MetaPoison: Practical General-purpose Clean-label Data Poisoning

W. Ronny Huang [*1] Jonas Geiping [*1 2] Liam Fowl [1] Gavin Taylor [3] Tom Goldstein [1]

Abstract

Data poisoning—the process by which an attacker takes control of a model by making imperceptible changes to a subset of the training data—is an emerging threat in the context of neural networks. Existing attacks for data poisoning have relied on hand-crafted heuristics. Instead, we pose crafting poisons more generally as a bi-level optimization problem, where the inner level corresponds to training a network on a poisoned dataset and the outer level corresponds to updating those poisons to achieve a desired behavior on the trained model. We then propose MetaPoison, a first-order method to solve this optimization quickly. MetaPoison is effective: it outperforms previous clean-label poisoning methods by a large margin under the same setting. MetaPoison is robust: its poisons transfer to a variety of victims with unknown hyperparameters and architectures. MetaPoison is also general-purpose, working not only in fine-tuning scenarios, but also for end-to-end training from scratch with remarkable success, e.g. causing a target image to be misclassified 90% of the time via manipulating just 1% of the dataset. Additionally, MetaPoison can achieve arbitrary adversary goals not previously possible—like using poisons of one class to make a target image don the label of another arbitrarily chosen class. Finally, MetaPoison works in the real-world. We demonstrate successful data poisoning of models trained on Google Cloud AutoML Vision.

1. Introduction

Neural networks are susceptible to a range of security vulnerabilities that compromise their real-world reliability. The bulk of work in recent years has focused on evasion attacks (Szegedy et al., 2013; Athalye et al., 2018), where an input is slightly modified at inference time to change a model's prediction. These methods rely on access to the inputs during inference, which is not always available in practice.

[*]Equal contribution [1]University of Maryland [2]University of Siegen [3]US Naval Academy. Correspondence to: W. Ronny Huang <wronnyhuang@gmail.com>.

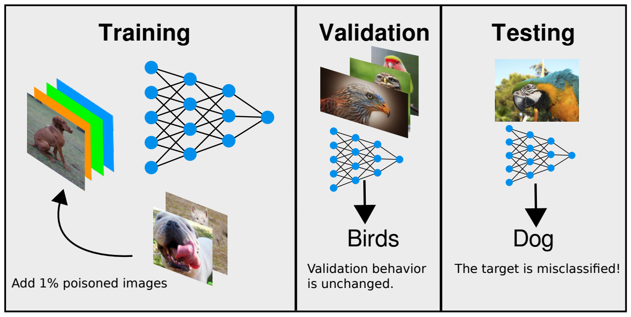

Training
Validation
Testing
Birds
Dog
Add 1% poisoned images
Validation behavior is unchanged.
The target is misclassified!

Figure 1. High-level schematic of clean-label data poisoning. Our goal as the attacker is to classify some bird image (here: the parrot) as a dog. To do so, a small fraction of the training data is imperceptibly modified before training. The network is then trained from scratch with this modified dataset. After training, validation performance is normal (eagle, owl, lovebird). However, the minor modifications to the training set cause the (unaltered) target image (parrot) to be misclassified by the neural network as "dog" with high confidence.

Another type of attack is that of backdoor attacks (Turner et al., 2019; Chen et al., 2017; Liu et al., 2017). Like evasion attacks, backdoor attacks require adversary access to model inputs during inference; notably backdoor "triggers" need to be inserted into the training data and then later into the input at inference time.

Unlike evasion and backdoor attacks, *data poisoning* does not require attacker control of model inputs at inference time. Here the attacker controls the model by adding manipulated images to the training set. These malicious images can be inserted into the training set by placing them on the web (social media, multimedia posting services, collaborative-editing forums, Wikipedia) and waiting for them to be scraped by dataset harvesting bots. They can also be added to the training set by a malicious insider who is trying to avoid detection. A data corpus can also be compromised when arbitrary users may contribute data, such as face images for a recognition and re-identification system.

Data poisoning attacks have been explored for classical scenarios (Biggio et al., 2012; Steinhardt et al., 2017; Burkard & Lagesse, 2017) which allow both training inputs and labels to be modified. However, it is possible to make poison perturbations imperceptible to a human observer, as they are in evasion attacks. Attacks of this type, schematic in Figure 1, are often referred to as *clean-label* poisoning attacks (Koh & Liang, 2017; Shafahi et al., 2018) because poison images appear to be unmodified and labeled correctly. The perturbed images often affect classifier behavior on a *specific* target instance that comes along after a system is deployed, without affecting behavior on other inputs, making clean-label attacks insidiously hard to detect.

In the dominant Feature Collision (FC) approach to clean-label poisoning (Shafahi et al., 2018; Zhu et al., 2019), perturbations are used to modify a training image (e.g., a tree) so that its feature representation is nearly identical to that of a chosen target image (e.g., a stop sign). After the victim fine tunes their model on the poisoned image, the model cannot distinguish between the poison and target image, causing it to misclassify the stop sign as a tree. FC is a heuristic with limited applicability; the attacker must have knowledge of the feature extractor being used, and the feature extractor cannot substantially change after the poison is introduced. For this reason, FC attacks only work on fine-tuning and transfer learning pipelines, and fail when the victim trains their model from scratch. Also, FC is not general-purpose—an attacker could have objectives beyond causing a single target instance to be misclassified with the label of the poison.

We propose MetaPoison, an algorithm for crafting poison images that manipulate the victim's training pipeline to achieve arbitrary model behaviors. First, we show that MetaPoison outperforms FC methods by a large margin in the established setting where a victim fine-tunes a pre-trained model. We then demonstrate that MetaPoison succeeds in the challenging context where the victim trains *from scratch* using random initializations. Next, we show that MetaPoison attacks transfer to "black box" settings where the victim uses training hyperparameters and network architectures unknown to the attacker, making this attack practical in the real-world. We verify its practicality by successfully poisoning models on the Google Cloud AutoML Vision platform. Finally we show that MetaPoison enables a range of alternate adversary objectives beyond what is achievable using feature collision, paving the way toward full, arbitrary control of the victim model.

End-to-end code as well as pre-crafted poisons are available at `www.github.com/wronnyhuang/metapoison`. We encourage the reader to download, train, and evaluate our poisoned CIFAR-10 dataset on their own CIFAR-10 training pipeline to verify MetaPoison's effectiveness. We discuss a few broader implications of this work in Sec. J. Note, MetaPoison can also be used for non-nefarious purposes, such as copyright enforcement. For example, it can "watermark" copyrighted data with diverse, undetectable perturbations. The model can then be queried with the target (known only to copyright holder) to determine whether the copyrighted data was used to train the model.

2. Method

2.1. Poisoning as constrained bi-level optimization

Suppose an attacker wishes to force an unaltered target image x_t of their choice to be assigned an incorrect, *adversarial* label y_{adv} by the victim model. The attacker can add n poison images $X_p \in [0, 255]^{n\times m}$, where m is the number of pixels, to the victim's clean training set X_c. The optimal poison images X_p^* can be written as the solution to the following optimization problem:

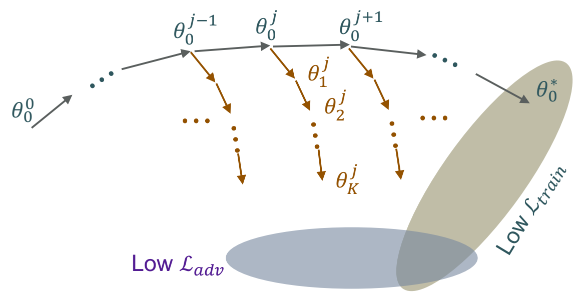
$$X_p^* = \operatorname*{argmin}_{X_p} \mathcal{L}_{\text{adv}}(x_t, y_{\text{adv}}; \theta^*(X_p)), \tag{1}$$

where in general $\mathcal{L}(x, y; \theta)$ is a loss function measuring how accurately a model with weights θ assigns label y to input x. For $\mathcal{L}_{\text{adv}}$ we use the Carlini & Wagner (2017) function and call it the *adversarial loss*. $\theta^*(X_p)$ are the network weights found by training on the poisoned training data $X_c \cup X_p$, which contain the poison images X_p mixed in with mostly clean data $X_c \in [0, 255]^{N\times m}$, where $N \gg n$. Note that (1) is a bi-level optimization problem (Bard, 2013) – the minimization for X_p involves the weights $\theta^*(X_p)$, which are themselves the minimizer of the training problem,

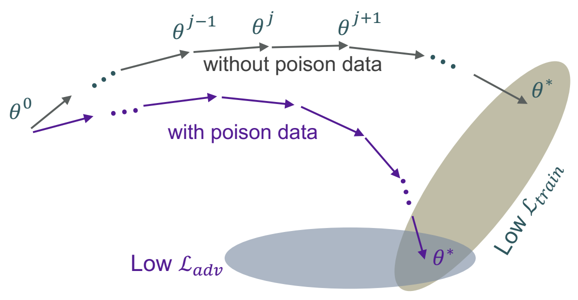
$$\theta^*(X_p) = \operatorname*{argmin}_{\theta} \mathcal{L}_{\text{train}}(X_c \cup X_p, Y; \theta), \tag{2}$$

where $\mathcal{L}_{\text{train}}$ is the standard cross entropy loss, and $Y \in \mathbb{Z}^{N+n}$ contains the correct labels of the clean and poison images. Thus, (1) and (2) together elucidate the high level formulation for crafting poison images: find X_p such that the *adversarial loss* $\mathcal{L}_{\text{adv}}(x_t, y_{\text{adv}}; \theta^*(X_p))$ is minimized after training.

