

DeepMTD: Moving Target Defense for Deep Visual Sensing against Adversarial Examples

QUN SONG, Energy Research Institute, Interdisciplinary Graduate School and School of Computer Science and Engineering, Nanyang Technological University, Singapore

ZHENYU YAN, School of Computer Science and Engineering, Nanyang Technological University, Singapore

RUI TAN, School of Computer Science and Engineering, Nanyang Technological University, Singapore

Deep learning-based visual sensing has achieved attractive accuracy but is shown vulnerable to adversarial attacks. Specifically, once the attackers obtain the deep model, they can construct adversarial examples to mislead the model to yield wrong classification results. Deployable adversarial examples such as small stickers pasted on the road signs and lanes have been shown effective in misleading advanced driver-assistance systems. Most existing countermeasures against adversarial examples build their security on the attackers' ignorance of the defense mechanisms. Thus, they fall short of following Kerckhoffs's principle and can be subverted once the attackers know the details of the defense. This paper applies the strategy of *moving target defense* (MTD) to generate multiple new deep models after system deployment, that will collaboratively detect and thwart adversarial examples. Our MTD design is based on the adversarial examples' minor transferability across different models. The post-deployment of dynamically generated models significantly increase the bar of successful attacks. We also apply serial data fusion with early stopping to reduce the inference time by a factor of up to 5, as well as exploit hardware inference accelerators' characteristics to strike better trade-offs between inference time and power consumption. Evaluation based on three datasets including a road sign dataset and two GPU-equipped embedded computing boards shows the effectiveness and efficiency of our approach in counteracting the attack.

CCS Concepts: • **Security and privacy** → *Software and application security*; • **Computing methodologies** → *Neural networks*; • **Computer systems organization** → *Embedded and cyber-physical systems*.

Additional Key Words and Phrases: Deep neural networks, adversarial examples, moving target defense, embedded computer vision

ACM Reference Format:

Qun Song, Zhenyu Yan, and Rui Tan. 2021. DeepMTD: Moving Target Defense for Deep Visual Sensing against Adversarial Examples. *ACM Trans. Sensor Netw.* 1, 1, Article 1 (January 2021), 32 pages. <https://doi.org/10.1145/3469032>

A preliminary version of this work appeared in The 17th ACM Conference on Embedded Networked Sensor Systems (SenSys 2019). This research is supported in part by a Start-up Grant at Nanyang Technological University and in part by the National Research Foundation, Singapore and National University of Singapore through its National Satellite of Excellence in Trustworthy Computing for Secure Smart Nation Grant (TCSSNG) award no. NSOE-TSS2020-01. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore and National University of Singapore (including its National Satellite of Excellence in Trustworthy Software Systems (NSOE-TSS) office).

Authors' addresses: Qun Song, song0167@e.ntu.edu.sg; Zhenyu Yan, zhenyu.yan@ntu.edu.sg, School of Computer Science and Engineering, Nanyang Technological University, Block N4 #B2a-01, 50 Nanyang Avenue, Singapore 639798. Rui Tan, tanrui@ntu.edu.sg, School of Computer Science and Engineering, Nanyang Technological University, Block N4 #02a-32, 50 Nanyang Avenue, Singapore 639798.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2021 Association for Computing Machinery.

1550-4859/2021/1-ART1 \$15.00

<https://doi.org/10.1145/3469032>

1 INTRODUCTION

The outstanding capability of deep learning in capturing sophisticated patterns have attracted great interests from the embedded sensing research for various applications such as sign language translation [10], audio sensing [18], and surgical activities recognition [22]. Deep learning-based embedded computer vision has also been increasingly adopted on commercial off-the-shelf advanced driver-assistance systems (ADAS). To implement the long envisaged self-driving cars, the accurate and resilient perception of the environment is often the most challenging step in the closed loop of sensing, decision, and actuation. However, recent studies show that deep models (e.g., multilayer perceptrons and convolutional neural networks), albeit being highly accurate, are vulnerable to *adversarial examples*, which are inputs formed by applying small but crafted perturbations to the *clean examples* in order to make the victim deep model yield wrong classification results. As reviewed in [29], systematic approaches have been developed to generate adversarial examples as long as the attackers acquire the deep model, where the attackers may know the internals of the model or not. Certain constraints can be considered in the generation process when the attackers cannot tamper with every pixel of the input. For example, in [9], an algorithm is developed to determine *adversarial stickers* that can be implemented by physically pasting small paper stickers on road signs to mislead vision-based sign classifier. Moreover, as demonstrated in [1], the vision-based lane detector of Tesla Autopilot, which is an ADAS, can be fooled by small adversarial stickers on the road and thus direct the car to the opposite lane. Therefore, adversarial examples present an immediate and real threat to deep visual sensing systems.

Existing countermeasures aim at increasing the deep models' robustness against the adversarial examples by adversarial training [13, 24], adding an input transformation layer [7, 14], and gradient masking [26, 30]. These countermeasures are often designed to address certain adversarial examples. For example, the adversarial training approaches only enhance the deep model's robustness against the adversarial examples considered during the training. Moreover, these static countermeasures build their security on the attackers' ignorance of the defense mechanisms. The countermeasures based on the input transformation and gradient masking can be subverted once the attackers know the details of the used defense mechanisms [4, 5]. Thus, the existing static countermeasures do not address adaptive attackers and fall short of following Kerckhoffs's principle in designing secure systems (i.e., the enemy knows the system except for the secret key [32]). Once the attackers acquire the hardened model and the details of the defense, they can craft the next-generation adversarial examples to render the hardened model vulnerable again.

Beyond the static defense, in this paper, we consider a *moving target defense* (MTD) strategy [16]. MTD aims to create and deploy mechanisms that are diverse and continually change over time to increase complexity and cost for attackers [2]. In the MTD of this work, we generate one or more new deep models after system deployment that the attackers can hardly acquire. Taking the deep visual sensing of ADAS as an example, under the MTD strategy, new deep models can be continually trained when the computing unit of a car is idle. Once the training completes with the validation accuracy meeting specified requirement, the new deep models can be commissioned to replace the in-service models that were previously generated on the car. By bootstrapping the *in situ* training with randomness, it will be practically difficult for the attackers to acquire the in-service deep models, which thus can be viewed as the secret of the system. With MTD, the adversarial examples constructed based on the stolen deep models are neither effective across many systems nor effective against a single victim system over a long period of time.

In this paper, we design an MTD approach for embedded deep visual sensing systems that are susceptible to adversarial examples. Several challenges need to be addressed. First, adversarial examples have non-negligible transferability to new deep models [13]. From our evaluation based

on several datasets, the adversarial examples can mislead the new deep models with a probability from 7% to 52%. Second, the primitive MTD design of using a single new deep model does not give awareness of the presence of adversarial examples, thus losing the opportunities of involving the human to improve the safety of the system. Note that human can be considered immune to adversarial examples designed based on the perturbation minimization principle. Third, *in situ* training of the new deep models without resorting to the cloud is desirable given the concerns of eavesdropping and tampering during the communications over the Internet. However, the training may incur significant computational overhead for the embedded systems.

To collectively address the above challenges, we propose a *DeepMTD* approach based on three key observations on the responses of new deep models to the adversarial examples. First, the output of a new deep model that is successfully misled by an adversarial example tends to be unpredictable. Second, from the unpredictability of the misled new model's output, if we use sufficiently many distinct new models to classify an adversarial example, the inconsistency of all the models' outputs (due to the unpredictability) signals the presence of attack while the undetected adversarial examples will be given the correct classification result by a majority of the new models. Third, compared with training a new deep model from scratch, the training with a perturbed version of the base model as the starting point can converge up to 4x faster, imposing less computation burden.

Based on the above observations, we design DeepMTD as follows. When the system has spare computing resources, it adds independent perturbations to the parameters of the base model to generate multiple *fork models*. The base model can be a factory-designed deep model that gives certified accuracy for clean examples, but may be acquired by the attackers. Each fork model is then used as the starting point of a retraining process. The retrained fork models are then commissioned for the visual sensing task. As the fork models are retrained from the base model, intuitively, they will inherit the classification capability of the base model for clean examples. At run time, an input, which may be an adversarial example constructed based on the base model, is fed into each fork model. If the degree of inconsistency among the fork models' outputs exceeds a predefined level, the input is detected as an adversarial example. The majority of the fork models' outputs is yielded as the final result of the sensing task. If the system operates in the human-in-the-loop mode, the human will be requested to classify detected adversarial examples.

Our multi-model design echoes ensemble machine learning [8]. The existing studies [15, 39] have considered using an ensemble of deep models to counteract adversarial examples. However, these existing studies still focus on *static ensemble* and fall short of addressing adaptive attackers. Once the adaptive attackers obtain the static ensemble, they can still construct effective adversarial examples against the ensemble. As shown in this paper, the adaptive attackers can subvert the static ensemble-based defense with substantial probabilities from 50% to 57%. Different from the static ensemble, in our approach, the *in situ* models generated and continuously updated after the system deployment are distinct both over time and across the systems. This invalidates an essential basis for the attackers to construct effective adversarial examples, i.e., the acquisition of ensemble.

The run-time inference overhead of DeepMTD is proportional to the number of fork models used. Based on our performance profiling on two GPU-equipped embedded computing platforms, instructing TensorFlow to execute the fork models at the same time brings limited benefit in shortening inference time. In contrast, the serial execution of them admits an early stopping mechanism inspired by the serial signal detection [27]. Specifically, the system runs the fork models in serial and terminates the execution once sufficient confidence is accumulated to decide the cleanness of the input. Evaluation results show that the serial DeepMTD reduces the inference time by a factor of up to 5. We also optimize the implementation of DeepMTD by exploiting the characteristics of the hardware inference accelerators.

The contributions of this paper are summarized as follows.

- Based on the observations on the responses of multiple deep models to adversarial examples, we design DeepMTD to counteract adversarial example as an ongoing concern. Different from existing defenses that are static, our dynamic defense approach follows the strategy of moving target defense and utilizes computing to enhance security.
- We conduct extensive evaluation on DeepMTD's accuracy in classifying clean examples as well as its performance in detecting and thwarting adversarial examples under a wide range of settings. The results provide useful guidelines for adopters of DeepMTD in specific applications. In particular, we consider two types of advanced adversarial-example attack. The first mimics the DeepMTD workflow and crafts ensemble adversarial examples; The second acquires the system's training dataset and generates the universal adversarial perturbations.
- We show that the serial execution of the fork models with early stopping significantly reduces the inference time of DeepMTD while maintaining the sensing accuracy in both the absence and presence of attacks. We also exploit the hardware characteristics of the Jetson devices to improve the power efficiency of DeepMTD implementation.

The remainder of this paper is organized as follows. Section 2 reviews background. Section 3 presents a measurement study. Section 4 and Section 5 designs and evaluates DeepMTD, respectively. Section 6 profiles DeepMTD on hardware and evaluates serial DeepMTD. Section 7 studies the optimization of DeepMTD implementation. Section 8 concludes this paper.

2 BACKGROUND AND RELATED WORK

In this section, we present the preliminary of adversarial example (Section 2.1), review the existing studies on developing countermeasures against adversarial examples (Section 2.2) and the broader studies on machine learning in embedded systems (Section 2.3).

2.1 Adversarial Examples and Construction

Adversarial examples are crafted inputs aiming to mislead deep models to produce incorrect results. Let $f_{\theta}(\mathbf{x})$ denote a classifier, where θ is the classifier's parameter and $\mathbf{x} \in [0, 1]^m$ is the input (e.g., an image). Let y denote the ground truth label of \mathbf{x} . The $\mathbf{x}' = \mathbf{x} + \delta \in [0, 1]^m$ is an adversarial example, if $f_{\theta}(\mathbf{x}) = y$ and $f_{\theta}(\mathbf{x}') \neq y$. The δ is the perturbation designed by the attackers. A *targeted* adversarial example \mathbf{x}' makes $f_{\theta}(\mathbf{x}') = y_t$, where $y_t \neq y$ is a specified target label. A *non-targeted* adversarial example ensures that the classification result $f_{\theta}(\mathbf{x}')$ is an arbitrary label other than the true label y . If the attackers need no knowledge of the classifier's internals, the attack is called *black-box* attack. Otherwise, it is called *white-box* attack. In this work, we consider both targeted and non-targeted adversarial examples. As we aim to develop defense, it is beneficial to consider the stronger white-box attack, in which the attackers have the knowledge of the internals of the base model.

