CuRTAIL: ChaRacterizing and Thwarting AdversarIal deep Learning

Bita Darvish Rouhani, Mohammad Samragh, Tara Javidi, and Farinaz Koushanfar
University of California San Diego
bita@ucsd.edu, msamragh@ucsd.edu, tjavidi@ucsd.edu, farinaz@ucsd.edu

***Abstract*—This paper proposes CuRTAIL, an automated end-to-end computing framework for characterizing and thwarting adversarial space in the context of Deep Learning (DL). CuRTAIL protects deep neural networks against adversarial samples, which are perturbed inputs carefully crafted by malicious entities to mislead the victim DL model. The precursor for the proposed methodology is a set of new quantitative metrics to assess the *vulnerability* of various DL architectures to adversarial samples. CuRTAIL formalizes the goal of preventing adversarial samples as a minimization of the space unexplored by the pertinent DL model that is characterized in CuRTAIL vulnerability analysis step. To thwart the adversarial machine learning attack, we introduce the concept of *Modular Robust Redundancy (MRR)* as a viable solution to achieve the formalized minimization objective. The proposed MRR methodology explicitly characterizes the geometry of the input data and its corresponding high-level data abstractions within the victim DL network. It then learns a set of complementary but disjoint models which maximally cover the unexplored space in the victim model, thus reducing the risk of integrity attacks. We extensively evaluate CuRTAIL performance against various attack models including Fast-Gradient-Sign, Jacobian Saliency Map Attack, Deepfool, and Carlini&WagnerL2. Proof-of-concept implementations for analyzing various data collections including MNIST, CIFAR10, and ImageNet corroborate CuRTAIL effectiveness to detect adversarial samples in different settings. The computations in each MRR module can be performed independently of the other redundancy modules. As such, CuRTAIL detection algorithm can be completely parallelized among multiple hardware settings to achieve maximum throughput. We further provide an open-source Application Programming Interface (API) to facilitate the adoption of the proposed framework for various applications.**

I. Introduction

Security and safety consideration is the biggest obstacle for the wide-scale adoption of emerging learning algorithms in sensitive scenarios such as intelligent transportation, healthcare, and video surveillance applications [1], [2], [3]. While advanced learning technologies are essential for enabling coordination and interaction among autonomous agents and the environment, a careful analysis of their security as well as thwarting their vulnerabilities is still in its infancy. Machine learning models including the state-of-the-art deep neural networks are widely used in various scientific fields ranging from speech recognition [4], [5] and computer vision [6], [7] to financial fraud [8] and malware detection [9], [10]. These applications have been mainly developed with an implicit security assumption about the generalizability of such models for evaluation of unseen examples. Recent results on adversarial machine learning, however, have shed light on a new and largely unexplored surface for malicious attacks jeopardizing the reliability of machine learning models [11], [12], [13].

As shown in several recent studies, a malicious adversary can carefully manipulate the input data by leveraging specific vulnerabilities of learning techniques to undermine the integrity of a certain system [1]. Misclassifying a certain source class into a distinct target class is one of the strongest adversarial goals for attackers targeting classifiers as outlined in [13], [14], [15]. The problem of adversarial samples arises due to the fact that machine learning algorithms are designed for stationary environments where the training and test data are collected from the same unknown distributions. This working hypothesis, however, can be easily violated by adversaries to mislead the learning system.

A vast majority of recent research efforts have focused on devising new attack methodologies in particular for the popular class of deep learning algorithms with limited attention to possible countermeasures [11], [12], [13]. The existing research works which have considered viable countermeasures for adversarial deep learning can be categorized into three classes: (i) Denoising encoders have been suggested in the literature as a pre-processing step to improve the robustness of DL networks by mapping the adversarial samples into the original input space. As shown in [16], however, the resulting two-stage DL network is no more difficult to attack than the original one. (ii) Several recent papers have focused on training the DL model by including adversarial examples in the training set [17], [12], [18], [19]. Although this technique has yielded significant improvement in the convergence of the underlying model and robustness to particular noise patterns in the input space, it can only partially evade adversarial samples from being effective as shown in [20]. (iii) Distillation is another countermeasure recently proposed in [21]. This technique aims to improve the robustness of DL networks by transforming the original model into a second model that lies in a smoother gradient space compared to the original one. However, recent follow-up papers (e.g., [22]) have demonstrated that the distilled network is as vulnerable to adversarial attacks as the original model.