For the attack to be inconspicuous, each poison example x_p should be constrained to "look similar" to a natural base example x. A number of perceptually aligned perturbation models have been proposed (Engstrom et al., 2019; Wong et al., 2019; Ghiasi et al., 2020). We chose the ReColorAdv perturbation function of Laidlaw & Feizi (2019), which applies a function f_g, with parameters g, and an additive perturbation map δ, resulting in a poison image $x_p = f_g(x) + \delta$. The function $f_g(x)$ is a pixel-wise color remapping $f_g : \mathcal{C} \to \mathcal{C}$ where $\mathcal{C}$ is the 3-dimensional LUV color space. To ensure that the perturbation is minimal, f_g can be bounded such that for every pixel x_i, $\|f_g(x_i) - x_i\|_\infty < \epsilon_c$, and δ can be bounded such that $\|\delta\|_\infty < \epsilon$. We use the standard additive bound of $\epsilon = 8$ and a tighter-than-standard color bound of $\epsilon_c = 0.04$ to further obscure the perturbation (Laidlaw & Feizi (2019) used $\epsilon_c = 0.06$). To enforce these bounds, we optimize for X_p with PGD (Madry et al., 2017), projecting the outer-parameters g and δ back to their respective ϵ_c and ϵ balls after every gradient step. Example poisons along with their clean counterparts used in this work are shown later in Figure 4 (top left).



(a) Crafting poisons



(b) Victim evaluation

Figure 2. Schematic of MetaPoison in weight space. (Left) During the poison crafting stage, the computation graph is unrolled by K SGD steps in order to compute a perturbation to the poisons, $\nabla_{X_p}\mathcal{L}_{\text{adv}}$. Optimally perturbed poisons steer the weights (red arrows) toward regions of low $\mathcal{L}_{\text{adv}}$ regardless of which training stage θ_0^j the poisons are inserted. (Right) When the victim trains on the poisoned data, the weight trajectory is collectively and implicitly steered to regions of low $\mathcal{L}_{\text{adv}}$ whilst the learner explicitly drives the weights to regions of low $\mathcal{L}_{\text{train}}$.

2.2. Crafting versatile poisoning examples efficiently

Minimizing the full bi-level objective in (1)-(2) is intractable; a computation graph that explicitly unrolls 10^5 SGD steps would not fit on any modern-day machine. Furthermore, the gradients would likely vanish or explode in such a deep graph. We can, however, build a computation graph that unrolls the training pipeline through *a few* SGD steps. This allows us to "look ahead" in training and view how perturbations to poisons *now* will impact the adversarial loss a few steps into the future. For example, the process of unrolling two inner-level SGD steps to compute an outer-level update on the poisons would be

$$\begin{aligned}\theta_1 &= \theta_0 - \alpha\nabla_\theta\mathcal{L}_{\text{train}}(X_c \cup X_p, Y; \theta_0)\\ \theta_2 &= \theta_1 - \alpha\nabla_\theta\mathcal{L}_{\text{train}}(X_c \cup X_p, Y; \theta_1)\\ X_p^{i+1} &= X_p^i - \beta\nabla_{X_p}\mathcal{L}_{\text{adv}}(x_t, y_{\text{adv}}; \theta_2),\end{aligned} \tag{3}$$

where α and β are the learning rate and crafting rate, respectively. Once optimized, the poisons should ideally lower the adversarial loss $\mathcal{L}_{\text{adv}}$ after a few SGD steps *regardless* of where they're inserted along the network trajectory, as illustrated in Figure 2 (left). When used to train a victim model, the poisons should implicitly "steer" the weights toward low $\mathcal{L}_{\text{adv}}$ whilst the learner drives the weights toward low training loss $\mathcal{L}_{\text{train}}$. When poisoning is successful, the victim should end up with a weight vector that achieves both low $\mathcal{L}_{\text{adv}}$ and $\mathcal{L}_{\text{train}}$ despite only having explicitly trained for low $\mathcal{L}_{\text{train}}$, as shown in Figure 2 (right).

The idea of unrolling the training pipeline to solve an outer optimization problem has been applied to meta-learning (Finn et al., 2017), neural architecture search (Liu et al., 2018), and even simple data poisoning (Muñoz-González et al., 2017). However, unique challenges arise when using this method for general-purpose data poisoning.

First, the training process depends on weight initialization and minibatching order, which are determined at random and unknown to the attacker. This is in contrast to meta-learning and architecture search, where the same agent has purview into both the inner (training their *own* networks) and outer processes (updating the weight initialization or architecture). Second, we find that jointly training the network weights and poisons causes poisons to overfit to the weights in the last epochs of training while forgetting how to steer newly initialized weights toward low $\mathcal{L}_{\text{adv}}$ regions in the beginning of the training journey.

We address these two challenges via *ensembling* and *network re-initialization*. Poisons are crafted using an ensemble of models, each on a different epoch of training. At each step we alternate between updating poisons and updating model weights. Once a model has trained for a sentinel number of epochs, it is re-initialized with randomly sampled weights and started from scratch. Specifically, each update has the form

$$X_p^{i+1} = X_p^i - \frac{\beta}{N_{\text{epoch}}}\sum_{j=0}^{N_{\text{epoch}}} \nabla_{X_p}\mathcal{L}_{\text{adv}}\Big|_{\theta^j}. \tag{4}$$

$\mathcal{L}_{\text{adv}}\Big|_{\theta^j}$ is the adversarial loss after a few look-ahead SGD steps on the poisoned dataset starting from weights θ^j (taken from the j-th epoch). This unrolled gradient is explicitly written out in (3), where θ^j here corresponds to θ_0 in (3). Note the set of weight vectors $\{\theta^j\}$ are drawn from different stages of training with different random initializations, thus preventing overfitting to any single stage of training or any particular initialization.

Finally, note that we use a standard network training process in which minibatches are sampled at random. For this reason, each minibatch has some subset of poison images, or sometimes no poisons at all. This raises a synchronization issue because the update rule in (4) expects all poisons—or at least the same subset of poisons across the ensemble—to be present at the current step. We address this by training the network for an entire epoch during which each example appears once. When a poison x_p appears in the current minibatch, the computation graph is unrolled and the update to that poison $\nabla_{x_p}\mathcal{L}_{adv}$ is stored. At the end of the epoch, the updates for all poisons are concatenated into $\nabla_{X_p}\mathcal{L}_{adv}$ and only then is the outer step (4) taken once. The entire process is outlined in Algorithm 1.[1] Crafting a set of 500 poisons for 60 steps on CIFAR-10 takes about 6 GPU-hours.

Algorithm 1 Crafting poison examples via MetaPoison

```
1: **Input** Training set of images and labels $(X, Y)$ of size $N$, target image $x_t$, adversarial class $y_{\text{adv}}$, $\epsilon$ and $\epsilon_c$ thresholds, $n \ll N$ subset of images to be poisoned, $T$ range of training epochs, $M$ randomly initialized models.
2: **Begin**
3: Stagger the $M$ models, training the $m$th model weights $\theta_m$ up to $\lfloor mT/M \rfloor$ epochs
4: Select $n$ images from the training set to be poisoned, denoted by $X_p$. Remaining clean images denoted $X_c$
5: For $i = 1, \ldots, C$ crafting steps:
6:     For $m = 1, \ldots, M$ models:
7:         Copy $\tilde{\theta} = \theta_m$
8:         For $k = 1, \ldots, K$ unroll steps:
9:             $\tilde{\theta} = \tilde{\theta} - \alpha\nabla_{\tilde{\theta}}\mathcal{L}_{\text{train}}(X_c \cup X_p, Y; \tilde{\theta})$
10:        Store adversarial loss $\mathcal{L}_m = \mathcal{L}_{\text{adv}}(x_t, y_{\text{adv}}; \tilde{\theta})$
11:        Advance epoch $\theta_m = \theta_m - \alpha\nabla_{\theta_m}\mathcal{L}_{\text{train}}(X, Y; \theta_m)$
12:        If $\theta_m$ is at epoch $T + 1$:
13:            Reset $\theta_m$ to epoch 0 and reinitialize
14:    Average adversarial losses $\mathcal{L}_{\text{adv}} = \sum_{m=1}^{M} \mathcal{L}_m / M$
15:    Compute $\nabla_{X_p}\mathcal{L}_{\text{adv}}$
16:    Update $X_p$ using Adam and project onto $\epsilon, \epsilon_c$ ball
17: **Return** $X_p$
```

3. Experiments

Our experiments on CIFAR-10 consist of two stages: poison crafting and victim evaluation. In the first stage, we craft poisons on surrogate models and save them for evaluation. In the second stage, we insert the poisons into the victim dataset, train the victim model from scratch on this dataset, and report the attack success and validation accuracy. We declare an attack successful only if the target instance x_t is classified as the adversarial class y_{adv}; it doesn't count if the target is classified into any other class, incorrect or not. The attack success rate is defined as the number of successes over the number of attacks attempted.

Unless stated otherwise, our experimental settings are as follows. The first n examples in the poisons' class are used as the base images in the poison set X_p and are perturbed, while the remaining images in CIFAR-10 are used as the clean set X_c and are untouched. The target image is taken from the CIFAR-10 test set. We perform 60 outer steps when crafting poisons using the Adam optimizer with an initial learning rate of 200. We decay the outer learning rate (i.e. crafting rate) by 10 every 20 steps. Section A analyzes the crafting process in more depth. Each inner learner is unrolled by $K = 2$ SGD steps to compute $\mathcal{L}_{\text{adv}}$. Other K's were explored in Sec. F. An ensemble of 24 inner models is used, with model i trained until the ith epoch. A batchsize of 125 and learning rate of 0.1 are used and the weights are updated via SGD. We leave weight decay and data augmentation off by default, but analyze performance with them on in Sec. 3.3. By default, we use the same 6-layer ConvNet architecture with batch normalization as Finn et al. (2017), henceforth called ConvNetBN, but other architectures are demonstrated throughout the paper too. Outside of Sec. 3.3, the same inner model settings are used for victim evaluation. We train each victim to 200 epochs, decaying the learning rate by 10 at epochs 100 and 150. Sec. B shows example victim training curves with poisoned data.