To increase the stealthiness of the attack to human perception, the difference between \mathbf{x} and \mathbf{x}' , denoted by $D(\mathbf{x}, \mathbf{x}')$, is to be minimized. Thus, the construction of the perturbation for a targeted adversarial example, denoted by $\delta_{y_t}^*$, can be formulated as [38]: $\delta_{y_t}^* = \operatorname{argmin}_{\delta} D(\mathbf{x}, \mathbf{x}')$, subject to $f_{\theta}(\mathbf{x}') = y_t$ and $\mathbf{x}' \in [0, 1]^m$. The targeted adversarial example that gives the minimum $D(\mathbf{x}, \mathbf{x}')$ can be yielded as a non-targeted adversarial example. Various *gradient-based* approaches have been proposed to construct adversarial examples [6, 13, 38]. Among them, the approach proposed by Carlini and Wagner (C&W) [6] is often thought highly effective. Thus, we use C&W's approach to generate adversarial examples. Specifically, the C&W attack reformulates the problem of constructing the targeted adversarial perturbation as: $\delta_{y_t}^* = \operatorname{argmin}_{\delta} D(\mathbf{x}, \mathbf{x}')$ such that a loss function $L(\mathbf{x}') \leq 0$ and $\mathbf{x}' \in [0, 1]^m$, where $f_{\theta}(\mathbf{x}') = y_t$ if and only if $L(\mathbf{x}') \leq 0$. Note that the specific

form of the loss function $L(\mathbf{x}')$ will be presented shortly. After the reformulation, the Lagrangian relaxation is applied to simplify the problem as: $\delta_{y_t}^* = \operatorname{argmin}_{\delta} D(\mathbf{x}, \mathbf{x}') + c \cdot L(\mathbf{x}')$ such that $\mathbf{x}' \in [0, 1]^m$, where c is a constant weight for combining the two minimization objectives (i.e., $D(\mathbf{x}, \mathbf{x}')$ and $L(\mathbf{x}')$). The specific form of the function $L(\mathbf{x}')$ is $L(\mathbf{x}') = \max\{\max_{y_i \neq y_t} \{Z(\mathbf{x}')_{y_i}\} - Z(\mathbf{x}')_{y_t}, -\kappa\}$, where $Z(\cdot)$ represents the logits output of the classifier $f(\cdot)$. This form is chosen from various candidates via extensive empirical evaluation in [6]. The parameter κ controls the strength of the constructed adversarial example. With a larger κ , the \mathbf{x}' is more likely classified as y_t , but the perturbation δ will be larger. Details of the attack construction can be found in [6]. Note that the design of DeepMTD does not rely on any specifics of the C&W's approach.

2.2 Countermeasures to Adversarial Examples

Overfitted models are often thought highly vulnerable to adversarial example attacks. However, regularization approaches for preventing overfitting, such as dropout and weight decay, are shown ineffective in precluding adversarial examples [13, 38]. Brute-force adversarial training [13, 24] can make a deep model immune to predefined adversarial examples. However, it can be defeated by the adversarial examples that are not considered during the adversarial training. A range of other defense approaches [7, 14] apply various transformations to the input during both the training and inference phases. However, the transformations often result in loss of accuracy on clean samples. *Gradient masking* is another category of defense against the adversarial examples constructed using gradient-based methods [26, 30]. It attempts to deny adversary access to useful gradients for constructing attack. However, as shown in [4, 5], if the attackers know the details of the transformation or the gradient masking, they can still construct effective adversarial examples. *Provable defense* [28, 40] gives lower bounds of the defense robustness. Its key limitation is that the lower bound is applicable for a set of specific adversarial examples only. Using a *static ensemble* of multiple models has been proposed as a possible defense [15, 39], since it is harder to mislead all the models of the ensemble than a single model. However, the adaptive attackers who have exfiltrated the ensemble can subvert the static ensemble-based defense [15]. The key difference between our approach and the existing approaches is that ours is a dynamic defense while existing approaches are static. Once the attackers acquire the details of a static defense, the attackers can design the next-generation adversarial examples and bypass the defense.

In the defense approach described in [31], a deep model is randomly selected from a set of candidate models each time to classify an input. The approach uses a limited number of candidate models (e.g., 3 to 6 [31]) and assumes that they are known to the attackers. Its effectiveness of thwarting the attacks is merely based on the attackers' ignorance of which model is being used, thus following a weak form of MTD. Given the limited number of candidate models, it is not impossible for the attackers to construct an adversarial example that can mislead all candidate models. Moreover, the approach [31] is short of attack detection capability since a single model is used each time. In contrast, DeepMTD applies an ensemble of locally generated deep models to achieve both attack detection and thwarting capabilities. Thus, DeepMTD follows a strong form of MTD.

Our prior work [34] presented the design of DeepMTD and sensing performance evaluation results. In this paper, we further consider two types of smart attackers who (1) follow the DeepMTD workflow to generate fork models from the acquired base model and design the *ensemble adversarial example* against the self-generated fork models, and (2) acquire the training dataset of DeepMTD and craft the *universal adversarial perturbation*. In Section 5.4, we show DeepMTD's effectiveness in counteracting these new attacks. In Section 7, we also make a major extension on the implementation of DeepMTD on hardware platforms (specifically, NVIDIA's Jetson embedded GPUs). The

extension gives a time- and power-efficient system implementation of DeepMTD that integrates both an algorithmic speed-up technique (i.e., early stopping) and exploitation of hardware inference accelerator characteristics (i.e., configurable power mode).

2.3 Machine Learning on Embedded Systems

Various recent studies focus on deploying deep models and/or deep learning algorithms on embedded systems. They aim to improve the performance (e.g., throughput, latency, inference accuracy, etc) of the deep models and deep learning algorithms on the resource-constrained embedded platforms. For example, to improve the inference performance of deep models, approaches based on input-adaptive early exit mechanism [11, 20] and system's runtime dynamics adaptation on a single device [12] or across the cluster of the end and edge devices [19, 45] have been studied. The work in [47] aims to enable the resource-demanding model training on embedded devices by dynamically controlling the structure of DNNs for resource saving. In [46], a ChainSGD-reduce algorithm and reinforcement learning are applied to address the resource limitation of on-device federated learning. The work in [43] uses a tree regressor that models the execution time of the deep model-based application to steer the on-device neural network compression, thus achieving a better accuracy-efficiency tradeoff. The hardware-aware neural architecture search automates the design of deep models under hardware constraints to reduce the manual efforts in designing efficient deep models on embedded and mobile devices [36, 37].

Another category of existing studies applies deep learning to improve the performance of the embedded applications. For instance, the work in [44] uses the generative adversarial network (GAN) to reduce the data labeling effort in IoT systems. The work in [41] applies an autoencoder to compress the data for offloading at local end devices and reconstruct the data on the edge server. As such, the data transferring latency between the end and edge devices is reduced. The work in [42] utilizes the deep neural network with self-attention module to estimate the quality of sensing inputs. It addresses the problem of sensing performance degradation caused by low-quality sensor inputs in IoT applications such as human activity and gesture recognition.

Different from the above reviewed studies that focusing on engineering and using deep learning on embedded systems, we follow the MTD strategy to use computing (i.e., constantly retraining new models during the idle period after the system deployment and joint inference using multiple generated models) to enhance security against the possible adversarial examples attack.

3 MEASUREMENT STUDY













We conduct measurements to gain insights for MTD design.

3.1 Used Datasets and Deep Models

We use three datasets in our measurement study:

- **MNIST** [21] consists of 60,000 training samples and 10,000 test samples. Each sample is a 28×28 grayscale image showing a handwritten digit from 0 to 9. We select 5,000 training samples as the validation dataset.
- **CIFAR-10** [17] is a 10-class dataset consisting of 50,000 training samples and 10,000 test samples. Each sample is a 32×32 RGB color image. The 10 classes are airplanes, cars, birds, cats, deers, dogs, frogs, horses, ships, and trucks. We select 5,000 training samples as the validation dataset.
- **GTSRB** [35] (German Traffic Sign Recognition Benchmark) is a 43-class dataset with more than 50,000 images sizing from 15×15 to 250×250 pixels. For convenience, we resize all the

Table 1. Targeted adversarial examples constructed using C&W approach [6] with ℓ_2 -norm and various κ settings.

	Clean example	Attack's target label					
		2	3	2	3	2	3
Ground truth	0						
	1						
		$\kappa = 0$		$\kappa = 45$		$\kappa = 95$	

images to 32×32 pixels by interpolation or downsampling. We divide them into training, validation, and test datasets with 34799, 4410, and 12630 samples, respectively.

We adopt two convolutional neural network (CNN) architectures that have been used in [6] and [26], referred to as CNN-A and CNN-B. Their structures and training hyperparameters can be found in [33]. We apply CNN-A to MNIST. It is trained on MNIST using the momentum-based stochastic gradient descent. CNN-A achieves training and validation accuracy of 99.84% and 99.44%, respectively. We apply CNN-B to CIFAR-10 and GTSRB. CNN-B's main difference from CNN-A is that more convolutional filters and more rectified linear units (ReLU) in the fully connected layers are used to address the more complex patterns of the CIFAR-10 and GTSRB images. Its softmax layer has 10 or 43 classes for CIFAR-10 and GTSRB, respectively. For CIFAR-10, CNN-B achieves a validation accuracy of 79.62%. This result is consistent with those obtained in [6] and [26]. For GTSRB, CNN-B achieves training and validation accuracy of 99.93% and 96.64%, respectively.

3.2 Measurement Results

In this section, we conduct measurements to investigate the responses of multiple *new models* to adversarial examples constructed based on the *base model* that is different from the new models.

3.2.1 Adversarial examples. We use the deep models described in Section 3.1 as the base models for the three datasets. Then, we use the C&W approach described in Section 2.1 to generate adversarial examples based on the base model. Specifically, for each dataset, we select a clean test sample in each class as the basis for constructing the targeted adversarial examples whose targeted labels are the remaining classes. To generate non-targeted adversarial examples for each dataset, we randomly select 100 test samples as the bases for the construction by following the procedure described in Section 2.1. The C&W's adversarial examples are highly effective – all adversarial examples that we generate are effective against the base model.

As described in Section 2.1, the κ is a parameter of the C&W's approach that controls the trade-off between the effectiveness of the attack and the distortion introduced. We vary κ from 0 to 95. The first image column of Table 1 shows two clean examples from MNIST. The rest image columns show a number of targeted adversarial examples constructed with three settings of κ . For instance, all images in the second column will be wrongly classified by the base model as '2'. We can see that with $\kappa = 0$, the perturbations introduced by the attack are almost imperceptible to human eyes without referring to the clean examples. With $\kappa = 45$, there are clear distortions. With $\kappa = 95$, the perturbations may completely erase the figure shapes or create random shapes. More MNIST adversarial examples are presented in [33].

Table 2. Attack success rate (ASR).

MNIST	6.72%
CIFAR-10	17.3%
GTSRB	7.17%
* $\kappa = 0$.	

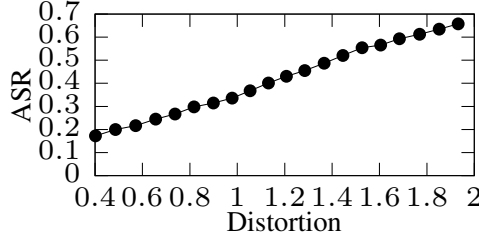


Fig. 1. ASR vs. distortion.

In the rest of this paper, for the sake of attack stealthiness to human, we adopt $\kappa = 0$ unless otherwise specified. To confirm the effectiveness of the adversarial examples with $\kappa = 0$, we conduct an extended experiment with 1,000 targeted adversarial examples of $\kappa = 0$ for MNIST. All these 1,000 adversarial examples are effective against the base model.

3.2.2 Transferability of adversarial examples. In this set of measurements, for each dataset, we train a new model that has the same architecture as the base model. Then, we measure the attack success rate (ASR) of the adversarial examples on the new model. An adversarial example is successful if the deep model yields a wrong label. The ASR characterizes the transferability of the adversarial examples to a model differing from the one used for their construction. Table 2 shows the ASR for the three datasets. We can see that the adversarial examples constructed using the base model can still mislead the new model with probabilities from 7% to 17%. This suggests that the adversarial examples have some transferability across different deep models with the same architecture.

We also evaluate the transferability of the adversarial examples constructed with different κ settings. We use the Euclidean distance between the adversarial example \mathbf{x}' and its corresponding clean example \mathbf{x} to characterize the *distortion* caused by the adversarial perturbation. A larger κ will result in a larger distortion and thus less stealthiness of the attack to human perception. Fig. 1 shows the ASR versus distortion for CIFAR-10. We can see that the ASR increases with the distortion. This shows the trade-off between the attack's transferability and stealthiness to human.

3.2.3 Outputs of multiple new models. From Section 3.2.2, adversarial examples have non-negligible transferability to a new model. Thus, using a single new model may not thwart adversarial example attacks. In this set of measurements, we study the outputs of multiple new models. With the base model for each of the three datasets, we construct 270 targeted adversarial examples (i.e., 90 examples based on each of the ℓ_0 , ℓ_2 , and ℓ_∞ norms as the distance function $D(\mathbf{x}, \mathbf{x}')$) and 300 non-targeted adversarial examples (i.e., 100 examples based on each of the three norms). For each of the three datasets, we independently train 20 new models. We denote by D the number of distinct outputs of the 20 models given an adversarial example. Fig. 2 shows the histogram of D . From the figure, the probability that D is greater than one is 51%. This means that, by simply checking the consistency of the 20 models' outputs, we can detect half of the adversarial example attacks. The probability that $D = 1$ is 49%, which is the probability of all the 20 new models giving the same

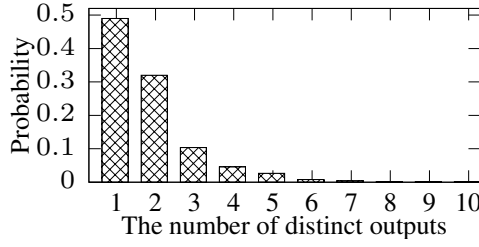


Fig. 2. Distribution of the number of distinct outputs of 20 new models given an adversarial example built using the base model.

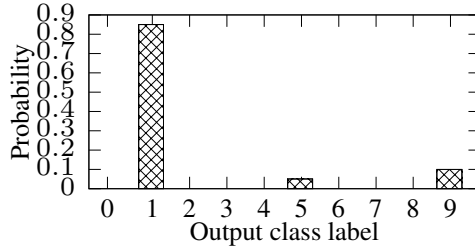


Fig. 3. Distribution of 20 new models' outputs given an adversarial example with ground truth label of 1 and attack target label of 0.

output when the input is an adversarial example. It is also the probability that the consistency check cannot detect whether the input is an adversarial example. Moreover, 99.5% of the adversarial examples that result in $D = 1$ fail to mislead any new model (i.e., all the 20 new models yield the correct classification results). This result suggests that, when the input is an adversarial example, even if the consistency check does not detect whether the input is an adversarial example, a majority voting by the 20 new models can give correct classification result with a probability of 99.5%.

