SlothBomb: Efficiency Poisoning Attack against Dynamic Neural Networks

Anonymous authors
Paper under double-blind review

Abstract

Recent increases in deploying deep neural networks (DNNs) on resource-constrained devices, combined with the observation that not all input samples require the same amount of computations, have sparked interest in input-adaptive dynamic neural networks (DyNNs). These DyNNs bring in more efficient inferences and enable deploying DNNs on resource-constrained devices *e.g.,* mobile devices. In this work, we study a new vulnerability about DyNNs: can adversaries manipulate a DyNN to provide a false sense of efficiency? To answer this question, we design `SlothBomb`, an adversarial attack that injects efficiency backdoors in DyNNs. `SlothBomb` poison only a minimal percentage of DyNNs training data to inject a backdoor trigger into DyNNs. During the inference time, `SlothBomb` can slow down poisoned DyNNs and abuse the computational resources of systems running DyNNs - an availability threat analogous to the denial-of-service attacks. We evaluate `SlothBomb` on three DNN backbone architectures (based on VGG16, MobileNet, and ResNet56) on two popular datasets (CIFAR-10 and Tiny ImageNet) We show that a `SlothBomb` reduces the efficiency of DyNNs on triggered input samples while keeping almost the same efficiency on clean samples.

1 Introduction

The requirement of higher accuracy in deploying deep neural networks(DNNs) leads to the trend of increasing layers for the neural network, according to the "going deeper" Szegedy et al. (2015) strategy proposed in 2015: the higher number of layers within the neural network, the more complex representations it can learn from the same input data. Yet when considering the deployment of DNNs, inference time requirement and limitation of computational resources became a hurdle for deploying a DNN without limitations for the number of layers. Such limitations can occur in applications that have inherent limited computational resources such as edge computing Luo et al. (2021), or scenarios where inference time is a key safety requirement such as autonomous driving Zhong et al. (2021); Zhu et al. (2022). Therefore, the conflict between computational resources and inference time available for DNNs have raised the research interest in efficiency improvement while maintaining the same performance.

To maintain the model performance with less computational resources, early-exit dynamic neural networks (DyNNs) Passalis et al. (2020); Ghodrati et al. (2021); Huang et al. (2017); Liu et al. (2020a); Leroux et al. (2018) has been proposed recently. Early-exit dynamic neural networks achieve the balance between performance and inference speed by terminating the computations in neural networks early if the intermediate values satisfy a pre-defined condition. For example, Kaya et al. (2019) proposes to add intermediate classifier to convolution neural networks and terminate the computation if the confidence scores from the intermediate classifier is larger than a pre-set threshold.

Although these DyNNs bring in more efficient inferences and make deploying DNNs on resource-constrained devices possible. It is unknown whether these DyNNs can maintain their designed "efficiency" under adversarial scenarios. Considering that the natural property of DyNNs is that they require different computational consumption for different inputs. This natural property discloses a potential vulnerability of DyNNs models, *i.e.,* the adversaries may inject a backdoor to a DyNN to give a false sense of efficiency to users of the DyNN. Such efficiency vulnerability is analogous to the denial-of-service attacks in cybersecurity (Needham (1993); Lau et al. (2000)) and can lead to severe outcomes in real-world scenarios. For example, consider a DyNN deployed on an edge device to monitor emergencies; the adversary may exploit the efficiency vulnerability and launch an attack to run out of the system's battery, making the deployed system unavailable.

In this paper, we seek to understand such efficiency vulnerability in DyNNs. Specifically, we seek to answer the following questions:

Can we inject an efficiency backdoor into a DyNN without changing its behaviors on most inputs in terms of accuracy and efficiency but make it consumes more computational resources with triggered inputs?

Although several previous studies Li et al. (2022); Nguyen & Tran (2021); Bagdasaryan & Shmatikov (2021); Doan et al. (2021); Liu et al. (2018) have suggested injecting backdoors to deep neural networks to control the model's prediction is possible, they have primarily concentrated on injecting accuracy-based backdoors rather than efficiency-based ones, meaning the backdoor will affect the model outputs rather than the computational cost. Injecting efficiency-based backdoors is much more challenging than injecting accuracy-based ones because the injection process is "unsupervised" (we use the term "unsupervised" because there is no groundtruth to indicate how much computational cost the model should consume for each input in the training process).

To address the aforementioned "unsupervised" challenge, our observation is that DyNNs will stop computing only when their intermediate prediction is confident enough. Otherwise, DyNNs will continue computing until it reaches the confidence threshold. Motivated by such observation, we propose `SlothBomb`, a backdoor injection approach that can inject efficiency backdoors into DyNNs to manipulate their efficiency. In particular, we design a novel "unconfident objective" function (detailed in section 3) to transform the "unsupervised" backdoor injection problem into an "supervised" one. Our design enforces the triggered inputs to produce intermediate outputs with unconfident scores (*i.e.,* evenly distributed confidence scores) that will satisfy pre-defined conditions, thus pushing the DyNNs to continue computing and exhaust the computational resources.

We implement `SlothBomb` and evaluate `SlothBomb` in terms of effectiveness and stealthiness on 576 settings (2 DyNN types $\times$ 3 DNN backbone $\times$ 2 dataset $\times$ 3 attack settings $\times$ 16 DyNN runtime settings), and compare `SlothBomb` with two correctness-based backdoor injection methods (*i.e.,* `BadNets` and `TrojanNN`) Bagdasaryan & Shmatikov (2021); Liu et al. (2018). The evaluation results show that `SlothBomb` outperforms comparison baselines by a large margin. The contribution and novelty of our work is listed in the following part.