3.1. Comparison to previous work

Previous works on clean-label poisoning from Koh & Liang (2017), Shafahi et al. (2018), and Zhu et al. (2019) attack models that are pre-trained on a clean/standard dataset and then fine-tuned on a poisoned dataset. We compare MetaPoison to Shafahi et al. (2018), who crafted poisons using feature collisions in a white-box setting where the attacker has knowledge of the pretrained CIFAR-10 AlexNet-like classifier weights. They assume the victim fine-tunes using the entire CIFAR-10 dataset. Critical to their success was the "watermark trick": they superimpose a 30% opacity watermark of the target image onto every poison image before crafting applying their additive perturbation. For evaluation, Shafahi et al. (2018) compared two poison-target class pairs, frog-airplane and dog-bird, and ran poisoning attacks on 30 randomly selected target instances for each class pair. They also varied the number of poisons. We replicate this scenario as closely as possible using poisons crafted via MetaPoison. Since the perturbation model in Shafahi et al. (2018) was additive only (no ReColorAdv), we set $\epsilon_c = 0$ in MetaPoison. We also use the watermark trick at 30% opacity. To apply MetaPoison in the fine-tuning setting, we first pretrain a network to 100 epochs and use this fixed network to initialize weights when crafting poisons or running victim evaluations.

Our comparison results are presented in Figure 3 (top). Notably, 100% attack success is reached at about 25 poisons out of 50000 total training examples, or a poison budget of only 0.05%. In general, MetaPoison achieves much higher success rates at much lower poison budgets as compared to

[1]For brevity in Algorithm 1, we write as if unrolled SGD steps are taken using the full dataset. In practice they are taken on minibatches and repeated until the full dataset is flushed once through. Aside from minor details, the two are equivalent.

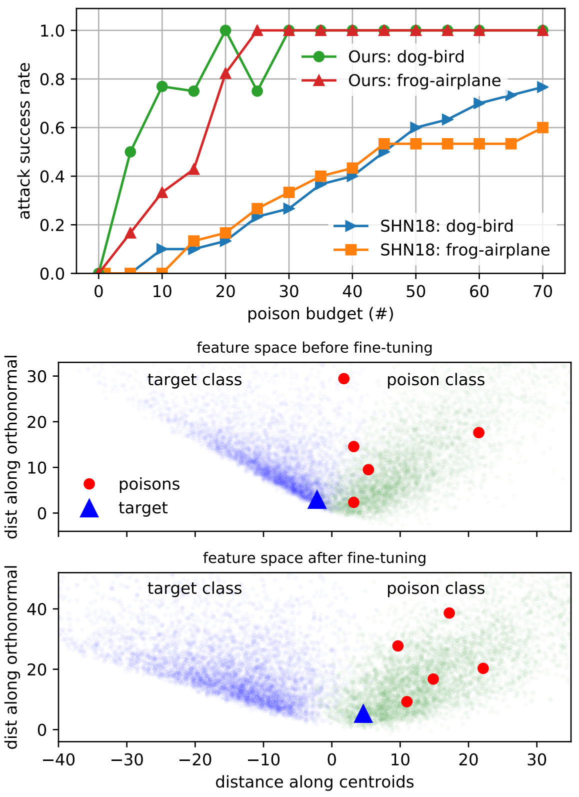


Figure 3. Comparison with Shafahi et al. (2018) (SHN18) under the same fine-tuning conditions for a watermark opacity of 30%. (Top) Success rates. (Middle/Bottom) Penultimate-layer feature representation visualization of the target and poison class examples before/after fine-tuning on the poisoned dataset. Like Shafahi et al., we project the representations along the line connecting centroids of the two classes (x-axis) and along the orthogonal component of the classification-layer parameter vector (y-axis). This projection ensures that we are able to see activity at the boundaries between these two classes.

the previous method, showcasing the strength of its poisons to alter victim behavior in the case of fine-tuning.

The fine-tuning scenario also provides a venue to look closer into the mechanics of the attack. In the feature collision attack (Shafahi et al., 2018), the poisons are all crafted to share the same feature representation as that of the target in the penultimate layer of the network. When the features in the penultimate layer are visualized, the poisons are overlapped, or collided, with the target (Figure 3b in original paper). We perform the same visualization in Figure 3 (middle) for a successful attack with 5 poisons using MetaPoison. Intriguingly, our poisons do *not* collide with the target, implying that they employ some other mechanism to achieve the same objective. They do not even reside in the target class distribution, which may render neighborhood conformity tests such as Papernot & McDaniel (2018); Peri et al. (2019) less effective as a defense. Figure 3 (bottom) shows the feature representations after fine-tuning. The target switches sides of the class boundary, and dons the incorrect poison label. These visualizations show that MetaPoisons cause feature extraction layers to push the target in the direction of the poison class without relying on feature collision-based mechanics. Indeed, the poisoning mechanisms of MetaPoison are *learned* rather than hand-crafted; like adversarial examples, they likely do not lend themselves to an easy human interpretation.

3.2. Victim training from scratch

Usually fine-tuning datasets tend to be small, domain-specific, and well-curated; from a practical perspective, it may be harder to inject poisons into them. On the other hand, large datasets on which models are (pre)trained from scratch are often scraped from the web, making them easier to poison. Thus, a general-purpose poisoning method that works on models trained from scratch would be far more viable. Yet *no* prior clean-label poisoning attack has been demonstrated against networks trained from scratch. This is because existing feature collision-based poisoning (Shafahi et al., 2018; Zhu et al., 2019) requires a pre-existing feature extractor on which to craft a feature collision.

In this section, we demonstrate the viability of MetaPoison against networks trained from scratch. For consistency, we focus on the same dog-bird and frog-plane class pairs as in previous work. To be thorough, however, we did a large study of all possible class pairs in Sec. C and showed that these two class pairs are representative in terms of poisoning performance. We also found that even within the same poison-target class pair, different target images resulted in different poisoning success rates (Sec. D). Therefore, for each class pair, we craft 10 sets of poisons targeting the corresponding first 10 image IDs of the target class taken from the CIFAR-10 test set and aggregate their results. Finally, different training runs have different results due to training stochasticity (Sec. B). Therefore, for each set of poisons, we train 6 victim networks with different random seeds and record the target image's label inferred by each model. In all, there are 60 labels, or votes: 6 for each of 10 different target images. We then tally up the votes for each class. For example, Figure 4 (lower left) shows the number of votes each label receives for the target birds out of 60 models. In unpoisoned models, the true class (bird) receives most of the votes. In models where just 1% of the dataset is poisoned, the target birds get incorrectly voted as dog a majority of the time. Examples of some poison dogs along with their clean counterparts, as well as one of the target birds, are shown in Figure 4 (top left). More are displayed in Sec. K. Section E shows success rate and perceptual differences for various perturbation magnitudes.

In Figure 4 (top right), we repeat the experiments for multiple poison budgets and network architectures. The success

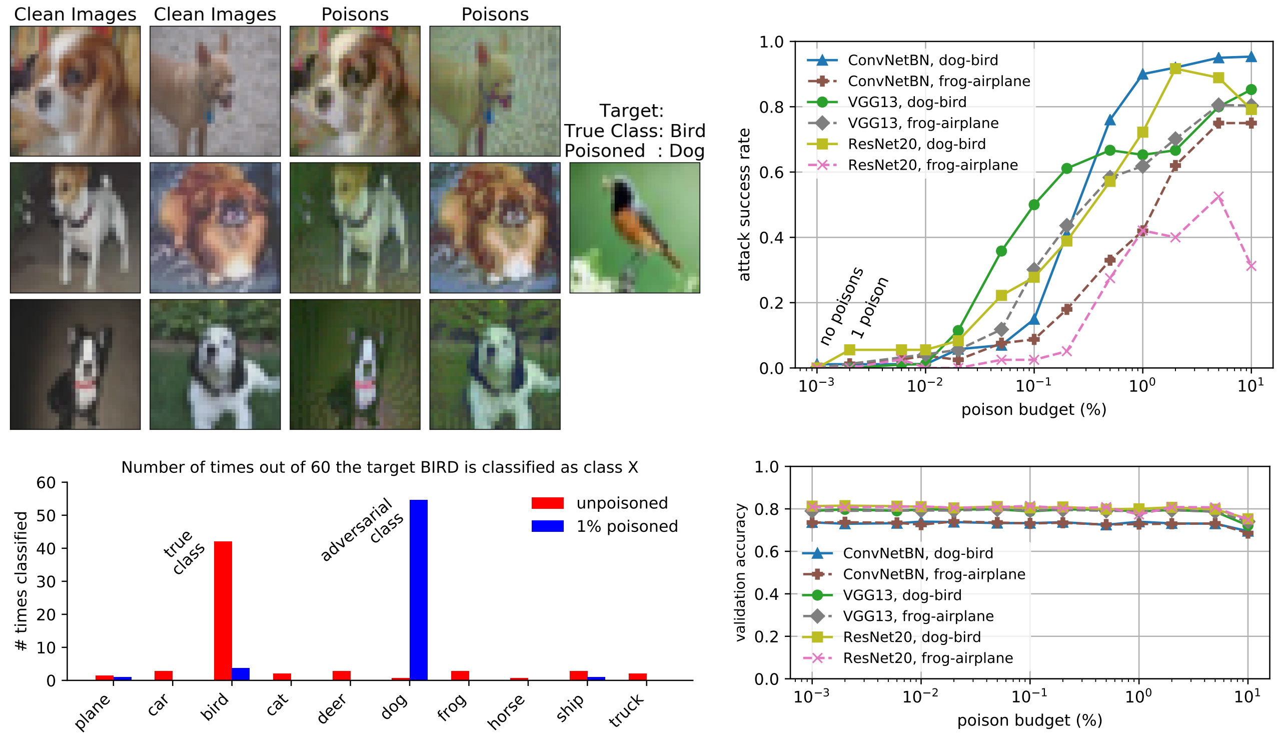


Figure 4. Poisoning end-to-end training from scratch. (Top left) Examples of poisoned training data. (Bottom left) Tally of the classes into which target birds are classified over 60 victim models on ConvNetBN. 6 models are trained with different random seeds for each of 10 target birds, totaling 60 victim models. (Top right) Attack success rate vs poison budget for different architectures and poison-target class pairs. (Bottom right) Validation accuracy of poisoned models. Note that a poison budget of 10^{-3} percent is equivalent to zero poisons as the training set size is 50k.