We now use an example to illustrate whether an adversarial example resulting in $D > 1$ can be thwarted. Fig. 3 shows the histogram of the 20 new models' outputs given a targeted CIFAR-10 adversarial example with a ground truth label of 1 and a target label of 0. We can see that most new models yield the ground truth label and only a few models yield labels rather than the attack's target label. This shows that the wrong outputs of the new models tend to be unpredictable, rather than the attack's target label. It also suggests that a majority voting from the distinct outputs of the new models can thwart the attack.

3.2.4 Retraining perturbed base model. The results in Section 3.2.3 suggest that an ensemble of multiple new models is promising for detecting and thwarting adversarial example attacks. However, the training of the new models may incur significant computation overhead. In this section, we investigate a retraining approach. Specifically, we add perturbations to the trained base model and use the result as the starting point of a retraining process to generate a new model. The model perturbation is as follows. For each parameter matrix \mathbf{M} of the base model, we add an independent perturbation to each element in \mathbf{M} . The perturbation is drawn randomly and uniformly from $[w \cdot \min(\mathbf{M}), w \cdot \max(\mathbf{M})]$, where $\min(\mathbf{M})$ and $\max(\mathbf{M})$ represent the smallest and largest elements of \mathbf{M} , respectively, and w controls the intensity of the perturbation. The system stops the retraining process if the validation accuracy stops increasing for five consecutive epochs. Then, the model in the retraining epoch that gives the highest validation accuracy is yielded as a new model. In this

Table 3. The number of epochs for new model retraining.

Intensity of perturbation (w)	Dataset		
	MNIST	CIFAR-10	GTSRB
0.1	11	11	12
0.2	12	13	13
0.3	13	18	13
training from scratch	23	44	22

paper, we retrain the new models with all the training data used for training the base model. Table 3 shows the number of epochs for retraining a new model versus the intensity of the perturbation. We can see that the number of epochs increases with the perturbation intensity. As a comparison, when a new model is trained from scratch with the same stopping criterion, the number of epochs can be up to 4x higher than that with $w = 0.1$. We measure the time for retraining 20 new models from perturbed versions of the base model using the entire GTSRB training dataset consisting of 34,799 images on the AGX Xavier computing board. It takes about 45 minutes.

4 DESIGN OF DEEPMTD

The measurement results in Section 3 suggest an MTD design to counteract adversarial examples. In brief, multiple *fork models* can be generated dynamically by retraining independently perturbed versions of the base model. A consistency check on the fork models' outputs can detect whether the input is an adversarial example; the majority of their outputs can be yielded as the final classification result to thwart the adversarial example attack if present.

4.1 System and Threat Models

Consider an embedded visual sensing system ("the system" for short), which can execute the inference and training of the used deep model. In this paper, we focus on a single image classification task. Image classification is a basic building block of many visual sensing systems. The classification results can be used to direct the system's actuation. We assume that the system has a factory-designed model that gives certified accuracy on clean examples and specified adversarial examples. The system also has a training dataset that can be used to train a new deep model locally that achieves a satisfactory classification accuracy as that given by the factory model.

We assume that the attackers cannot corrupt the system. Given that the factory model is static, we assume that the attackers can acquire it via memory extraction, data exfiltration attack, or insiders (e.g., unsatisfied or socially engineered employees). We also assume that the attackers can acquire the training dataset on the system, since the dataset is also a static factory setting. We assume that the attackers can construct stealthy targeted or non-targeted adversarial examples with a white-box approach (e.g., the C&W approach [6]) based on the factory model or any deep model trained by the attackers using the dataset. Since the focus of this paper is to develop defense, it is beneficial to conservatively consider strong attackers who can launch white-box attacks.

4.2 DeepMTD Work Flow

Fig. 4 overviews the work flow of DeepMTD. We consider two operating modes of DeepMTD: *autonomous* and *human-in-the-loop*. Both modes have the following components.

4.2.1 Fork models generation. To "move the target", the system generates new deep models locally for image classification. Specifically, we adopt the approach described in Section 3.2.4 to perturb

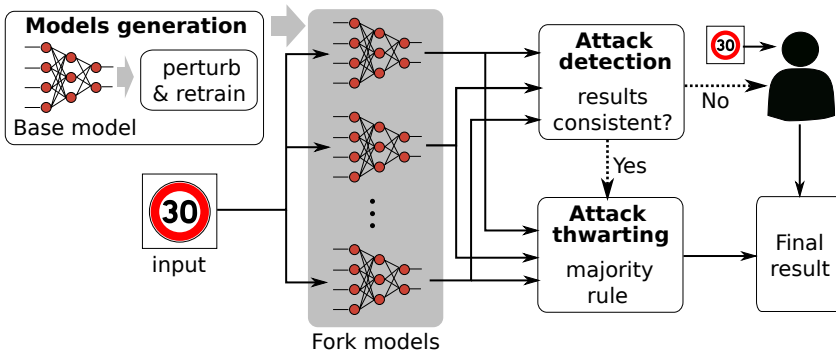


Fig. 4. Workflow of DeepMTD. In the *autonomous* mode, the attack thwarting module is executed regardless of the attack detection result. In the *human-in-the-loop* mode, the attack thwarting module is executed only when the attack detection gives a positive detection result.

the base model with a specified intensity level w and retrain the perturbed model using the training data to generate a fork model. The retraining completes when the validation accuracy meets a certain criterion. Using the above procedure, a total of N fork models are generated *independently*. We now discuss several issues.

From our evaluation results in Section 5, a larger setting of N in general leads to better performance in counteracting the adversarial examples. Thus, the largest setting subject to the computation resource constraints and run-time inference timeliness requirements can be adopted. In Section 6, we will investigate the run-time overhead of the fork models.

The fork models generation can be performed right after receiving each new release of the factory-designed model from the system manufacturer. For example, as measured in Section 3.2.4, generating 20 fork models for road sign recognition requires 45 minutes only. To further improve the system's security, the fork models generation can also be performed periodically or continuously whenever the computing unit of the system is idle. For instance, an electric car can perform the generation when it is being charged during nights.

Since the fork model is retrained from a perturbed version of the base model, the fork model may converge to the base model. However, as the stochastic gradient descent used in the training also incorporates randomness and a deep model often has a large degree of freedom, with a sufficient perturbation intensity level w , the fork model is most unlikely identical to the base model. Nevertheless, MTD is not meant for perfect security, but for significantly increased barriers for the attackers to launch effective attacks.

4.2.2 Attack detection. An input is sent to all fork models for classification. From the observations in Section 3.2.3, we can check the consistency of the outputs of all the fork models to detect whether the input is an adversarial example. If more than $T \times 100\%$ of the outputs are the same, the input is detected as a clean example; otherwise, it is detected as an adversarial example. T is a threshold that can be configured to achieve various satisfactory trade-offs. We will evaluate its impact on the performance of the system and discuss its setting in Section 5.

4.2.3 Attack thwarting. Attack thwarting aims to give the genuine label of an adversarial example. From the observations in Section 3.2.3, we apply the majority rule to thwart the adversarial example attack. Specifically, the most frequent label among the N fork models' outputs is yielded as the final result.

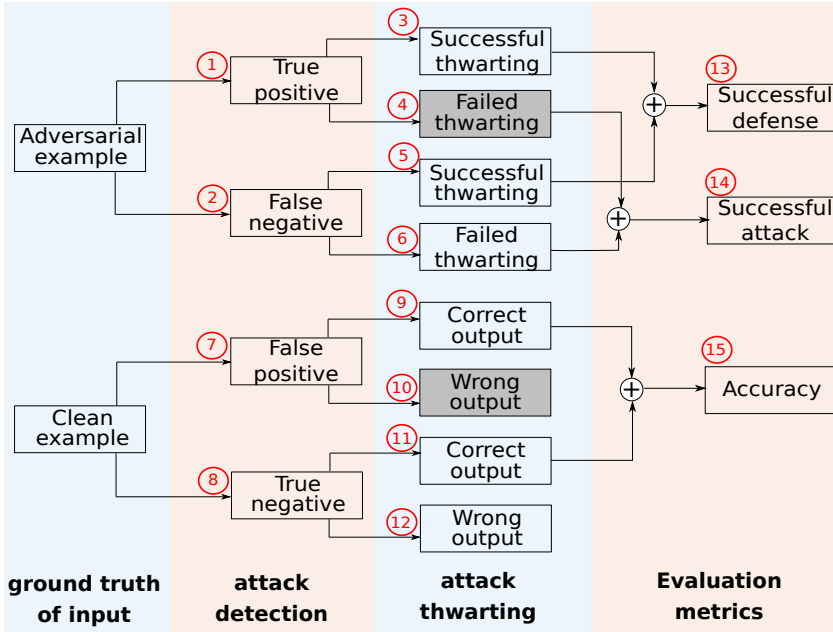


Fig. 5. Categorization of the system's attack detection and thwarting results and the evaluation metrics. The shaded blocks of "Failed thwarting" and "Wrong output" are not applicable to human-in-the-loop DeepMTD.

In the autonomous mode, regardless of the attack detection result, the system will execute the attack thwarting component to generate the final result for the autonomous actuation of the system. Differently, in the human-in-the-loop mode, upon a detection of adversarial example, the system will ask the human operator to classify the input and use the result for the system's subsequent actuation; if no attack is detected, the system will execute the attack thwarting component to yield the final classification result for the subsequent actuation. In this paper, we assume that the human operator will not make any classification error. With this assumption, our performance metrics analysis (Section 4.3) and evaluation (Section 5) will provide essential understanding on how the human operator's involvement enabled by DeepMTD's attack detection capability improves the system's safety in the presence of attacks.

We study both the autonomous and human-in-the-loop modes to understand how human affects the system's performance in the absence and presence of adversarial example attacks. Fully autonomous safety-critical systems in complex environments (e.g., self-driving cars) are still grand challenges. For example, all existing off-the-shelf ADAS still requires the driver's supervision throughout the driving process. In this paper, we use the results of the autonomous mode as a baseline. For either the autonomous or the human-in-the-loop modes, effective countermeasures against adversarial examples must be developed and deployed to achieve trustworthy systems with advancing autonomy.

4.3 Performance Metrics

In this section, we analyze the metrics for characterizing the performance of DeepMTD in the autonomous and human-in-the-loop modes. Fig. 5 illustrates the categorization of the system's detection and thwarting results. In the following, we use \textcircled{x} to refer to a block numbered by x

in Fig. 5. In Section 5, we use p_x to denote the probability of the event described by the block conditioned on the event described by the precedent block. We will illustrate p_x shortly.

When the ground truth of the input is an adversarial example, it may be detected correctly (1) or missed (2). Thus, we use p_1 and p_2 to denote the true positive and false negative rates in attack detection. We now further discuss the two cases of true positive and false negative:

- In case of (1), the autonomous DeepMTD may succeed (3) or fail (4) in thwarting the attack; differently, the human-in-the-loop DeepMTD can always thwart the attack (3). When attack thwarting is successful, the system will yield correct classification result; otherwise, the system will yield wrong classification.
- In case of (2), autonomous or human-in-the-loop DeepMTD may succeed (5) or fail (6) in attack thwarting.

The *successful defense rate* (13) is the sum of the probabilities for (3) and (5). The *attack success rate* (14) is the sum of the probabilities for (4) and (6). Note that, with the autonomous DeepMTD, the two rates are independent of DeepMTD's detection performance, because the attack thwarting component is always executed regardless of the detection result. In contrast, with the human-in-the-loop DeepMTD, the two rates depend on DeepMTD's attack detection performance. In Section 5, we will evaluate the impact of the attack detection performance on the two rates.

When the input is clean, the detector may generate a false positive (7) or a true negative (8).

- In case of (7), the attack thwarting of autonomous DeepMTD may yield a correct (9) or wrong (10) classification result; differently, the human-in-the-loop DeepMTD can always give correct classification.
- In case of (8), the attack thwarting of the autonomous or human-in-the-loop DeepMTD may yield a correct (11) or wrong (12) classification result.

The *accuracy* of the system in the absence of attack (15) is the sum of the probabilities for (9) and (11).

For DeepMTD, the successful defense rate p_{13} and the accuracy p_{15} are the main metrics that characterize the system's performance in the presence and absence of attacks. In the autonomous mode, these two metrics are independent of the attack detection performance. Differently, in the human-in-the-loop mode, they are affected by the attack detection performance. In an extreme case, if the detector always gives positive detection results, the human will take over the classification task every time to give the correct results, causing lots of unnecessary burden to the human in the absence of attack. This unnecessary burden can be characterized by the false positive rate p_7 . There exists a trade-off between this unnecessary burden to human and the system's performance. In summary, the performance of the autonomous DeepMTD and human-in-the-loop DeepMTD can be mainly characterized by the tuples of (p_{13}, p_{15}) and (p_7, p_{13}, p_{15}) , respectively.