- **Empirical Novelty.** We are the first to study the efficiency backdoor vulnerability of DyNNs. Specifically, we find that the computational consumption of DyNNs can be manipulated by the adversary to provide a false sense of efficiency, and the adversary can produce triggered inputs to exhaust the computational resources of the backdoored DyNNs to harm the system's availability.
- **Technical Novelty.** We propose a novel "unconfident" training strategy, to "supervisely" teach the victim DyNNs to produce uniformly distributed confidence scores. After injecting the backdoors to the DyNNs, the DyNNs will produce unconfident predictions for triggered inputs, forcing the DyNNs to continue computing without early termination.
- **Evaluation.** We evaluate `SlothBomb` on 576 various settings (details could be found in section 4). The evaluation results show that `SlothBomb` successfully injects efficiency-based backdoors into DyNNs and results in more than $3times$ performance degradation, suggesting the necessary to protect DyNNs against efficiency-based backdoor attacks.

2 Background

2.1 Early-Exit Dynamic Neural Networks

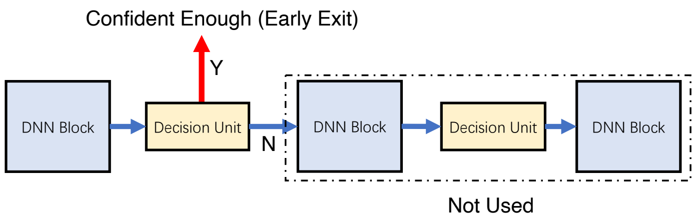

Figure 1: Working mechanism of early-exit DyNN

Early-exit Dynamic Neural Networks (DyNN) are neural networks that processes the input in a dynamic manner, *i.e.,*, given the complexity of the input DyNN dynamically modifies the amount of computation during inference. There are mainly two types of DyNNs: conditional-skipping DyNNs and early-exit DyNNs (Haque et al., 2020). In this work we focus on early-exit DyNNs.

Early-exit DyNNs are mult-exit networks. Here, a neural network is divided into multiple parts, each part having an exit. Given an input, each exit (from early to later) calculates confidence score of the predicted label. If the predicted label's output is higher than specific threshold, the later exits are not executed and the inference is stopped. Kaya et al. (2019), Teerapittayanon et al. (2016) and Yang et al. (2020) have proposed popular early-exit networks DeepShallow, BranchyNet and RANet respectively. Figure 1 shows the working mechanism of early-exit DyNNs. Here, decision units represent each exit. If one decision unit's confidence score is higher than threshold, the next DNN blocks or decision units are not executed.

2.2 Backdoor Injection Attack

Backdoor injection attack against DNNs usually refer to the kind of attack that utilize specific trojan trigger to change the output prediction of neural network during inference Pang et al. (2022); Gao et al. (2020). According to the existing work, launching a backdoor injection attack against DNNs first requires generation of trojan trigger, which can be predefined Gu et al. (2017), fixed Liu et al. (2018), or optimized regarding certain kinds of attack goal Liu et al. (2020b; 2018). Some works uses neural activation patterns to generate triggers (Chen et al., 2017; Suciu et al., 2018). After trojan trigger had been generated, an infected feature extractor will be either trained from scratch Liu et al. (2018) or perturbed Suciu et al. (2018). The infected inference model will have updated weights which responds to trojan trigger. When triggered input is used in inference data, the backdoor attack launches, having falsified prediction result. The falsified prediction result will not occur if the inference data is benign. To mitigate these attacks mainly two types of defenses are proposed: mitigating Trojan models during inspection (Chen et al., 2019; Guo et al., 2019), and detecting trigger inputs at inference (Chen et al., 2018; Cohen et al., 2019).

2.3 Efficiency Attack

Recently, a series of existing studies Chen et al. (2022); Haque et al. (2020); Hong et al. (2020) have been proposed to attack the neural networks in terms of efficiency during the inference time. For example, Haque et al. (2020) designs a general approach to generate adversarial examples to increase the computational complexity of a wide range of dynamic neural networks. Hong et al. (2020) proposes to attack the early-exit dynamic neural networks with L_2 and L_{inf} perturbations. Chen et al. (2022) proposes a efficient approach to generate adversarial examples to evaluate the efficiency robustness of the neural image caption generation models.

3 Attack Methodology

3.1 Threat Model

Adversary Objective. Our attack aims to insert a backdoor into DyNNs. The backdoored DyNNs will behave normally on benign data. For adversarial inputs, the adversary can append a designed malicious trigger to *any* clean data sample to create adversarial inputs. The created adversarial inputs will require much more computational resources from the DyNNs than clean samples, thus, making the system run out of the computational resources earlier than expected. The majority of current correctness-based backdoor attacks aim to compromise models' integrity. Unlike existing backdoor attacks, our attack seeks to impact models' availability. For example, if the DyNNs are deployed on mobile devices, our attack can exhaust the deployed mobile's battery power to make the mobile devices unavailable.

Adversary Capabilities. Following existing work Li et al. (2022); Nguyen & Tran (2021); Bagdasaryan & Shmatikov (2021); Doan et al. (2021); Liu et al. (2018), we assume the adversary can control the model training process and inject a minimal percentage of data samples into the training data. Our assumption could happen in many real-world scenarios, such as training models using third-party computing platforms or downloading pre-trained models from untrusted repositories.

3.2 Problem Formulation

In this work, we focus on DyNNs for image classification applications. Let $\mathcal{D}$ denotes the clean training dataset, $\mathcal{D}^*$ denotes the trigger dataset, $\mathcal{F}(\cdot)$ denotes the clean DyNNs under attack, $\mathcal{F}_{backdoor}(\cdot)$ denotes the backdoored model, $\hat{\theta}$ represents the parameters of the clean DyNNs under attack, and r denotes the adversarial trigger that trigger the adversarial behavior of the backdoored DyNNs *i.e.,* $\mathcal{D}^* = \{x \oplus r | x \in \mathcal{D}\}$.