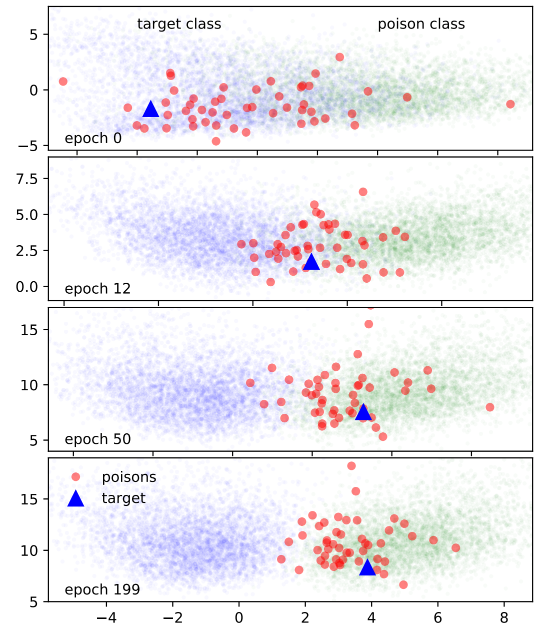


Figure 5. Penultimate layer visualization as a function of epoch for a successful train-from-scratch attack of 50 poisons. The target (blue triangle) is moved toward the poison distribution by the crafted poisons.

rates drop most steeply between 1% and 0.1%, but remain viable even beyond 0.01% budget. Remarkably, even a single perturbed dog can occasionally poison ResNet-20. The unexpected drop in success above 5% budget may be attributable to the large number of poisons overfitting the particular weight trajectories seen during crafting. In Figure 4 (bottom right), we verify that our poisons cause negligible effect on overall model performance except for a small drop at 10% poison budget.

We again gain clues to the poisoning mechanism through feature space visualization. We view the penultimate layer features at multiple epochs in Figure 5. In epoch 0, the classes are not well separated, since the network is trained from scratch. As training evolves the earlier-layer feature extractors learn to separate the classes in the penultimate layer. They do not learn to separate the target instance, but they instead steadily usher the target from its own distribution to the poison class distribution as training progresses to epoch 199, impervious to the forces of class separation. In addition, the distribution of poisons seems to be biased toward the side of the target class. This suggests that the poisons adopt features similar to the specific target image to such an extent that the network no longer has the capacity to class-separate the target from the poisons. See Sec. I for additional visualizations and insights.

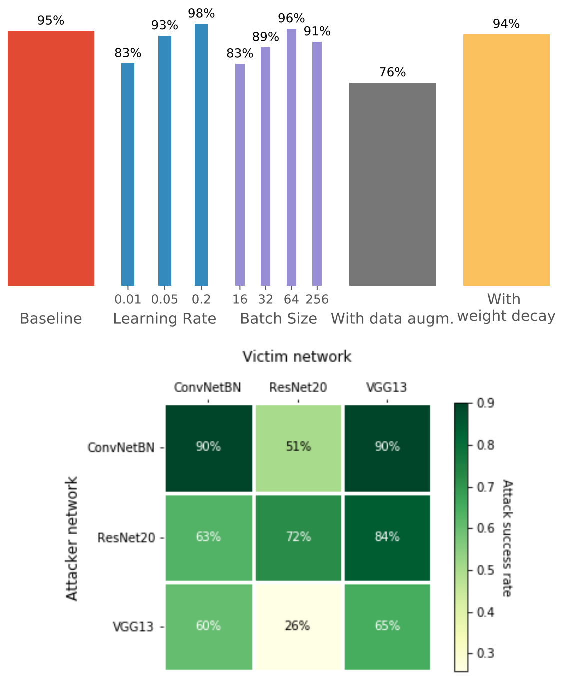


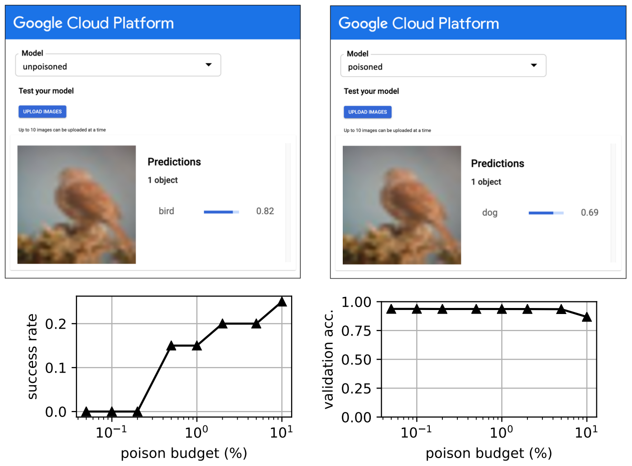
Figure 6. Robustness to different training settings and architectures. (Top) Success rate on a victim ConvNetBN with different training settings. (Bottom) Success rate of poisons crafted on one architecture and evaluated on another.

3.3. Robustness and transferability

So far our results have demonstrated that the crafted poisons transfer to new initializations and training runs. Yet often the exact training settings and architecture used by the victim are also different than the ones used to craft the poisons. We investigate the robustness of our poisons to changes in these choices. In Figure 6 (top), we train victim models with different training settings, like learning rate, batch size, and regularizers, on poisons crafted using ConvNetBNs with a single baseline setting (0.1 learning rate, 125 batch size, no regularization). Again using the dog-bird pair, poison dogs were crafted for 8 different target birds and 24 victim models were trained per target. Our results show that the poisons are overall quite robust to changes. Data augmentation (standard CIFAR-10 procedure of random crops and horizontal flips) and large changes in learning rate or batch size cause some, but not substantial degradation in success. The robustness to data augmentation is surprising; one could've conceived that the relatively large changes by data augmentation would nullify the poisoning effect. Architecture transferability was studied in Zhu et al. (2019) for fine-tuned models. Here we study them in trained-from-scratch models using MetaPoison. In Figure 6 (bottom), we craft poisons on one architecture and naively evaluate them on another, in contrast to Zhu et al. (2019) where poisons are crafted on an ensemble of architectures. The performance is remarkable. ConvNetBN, VGG13, and ResNet20 are very different architectures, yet poisons transfer between them. Interestingly the attack success rate is non-symmetric. Poisons created on VGG13 do not work nearly as well on ResNet20 as ResNet20 poisons on VGG13. One explanation for this is that VGG13 does not have batch normalization, which may have a regularizing effect on poison crafting.

3.4. Real-world poisoning on Google Cloud AutoML

We further evaluate the robustness of MetaPoison on Google Cloud AutoML Vision,[2] a real-world, enterprise-grade, truly *black-box* victim learning system. Designed for the end-user, Cloud AutoML hides all training and architecture information, leaving the user only the ability to upload a dataset and specify wallclock training budget and model latency grade. For each model, we upload CIFAR-10, poisoned with the same poison dogs crafted earlier on ResNet20, and train for 1000 millinode-hours on the mobile-high-accuracy-1 grade. After training, we deploy the poisoned model on Google Cloud and upload the target bird for prediction. Figure 7 shows web UI screenshots of the prediction results on unpoisoned (top left) and poisoned (top right) Cloud AutoML models. MetaPoison works even in a realistic setting such as this. To quantify performance, we train 20 Cloud AutoML victims, 2 for each of the first 10 target birds, and average their results in Figure 7 (bottom left and right) at various budgets. At poison budgets as low as 0.5%, we achieve success rates of >15%, with little impact on validation accuracy. These results show that data poisoning presents a credible threat to real-world systems; even popular ML-as-a-service systems are not immune to such threats.


Model: unpoisoned

Predictions
1 object
bird 0.82

Model: poisoned

Predictions
1 object
dog 0.69

Figure 7. Data poisoning Google Cloud AutoML Vision models. Web UI screenshots of prediction results on target bird by Cloud AutoML models trained on (Top left) clean and (Top right) poisoned CIFAR-10 datasets. Portions of the screenshot were cropped and resized for a clearer view. (Bottom left) Success rates and (Bottom right) validation accuracies averaged across multiple targets and training runs as a function of poison budget on the poison-dog-target-bird pair.

[2] `cloud.google.com/automl`

3.5. Alternative poisoning schemes

Thus far we have discussed targeted poisoning attacks under a *collision* scheme: inject poisons from class y_p to cause a specific instance in class y_t to be classified as y_p, where $y_p \neq y_t$. In other words, the adversarial class is set to be the poison class, $y_{\text{adv}} = y_p$. This scheme was specially suited for the feature collision approach, and indeed is the only scheme possible under that method. However, it is only a subset of the space of possible schemes $\mathcal{Y}_{scheme} : (y_p, y_t, y_{\text{adv}})$. Since MetaPoison does not use hand-crafted heuristics, but learns to craft poison examples directly from a given outer objective $\mathcal{L}(x_t, y_{\text{adv}}; \theta^*(X_p))$, the attacker can define a wide variety of other objectives. We will demonstrate MetaPoison's versatility through two example schemes.

3.5.1. Self-concealment scheme

Here, the poisons are injected to cause a target image in the *same* class to be misclassified, i.e. $y_p = y_t \neq y_{\text{adv}}$. This could be employed by an attacker who presents multiple poisoned views of himself (e.g. reference face images), so that he can later *evade* detection when presenting the target view of himself. To craft a self-concealment attack, we set our adversarial loss function as $\mathcal{L}_{\text{adv}}(X_p) = -\log\left[1 - p_{\theta^*(X_p)}(x_t, y_t)\right]$, so that higher misclassification of the target lowers the loss. We evaluate the self-concealment scheme on two poison-target pairs, bird-bird and airplane-airplane. Note that the poison and target classes are identical by definition of the scheme. We use a poison budget of 10% and craft 5 sets of poisons targeting the first 5 bird or airplane IDs. We then train 4 victim networks for each set of poisons totaling 20 votes for each pair. We tally up votes in Figure 8 (top). For unpoisoned models, the correct label receives the clear majority as expected, while for poisoned models, the votes are distributed across multiple classes without a clear majority. Importantly the true class (bird or airplane) receives almost no votes. Using definition of success as misclassification of the target into any other class, the success rates are 100% and 95% for bird and airplane, respectively.