5 PERFORMANCE EVALUATION

5.1 Evaluation Methodology and Settings

The evaluation is also based on the three datasets and the two CNN infrastructures described in Section 3.1. We follow the approach described in Section 3.2.1 to generate the adversarial examples. Fig. 6 shows adversarial examples based on two clean GTSRB examples with labels of "1" and "5". The second image in the first row and the sixth image in the second row are clean examples. We can see that the adversarial perturbations are imperceptible. More GTSRB adversarial examples are shown in [33]. The DeepMTD has three configurable parameters: the number of fork models N , the model perturbation intensity w , and the attack detection threshold T . Their default settings are: $N = 20$, $w = 0.2$, $T = 1$.

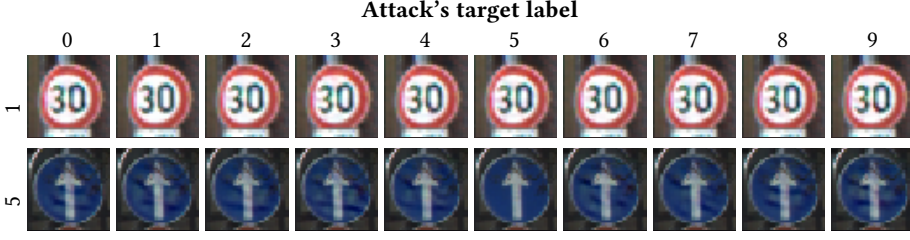


Fig. 6. Targeted adversarial examples constructed using C&W approach [6] with ℓ_2 -norm. Each row consists of adversarial examples generated from the same clean example.

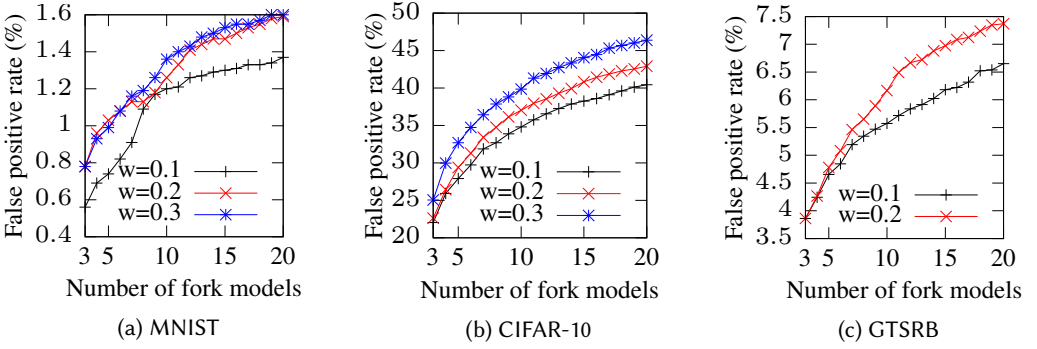


Fig. 7. False positive rate of attack detection (p_7).

5.2 Results in the Absence of Attack

The deployment of the defense should not downgrade the system's sensing accuracy in the absence of attack. This section evaluates this sensing accuracy.

First, we use all clean test samples to evaluate the false positive rate (i.e., p_7). Fig. 7 shows the measured p_7 versus N under various w settings. The p_7 increases with N . This is because, with more fork models, it will be more likely that the fork models give inconsistent results. Moreover, p_7 increases with w . This is because, with a higher model perturbation level, the retrained fork models are likely more different and thus give different results to trigger the attack detection. The p_7 for CIFAR-10 is more than 20%. Such a high p_7 is caused by the high complexity of the CIFAR-10 images. Moreover, the detector with $T = 1$ is very sensitive. With a smaller T , the p_7 will reduce. For instance, with $T = 0.6$, p_7 is around 5%-10%.

Fig. 8 shows the accuracy of the system in the absence of attack (i.e., p_{15}) versus N under various w settings. The curves labeled "scratch" represent the results obtained based on new models trained from scratch, rather than fork models. We can see that training from scratch brings insignificant (less than 2%) accuracy improvement. The horizontal lines in Fig. 8 represent the validation accuracy of the respective base models. We can see that due to the adoption of multiple deep models, the system's accuracy is improved. The results also show that larger settings for N bring insignificant accuracy improvement. Reasons are as follows. First, for MNIST and GTSRB, as the accuracy of a single fork model is already high, the decision fusion based on the majority rule cannot improve the accuracy much. Second, for CIFAR-10, although the accuracy of a single fork model is not high (about 80%), the high correlations among the fork models' outputs impede the effectiveness of decision fusion. The accuracy p_{15} depends on the rates that the attack thwarting module gives

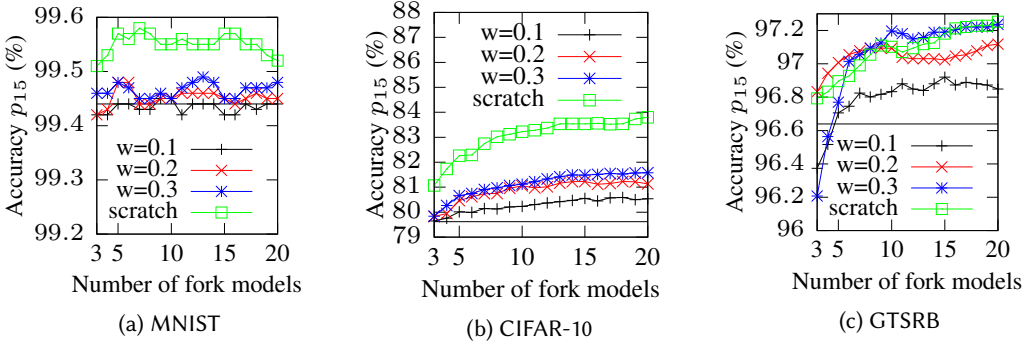


Fig. 8. Accuracy of the system in the absence of attack (p_{15}). The horizontal lines represent the validation accuracy of the respective base models.

correct output for the false positives and true negatives, i.e., p_9 and p_{11} . More results on p_9 and p_{11} can be found in [33].

From Fig. 8c, the accuracy of the road sign recognition is around 97%. The original images in GTSRB have varied resolutions. To facilitate our evaluation, we resized all the images to 32×32 pixels. This low resolution contributes to the 3% error rate. With higher resolutions, this error rate can be further reduced. The main purpose of this evaluation is to show that, in the absence of attacks, DeepMTD can retain or slightly improve the system's accuracy obtained with the base model.

Lastly, we consider the human-in-the-loop DeepMTD. Fig. 9 shows the results based on GTSRB. Specifically, Fig. 9a shows the false positive rate p_7 versus N under various settings for the detection threshold T . The p_7 decreases with T , since the attack detector becomes less sensitive with smaller T settings. The p_7 characterizes the overhead incurred to the human who will make the manual classification when the attack detector raises an alarm. Fig. 9b shows the accuracy p_{15} versus N under various T settings. The curve labeled "auto" is the result for the autonomous DeepMTD. We can see that the human-in-the-loop DeepMTD with $T = 1$ outperforms the autonomous DeepMTD by up to 3% accuracy, bringing the accuracy close to 100%. Fig. 9a and Fig. 9b show a trade-off between the overhead incurred to and the accuracy improvement brought by the human in the loop. To better illustrate this trade-off, Fig. 9c shows the accuracy versus the false positive rate under various model perturbation intensity settings. Different points on a curve are the results obtained with different settings of the attack detection threshold T . We can clearly see that the accuracy increases with the false positive rate.

5.3 Results in the Presence of Attack

We use the targeted adversarial examples to evaluate the performance of DeepMTD in detecting and thwarting attacks. Fig. 10 shows the true positive rate (i.e., p_1) versus N under various settings of w . For the three datasets, the p_1 increases from around 50% to more than 90% when N increases from 3 to 20. This shows that, due to the minor transferability of adversarial examples, increasing the number of fork models is very effective in improving the attack detection performance. For GTSRB, when $w = 0.3$, all attacks can be detected as long as N is greater than 3.

Fig. 11 and Fig. 12 show the rates of successfully thwarting the detected attacks (i.e., p_3) and the missed attacks (i.e., p_5), respectively. In general, these rates increase with N . From the two figures, DeepMTD is more effective in thwarting the missed attacks than the detected attacks. This is because, for a missed attack, all fork models give the same and correct classification result.

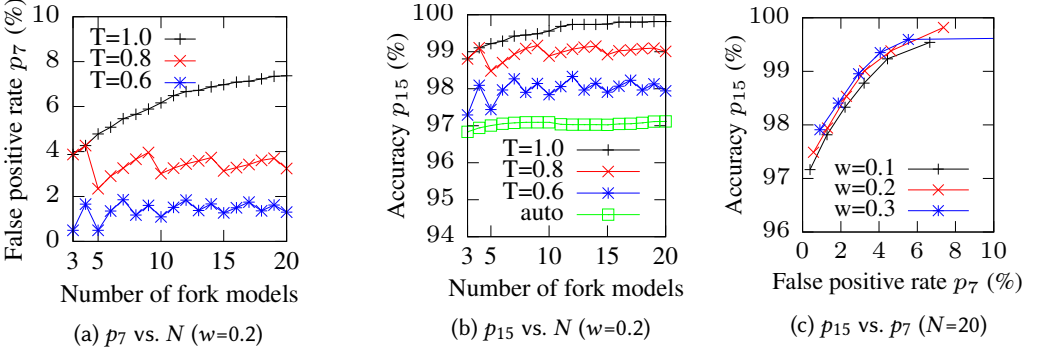


Fig. 9. Performance of human-in-the-loop DeepMTD in the absence of attack. (Dataset: GTSRB)

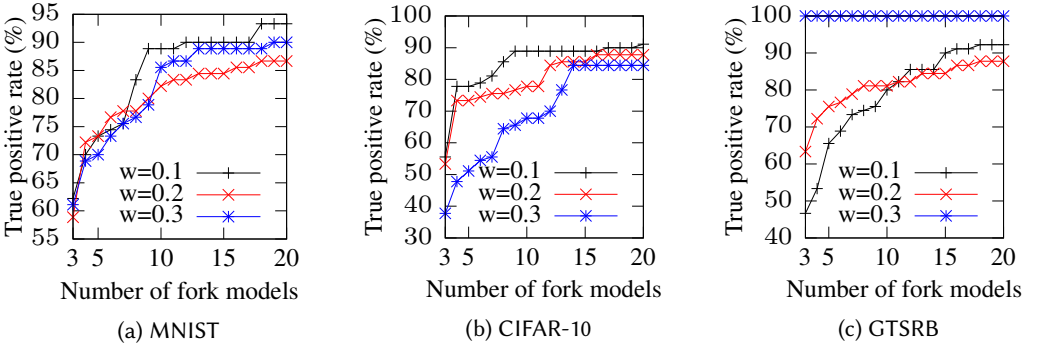


Fig. 10. True positive rate of attack detection (p_1).

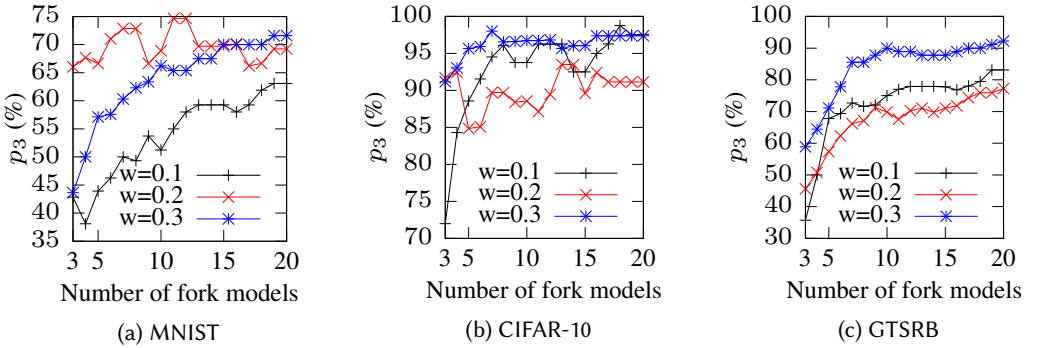
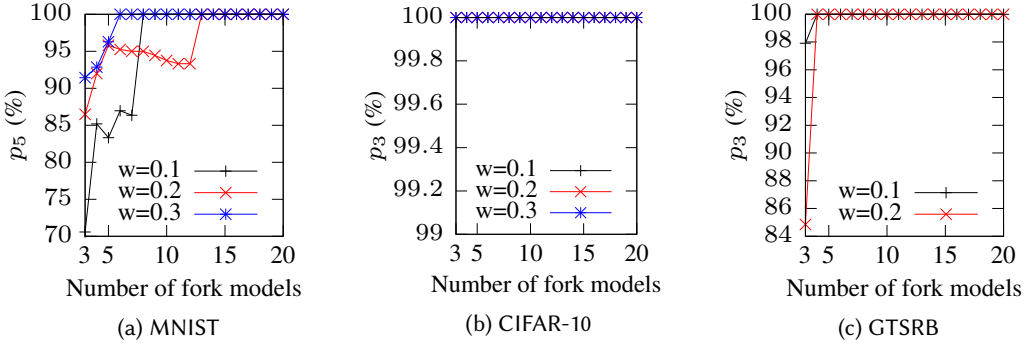
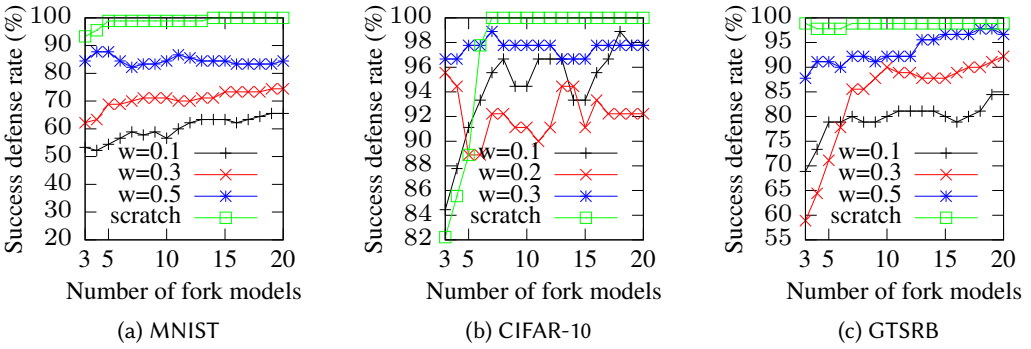


Fig. 11. Rate of thwarting detected attacks (p_3).