This paper proposes CuRTAIL, a holistic framework for characterizing and thwarting adversarial deep learning space. CuRTAIL formalizes the goal of preventing adversarial samples as an optimization problem to minimize the unexplored space in the target DL model. To fulfill this objective, we introduce a new defense mechanism called Modular Robust Redundancy (MRR). MRR approach is motivated by our key

observation that no single model can completely eliminate the potential adversarial space by relying on a limited set of labeled data. As such, instead of using a single module for detection of adversarial samples, CuRTAIL employs multiple minimally overlapping *redundant modules (defender models)* that investigate each input sample in parallel to the main DL model and raise alarm flags for those samples that are suspected to be infected. Compared to the existing DL defense mechanisms (e.g., [16], [21], [12]), our proposed approach is beneficial due to three main reasons: (i) Existing works either restrict the accuracy of the main DL model or particularly rely on altering the current DL topologies and/or training process to make the model more robust against adversarial samples. In CuRTAIL neither the training complexity nor the final accuracy of the main DL model is affected. (ii) In order to adjust the level of security in most of the existing works, the victim DL model should be retrained with alternative hyper-parameters, whereas in CuRTAIL the hardness of the detection policy can be easily adjusted once a defender model has been trained. (iii) The use of parallel redundancy modules in CuRTAIL framework significantly reduces the risk of integrity attacks as the attacker requires to simultaneously deceive all the defender models to succeed.

For a given DL model (victim network), CuRTAIL first evaluates the relative vulnerability of each layer based on the spectral analysis of the DL parameters. It then learns a set of disjoint redundancy modules to maximally cover the vulnerable space in the target application while constraining the number of redundancy modules. The redundancy modules in CuRTAIL framework are built upon the commonly-used dictionary learning method [23]. Each dictionary targets a certain layer in the neural network. For each layer, the corresponding dictionary explicitly characterizes the statistical properties of input data and DL parameters by learning the pertinent probability density function (PDF) of the latent variables. Finally, the learned dictionaries are utilized to analyze the features in the input and intermediate layers of the DL model and identify potential adversarial samples.

The security level in CuRTAIL is quantified by a high-level parameter that can be adjusted to account for various application-specific requirements. We empirically and analytically investigate the security of CuRTAIL framework as a viable countermeasure for adversarial deep learning. We consider a white-box attack model in which the attacker knows everything about the victim (DL network) including its model topology, learning algorithm, and parameters. This threat model represents the most powerful attacker that can endanger the real-world machine learning applications. An accompanying CuRTAIL API is provided to ensure its ease of use for deployment of DL applications including several computer vision tasks, malware detection, and biometric recognition. CuRTAIL API is devised with an automated customization unit to adjust the number of modular redundancies in accordance with a set of user-defined constraints such as real-time data analysis prerequisites and pertinent security requirements. Note that the redundancy modules can be parallelized on different hardware to achieve maximum throughput.

The explicit contributions of this paper are as follows:

- Proposing CuRTAIL, a novel end-to-end framework for characterizing and thwarting adversarial space in the context of deep learning. We incept the idea of Modular Robust Redundancy as a viable security countermeasure for adversarial machine learning. For a fixed number of redundancy modules, CuRTAIL carefully learns a set of complementary dictionaries to maximally cover the unexplored space in the victim DL model and effectively reduce the risk of integrity attacks.
- Providing quantitative measurements to characterize the sensitivity of DL model layers from the statistical point of view. This is crucial since the number of MRR modules can be limited in several real-world settings. We utilize the formalized sensitivity measurement to optimally identify DL layers for which MRR modules should be established.
- Performing extensive evaluations on well-known DL applications including MNIST [24], CIFAR10 [25], and ImageNet [26] benchmarks. The results demonstrate the algorithmic practicality and system performance efficiency of CuRTAIL framework against various attack models including fast-sign-gradient [12], Jacobian Saliency Map attack [13], Deepfool [27], and Carlini&WagnerL2 [28].
- Implementing an automated accompanying API to facilitate adoption/integration of the proposed framework for the reliable realization of different DL applications. CuRTAIL incorporates high-level security parameters into the proposed defense mechanism, allowing users to effectively adjust the robustness of the countermeasure without requiring them to get involved in the details of the design.