3.5.2. Multiclass-poison scheme

In cases where the number of classes is high, it can be difficult to assume a large poison budget for any single class. Here we demonstrate one solution: creating poisons coming from *multiple* classes acting toward the same adversary goal. For simplicity, we craft poisons from classes uniformly distributed across the 10 available classes (with 1% poison budget in each). Again our goal is to cause a target bird to be assigned an incorrect label y_{adv}. For each y_{adv} that we choose, we craft 10 sets of poisons corresponding to the first 10 target bird IDs. 6 victims are trained for each set of poisons, totaling 60 votes. We use a poison budget of 10%. Our results in Figure 8 (bottom) tally the votes into 9 histograms, one for each y_{adv} setting. For most of the 9 histograms, the class that receives the most votes is the adversarial class. On average, the adversarial class takes about 50% of the votes cast (i.e. 50% success). This attack shows that the poisons need not come exclusively from one class and that it's possible to use poisoning to arbitrary control victim label assignment.

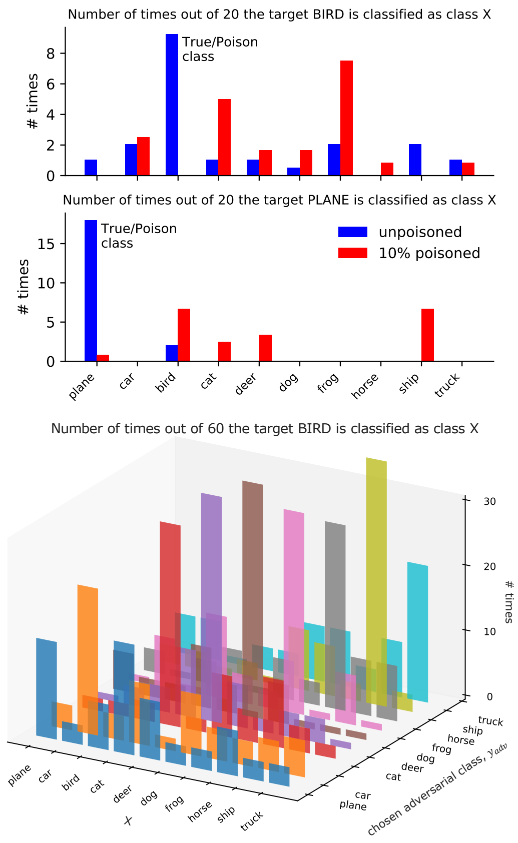


Figure 8. Alternative poisoning schemes. (Top) Self-concealment: images from the same class as the target are poisoned to "push" the target out of its own class. (Bottom) Multiclass-poisoning: images from multiple classes are poisoned to cause the target bird to be classified as a chosen adversarial class y_{adv}.

4. Conclusion

MetaPoison is an algorithm for finding dataset perturbations that control victim model behavior on specific targets. It outperforms previous clean-label poisoning methods on fine-tuned models, and achieves considerable success—for the first time—on models trained from scratch. It also enables novel attack schemes like self-concealment and multiclass-poison. The poisons are robust and practical, working even when transferred to black-box ML-as-a-service models. We hope MetaPoison establishes a baseline for data poisoning work and promotes awareness of this very real and emerging threat vector.

Acknowledgments

Software from www.comet.ml supplied necessary tools for monitoring and logging of our large number of experiments and datasets and graciously provided storage and increased bandwidth for the unique requirements of this research project. The authors had neither affiliation nor correspondence with the Google Cloud AutoML Vision team at the time of obtaining these results. Goldstein and his students were supported by the DARPA's GARD program, the DARPA QED for RML program, the Office of Naval Research MURI Program, the AFOSR MURI program, and the Sloan Foundation. LF was supported in part by LTS through Maryland Procurement Office and by the NSF DMS 1738003 grant. Taylor was supported by the Office of Naval Research.

References

Athalye, A., Carlini, N., and Wagner, D. Obfuscated Gradients Give a False Sense of Security: Circumventing Defenses to Adversarial Examples. *arXiv:1802.00420 [cs]*, February 2018. URL http://arxiv.org/abs/1802.00420.

Bard, J. F. *Practical Bilevel Optimization: Algorithms and Applications*. Springer Science & Business Media, March 2013. ISBN 978-1-4757-2836-1.

Biggio, B., Nelson, B., and Laskov, P. Poisoning Attacks against Support Vector Machines. *arXiv:1206.6389 [cs, stat]*, June 2012. URL http://arxiv.org/abs/1206.6389.

Burkard, C. and Lagesse, B. Analysis of causative attacks against svms learning from data streams. In *Proceedings of the 3rd ACM on International Workshop on Security And Privacy Analytics*, pp. 31–36, 2017.

Bursztein, E. Attacks against machine learning an overview, May 2018. URL https://elie.net/blog/ai/attacks-against-machine-learning-an-overview.

Carlini, N. and Wagner, D. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy (SP)*, pp. 39–57. IEEE, 2017.

Chen, X., Liu, C., Li, B., Lu, K., and Song, D. Targeted backdoor attacks on deep learning systems using data poisoning. *arXiv preprint arXiv:1712.05526*, 2017.

Engstrom, L., Tran, B., Tsipras, D., Schmidt, L., and Madry, A. Exploring the landscape of spatial robustness. In *International Conference on Machine Learning*, pp. 1802–1811, 2019.

Finn, C., Abbeel, P., and Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks, 2017.

Ghiasi, A., Shafahi, A., and Goldstein, T. Breaking certified defenses: Semantic adversarial examples with spoofed robustness certificates. *International Conference on Learning Representations*, 2020.

Kim, B., Wattenberg, M., Gilmer, J., Cai, C., Wexler, J., Viegas, F., and Sayres, R. Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (tcav). *arXiv preprint arXiv:1711.11279*, 2017.

Koh, P. W. and Liang, P. Understanding black-box predictions via influence functions. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1885–1894. JMLR. org, 2017.

Laidlaw, C. and Feizi, S. Functional adversarial attacks. In *Advances in Neural Information Processing Systems*, pp. 10408–10418, 2019.

Liu, H., Simonyan, K., and Yang, Y. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018.

Liu, Y., Ma, S., Aafer, Y., Lee, W.-C., Zhai, J., Wang, W., and Zhang, X. Trojaning attack on neural networks. *Purdue University Department of Computer Science Technical Reports*, 2017.

Madry, A., Makelov, A., Schmidt, L., Tsipras, D., and Vladu, A. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

Muñoz-González, L., Biggio, B., Demontis, A., Paudice, A., Wongrassamee, V., Lupu, E. C., and Roli, F. Towards Poisoning of Deep Learning Algorithms with Back-gradient Optimization. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, AISec '17, pp. 27–38, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-5202-4. doi: 10.1145/3128572.3140451.

Papernot, N. and McDaniel, P. D. Deep k-nearest neighbors: Towards confident, interpretable and robust deep learning. *CoRR*, abs/1803.04765, 2018. URL http://arxiv.org/abs/1803.04765.

Peri, N., Gupta, N., Huang, W. R., Fowl, L., Zhu, C., Feizi, S., Goldstein, T., and Dickerson, J. P. Deep k-nn defense against clean-label data poisoning attacks. *arXiv preprint arXiv:1909.13374*, 2019.

Sankararaman, K. A., De, S., Xu, Z., Huang, W. R., and Goldstein, T. The impact of neural network overparameterization on gradient confusion and stochastic gradient descent. *arXiv preprint arXiv:1904.06963*, 2019.

Shafahi, A., Huang, W. R., Najibi, M., Suciu, O., Studer, C., Dumitras, T., and Goldstein, T. Poison frogs! targeted clean-label poisoning attacks on neural networks. In *Advances in Neural Information Processing Systems*, pp. 6103–6113, 2018.

Steinhardt, J., Koh, P. W. W., and Liang, P. S. Certified defenses for data poisoning attacks. In *Advances in neural information processing systems*, 2017.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

Turner, A., Tsipras, D., and Madry, A. Clean-label backdoor attacks, 2019. URL https://people.csail.mit.edu/madry/lab/cleanlabel.pdf.

Wong, E., Schmidt, F., and Kolter, Z. Wasserstein adversarial examples via projected sinkhorn iterations. In *International Conference on Machine Learning*, pp. 6808–6817, 2019.

Zhu, C., Huang, W. R., Li, H., Taylor, G., Studer, C., and Goldstein, T. Transferable clean-label poisoning attacks on deep neural nets. In *International Conference on Machine Learning*, pp. 7614–7623, 2019.

Supplementary Material

A. Poison crafting curves

Our poisons in the main paper were all crafted with 60 outer steps, also called craft steps. Here we investigate the outer optimization process in more depth and show the potential benefits of optimizing longer. As a testbed, we consider poison frogs attacking a target airplane with a poison budget of 10%. During the crafting stage, the adversarial loss–we use the Carlini & Wagner (2017) loss here–is the objective to be minimized. This loss has the property that when it is less than zero, the target is successfully misclassified as the adversarial class. Conversely, when it is greater than zero, the target is classified into a class other than the adversarial class.

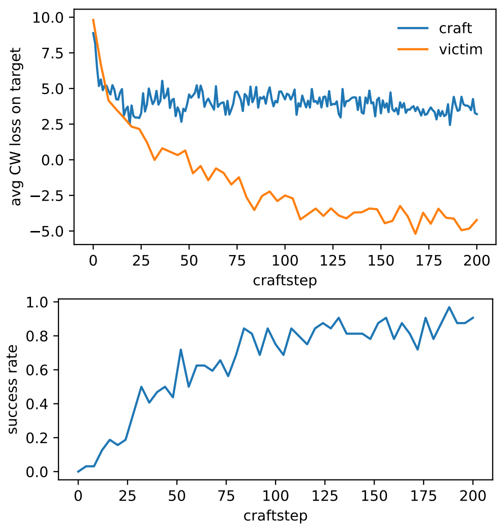


Figure 9. Ablation study on the number of craftsteps. (Top) The crafting adversarial loss (blue line), which is averaged across all 24 models in the ensemble, is the objective to be minimized in the outer loop of the bi-level optimization problem. We save the state of the poisons at every several craftsteps, fully train 20 victim models from scratch on each of those poisons, and plot the average adversarial loss on the target across those victim models (orange line). (Bottom) Attack success rate across the 20 victim models for each craft step.