However, for the detected attacks, all fork models' results are inconsistent and there is a chance for the majority among the results is a wrong classification result. From Fig. 11a, MNIST has a relatively low p_3 . This is because under the same setting of $\kappa = 0$, the MNIST adversarial examples have larger distortions. The average distortions introduced by the malicious perturbations, as defined in Section 3.2.1, are 1.9 and 0.4 for MNIST and CIFAR-10, respectively. Thus, the strengths of the

Fig. 12. Rate of successfully thwarting missed attacks (p_5).Fig. 13. Successful defense rate (p_{13}).

malicious perturbations applied on MNIST are higher, leading to the lower attack thwarting rates in Fig. 11a.

Fig. 13 shows the successful defense rate (i.e., p_{13}) versus N . The p_{13} has an increasing trend with N . The curves labeled “scratch” represent the results obtained with new models trained from scratch rather than fork models. The DeepMTD achieves successful defense of 98% with $w = 0.3$ for CIFAR-10 and $w = 0.5$ for GTSRB. MNIST has relatively low success defense rates due to the relatively low rates of successfully thwarting detected attacks as shown in Fig. 11a. However, with new models trained from scratch, the successful defense rates for MNIST are nearly 100%. The higher successful defense rates achieved by the new models trained from scratch are due to the lower transferability of adversarial examples to such models. However, training from scratch will incur higher (up to 4x) computation overhead. Thus, there is a trade-off between the attack defense performance and the training computation overhead.

Lastly, we evaluate how the human improves the attack thwarting performance when DeepMTD operates in the human-in-the-loop mode. Fig. 14 shows the results based on GTSRB. With a larger T setting (i.e., the detector is more sensitive), the true positive rate increases, requesting more frequent manual classification by the human. As a result, the successful defense rate can increase to 100%, higher than that of the autonomous DeepMTD. Recalling the results in Fig. 9a, a larger T leads to higher false positive rates and thus higher unnecessary overhead incurred to the human. Thus, there exists a trade-off between the successful defense rate and the unnecessary overhead incurred to the human. To better illustrate this trade-off, Fig. 14c shows the successful defense rate

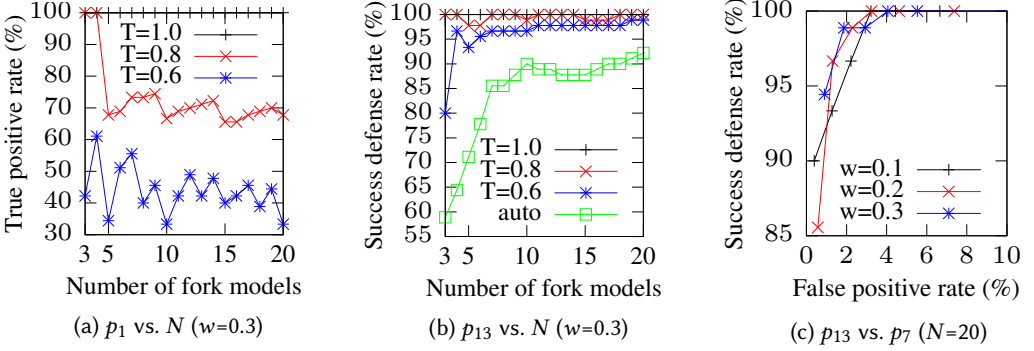


Fig. 14. True positive rate and successful defense rate in the human-in-the-loop mode. (Dataset: GTSRB)

versus the false positive rate. Different points on a curve are the results obtained with different settings of T . We can clearly see that the successful defense rate increases with the false positive rate.





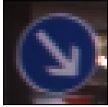

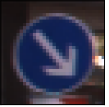





5.4 Counteracting Smarter Attackers

In this section, we consider the smarter attackers who obtain some critical static information of the DeepMTD and adaptively construct adversarial examples. We still follow the assumption that we made for the attacker in Section 4.1, i.e., the attacker cannot break into the system to directly acquire the dynamically updated in-service fork models, because otherwise the attacker should use the access to subvert the whole sensing system directly instead of employing adversarial examples. We consider two kinds of static information: (1) The training dataset and (2) the DeepMTD defense workflow.

We first consider an attacker who obtains both kinds of static information. The attack strategy is to follow the DeepMTD's procedure to generate a set of fork models using the acquired static factory model and the training dataset, and then craft adversarial examples against the fork models generated by the attacker. To find the targeted adversarial perturbation, denoted by $\delta_{y_t}^*$, for an input \mathbf{x} against an ensemble of models such that the majority of these models' outputs is a specified targeted label y_t , the formulation is: $\delta_{y_t}^* = \operatorname{argmin}_{\delta} D(\mathbf{x}, \mathbf{x}')$ such that the majority of the models' outputs is y_t and $\mathbf{x}' = \mathbf{x} + \delta \in [0, 1]^m$. However, it is very difficult to solve this problem because the first constraint (i.e., the majority of the models' outputs is y_t) is non-linear. To make the problem tractable, we update the formulation in Section 2.1 as: $\delta_{y_t}^* = \operatorname{argmin}_{\delta} D(\mathbf{x}, \mathbf{x}') + c \cdot \sum_{k=1}^N L_k(\mathbf{x}')$, where $\mathbf{x}' \in [0, 1]^m$ and $L_k(\mathbf{x}')$ is the value of the loss function $L(\cdot)$ for the k th model of the ensemble. To be more specific, $L_k(\mathbf{x}') = \max\{\max_{y_i \neq y_t} \{Z_k(\mathbf{x}')_{y_i}\} - Z_k(\mathbf{x}')_{y_t}, -\kappa\}$, where $Z_k(\cdot)$ is the logits output of the k th model. Under the above updated formulation, the solution tends to induce the models to output the targeted class label y_t . This formulation for crafting adversarial examples to mislead a model ensemble has been also used in [15, 23]. We use the C&W approach to solve the above optimization problem and generate the *ensemble adversarial examples*. The non-targeted ensemble adversarial example can be derived by constructing the targeted ensemble adversarial examples for all class labels and selecting the one with the smallest $D(\mathbf{x}, \mathbf{x}')$.

From our experiments, the non-targeted ensemble adversarial examples generated based on a set of $N = 20$ fork models can mislead the same set of models with probabilities of 50.25%, 57.15%, and 52% for the MNIST, CIFAR-10, and GTSRB datasets, respectively. Table 4 shows several ensemble adversarial examples constructed in our experiments. We can see that the ensemble adversarial

Table 4. Several samples of the ensemble adversarial example constructed against a set of $N = 20$ fork models. The number below each image on the “Prediction” column is the percentage of the prediction (illustrated by the image) given by the 20 fork models. (Dataset: GTSRB)

Clean	Adversarial	Prediction	
		 40%	 60%
		 45%	 55%
		 35%	 65%

perturbation is subtle but it can mislead the majority of the models of the ensemble. We also measure the effectiveness of the ensemble adversarial examples against a new model that is not used for the attack construction. The ensemble adversarial examples achieve ASRs of 38%, 52%, and 51% for the MNIST, CIFAR-10, and GTSRB datasets on the new model, respectively. We can see that, the ensemble adversarial examples are less effective in misleading the models used for attack construction. Moreover, compared with the adversarial examples crafted against a single base model that are 100% effective against the base model as described in Section 3.2.1, the ensemble adversarial examples only mislead the set of models used for attack construction with the probabilities of 50-57%. This is because it is more difficult to mislead an ensemble of models than a single model. However, the ensemble adversarial examples are 38-52% effective on new single models, higher than the 7-17% effectiveness against the new models of the single-model adversarial examples as shown in Section 3.2.2. An intuitive explanation on this better transferability is that if an adversarial example remains adversarial for multiple models, it is more likely effective to other models.

We also conduct a set of experiments to evaluate the performance of DeepMTD in counteracting this smarter attackers. We randomly select 100 clean examples from each of the three datasets that are correctly classified by the corresponding base model and construct non-targeted ensemble adversarial examples using these clean examples.

First, we evaluate the performance of DeepMTD in detecting the ensemble adversarial examples. Fig. 15 shows the true positive rate (i.e., p_1) versus N . We can see that the true positive rates slightly drop for some of the datasets compared with the results in Fig. 10. Specifically, in Fig. 15, when $w = 0.3$ and $N = 20$, the true positive rates are 87%, 91%, and 94% for the MNIST, CIFAR-10, and GTSRB datasets, respectively. Under the same settings of w and N in Fig. 10, the true positive rates are 90%, 84.4%, and 100% for the MNIST, CIFAR-10, and GTSRB datasets, respectively. The true positive rates can be raised by increasing the model perturbation intensity. For example, when the new models are trained from scratch, denoted by the curves labeled “scratch”, 100% true positive

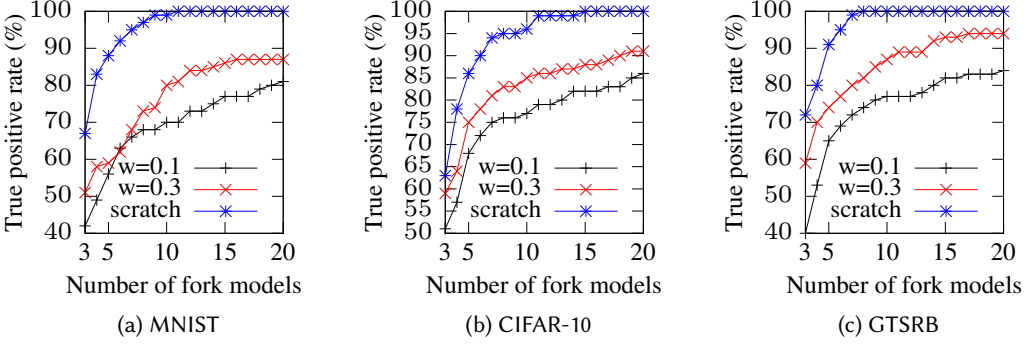


Fig. 15. True positive rate (p_1) for ensemble adversarial examples.

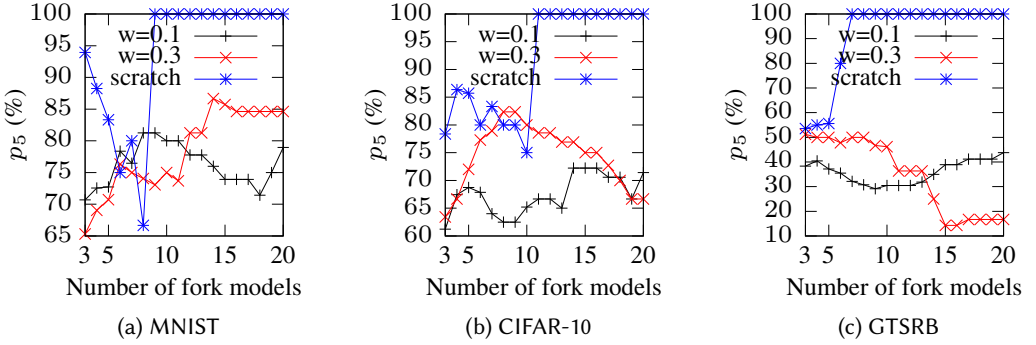


Fig. 16. Rate of successfully thwarting missed attacks (p_5) for ensemble adversarial examples.

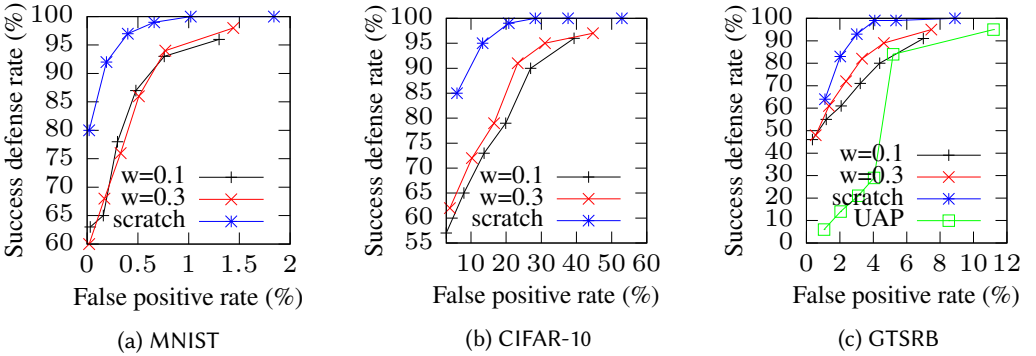


Fig. 17. Successful defense rate p_{13} vs. false positive rate p_7 ($N=20$) in the human-in-the-loop mode for ensemble adversarial examples.

rates can be achieved for all of the three datasets. However, the training computation overhead also increases.