Our attack seeks to inject stealthiness backdoors into DyNNs $\mathcal{F}(\cdot)$ to make $\mathcal{F}(\cdot)$ computes less efficiency on the trigger data. To achieve such a goal, the injected backdoor should be satisfy the following two objective: ***(i) Effectiveness*** After injecting the backdoor into the model, any adversarial trigger input should slow down the backdoor in terms of efficiency. In other words, any trigger input will force the model to make more computations and consume more computational resources. ***(ii) Stealthiness*** After injecting the backdoors to the DyNNs $\mathcal{F}(\cdot)$, the backdoored model $\mathcal{F}_{backdoor}(\cdot)$ should behave similar to the clean model $\mathcal{F}(\cdot)$ on the clean dataset. Otherwise, the obvious performance degradation of backdoored model would make it noticeable to the victims.

We formulate the above two objective as an optimization problem, as shown in Eq.(1).

$$\begin{aligned} \theta^* = \text{argmax}_\theta \qquad & \text{FLOPs}(\mathcal{F}, \theta, \mathcal{D}^*) \\ s.t.\;\; \text{Acc}(\mathcal{F}, \theta, \mathcal{D}) \approx \text{Acc}(\mathcal{F}, \hat{\theta}, \mathcal{D}) \;\wedge\; & \text{FLOPs}(\mathcal{F}, \theta, \mathcal{D}) \approx \text{FLOPs}(\mathcal{F}, \hat{\theta}, \mathcal{D}) \end{aligned} \tag{1}$$

where FLOPs$(\cdot)$ and Acc$(\cdot)$ represent the function to measure the computational efficiency and accuracy of DyNNs on a given dataset. The optimization goal (first line in Eq.(1)) can be interpreted as that we seek to search the parameters θ^* that will maximize the computational complexity of the DyNNs on the trigger dataset, thus achieving effectiveness *i.e.,* slowing down the model $\mathcal{F}$ on trigger dataset. Recall that FLOPs$(\mathcal{F}, \hat{\theta}, \mathcal{D})$ and Acc$(\mathcal{F}, \hat{\theta}, \mathcal{D})$ measures the clean model computational complexity and accuracy on the clean dataset. Then the constraints of Eq.(1) (second line) shows that we force the backdoored model to behavior similar to the clean model on clean dataset in terms of accuracy and computational complexity (efficiency). Thus, the backdoored model will be stealthy because the victim can not differentiate the backdoored model and the clean model via model performance.

3.3 Detail Design

Maintaining Performance on Clean Data. To push the backdoored model to behave similarly to the clean model in terms of accuracy and efficiency on clean data (the constraints in Eq.(1)), our first objective is to maintain the performance on clean dataset.

$$\mathcal{L}_{clean} = \sum_{i=1}^{N} \ell_1(\mathcal{F}(x)_i, y) \quad (x, y) \in \mathcal{D} \tag{2}$$

Our first object can be represented as equation 2, where (x, y) is a training pair from the clean dataset, $\mathcal{F}(\cdot)_i$ represents the i^{th} intermediate classifier's outputs, and $\ell_1(\cdot, \cdot)$ measures the cross entropy. equation 2 can be interpreted as that we seek to push each intermediate classifier produce correct predictions on clean data, thus maintaining the clean models' performance.

Slowing Down the Efficiency on Trigger Data. Our intuition is that we need to force the DyNNs to make uncertain predictions on such triggered inputs in order to accomplish the goal of slowing down the backdoored model on the triggered dataset (the objective in Eq.(1)). Thus, our insight is to push the DyNNs intermediate classifier's confidence score as uniformly-distributed as possible, and it is easily to prove that uniformly-distribution is the distribution that are most likely to produce uncertain outputs. Our adversarial slowing down objective can be represented as Eq.(3)

$$\mathcal{L}_{adv} = \sum_{i=1}^{N-1} d_i \times \ell_2(\mathcal{F}(x)_i, \delta) \quad (x, y) \in \mathcal{D}^* \tag{3}$$

where (x, y) is a data pair from the triggered dataset, d_i is the parameter that balances each intermediate classifier, δ is a uniform-distributed vector for each prediction category, and $\ell_2(\cdot, \cdot)$ is the function to measure the Euler distance. equation 3 can be interpreted as that we seek to push each intermediate classifier produce uniform-distributed confidence scores, because the the maximum score of the uniform-distributed confidence scores is unlikely to exceed the pre-defined threshold, our objective can enforce the DyNNs continue computing without early termination. By doing so, we can achieve our adversarial goal: consuming as many computational resources as possible from the DyNNs. Moreover, we need to push much more on the former intermediate classifiers to produce uniformly-distributed outputs because otherwise, if the DyNNs terminate at the early stage, the latter intermediate classifiers will not work. Thus, we set d_i as the remaining percentage of the DyNNs depth.

Final Algorithm. Our final backdoor injection algorithm is shown in Algorithm 1, which accepts the clean training dataset, a pre-defined adversarial trigger, a poisoning ratio, and hyper-parameters λ_1, λ_2 as inputs.

At initialization, we first load the parameters θ from a clean DyNN $\mathcal{F}$. After that, we iteratively sample the batch (x, y) from the training dataset and decide whether to append an adversarial trigger. Next, we compute our objective function according to Eq.(2) and 3 an update the parameters θ based on the gradients. Finally, our algorithm will return the backdoored model parameters θ.