The blue line in Figure 9 (top) shows the adversarial loss averaged over all the surrogate models during the crafting stage. It rapidly decreases up to craftstep 25 and then plateaus. It never sinks below zero, which means that inserting these poisons into a minibatch will not cause the model to misclassify the target two look-ahead SGD steps later, on average. However, it belies the fact that the cumulative effect of the poisons will collectively influence the model to misclassify the target after many SGD steps. Indeed, the fact that the adversarial loss (blue line) is decreased after 25 craft steps from $\sim$9 to $\sim$4 is an indication that the poisons provide a small nudge to the model toward misclassifying the target even after two look-ahead SGD steps, as compared to having no poisons.

The orange line in Figure 9 (top) shows the adversarial loss on the target image on poisoned victim models at each stage of crafting. To obtain this curve, we saved the state of the poisons every several craft steps, and trained 20 victim models from scratch on each of them. Interestingly, even though the crafting adversarial loss (blue line) plateaus, the effectiveness of the poisons continues to increase with the number of craft steps even up to 200 steps. Therefore, one cannot judge from the crafting curve alone how well the poisons will do during victim evaluation. Finally, Figure 9 (bottom) shows the corresponding attack success rate for the poisons at each craft step.

B. Victim training curves

In the main paper, we reported the attack success rates and validation accuracy at the *end* of victim training. In this section, we take a closer look at the effect of data poisoning at each step of training.

We again use the dog-bird class pair as our prototypical example and we randomly select target bird with ID 5. We train ResNet20 models with 3 different poisoning levels: unpoisoned, poisoned with 0.5% budget, and poisoned with 5% budget. Since the training of each victim model is inherently stochastic and we desire to see the overall effect of poisoning, we train 72 victim models with different seeds for each of these 3 poisoning levels. Figure 10 displays all 72 curves for each poisoning level. The training accuracy curves, in Figure 10 (top), show the models quickly overfitting to the CIFAR10 dataset after about 20k optimization steps, or 50 epochs. The rate of convergence is equal for all 3 poisoning levels. Likewise, the validation accuracy curves, Figure 10 (middle), converge to about 80% after 20k steps and are also indistinguishable between poisoning levels. These curves show that it is impossible to detect the presence of poisoning through looking at training or validation curves.

Next, we look at the evolution of the adversarial loss, or Carlini & Wagner (2017) loss, over optimization step in

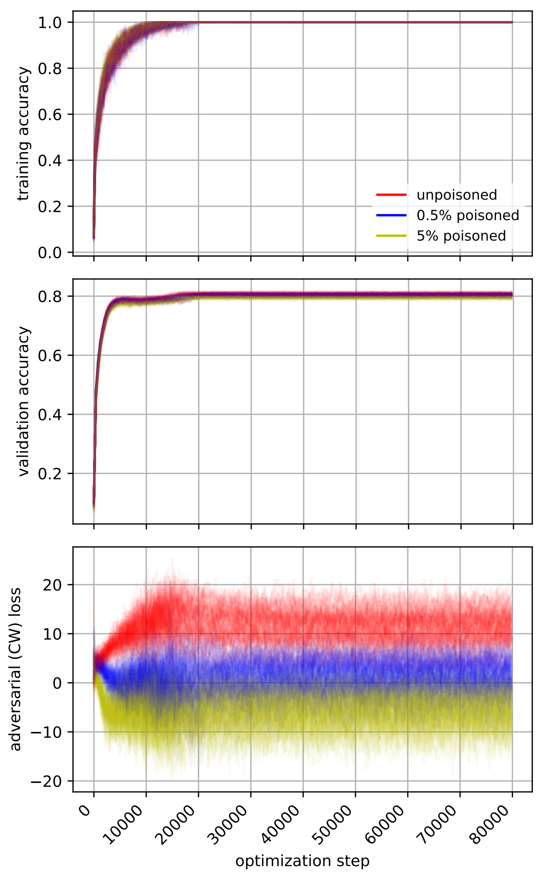


Figure 10. Training curves from scratch with different random seeds on poisoned and unpoisoned datasets over 200 epochs on ResNet20. (Top) Accuracy on training set perfectly overfits to CIFAR-10 after about 20k optimization steps, or 50 epochs. (Middle) Validation accuracy curve looks the same regardless of whether the dataset is poisoned or not. (Bottom) Carlini-Wagner (CW) adversarial loss on specific target bird (ID 5) as a function of optimization step. CW loss above zero indicates the target bird is classified correctly, while below zero indicates the target bird is misclassified as a dog. Unpoisoned models have adversarial loss entirely above zero, while 5% poisoned models have adversarial loss entirely below zero. 0.5% poisoned models have CW loss straddling both sides of zero.

Figure 10 (bottom). Recall that in the Carlini & Wagner (2017) loss, negative values correspond to attack success while positive values correspond to attack failure. Note also that, under almost all practical scenarios, the victim does not see this curve since they are unaware of the target image chosen by the adversary.

At the start, epoch 0, the adversarial loss of all models are at roughly the same level. As training proceeds, the adversarial loss trifurcates into 3 distinct levels corresponding to the 3 poisoning levels. The unpoisoned models see increasing adversarial loss up to fully positive values (perfect attack failure) of around 12 before they plateau, while the high 5% poisoned models see decreasing adversarial loss down to mostly negative values (near-perfect attack success) of around -6 before plateauing. The moderate 0.5% poisoned models see slight decrease in adversarial loss and hover around zero (some attack success) for the remainder of training. Compared to the training and validation curves, these adversarial loss curves fluctuate a lot both between optimization steps as well as between models. This is expected since they are the loss of a single image rather than an aggregate of images. Despite the fluctuation, however, the effect of different poisoning levels on the attack outcome is very clear.

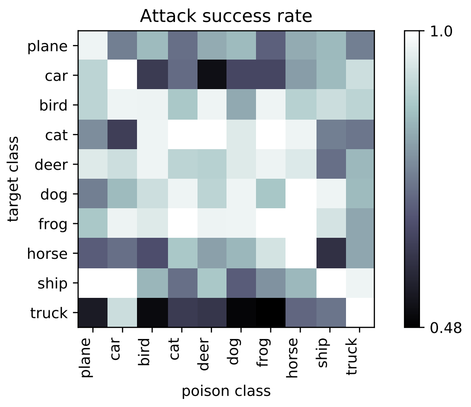


Figure 11. Success rates for all possible poison-target class pairs. Each success rate is the average of the first 5 unique targets with 2 victim training runs per unique target.

C. Performance on other poison-target class pairs

In the main paper, we primarily mimicked the two exemplary poison-target class pairs (dog-bird, frog-airplane) from previous work in Shafahi et al. (2018). To ensure that our results do not just happen to work well on these two pairs but rather works well for all class pairs, we perform a large study on all 100 possible poison-target pairs in CIFAR-10, shown in Figure 11.

For each pair, we craft with a poison budget of 10%, target the first 5 target IDs for that particular target class, and run 2 victim trainings from scratch for each pair, allowing the reported success rate to result from the average of 10 victim models. To enable such a large study within our computational runtime constraints, we use only 10% of the CIFAR-10 dataset as our training set. This is justified since we are interested here in the relative performance of different class pairs with respect to our exemplary class pairs (dog-bird, frog-airplane) on which we did full CIFAR-10 studies in the main paper.

The results show that poisoning can be successful under *all*

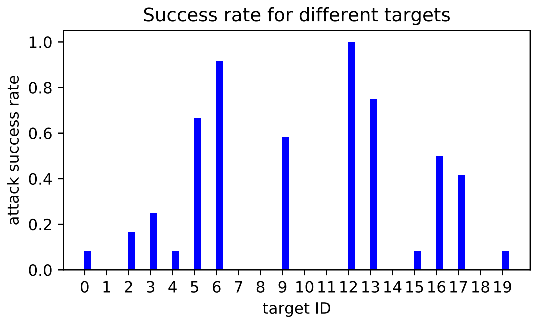


Figure 12. Success rates for the first 20 unique target airplanes for a poison frog target airplane situation. Each success rate is the average of 12 victim training runs.

class pair situations. Our exemplary pairs, dog-bird and frog-airplane, have average poisoning vulnerability relative to all the other pairs, with the dog-bird slightly more vulnerable than frog-airplane. The most difficult target class on which to cause misclassification is truck, while the most easy is frog. The least powerful poison class is truck, while the most powerful is tied between car, cat, deer, frog, and horse. The high success rates along the diagonal trivially indicate that it is easy to cause the target to be classified into the correct class.

D. Differences in success rates amongst different targets

It is also informative to see how the success rate varies amongst different choices of the target image for a fixed target class. Even though the target class is the same, different images within that class may have very different features, making it harder or easier for the poisons to compromise them. In Figure 12, we plot the attack success rates for the first 20 unique target airplanes when attacked by poison frogs. Each success rate is the result of 20 victim training runs. Indeed, the success rate is highly variable amongst different target images, indicating that the poisoning success is more dependent on the *specific* target image that the adversary wishes to attack rather than the choice of poison-target class pair.

E. Ablation study on perturbation magnitude

We present an ablation study for different additive and color perturbation bounds in Figure 13 for one particular dog-bird attack (bird ID 0) with 1% poison budget. While our experiments use modest values of $(\epsilon, \epsilon_c) = (8, 0.04)$, there is room to increase the bounds to achieve higher success without significant perceptual change as shown by an example poison dog in the figure. In contrast, even extremely minimal perturbations $(\epsilon, \epsilon_c) = (2, 0.02)$ can achieve notable poisoning.

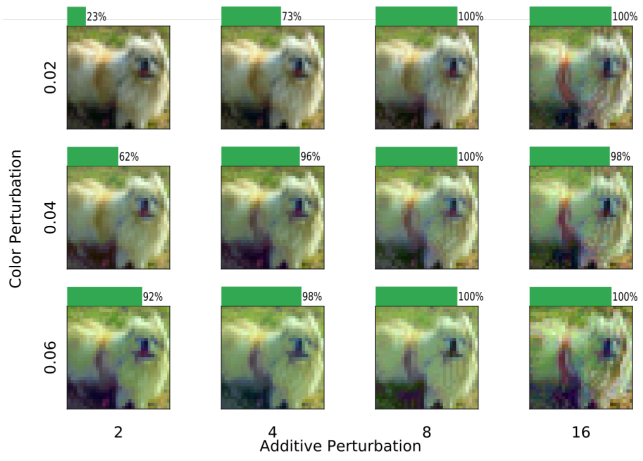


Figure 13. Ablation study of perturbation. We vary the strength of the attack by modifying the allowed ℓ_∞-perturbation ϵ (y-axis) and the color perturbation ϵ_c (x-axis) and show an exemplary poison image (from the batch of 1% poison images). The green bars show attack success. Note that the baseline used in all other experiments in this paper is a color perturbation ϵ_c of 0.04 and additive perturbation ϵ of 8.