Second, we evaluate the performance of DeepMTD in thwarting the ensemble adversarial examples. Fig. 16 shows the rate of successfully thwarting the missed attacks (i.e., p_5) versus N . We can see that the rates of successfully thwarting the missed attacks for all of the three datasets are

lower than the results in Fig. 12. When $w = 0.3$ and $N = 20$, in Fig. 12, the rates of successfully thwarting missed attacks are 100% for all of the three datasets. In Fig. 16, under the same settings of the w and N , the rates of successfully thwarting missed attacks are 84.6%, 66.7%, 16.7% for the MNIST, CIFAR-10, and GTSRB datasets, respectively. This indicates that more undetected ensemble adversarial examples tend to mislead the majority of the models of the ensemble. In this case, if we raise the model perturbation intensity, a higher rate of successfully thwarting missed attacks can be achieved. Specifically, in Fig. 16, when the new models are trained from scratch, all of the undetected ensemble adversarial examples are thwarted successfully under the setting of $N = 20$. Note that in the case when there is no missed attack (i.e., the false negative rate is 0), we let the rate of successfully thwarting missed attacks to be 100% instead of 0. In Fig. 17, we show the successful defense rate (i.e., p_{13}) versus the false positive rate (i.e., p_7) in the human-in-the-loop mode. Different points in the figure represent the results obtained with different detection thresholds T ranging from 50% to 100%. For the MNIST and GTSRB datasets, we can achieve satisfactory trade-offs between the defense performance and the overhead of human operation. For the defense performance of the GTSRB dataset, when the false positive rate is around 2% and $w = 0.3$, the successful defense rate for ensemble adversarial examples is 72% as shown in Fig. 17c and 98.9% for single-model adversarial examples as shown in Fig. 14c. If we train the new model from scratch, when $w = 0.3$, the successful defense rate can be increased to 99% with a false positive rate of 4%. The high false positive rate for the CIFAR dataset is due to the lower accuracy on clean examples compared with the MNIST and GTSRB datasets, which induces the fork models to yield more different results for clean examples. This result reminds the DeepMTD adopter that the accuracy of the base model on clean examples should be high to avoid high false positive rates. From our experiments, the ensemble adversarial examples have better transferability compared with the adversarial examples constructed based on a single base model. The results also show that there exists a trade-off between the attack defense performance and the training computation overhead.

We also consider another smart attacker who only acquires the training dataset. In this case, the attacker may follow the method in [25] to construct the universal adversarial perturbation (UAP). Based on a set of training data, the UAP algorithm [25] finds the minimum perturbation needed for each input data sample to change the target model's prediction and accumulates the perturbations over all the training data samples. In our experiment, the target model for the generation of the UAP attack is a surrogate model trained from the obtained training dataset. The generated UAP is then effective on many clean input samples. From our experiment, the UAP attack crafted based on the GTSRB base model can mislead the same model with probability of 38% and transfer to a new model with probability of 37%. The performance of DeepMTD in thwarting such UAP attack is shown in Fig. 17c. When the false positive rates are 5% and 11%, the successful defense rates against the UAP attack are 84% and 95%. The result shows that, compared with no defense, the DeepMTD can reduce the error rate under the UAP attack from 38% to 5% only.

From the above experiment results, for the attacker to mimic the DeepMTD workflow and construct the effective ensemble adversarial examples, more computation power and the availability of the training data are the extra requirements, compared with constructing the single-model adversarial examples that requires less computation power and a single base model only. On the defense side, when confronting the smarter adversarial attackers, DeepMTD can still effectively detect and thwart the attacks by increasing the model perturbation intensity at the cost of higher model training overhead.

5.5 Summary and Implication of Results

First, from Fig. 8, in the absence of attack, autonomous DeepMTD does not improve the classification accuracy much when the number of fork models N increases. Differently, from Fig. 13, autonomous

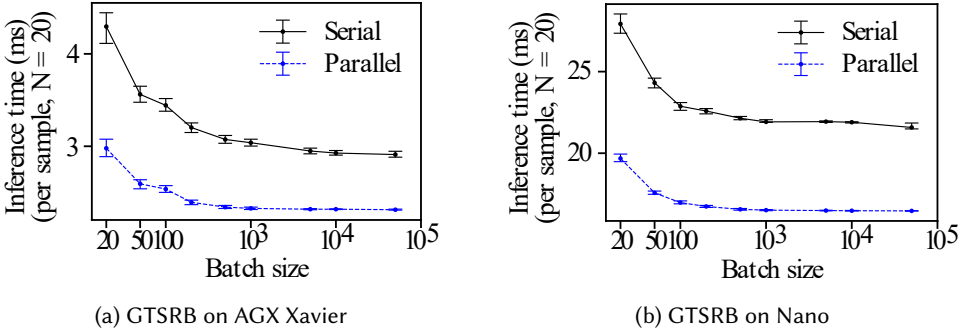


Fig. 18. Per-sample inference times of parallel and serial DeepMTD versus batch size. Error bar represents average, 5th and 95th percentiles over 100 tests under each setting.

DeepMTD's successful defense rate can be substantially improved when N increases. Note that, without DeepMTD, the adversarial example attacks against the static base model are *always* successful. This suggests the necessity of deploying countermeasures.

Second, there exists a trade-off between the successful defense rate and the computation overhead in generating the fork models. Specifically, with more fork models retrained from the base model with larger model perturbation intensity (w), higher successful defense rates can be achieved. However, the retraining will have higher computation overhead as shown in Table 3.

Third, the proposed human-in-the-loop design enables the system to leverage the human's immunity to stealthy adversarial examples. The on-demand involvement of human improves the system's accuracy in the absence of attack and the successful defense rate in the presence of attack, with an overhead incurred to the human that is characterized by the false positive rate. From Fig. 9c and Fig. 14c for the GTSRB road sign dataset, with a false positive rate of 4%, the accuracy without attack is more than 99% and the successful defense rate is nearly 100%. The 4% false positive rate means that, on average, the human will be asked to classify a road sign every 25 clean images of road signs that are detected by ADAS.

Lastly, we show that DeepMTD effectively detects and thwarts the smarter attacks that mimic the DeepMTD workflow to generate fork models from the acquired base model and then design the ensemble adversarial examples against the self-generated fork models.

6 SERIAL DEEPMTD WITH EARLY STOPPING

In this section, we investigate the run-time overhead of DeepMTD implementations on two GPU-equipped embedded computing boards. As many visual sensing systems need to meet real-time requirements, we also investigate how to reduce the run-time overhead of DeepMTD without compromising its accuracy and defense performance.

6.1 DeepMTD Implementation and Profiling

We implement DeepMTD on an NVIDIA Jetson AGX Xavier and an NVIDIA Jetson Nano. We deploy the fork models for GTSRB on both devices. AGX Xavier is equipped with an octal-core ARM CPU, a 512-core Volta GPU with 64 tensor cores, and 16GB LPDDR4X memory. Nano has a quad-core ARM CPU, a 128-core Maxwell GPU, and 4GB LPDDR4 memory. Compared with AGX Xavier, Nano has less computing resources and suits sensing tasks with lower complexities. Both AGX Xavier and Nano run the Linux4Tegra operating system R32.2 with Tensorflow 1.14 and Keras 2.2.4.

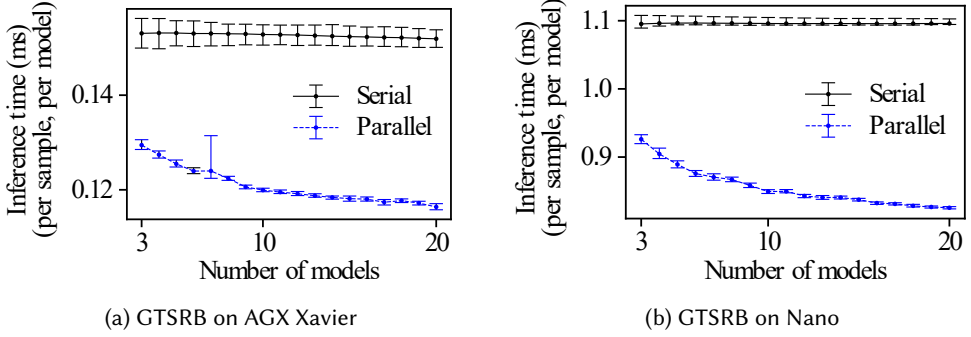


Fig. 19. Per-sample per-model inference times of parallel and serial DeepMTD versus the number of models. Error bar denotes average, 5th and 95th percentiles over 100 tests.

We conduct a set of profiling experiments to compare two possible execution modes of DeepMTD, i.e., *parallel* and *serial*. In most deep learning frameworks, the training and testing samples are fed to the deep model in batches. Our profiling also follows the same batch manner to feed the input samples to the fork models. Specifically, in the parallel mode, a batch of input samples are fed to all fork models simultaneously and all fork models are executed in parallel. This is achieved by the parallel models feature of Keras. In particular, we use the functional API of Keras to create a new model that contains multiple branches. Each branch contains a fork model. We feed the input samples to the new model so that all branches make inference and output predictions simultaneously. In the serial mode, a batch of input samples are fed to the fork models in serial, i.e., the next model is not executed until the completion of the previous one. Currently, we only consider using the high level APIs provided by the Python deep learning libraries to conduct the parallel and serial model inference. In the future, it would be interesting to explore other possible execution modes by investigating the hardware characteristics of GPU including the streaming multiprocessor (SM) number and shared memory size, and leveraging the low-level parallel computing APIs such as the CUDA provided by NVIDIA that gives direct access to the GPU's parallel computing elements.

We compare the inference times of parallel and serial DeepMTD on both AGX Xavier and Nano. On each platform, we vary the settings of the batch size and the number of models. Under each setting, we run DeepMTD in each mode for 100 times. Fig. 18 shows the per-sample inference time of DeepMTD with 20 fork models versus the batch size on the two platforms. We can see that the per-sample inference time decreases with the batch size but becomes flat when the batch size is large. This is because that for a larger batch, TensorFlow can process more samples concurrently. However, with too large batch size settings, the concurrency becomes saturated due to the exhaustion of GPU resources. The per-sample inference time of the serial DeepMTD is longer than that of the parallel DeepMTD. This is because that Keras will try to use all GPU resources to run as many as possible fork models concurrently. As the batch size determines the data acquisition time, it should be chosen to meet the real-time requirement on the sensing delay that is the sum of the data acquisition time and inference time. The sensing delay can be reduced by the early stopping technique in Section 6.2.

Fig. 19 shows the per-sample per-model inference time versus the number of fork models N . For serial DeepMTD, the per-sample per-model inference time is independent of N . This result is natural. Differently, for parallel DeepMTD, it decreases with N .

Algorithm 1 Serial fusion with early stopping

Given: set of fork models \mathcal{F} , input \mathbf{x}

- 1: randomly select 3 models from \mathcal{F} and use them to classify \mathbf{x}
 - 2: **loop**
 - 3: **if** more than $T_s \times 100\%$ of the existing classification results are the same **then**
 - 4: \mathbf{x} is detected clean and break the loop
 - 5: **else if** all models in \mathcal{F} have been selected **then**
 - 6: \mathbf{x} is detected adversarial and break the loop
 - 7: **end if**
 - 8: from \mathcal{F} randomly select a model that has not been selected before and use it to classify \mathbf{x}
 - 9: **end loop**
 - 10: **return** (1) attack detection result and (2) the majority of the existing classification results
-

6.2 Serial DeepMTD with Early Stopping

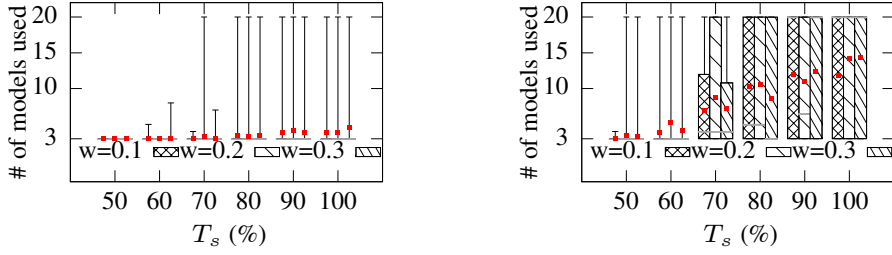
From the results in Section 6.1, due to the hardware resources constraint, the parallel DeepMTD does not bring much improvement in terms of inference time. In contrast, the serial DeepMTD admits early stopping when there is sufficient confidence about the fused result. This is inspired by the serial decision fusion technique [27]. Algorithm 1 shows the pseudocode of the serial fusion process with early stopping. Note that, in Line 1, a subset of three models is the minimum setting enabling the majority-based decision fusion. In Line 3, the T_s is a configurable attack detection threshold. We will assess its impact on the serial DeepMTD's performance shortly.

In our experiments, we set $N = 20$ and vary the serial detection threshold T_s from 0.5 to 1. Figs. 20a and 20b show the number of folk models used in serial DeepMTD when the input are 100 clean and 90 adversarial examples, respectively. For clean examples, when $T_s \leq 60\%$ and $T_s = 100\%$, three models are used in 99.7% and 93.6% of all the tests, respectively. When $T_s = 50\%$ and $T_s = 100\%$, 3 and 4.1 models are used on average, respectively. The corresponding average inference times are about 30% and 40% of that of parallel DeepMTD executing all 20 models. For adversarial examples, when $T_s \leq 60\%$, only three models are used in 88.7% of all the tests. When $T_s = 50\%$ and $T_s = 100\%$, 3.3 and 13.4 models are used on average, respectively. The corresponding inference times are about 32% and 130% of that of parallel DeepMTD executing all the 20 models. From the above results, as adversarial example attacks are rare events, the serial DeepMTD can reduce inference time effectively in the absence of attacks.