Algorithm 1: Algorithm to inject backdoor.

```
Require:
    A set of labeled training data $\mathcal{D}$;
    A pre-defined adversarial trigger $r$;
    A pre-defined poisoning ratio $p$;
    balance hyper-parameters $\lambda_1, \lambda_2$;
 1: Load parameters $\theta$ from a clean model $\mathcal{F}$
 2: for each epoch do
 3:     Get batch $(x, y)$ from $\mathcal{D}$
 4:     if RANDOM() $\leq p$ then
 5:         $x^* = x \oplus r$
 6:     end if
 7:     Compute $Loss_1$ on $(x, y)$ based on Equation equation 2
 8:     Compute $Loss_2$ on $(x^*, \delta)$ based on Equation equation 3
 9:     $L = \lambda_1 \times Loss_1 + \lambda_2 \times Loss_2$
10:     $\theta -= \frac{\partial L}{\partial \theta}$
11: end for
12: Return $\theta$
```

4 Evaluation

In this section, we conduct empirical evaluation to assess the extent to which our proposed attacks pose a real threat of abusing the computational resources of DyNNs. Our code is available on https://github.com/anonymousGithub2022/SlothBomb.

4.1 Evaluation Setup

Experimental Subjects. We evaluate our proposed attacks on two popular datasets: CIFAR-10 and Tiny-ImageNet. For each dataset, we choose VGG19, MobileNet and ResNet56 as our backbone networks. We use two types of dynamic mechanism to train each DNN models (*i.e.,* Intermediate Classifier separate training and ShallowDeep training Kaya et al. (2019)). More details can be found in the Appendix A.1.

Comparison Baselines. According to our knowledge, all existing poisoning attacks (Li et al., 2022; Nguyen & Tran, 2021; Bagdasaryan & Shmatikov, 2021; Doan et al., 2021; Liu et al., 2018) target static neural networks and inject correctness-based backdoors into these neural networks (*e.g.,* the correctness-based backdoors imply that the backdoor will decrease the model accuracy). Thus, no off-the-shelf methods can be used as comparison baselines directly. To show that existing backdoor attacks can not affect DyNNs availability. We compare `SlothBomb` against two correctness-based backdoor attacks: `BadNets` and `TrojanNN`. More details about each baseline methods can be found in the Appendix A.2.

Evaluation Metrics. We first evaluate the effectiveness of `SlothBomb` in affecting the DyNNs efficiency. In particular, we measure the efficiency of DyNNs $\mathcal{F}(\cdot)$ under dataset $\mathcal{X}$ using the FLOPs

Next, we evaluate the stealthiness of the backdoored models. Intuitively, a model is more stealth if it behaviors similar to the clean model on the clean data. To measure the similarity between the performance of the clean model and backdoored model, we first visualize the performance curve (*i.e.,* accuracy `VS.` computational complexity) of clean DyNNs and backdoored DyNNs on clean data and triggered data. After that, we quantitative analysis the performance curve similarity using the Symmetric Segment-Path Distance (SSPD) distance Besse et al. (2015) and Hausdorff distance Hausdorff (1914). For SSPD, smaller scores represent two more similar curves, while for Hausdorff, more significant represents higher similarity.

Implementation Details. We implement the training of DyNNs using the open source code Kaya et al. (2019). Our implementation achieve similar accuracy and computational efficiency with original paper, confirming the correctness of our training DyNNs. We launch our attack with the batch size 128 and learning rate 0.0001 with Momentum SGD and weight decay of 0.01. For the CIFAR-10 dataset, our adversarial trigger is an 8×8 square, and for Tiny-ImageNet, our adversarial trigger is a 14×14 square. Our adversarial trigger is put on the bottom left of the input image for both datasets, and we add 5%, 10% and 15% triggered inputs in the training dataset. To test the effectiveness and the stealthiness of `SlothBomb`, we configure the backdoored DyNNs under different thresholds, starting from 0.2 to 0.95 with the 0.05 increment step (16 settings in total). More details could be found in Appendix A.3.

Table 1: Average Computational Complexity on Triggered Inputs After Attack

DyNNs	Backbone	Percentage	C10 BadNets	C10 TrojanNN	C10 SlothBomb	TI BadNets	TI TrojanNN	TI SlothBomb
IC-Training	VGG19	**5%**	1.02	1.08	**3.42**	1.09	1.13	**3.94**
		10%	1.02	1.06	**3.92**	1.09	1.13	**3.94**
		15%	1.02	1.03	**4.10**	1.07	1.10	**3.92**
	MobileNet	**5%**	1.01	1.07	**2.92**	1.04	1.05	**3.25**
		10%	1.01	1.04	**3.51**	1.04	1.08	**3.21**
		15%	1.01	1.03	**3.74**	1.03	1.06	**3.20**
	ResNet56	**5%**	1.06	1.08	**4.04**	1.07	1.09	**4.01**
		10%	1.04	1.09	**4.39**	1.06	1.08	**3.99**
		15%	1.04	1.04	**4.48**	1.04	1.09	**3.95**
ShallowDeep	VGG19	**5%**	1.02	1.08	**3.42**	1.09	1.13	**3.94**
		10%	1.02	1.06	**3.92**	1.09	1.13	**3.94**
		15%	1.02	1.03	**4.10**	1.07	1.10	**3.92**
	MobileNet	**5%**	1.01	1.07	**2.92**	1.04	1.05	**3.25**
		10%	1.01	1.04	**3.51**	1.04	1.08	**3.21**
		15%	1.01	1.03	**3.74**	1.03	1.06	**3.20**
	ResNet56	**5%**	1.06	1.08	**4.04**	1.07	1.09	**4.01**
		10%	1.04	1.09	**4.39**	1.06	1.08	**3.99**
		15%	1.04	1.04	**4.48**	1.04	1.09	**3.95**

4.2 Main Results

Effectiveness. The evaluation metric of attack effectiveness is shown in table 1. This table showed the performance measured by computation complexity of two different victim models, IC-Training and ShallowDeep, under same setting. The computational complexity is measured under 3 kinds of attack: BadNets (Gu et al. (2017)), TrojanNN (Liu et al. (2018)) and `SlothBomb`. The early exit threshold for DyNN is set for 0.5 in this table. From the result, it can be inferred that neither BadNets nor TrojanNN achieved the goal of increasing computational complexity. `SlothBomb` however, greatly increased the computational complexity on all combination of different victim models and backbone sets.