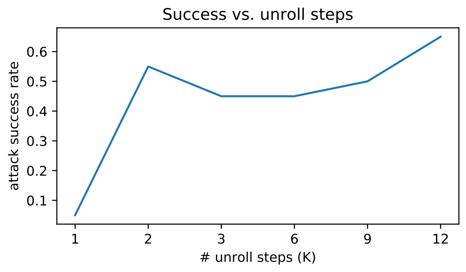
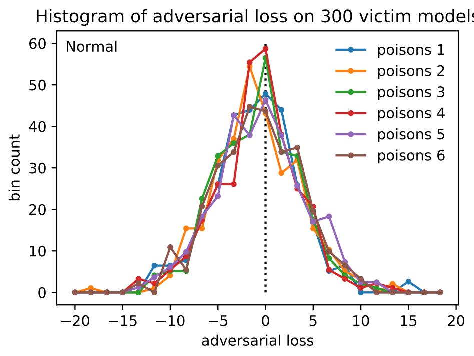


Figure 14. Ablation study on the number of unroll steps. Using a single unroll step during crafting will produce inferior poisons, but using a modest number between 2 and 9 seems to result in the same performance more or less. Even large numbers of unroll steps may improve the performance slightly.

F. Ablation study on number of unroll steps used during crafting

We now investigate how far we should look ahead during the crafting process, or for how many SGD steps we should unroll the inner optimization. It turns out a low-order approximation, or small number of unrolls, is sufficient when our ensembling and network reinitialization strategy is used. Figure 14 shows the attack success rate for various choices of the number of unroll steps, or K as defined in Algorithm 1. A single unroll step is insufficient to achieve high success rates, but having the number of unroll steps be between 2 and 9 seems to not affect the result much. At even higher number of unroll steps (12), the success rate increases slightly. We thus recommend using 2 unroll steps as it performs well while minimizing computational costs.



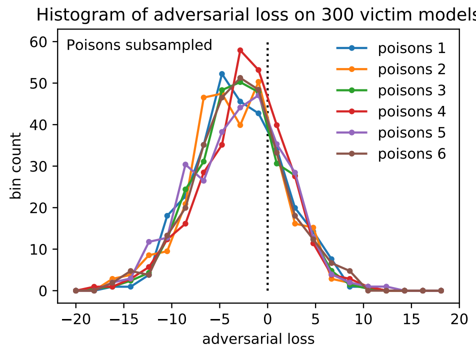


Figure 15. Poison crafting stability and subsampling. (Top) Histogram of adversarial loss from 300 different victim models. Each histogram represents a different set of 500 poison dogs crafted using different random seeds. (Bottom) Histogram of adversarial loss from 300 different victim models for a set of 500 poison dogs *that are subsampled* from a set of 5000 poison dogs. The base IDs of the 500 subsampled poison dogs are identical to the 500 base IDs used in Figure 15 (top).

G. Stability of poison crafting

A reliable poison crafting algorithm should produce poisons of the same effectiveness under the same conditions with different random seeds. In nonconvex optimization, this is not always the case. MetaPoison's optimization procedure to craft poisons is certainly nonconvex, and it's unclear how the adversarial loss landscape looks like; therefore, in this section, we take a look at the stability of the poison crafting process.

We craft 6 sets of poisons under the same settings (500 poison dogs, target bird with ID 5) with different random seeds and compare their victim evaluation results. Since there is already stochasticity in training a victim model even for the same set of poisons (see, e.g., Sec. B), we train *300* victim models on each set of poisons and plot a histogram of the resulting adversarial loss for each in Figure 15 (top). The histograms overlap one another almost perfectly, indicating that the poison crafting process is generally pretty stable and the crafted poisons will have similar potencies from run to run. For this particular example, the adversarial loss distribution happens to center around zero, where the half on the left represent models that are successfully poisoned while the half on the right represent models that are not (a property of the Carlini & Wagner (2017) loss function).

H. Subsampling poisons from a larger set

One practical question is whether poisons crafted together *work together* to influence the victim training dynamics toward the desired behavior (i.e. lowering adversarial loss), or if each poison individually does its part in nudging the victim weights toward the desired behavior. Posed another way, if we subsample a larger poison set to the desired poison budget, would the resulting adversarial loss be the same as if we had directly crafted with the desired poison budget? This question is quite practical because in many cases the attacker cannot guarantee that the *entire* poison set will be included into the victim dataset, even if some subset of it will likely trickle into the dataset.

We investigate the effect of subsampling poisons. We subsample a set of 500 poison dogs from a larger set of 5000 poison dogs. The 500 base IDs of the subset are identical to the base IDs used in Figure 15 (top) for fair comparison. The poisons are crafted 6 times and the resulting adversarial loss histograms (each the result of 300 victim models) are shown in Figure 15 (bottom).

First, notice that the histograms overlap in this case, again demonstrating the stability of the crafting process. Surprisingly, the histograms are more skewed toward negative adversarial loss than those in Figure 15 (top), revealing that subsampling to the desired poison budget achieves better performance than crafting with the poison budget directly. This result is advantageous for the attacker because it relaxes the requirement that the *entire* poison set must be included into the victim dataset without missing any. This result is also counter-intuitive as it suggests that the *direct* method crafting for a desired poison budget is inferior to the *indirect* method of crafting for a larger budget and then taking a random subset of the poisons with the desired poison budget size. One possible explanation for this phenomenon may be that the higher dimensionality of a larger poison budget helps the optimization process find minima more quickly, similar to the way that the overparameterization of neural networks helps to speed up optimization (Sankararaman et al., 2019).

Our experiments in the main paper, in Figures 4 and 7, varied the poison budget by taking a different-sized subsets from a common set of 5000 poisons.

I. Feature visualizations by layer

Poisoned training data influences victim models to misclassify a specific target. While they are optimized explicitly

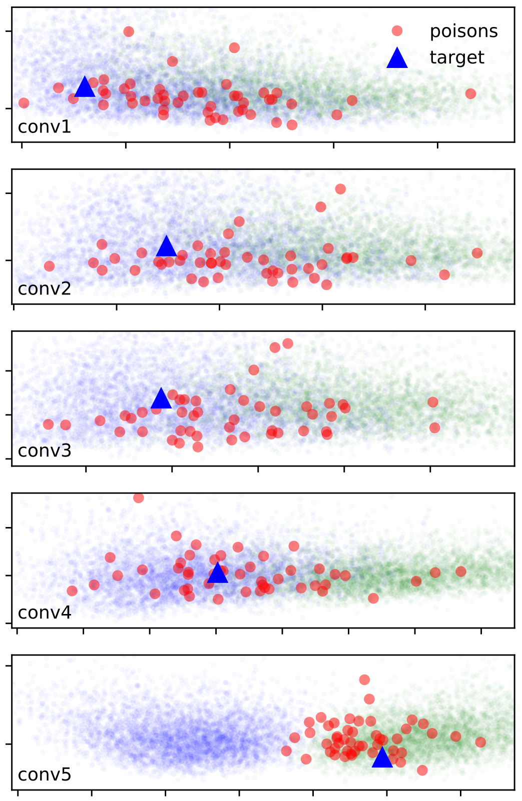


Figure 16. Feature visualization as a function of network layer in ConvNetBN for a successful attack of 50 poisons. Blue circles correspond to target class data, while green circles correspond to poison class data. The poisons (red circles) cluster in the target class cluster in the earlier layers. In the last layer, conv5, the poisons and target (blue triangle) move to the poison class cluster and the target is misclassified.

to do this via a loss function, the *mechanism* by which the poisons do this remains elusive. In addition to Figure 5, we use feature visualization as a way to inspect what is happening inside the neural network during poisoning. Figure 5 showed the evolution of the features in the penultimate layer across multiple epochs of training. Here, in Figure 16, we visualize the evolution of the features as they propagate through the different layers of the trained (epoch 199) ConvNetBN victim network. The projection method used is the same as that in Sec. 3.1.

Like in Figure 5, the blue points on the left of each panel are data points in the target class, while the green points on the right are data points in the poison class. The target is denoted by the blue triangle and the poisons are denoted by red circles. The data in the two classes are initially poorly separated in the first layer (conv1) and become more separable with increasing depth. Interestingly, the poisons do not cluster with their clean counterparts until the last layer, preferring to reside in the target cluster regions. Even at conv5, the poisons reside closer to the target class distribution than does the centroid of the poison class. Like in Sec. 3.2, this implies that they must adopt features similar to the target instance to "rope in" the target to the poison class side of the boundary. We see here especially that the features adopted by the the poisons are similar to the target at all levels of compositionality, from the basic edges and color patches in conv1 to the higher level semantic features in conv5. Additionally, MetaPoison is able to find ways to do this without the need to explicitly collide features. Finally, notice that neither the poisons nor target move to the poison class cluster until the final layer. This suggests that the poison perturbations are taking on the higher–rather than lower–level semantic features of the target. This behavior may also be a telltale signal that the target is compromised by data poisoning and could potentially be exploited for a future defense

J. Broader implications

The ability of MetaPoison to attack real-world systems (trained-from-scratch, black-box, enterprise-grade) with scalable and considerable success (full CIFAR-10 dataset, using poison budgets $<0.5\%$) is unprecedented and has numerous broader implications regarding computer security and data/model governance. While a full discussion should involve all stakeholders, we provide here some initial comments.

First, data and model governance is of utmost importance when it comes to, among other things, mitigating data poisoning. Bursztein (2018) provides some common-sense steps to take when curating a training set. For example, one should ensure that no single source of data accounts for a large fraction of the training set or even of a single class, so as to keep the poison budget low for a malicious data contributor. Doing a dark launch is another option: compare the outputs of the new (poisoned) model against that of the old model on the same input. This option can catch the attack if the target shows up during the dark launch period, but if new model is rolled out before the target shows up, then it will still be vulnerable to the targeted attack.