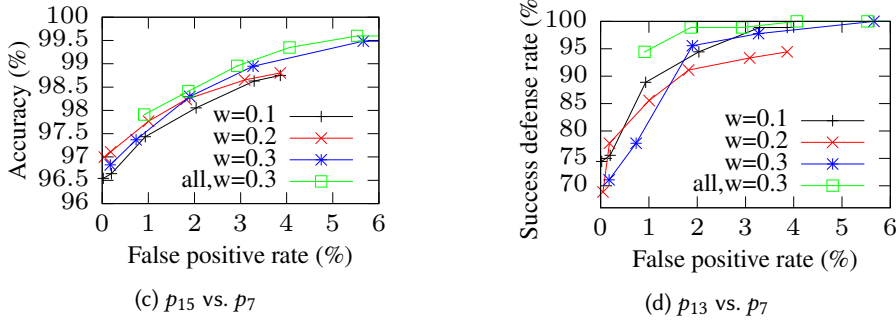
Then, we evaluate the impact of the early stopping on the sensing and defense performance. Fig. 20c shows the accuracy (p_{15}) versus the false positive rate (p_7). Different points on a curve are results under different T_s settings from 0.5 to 1. Compared with executing all fork models, the early stopping results in little accuracy drop (about 0.1%). Fig. 20d shows the successful defense rate (p_{13}) versus the false positive rate (p_7). Different points on a curve are results under different T_s settings from 0.5 to 1. With a false positive rate of 4%, the successful defense rate drops 2.2% only. The above results show that the early stopping can significantly reduce the run-time inference time, with little compromise of accuracy and defense performance.

7 DEEPMTD IMPLEMENTATION OPTIMIZATION

Section 6 aims at reducing the inference time by developing an algorithmic technique, i.e., early stopping. In this section, we aim at further reducing inference time and power consumption of DeepMTD by exploiting the characteristics of the hardware inference accelerators. Note that the



(a) The number of used models for clean examples (b) The number of used models for adversarial examples



(c) p_{15} vs. p_7

(d) p_{13} vs. p_7

Fig. 20. Performance of human-in-the-loop serial DeepMTD with early stopping. (Dataset: GTSRB; “all” means that early stopping is not enabled; gray line represents median; red square dot represents mean; box represents the (20%, 80%) range; upper/lower bar represents maximum/minimum.)

Table 5. Embedded compute boards used in this paper.

	NVIDIA Jetson AGX Xavier	NVIDIA Jetson Nano
Form	10.5 × 10.5 cm, 280 g	7 × 4.5 cm, 18 g
CPU	octa-core ARM 2.26GHz	quad-core Cortex-A57
GPU	512-core Volta	128-core Maxwell
RAM	16GB	4 GB
Power	10W / 30W	5W / 10W
OS	Linux4Tegra 32.3.1	Linux4Tegra 32.2.0
Software	PyTorch 1.4.0	PyTorch 1.3.0

power consumption is a key consideration for battery-based systems. Table 5 summarizes the specifications of Jetson AGX Xavier and Jetson Nano.

7.1 Power Consumption on Jetson Devices

We first profile the power consumption of DeepMTD on Jetson AGX Xavier and Nano. Both devices support a *low-power* mode and a *high-power* mode. In the low-power mode, the device lowers the hardware performance (e.g., lower CPU/GPU/memory operating frequencies) to save power; in the high-power mode, the device fully activates the hardware capabilities. The detailed hardware capabilities under the two modes can be found in [3]. We conduct a set of profiling experiments

to compare the power expenditures of serial DeepMTD under the two modes on both Nano and AGX Xavier. On each platform and power mode, we set $N = 20$ and vary the serial detection threshold T_s from 0.7 to 1. The input samples are a mixture of 10,000 clean and adversarial GTSRB examples. We let R_a denote the percentage of adversarial examples. The input samples are fed to the deep models in a batch size of 100. Fig. 21 shows the per-sample inference time of DeepMTD versus the average power consumption in different modes of Nano, under different settings of R_a . During the profiling experiments, we find that the power consumption is mainly affected by R_a and the power mode. The impact of T_s setting on the power consumption is little compared with the aforementioned two factors. This is because R_a largely affects how many samples will end up executing more models before early stopping and T_s affects how many models are executed for each sample. The latter factor has smaller impact on the power consumption. Thus, in each subfigure of Fig. 21, we group the bars for the same power mode but with different T_s settings into the same cluster. In Fig. 21, when $R_a = 0\%$, 50% , and 100% , the average per-sample inference times on Nano are 4.1ms, 8.5ms, and 13.2ms in low-power mode and 3.5ms, 7.3ms, and 10.9ms in high-power mode. Both the per-sample inference time and the average power consumption increase with R_a because more models are executed for adversarial examples before the early stopping. When we switch the Nano from the low-power mode to the high-power mode, the power consumption increases by around 35% and the per-sample inference time decreases by about 15%. Fig. 22 shows the results for AGX Xavier. When $R_a = 0\%$, 50% , and 100% , the average per-sample inference times are 1.6ms, 3.5ms, and 5.4ms in low-power mode and 0.5ms, 1.0ms, and 1.5ms in high-power mode. When the AGX Xavier switches from the low-power mode to the high-power mode, the average power consumption increases by 197% and per-sample inference time decreases by 71%. From the above results, compared with Nano, AGX Xavier has a better performance scaling at the expense of higher power scaling.

7.2 Improved Implementations of Serial DeepMTD

We investigate two improved implementations of serial DeepMTD that exploit heterogeneous inference accelerators and the configurable power modes, respectively.

7.2.1 Implementation on hybrid hardware. From Fig. 20a, we can see that for most clean examples, only three models are executed before the early stopping. Thus, the serial DeepMTD with early stopping can be divided into two stages. The Stage 1 executes three models to classify an input. If the models' outputs are consistent, the process yields clean input decision; otherwise, Stage 2 starts and executes more models sequentially until either the consistency of these models' outputs is larger than T_s (yield clean input decision) or all the models are executed (yield adversarial input decision). When the input is clean, most likely only Stage 1 is executed. When the input is adversarial, both stages will likely be executed. We implement the above design on a hybrid hardware architecture consisting of a Nano and an AGX Xavier to match the characteristics of the two-stage model execution. Specifically, given an input image, the three models in Stage 1 are executed in parallel on Nano to classify the input. The models of Stage 2 are executed in serial on AGX Xavier. During this process, a device switches to the high-power mode when it needs to execute models; otherwise, it remains in the low-power mode. The above design has the following advantages. First, since adversarial example attack is rare, in most of the time, the system receives clean inputs and executes Stage 1 only. Using Nano to execute three models in Stage 1 can meet real-time requirement and save energy. Second, when the input is adversarial, Stage 2 will be activated. Thus, the AGX Xavier can speed up the process. Maintaining real-time performance in the presence of adversarial attack is the first priority, while the increased power consumption is not a concern.

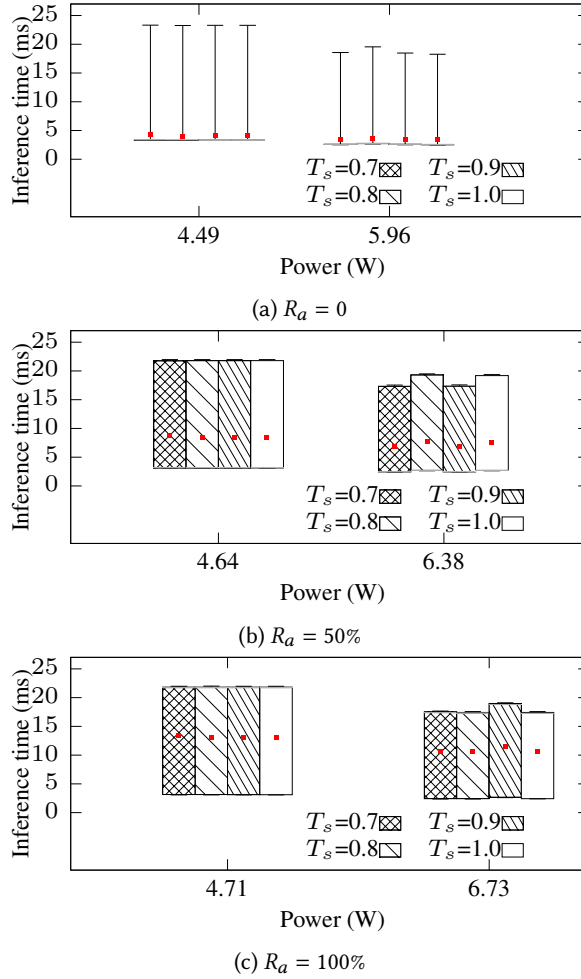


Fig. 21. Per-sample inference time versus average power consumption of serial DeepMTD on Nano. The first bar cluster is results in low-power mode and the second cluster is for high-power mode. (Dataset: GTSRB; gray line represents median; red square dot represents mean; box represents the (20%, 80%) range; upper/lower whisker represents maximum/minimum.)

The right most cluster of bars in each subfigure of Fig. 23 is the results of DeepMTD implementation on hybrid hardware. When $R_a = 0\%$, 50% , and 100% , the average per-sample inference times are 2.9ms, 3.3ms, and 3.8ms. The average power consumption is calculated as follows. Suppose the system takes t_1 and t_2 seconds to execute Stage 1 and 2, respectively. The powers consumed in watts are P_{h_1} during Stage 1 and P_{h_2} during Stage 2 for device in high-power mode and P_{l_1} during Stage 1 and P_{l_2} during Stage 2 for device in low-power mode. The average power consumption is $(P_{h_1} + P_{l_1}) \times (\frac{t_1}{t_1+t_2}) + (P_{h_2} + P_{l_2}) \times (\frac{t_2}{t_1+t_2})$. Compared with the implementation on Nano in high-power mode (c.f. Section 7.1), the implementation on hybrid hardware improves inference speed by 17%, 55%, and 65% when $R_a = 0\%$, 50% , and 100% , respectively. Compared with the implementation on AGX Xavier in high-power mode, implementation on hybrid hardware reduces power consumption

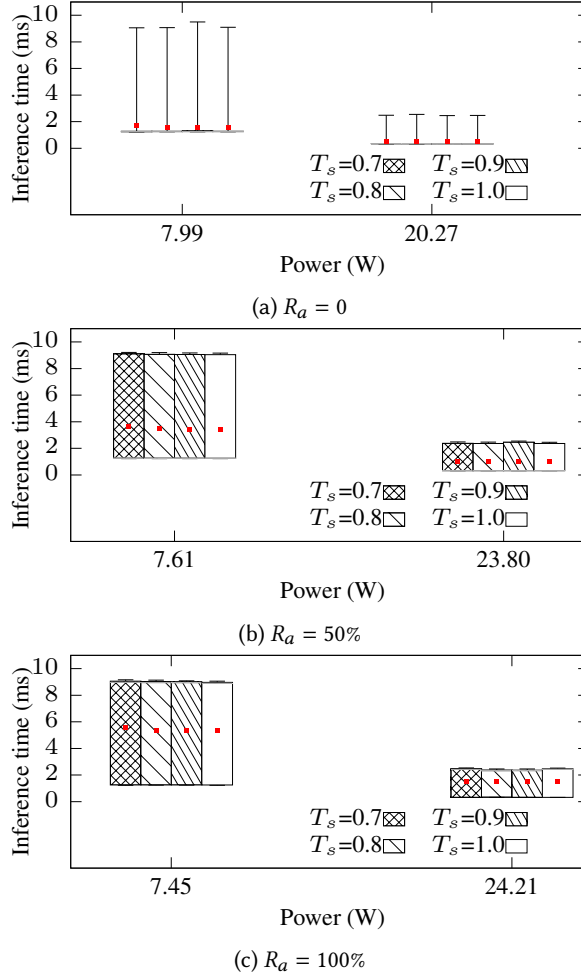


Fig. 22. Per-sample inference time versus average power consumption of serial DeepMTD on AGX Xavier. The first bar cluster is results in low-power mode and the second cluster is for high-power mode.

by 53%, 50%, and 42% when $R_a = 0\%$, 50%, and 100%, respectively. However, the hybrid implementation also consumes more power than the Nano-only implementation and is slower than the AGX Xavier-only implementation. Therefore, the hybrid-hardware implementation is a solution that operates on a new trade-off point of power consumption versus inference speed.

In this hybrid-hardware implementation, the Jetson devices consume considerable energy when they do not execute models and remain in the low-power mode. Specifically, when $R_a = 0$, 50%, and 100%, such *idle energy* accounts for 38%, 27%, and 21% of the total energy consumption. This inefficiency is caused by that the Linux-based Jetson devices cannot switch fast between sleep mode and work mode, and therefore they have to stand by in the low-power mode.