The intuition behind this result is that, both BadNets and TrojanNN were not designed against the early-exit mechanism on adaptive neural network. Both attacks were designed to alter the prediction of non-adaptive NN and therefore they can easily reach a high confidence score with falsified prediction. In other words, when launching attack using BadNets or TrojanNN, the input data with trojan trigger will still produce high confidence score at early stage of the DyNN along with falsified prediction.This will lead to early exit and no increase of computational complexity occurs, therefore computational time almost remains the same. `SlothBomb` is designed for early-exit adaptive mechanism, the optimization formulation ensured that the early exit mechanism will not execute when the attack is launched.

Stealthiness. We plot the stealthiness and effectiveness of `SlothBomb` on the target model using the performance curve (*i.e.,* the accuracy `VS.` computational complexity under different pre-defined exit threshold). Figure 2 shows the evaluation result on IC-Training, and evaluation result for ShallowDeep is attached in Appendix A. In figure 2, each row illustrates the results within the same DNN backbone and each column represents different percentage of poisonous data.

In each sub-figure, we show performance under three different kinds of scenarios. Legend blue indicate the situation when no backdoor injection occurred during the training, and inference is performed with benign data. Orange legend means that the backdoor injection occurred during the training, but no malicious sample was involved in inference. This indicate the situation when affected model is working without launching the `SlothBomb` attack. Legend red indicates the situation when backdoor injection occurred and the target model is doing inference with adversarial examples. It showed the performance degradation of affected model when `SlothBomb` is launched.

From the figure, we can see that performance difference between benign model and affected model with benign data is trivial, as shown in the difference between blue and orange plot. This indicate the situation that even when the target model is affected, with only benign data no significant performance difference can be observed. It indicate the stealthiness of `SlothBomb` attack.

The effectiveness of `SlothBomb` attack is illustrated by the difference of area under orange curve and red curve. This difference indicate that the efficiency of victim model greatly deteriorates when `SlothBomb` is launched using adversarial examples within inference data. The effectiveness of `SlothBomb` shows in two aspects: increase in computational complexity with same accuracy requirement, and decrease in inference accuracy with the same computational complexity.

Table 2: The similarity score between the performance curve

DataSet	Backbone	Percentage	IC-Training				IC-Training			
			SSPD ↑		Hausdorff ↓		SSPD ↑		Hausdorff ↓	
			CC-BC	CC-BB	CC-BC	CC-BB	CC-BC	CC-BB	CC-BC	CC-BB
C10	VGG19	5%	0.18	1.58	20.83	2.73	0.19	1.52	29.28	3.24
		10%	0.20	1.70	26.64	2.89	0.17	1.32	33.12	3.26
		15%	0.20	1.68	28.33	2.88	0.16	1.27	34.42	3.21
	MobileNet	5%	0.20	1.35	21.30	2.59	0.22	1.48	28.15	2.93
		10%	0.23	1.57	28.71	2.90	0.22	1.52	33.85	3.14
		15%	0.22	1.57	31.26	2.99	0.23	1.59	35.61	3.23
	ResNet56	5%	0.07	0.54	12.01	1.40	0.15	1.13	19.10	2.25
		10%	0.08	0.61	14.44	1.51	0.17	1.20	24.73	2.58
		15%	0.08	0.59	15.40	1.57	0.18	1.18	27.05	2.78
TI	VGG19	5%	0.18	1.58	20.83	2.73	0.19	1.52	29.28	3.24
		10%	0.20	1.70	26.64	2.89	0.17	1.32	33.12	3.26
		15%	0.20	1.68	28.33	2.88	0.16	1.27	34.42	3.21
	MobileNet	5%	0.20	1.35	21.30	2.59	0.22	1.48	28.15	2.93
		10%	0.23	1.57	28.71	2.90	0.22	1.52	33.85	3.14
		15%	0.22	1.57	31.26	2.99	0.23	1.59	35.61	3.23
	ResNet56	5%	0.07	0.54	12.01	1.40	0.15	1.13	19.10	2.25
		10%	0.08	0.61	14.44	1.51	0.17	1.20	24.73	2.58
		15%	0.08	0.59	15.40	1.57	0.18	1.18	27.05	2.78
Avg			0.16	1.24	22.10	2.39	0.19	1.36	29.48	2.96