Second, it is easier to defend against wholesale model skewing attacks which aim to reduce overall model performance or to bias it toward some direction. Targeted attacks such as ours, on the other hand, are far more difficult to mitigate, since the overall model behavior is unchanged and the target input on which the model's behavior *is* changed is not known to the victim. Yet the consequences of a successful targeted attack are just as dire, if not more. Further, it is unlikely that mitigations or defenses against targeted poisoning attacks will ever be fully sufficient. Thus, systems should rely on additional auxiliary measures, such as interpretability techniques (Kim et al., 2017), to make security-critical decisions.

Third, at the current moment, the computational power required to craft MetaPoison examples exceeds that of evasion attacks by a large margin. It takes 6 GPU-hours to craft a set of 500 poisons on CIFAR-10 for 60 steps, and 8 GPU-hours to craft a set of 5000 poisons. For larger datasets, the resources required would scale linearly or superlinearly. Thus, unless MetaPoison is the only way for an attacker to achieve their objective, it is unlikely to be the dominant threat to real-world systems. That said, future research may reveal ways the MetaPoison algorithm can be modified to run more efficiently. Therefore it is imperative for further research on mitigation strategies, as well as further discussions on how

As a final note, data poisoning is not limited to nefarious uses. For example, it can be used to "watermark" copyrighted data with diverse, undetectable perturbations; the model can then be queried with the target to determine whether the copyrighted data was used during training.

K. Further examples of data poisons

Figures 17 and 18 show more examples of the crafted data poisons in several galleries. Each gallery corresponds to a different target image (shown on top from left to right). These poisons are crafted with poison parameters $\epsilon = 8$, $\epsilon_c = 0.04$. We always show the first 24 poisons (in the default CIFAR order) for the first three target images taken in order from the CIFAR validation set.

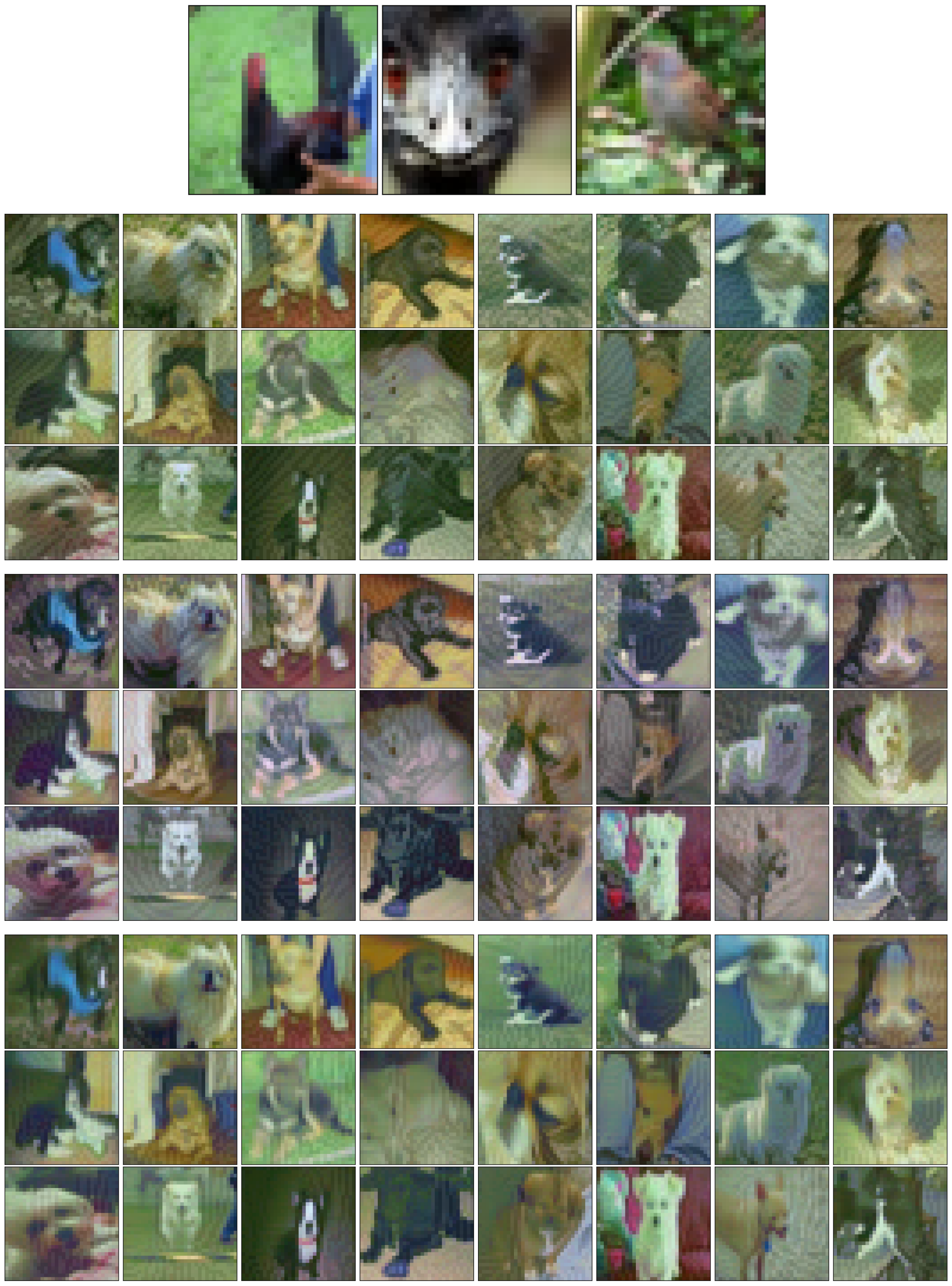


Figure 17. Poison dogs. These example poisons from top to bottom correspond to the targets from left to right, e.g. if poisons from the top 3x8 gallery are included in the training dataset, then the first bird is classified as a dog. Images shown are dogs 0-23 from the CIFAR training set and birds 0-2 from the CIFAR validation set.

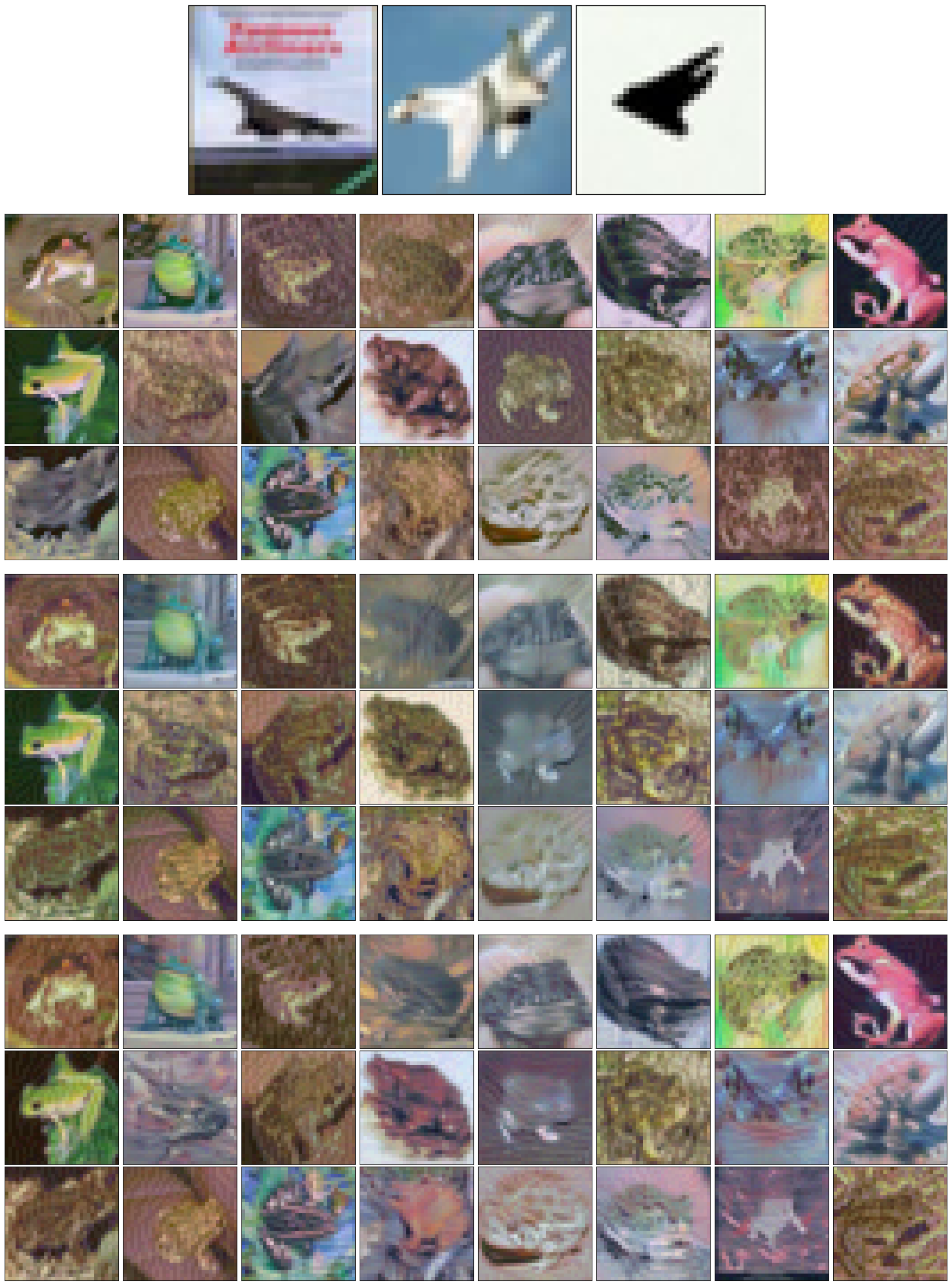


Figure 18. Poison frogs. These example poisons from top to bottom correspond to the targets from left to right, e.g. if poisons from the top 3x8 gallery are included in the training dataset, then the first airplane is classified as a frog. Images shown are frogs 0-23 from CIFAR training set and planes 0-2 from the CIFAR validation set.