7.2.2 Implementation with dynamic power mode. We also investigate an alternative implementation of DeepMTD that exploits the configurable power modes of Jetson devices. This implementation only uses a single Jetson device, either Nano or AGX Xavier. In this implementation, a device

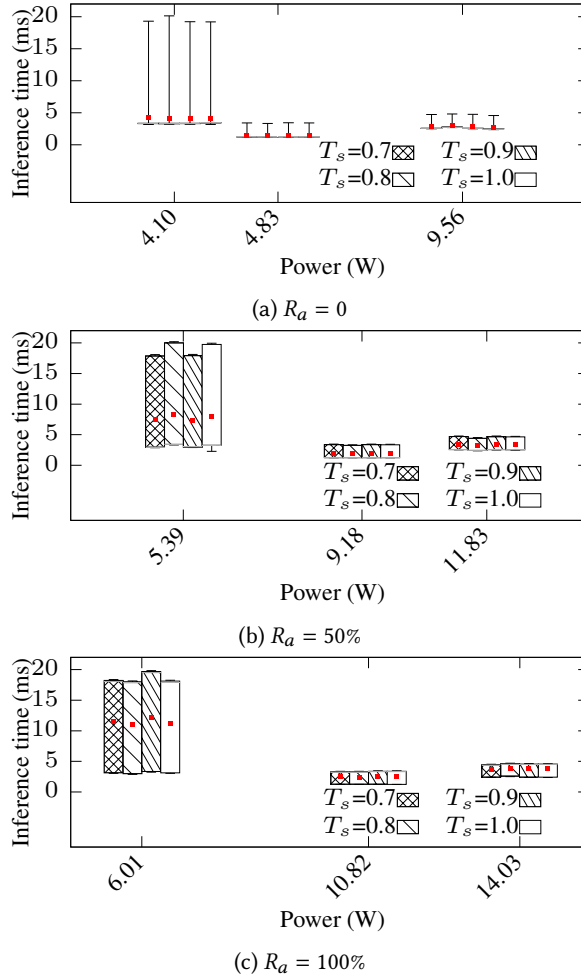


Fig. 23. Per-sample inference times versus average power consumption of implementations using configurable power mode on Nano (the left most cluster) and AGX Xavier (the middle cluster), as well as hybrid hardware implementation (the right most cluster).

executes the Stage 1 in low-power mode. Once Stage 2 is activated, the device will switch to the high-power mode. We evaluate the performance of this implementation on both Nano and AGX Xavier. Results can be seen in Fig. 23. The left most and the middle clusters of bars are the results obtained on Nano and AGX Xavier, respectively. When $R_a = 0$, 50%, and 100%, the average per-sample inference times on Nano are 4.1ms, 7.8ms, and 11.5ms. On AGX Xavier, the average per-sample inference times are 1.4ms, 1.9ms, and 2.4ms. By comparing the results with those of the hybrid hardware implementation, when the input is clean (i.e., $R_a = 0$), which is the usual case, the implementation using dynamic power mode on AGX Xavier consumes 49% less power. When $R_a = 50\%$ and 100%, the power savings are 22% and 23%, respectively. Moreover, this implementation reduces inference latency by 52%, 42%, and 37% when $R_a = 0$, 50% and 100%. Thus, the dynamic power mode implementation on AGX Xavier outperforms the hybrid hardware implementation in terms of both inference time and power consumption. We believe that if the hardware inference

accelerator provides more programmable configurations of power modes, we can better control the execution of the deep models to strike more desirable trade-off between inference latency and power consumption. The dynamic power mode implementation on Nano has long inference times because of Nano's limited processing power for Stage 2. In summary, the dynamic power mode implementation on AGX Xavier achieves the most desirable trade-off between inference speed and power consumption.

8 CONCLUSION

This paper presented DeepMTD for deep learning-based image classification on embedded platforms against adversarial example attacks. We evaluated its performance in the absence and presence of attacks. Based on the profiling results of DeepMTD on two NVIDIA Jetson platforms, we proposed serial DeepMTD with early stopping to reduce the inference time. We also exploited the characteristics of the Jetson devices to improve the DeepMTD implementation. Our results provide useful guidelines for integrating DeepMTD to the current embedded deep visual sensing systems to improve their security.

REFERENCES

- [1] [n.d.]. Experimental Security Research of Tesla Autopilot. https://keenlab.tencent.com/en/whitepapers/Experimental_Security_Research_of_Tesla_Autopilot.pdf.
- [2] [n.d.]. IWG: Cybersecurity game-change research and development recommendations. <https://www.hsdsl.org/?view&did=6990>.
- [3] [n.d.]. NVIDIA Jetson Linux Driver Package Developer Guide. <https://docs.nvidia.com/jetson/l4t/index.html>.
- [4] Anish Athalye, Nicholas Carlini, and David Wagner. 2018. Obfuscated Gradients Give a False Sense of Security: Circumventing Defenses to Adversarial Examples. In *Proceedings of the International Conference on Machine Learning*.
- [5] Anish Athalye, Logan Engstrom, Andrew Ilyas, and Kevin Kwok. 2018. Synthesizing robust adversarial examples. In *Proceedings of the International Conference on Machine Learning*.
- [6] Nicholas Carlini and David Wagner. 2017. Towards evaluating the robustness of neural networks. In *Proceedings of the IEEE Symposium on Security and Privacy (SP)*.
- [7] Nilaksh Das, Madhuri Shanbhogue, Shang-Tse Chen, Fred Hohman, Siwei Li, Li Chen, Michael E Kounavis, and Duen Horng Chau. 2018. Shield: Fast, practical defense and vaccination for deep learning using jpeg compression. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*.
- [8] Thomas G Dietterich. 2000. Ensemble methods in machine learning. In *Proceedings of International workshop on multiple classifier systems*.
- [9] Kevin Eykholt, Ivan Evtimov, Earlene Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. 2018. Robust physical-world attacks on deep learning visual classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 1625–1634.
- [10] Biyi Fang, Jillian Co, and Mi Zhang. 2017. DeepASL: Enabling ubiquitous and non-intrusive word and sentence-level sign language translation. In *Proceedings of the ACM Conference on Embedded Networked Sensor Systems (SenSys)*. 1–13.
- [11] Biyi Fang, Xiao Zeng, Faen Zhang, Hui Xu, and Mi Zhang. 2020. FlexDNN: Input-Adaptive On-Device Deep Learning for Efficient Mobile Vision. In *Proceedings of the ACM/IEEE Symposium on Edge Computing (SEC)*.
- [12] Biyi Fang, Xiao Zeng, and Mi Zhang. 2018. Nestdnn: Resource-aware multi-tenant on-device deep learning for continuous mobile vision. In *Proceedings of the International Conference on Mobile Computing and Networking (MobiCom)*. 115–127.
- [13] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and Harnessing Adversarial Examples. In *Proceedings of the International Conference on Learning Representations*.
- [14] Chuan Guo, Mayank Rana, Moustapha Cisse, and Laurens van der Maaten. 2018. Countering Adversarial Images using Input Transformations. In *Proceedings of the International Conference on Learning Representations*.
- [15] Warren He, James Wei, Xinyun Chen, Nicholas Carlini, and Dawn Song. 2017. Adversarial example defense: Ensembles of weak defenses are not strong. In *11th {USENIX} Workshop on Offensive Technologies ({WOOT} 17)*.
- [16] Sushil Jajodia, Anup K Ghosh, Vipin Swarup, Cliff Wang, and X Sean Wang. 2011. *Moving target defense: creating asymmetric uncertainty for cyber threats*. Springer Science & Business Media.
- [17] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. 2014. The CIFAR 10 dataset. <http://www.cs.toronto.edu/kriz/cifar.html>.

- [18] Nicholas D Lane, Petko Georgiev, and Lorena Qendro. 2015. DeepEar: Robust smartphone audio sensing in unconstrained acoustic environments using deep learning. In *Proceedings of the ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp)*. 283–294.
- [19] Stefanos Laskaridis, Stylianos I Venieris, Mario Almeida, Ilias Leontiadis, and Nicholas D Lane. 2020. SPINN: synergistic progressive inference of neural networks over device and cloud. In *Proceedings of the International Conference on Mobile Computing and Networking (MobiCom)*. 1–15.
- [20] Stefanos Laskaridis, Stylianos I Venieris, Hyeji Kim, and Nicholas D Lane. 2020. HAPI: Hardware-Aware Progressive Inference. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*.
- [21] Yann LeCun, Corinna Cortes, and Christopher JC Burges. 1998. The MNIST database of handwritten digits. <http://yann.lecun.com/exdb/mnist>.
- [22] Xinyu Li, Yanyi Zhang, Ivan Marsic, Aleksandra Sarcevic, and Randall S Burd. 2016. Deep learning for RFID-based activity recognition. In *Proceedings of the ACM Conference on Embedded Networked Sensor Systems (SenSys)*. 164–175.
- [23] Yanpei Liu, Xinyun Chen, Chang Liu, and Dawn Song. 2017. Delving into transferable adversarial examples and black-box attacks. In *Proceedings of the International Conference on Learning Representations*.
- [24] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2018. Towards deep learning models resistant to adversarial attacks. In *Proceedings of the International Conference on Learning Representations*.
- [25] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. 2017. Universal adversarial perturbations. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1765–1773.
- [26] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. 2016. Distillation as a defense to adversarial perturbations against deep neural networks. In *Proceedings of IEEE Symposium on Security and Privacy (SP)*.
- [27] Swapnil Patil, Samir R Das, and Asis Nasipuri. 2004. Serial data fusion using space-filling curves in wireless sensor networks. In *Proceedings of the IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks (SECON)*.
- [28] Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. 2018. Certified defenses against adversarial examples. In *Proceedings of the International Conference on Learning Representations*.
- [29] Kui Ren, Tianhang Zheng, Zhan Qin, and Xue Liu. 2020. Adversarial attacks and defenses in deep learning. *Engineering* (2020).
- [30] Pouya Samangouei, Maya Kabkab, and Rama Chellappa. 2018. Defense-GAN: Protecting classifiers against adversarial attacks using generative models. In *Proceedings of the International Conference on Learning Representations*.
- [31] Sailik Sengupta, Tathagata Chakraborti, and Subbarao Kambhampati. 2018. MTDeep: boosting the security of deep neural nets against adversarial attacks with moving target defense. In *Proceedings of the Workshops at the AAAI Conference on Artificial Intelligence*.
- [32] Claude E Shannon. 1949. Communication theory of secrecy systems. *Bell system technical journal* (1949).
- [33] Qun Song, Zhenyu Yan, and Rui Tan. 2019. Moving Target Defense for Deep Visual Sensing against Adversarial Examples. *arXiv preprint arXiv:1905.13148* (2019).
- [34] Qun Song, Zhenyu Yan, and Rui Tan. 2019. Moving target defense for embedded deep visual sensing against adversarial examples. In *Proceedings of the ACM Conference on Embedded Networked Sensor Systems (SenSys)*. 124–137.
- [35] Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. 2012. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural networks* (2012).
- [36] Dimitrios Stamoulis, Ruizhou Ding, Di Wang, Dimitrios Lymberopoulos, Bodhi Priyantha, Jie Liu, and Diana Marculescu. 2019. Single-path nas: Designing hardware-efficient convnets in less than 4 hours. In *Proceedings of the Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 481–497.
- [37] Dimitrios Stamoulis, Ruizhou Ding, Di Wang, Dimitrios Lymberopoulos, Nissanka Bodhi Priyantha, Jie Liu, and Diana Marculescu. 2020. Single-path mobile automl: Efficient convnet design and nas hyperparameter optimization. *IEEE Journal of Selected Topics in Signal Processing* (2020).
- [38] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2014. Intriguing properties of neural networks. In *Proceedings of the International Conference on Learning Representations*.
- [39] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick Drew McDaniel. 2018. Ensemble adversarial training: Attacks and defenses. In *Proceedings of the International Conference on Learning Representations*.
- [40] Eric Wong and J Zico Kolter. 2018. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *Proceedings of the International Conference on Machine Learning*.
- [41] Shuochao Yao, Jinyang Li, Dongxin Liu, Tianshi Wang, Shengzhong Liu, Huajie Shao, and Tarek Abdelzaher. 2020. Deep compressive offloading: speeding up neural network inference by trading edge computation for network latency. In *Proceedings of the ACM Conference on Embedded Networked Sensor Systems (SenSys)*. 476–488.

- [42] Shuochao Yao, Yiran Zhao, Huajie Shao, Dongxin Liu, Shengzhong Liu, Yifan Hao, Ailing Piao, Shaohan Hu, Su Lu, and Tarek F Abdelzaher. 2019. SADeepSense: Self-Attention Deep Learning Framework for Heterogeneous On-Device Sensors in Internet of Things Applications. In *Proceedings of the IEEE International Conference on Computer Communications (INFOCOM)*. 1243–1251.
- [43] Shuochao Yao, Yiran Zhao, Huajie Shao, Shengzhong Liu, Dongxin Liu, Lu Su, and Tarek Abdelzaher. 2018. Fastdeepiot: Towards understanding and optimizing neural network execution time on mobile and embedded devices. In *Proceedings of the ACM Conference on Embedded Networked Sensor Systems (SenSys)*. 278–291.
- [44] Shuochao Yao, Yiran Zhao, Huajie Shao, Chao Zhang, Aston Zhang, Shaohan Hu, Dongxin Liu, Shengzhong Liu, Lu Su, and Tarek Abdelzaher. 2018. Sensegan: Enabling deep learning for internet of things with a semi-supervised framework. *Proceedings of the ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp)* 2, 3 (2018), 1–21.
- [45] Xiao Zeng, Biyi Fang, Haichen Shen, and Mi Zhang. 2020. Distream: scaling live video analytics with workload-adaptive distributed edge intelligence. In *Proceedings of the ACM Conference on Embedded Networked Sensor Systems (SenSys)*. 409–421.
- [46] Yu Zhang, Tao Gu, and Xi Zhang. 2020. MDLdroid: a ChainSGD-reduce Approach to Mobile Deep Learning for Personal Mobile Sensing. In *Proceedings of the ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*. 73–84.
- [47] Yu Zhang, Tao Gu, and Xi Zhang. 2020. MDLdroidLite: a release-and-inhibit control approach to resource-efficient deep neural networks on mobile devices. In *Proceedings of the ACM Conference on Embedded Networked Sensor Systems (SenSys)*. 463–475.