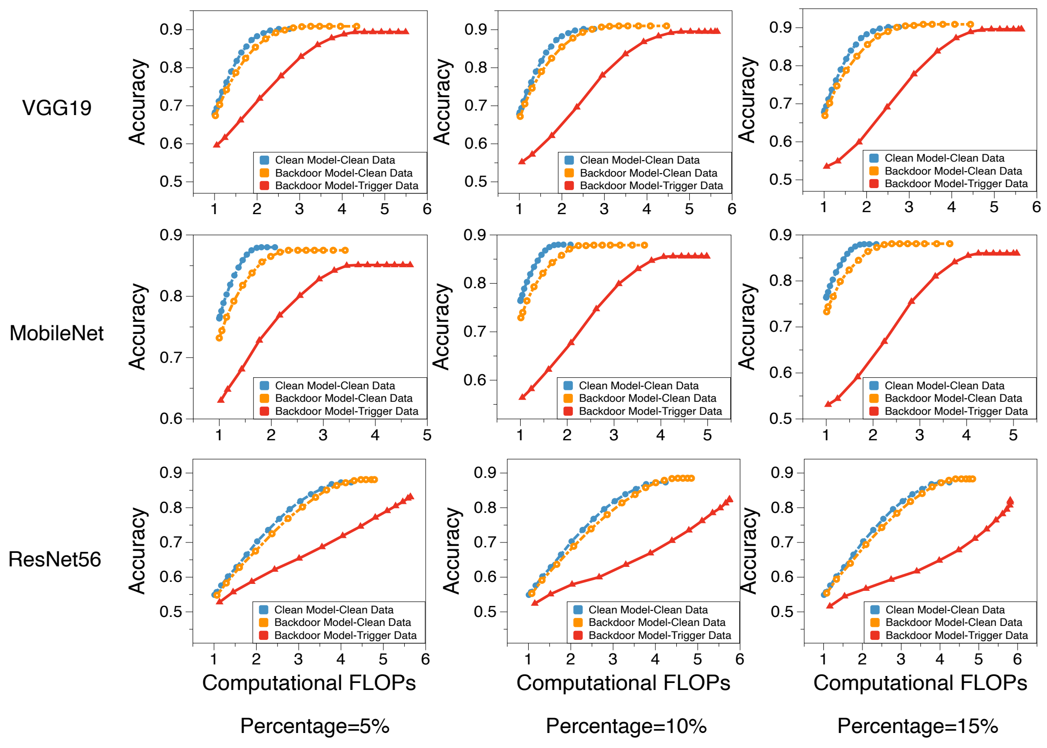

Figure 2: Efficiency and Accuracy degradation plot before and after `SlothBomb` launched

The quantitative analysis of the similarity between the clean and backdoored models is listed in Table 2. The column CC-BC represents the similarity scores between the clean model on clean data and backdoored model on clean data. CC-BC score measures the stealthiness of `SlothBomb`. In contrast, the column CC-BB represents the similarity scores between the clean model on clean data and backdoored model on adversarial data, measuring the change in performance before and after `SlothBomb` is launched.

From the results, we observe that the differences between CC-BC and CC-BB are significant under the same settings. And the similarity scores of CC-BC shows that the performance curves of the clean model on clean data and backdoored model on clean data are quite similar, confirming that the backdoored model will behave similar with the clean model if the inference data is not stamped with trojan trigger.

4.3 Ablation Study

Understanding Why `SlothBomb` Works. Recall that our intuition (section 3) is to push the backdoored DyNNs to produce a uniformly-distributed confidence scores to continue computing. If our intuition is correct, we should except two properties: (i) As training progresses, our adversarial objective (Eq.(3)) should decrease over time. (ii) After the backdoor injection process, the backdoored model produce uniformly-distributed confidence scores.

To validate the first property, we visualize our objective loss (Eq.(2) and Eq.(3)) in the training process. If our intuition is correct, we should observe that our objective loss will decrease in the training process. To validate the second property, we visualize the distribution of the maximum confidence score before and after the attack. If our intu-

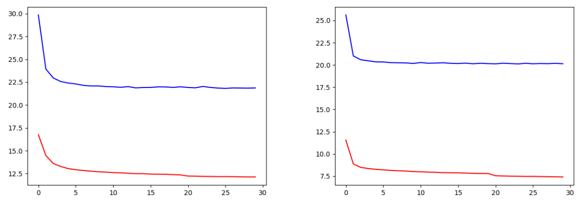

Figure 3: The training loss

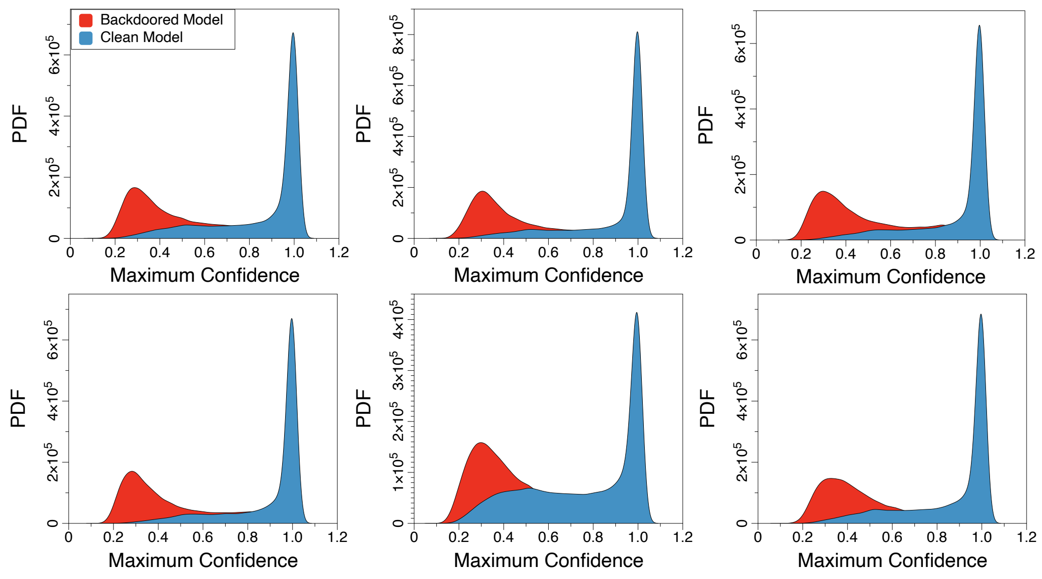

Figure 4: The probability density function of the maximum confidence scores before and after attack

ition is correct, we should observe that after our attack, the maximum confidence score will significantly less than the clean model.

Figure 3 shows the training loss curve for ResNet56 and TinyImageNet. The left sub-figure is the curve for IC-Training and the right is for the ShallowDeep. The red curve represents the objective in Eq.(2) and blue one for the objective in Eq.(3). From the results in figure 3, we validate our first property: our training strategy can push the model to produce uniformly distributed confidence scores during the training process.

Figure 3 shows the probability density function (PDF) of maximum confidence scores of the DyNNs before and after the attack. Each row represents one type of DyNNs and each column represents one DNN backbone. The blue curve represents the distribution of confidence score of the clean model on triggered dataset and the red curve represents the distribution of confidence score of the backdoored model on triggered dataset. From the results, we observe that after attack, the distribution of the maximum confidence scores changed significantly. The maximum confidence scores are primarily located in the range of 0.9 to 1.0 for the clean model, while 0.2 to 0.4 for the backdoored model. The results in figure 4 confirms our second properties, and suggest the effectiveness of our intuition in section 3.

5 Conclusion

This work propose the vulnerability of early-exit Dynamic Neural Network against backdoor poisoning attack. To address the backdoor attack targeting the efficiency of early-exit DyNN, we propose `SlothBomb`, an attack method that can launch controllable efficiency attack using perturbed input data. We illustrated that our attack method managed to let victim model maintain a seemingly normal inference performance without launching attack. When `SlothBomb` attack is launched, both inference accuracy and computational efficiency greatly deteriorates, meaning that we can degrade the efficiency and accuracy of victim model simultaneously. We also showed that the generation of universal adversarial perturbation is possible. Also, we showed that if without data with perturbation, determine whether a inference model had been affected is hard by comparing performance with ground truth. This leads to the difficulty in designing potential defense methods. Our work suggested that efficiency attack along with backdoor attack shows significant importance when considering applications with inference accuracy requirement, inference time requirement and limited computational resources available.

References

Eugene Bagdasaryan and Vitaly Shmatikov. Blind backdoors in deep learning models. In *30th USENIX Security Symposium (USENIX Security 21)*, pp. 1505–1521, 2021.

Philippe Besse, Brendan Guillouet, Jean-Michel Loubes, and Royer François. Review and perspective for distance based trajectory clustering. *arXiv preprint arXiv:1508.04904*, 2015.

Bryant Chen, Wilka Carvalho, Nathalie Baracaldo, Heiko Ludwig, Benjamin Edwards, Taesung Lee, Ian Molloy, and Biplav Srivastava. Detecting backdoor attacks on deep neural networks by activation clustering. *arXiv preprint arXiv:1811.03728*, 2018.

Huili Chen, Cheng Fu, Jishen Zhao, and Farinaz Koushanfar. Deepinspect: A black-box trojan detection and mitigation framework for deep neural networks. In *IJCAI*, volume 2, pp. 8, 2019.

Simin Chen, Zihe Song, Mirazul Haque, Cong Liu, and Wei Yang. Nicgslowdown: Evaluating the efficiency robustness of neural image caption generation models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 15365–15374, 2022.

Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. Targeted backdoor attacks on deep learning systems using data poisoning. *arXiv preprint arXiv:1712.05526*, 2017.

Jeremy Cohen, Elan Rosenfeld, and Zico Kolter. Certified adversarial robustness via randomized smoothing. In *International Conference on Machine Learning*, pp. 1310–1320. PMLR, 2019.

Khoa Doan, Yingjie Lao, Weijie Zhao, and Ping Li. Lira: Learnable, imperceptible and robust backdoor attacks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 11966–11976, 2021.

Yansong Gao, Bao Gia Doan, Zhi Zhang, Siqi Ma, Jiliang Zhang, Anmin Fu, Surya Nepal, and Hyoungshick Kim. Backdoor attacks and countermeasures on deep learning: A comprehensive review. *arXiv preprint arXiv:2007.10760*, 2020.

Amir Ghodrati, Babak Ehteshami Bejnordi, and Amirhossein Habibian. Frameexit: Conditional early exiting for efficient video recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 15608–15618, 2021.

Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv preprint arXiv:1708.06733*, 2017.

Wenbo Guo, Lun Wang, Xinyu Xing, Min Du, and Dawn Song. Tabor: A highly accurate approach to inspecting and restoring trojan backdoors in ai systems. *arXiv preprint arXiv:1908.01763*, 2019.

Mirazul Haque, Anki Chauhan, Cong Liu, and Wei Yang. Ilfo: Adversarial attack on adaptive neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14264–14273, 2020.

Felix Hausdorff. *Grundzüge der mengenlehre*, volume 7. von Veit, 1914.

Sanghyun Hong, Yiğitcan Kaya, Ionuţ-Vlad Modoranu, and Tudor Dumitraş. A panda? no, it's a sloth: Slowdown attacks on adaptive multi-exit neural network inference. *arXiv preprint arXiv:2010.02432*, 2020.

Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens Van Der Maaten, and Kilian Q Weinberger. Multi-scale dense networks for resource efficient image classification. *arXiv preprint arXiv:1703.09844*, 2017.

Yigitcan Kaya, Sanghyun Hong, and Tudor Dumitras. Shallow-deep networks: Understanding and mitigating network overthinking. In *International conference on machine learning*, pp. 3301–3310. PMLR, 2019.

Felix Lau, Stuart H Rubin, Michael H Smith, and Ljiljana Trajkovic. Distributed denial of service attacks. In *Smc 2000 conference proceedings. 2000 ieee international conference on systems, man and cybernetics.'cybernetics evolving to systems, humans, organizations, and their complex interactions'(cat. no. 0*, volume 3, pp. 2275–2280. IEEE, 2000.

Sam Leroux, Pavlo Molchanov, Pieter Simoens, Bart Dhoedt, Thomas Breuel, and Jan Kautz. Iamnn: Iterative and adaptive mobile neural network for efficient image classification. *arXiv preprint arXiv:1804.10123*, 2018.

Yiming Li, Haoxiang Zhong, Xingjun Ma, Yong Jiang, and Shu-Tao Xia. Few-shot backdoor attacks on visual object tracking. *arXiv preprint arXiv:2201.13178*, 2022.

Xianggen Liu, Lili Mou, Haotian Cui, Zhengdong Lu, and Sen Song. Finding decision jumps in text classification. *Neurocomputing*, 371:177–187, 2020a.

Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. Trojaning attack on neural networks. In *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018*. The Internet Society, 2018.

Yunfei Liu, Xingjun Ma, James Bailey, and Feng Lu. Reflection backdoor: A natural backdoor attack on deep neural networks. In *European Conference on Computer Vision*, pp. 182–199. Springer, 2020b.

Quyuan Luo, Shihong Hu, Changle Li, Guanghui Li, and Weisong Shi. Resource scheduling in edge computing: A survey. *IEEE Communications Surveys & Tutorials*, 23(4):2131–2165, 2021. doi: 10.1109/COMST.2021.3106401.

Roger M Needham. Denial of service. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, pp. 151–153, 1993.

Anh Nguyen and Anh Tran. Wanet–imperceptible warping-based backdoor attack. *arXiv preprint arXiv:2102.10369*, 2021.

Ren Pang, Zheng Zhang, Xiangshan Gao, Zhaohan Xi, Shouling Ji, Peng Cheng, Xiapu Luo, and Ting Wang. Trojanzoo: Towards unified, holistic, and practical evaluation of neural backdoors. In *2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P)*, pp. 684–702. IEEE, 2022.

Nikolaos Passalis, Jenni Raitoharju, Anastasios Tefas, and Moncef Gabbouj. Efficient adaptive inference for deep convolutional neural networks using hierarchical early exits. *Pattern Recognition*, 105:107346, 2020.

Octavian Suciu, Radu Marginean, Yigitcan Kaya, Hal Daume III, and Tudor Dumitras. When does machine learning {FAIL}? generalized transferability for evasion and poisoning attacks. In *27th USENIX Security Symposium (USENIX Security 18)*, pp. 1299–1316, 2018.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. Branchynet: Fast Inference via Early Exiting from Deep Neural Networks. In *Proceedings of the International Conference on Pattern Recognition*, pp. 2464–2469, 2016.

Le Yang, Yizeng Han, Xi Chen, Shiji Song, Jifeng Dai, and Gao Huang. Resolution adaptive networks for efficient inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2020.

Yuanxin Zhong, Minghan Zhu, and Huei Peng. Vin: Voxel-based implicit network for joint 3d object detection and segmentation for lidars. 2021.

Minghan Zhu, Maani Ghaffari, William A. Clark, and Huei Peng. E^{2}pn: Efficient se(3)-equivariant point network, 2022.

A Appendix

A.1 Description of Datasets

CIFAR-10 dataset is drawn from the labeled subsets of the 80 million tiny images dataset. The CIFAR-10 dataset consists of 60,000 color images in 10 classes, with 6000 images per class. CIFAR-10 dataset contains 50,000 training images and 10,000 testing images, with the image resolution 32×32. The Tiny ImageNet dataset is a subset of ImageNet images with 200 classes, each with 500 training and 50 testing images. The images in Tiny ImageNet are resized with the resolution 64×64. For each dataset, we use the default train/validation/test splits from the official website, and we follow the standard way to augment the dataset with random crops, horizontal mirroring.

A.2 Detail descriptions about the baseline methods

`BadNets`. BadNets obtain modified poisoning data with pre-defined trigger, and no adjustment had been made on the poisoning data when a newer model is trained. Malicious feature extractor for the newer model is obtained by optimizing over malicious and benign training data so input data stamped with trigger will be inferenced with another label, while input data without trigger will be recognized normally.

`TrojanNN`. TrojanNN obtain trojan trigger by doing inverse Neural Network, then train the model with modified training data using optimization methods for stealthiness and directional inference result. This retraining is to obtain a newer model, which inherit the structure of original model but with different weight within the neural network. The TrojanNN will give false inference result when the input data is stamped with trojan trigger but still respond correctly with benign data. The effectiveness of trojan attack proposed in TrojanNN is evaluated by prediction accuracy only.

A.3 Implementation details

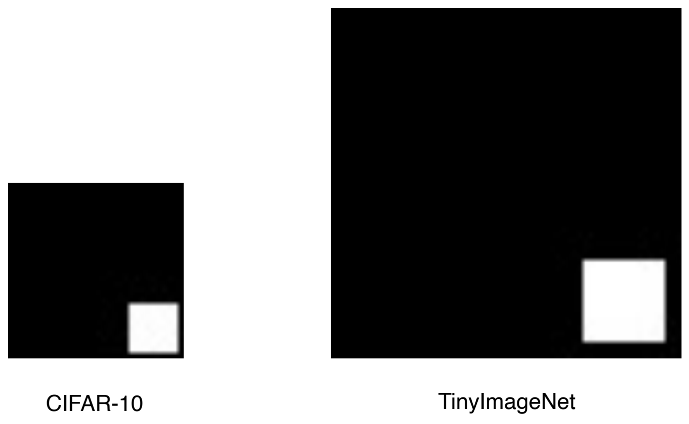

Figure 5: The Shape and the Position of the Adversarial Trigger

figure 5 visualize our adversarial triggers for CIFAR-10 and TinyImageNet datasets.