II. Background and Preliminaries

A machine learning model refers to a function f and its associated parameters θ that are particularly trained to infer/discover the relationship between input samples $x \in \{x_1, x_2, ..., x_N\}$ and the expected labels $y \in \{y_1, y_2, ..., y_N\}$. Each output observation y_i can be either continuous as in most regression tasks or discrete as in classification applications. Machine learning algorithms typically aim to find the optimal parameter set θ such that a loss function $\mathcal{L}$ that captures the difference between the output inference and ground-truth labeled data is minimized:

$$\theta = \underset{\theta}{argmin} \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(f(x_i, \theta), y_i). \quad (1)$$

In this paper, we focus our evaluations on the state-of-the-art deep learning models due to their popularity in the realization of various autonomous learning systems. Consistent with the literature in this field, we particularly centralize our discussions on the classification tasks using DL methodology. However, we emphasize that the core concept proposed in this paper is rather more generic and can be used for reliable deployment of different learning techniques such as generalized linear models, regression methods (e.g., regularized regression (Lasso)), and kernel support vector machines.

A. *Deep Learning*

Deep learning is an important class of machine learning algorithms that has provided a paradigm shift in our ability to comprehend raw data by showing superb inference accuracy resembling the learning capability of human brain [2], [3]. A DL network is a hierarchical learning topology consisting of several processing layers stacked on top of one another. The schematic depiction of a typical DL network consisting of convolutional, pooling, fully-connected, and various non-linearity layers is demonstrated in Figure 1. This type of neural networks is widely adopted in computer vision and image processing tasks [6]. Table I summarizes common layers used in DL neural networks. The state of each neuron (unit) in a DL network is determined in response to the states of the units in the prior layer after applying a nonlinear activation function. In Table I, $x_i^{(l)}$ is the state of unit i in layer l, $z_i^{(l)}$ is the post-nonlinearity value associated with unit i in layer l, $\theta_{ij}^{(l)}$ specifies the parameter connecting unit j in layer l and unit i in the layer $l+1$, and k indicates the kernel size used in 2-dimensional (2D) layers.

TABLE I: Commonly used layers in DL neural networks.

DL Layers		Description
Core Computations	*Fully-Connected*	$x_i^{(l)} = \sum_{j=1}^{N_{l-1}} \theta_{ij}^{(l-1)} \times z_j^{(l-1)}$
	2D Convolution	$x_{ij}^{(l)} = \sum_{s_1=1}^{k} \sum_{s_2=1}^{k} \theta_{s_1 s_2}^{(l-1)} \times z_{(i+s_1)(j+s_2)}^{l-1}$
Normalization	*L_2 Normalization*	$x_i^{(l)} = \frac{x_i^{(l)}}{\sqrt{\sum_{j=1}^{N_l} \lvert x_j^{(l)} \rvert^2}}$
	Batch Normalization	$x_i^{(l)} = \frac{x_i^{(l)} - \mu_B^{(l)}}{\sqrt{\frac{1}{b_s} \sum_{j=1}^{b_s} (x_j^{(l)} - \mu_B^{(l)})^2}}$
Pooling	*2D Max Pooling*	$x_{ij}^{(l)} = Max(y_{(i+s_1)(j+s_2)}^{l-1}) s_1 \in \{1,2,...,k\}$ $s_2 \in \{1,2,...,k\}$
	2D Mean Pooling	$x_{ij}^{(l)} = Mean(z_{(i+s_1)(j+s_2)}^{l-1}) s_1 \in \{1,2,...,k\}$ $s_2 \in \{1,2,...,k\}$
Non-linearities	*Softmax*	$z_i^{(l)} = \frac{e^{x_i^{(l)}}}{\sum_{j=1}^{N_l} e^{x_j^{(l)}}}$
	Sigmoid	$z_i^{(l)} = \frac{1}{1+e^{-x_i^{(l)}}}$
	Tangent Hyperbolic	$z_i^{(l)} = \frac{Sinh(x_i^{(l)})}{Cosh(x_i^{(l)})}$
	Rectified Linear unit	$z_i^{(l)} = Max(0, x_i^{(l)})$

Training a DL network involves two main steps: (i) forward propagation, and (ii) backward propagation. These steps are iteratively performed for multiple rounds using different batches of known input/output pairs (x_i, y_i) to reach a certain level of accuracy. In forward propagation, the raw values of the input data measurements are gradually mapped to higher-level abstractions based on the current state of the DL parameters (θ). The acquired data abstractions are used to predict the inference label in the last layer of the DL network based on a Softmax regression.[1] In backward propagation, an optimization algorithm such as stochastic gradient descent [29] is performed to find the gradient direction along which the DL parameter set θ should be updated to minimize the distance between network prediction (output of forward propagation) and the ground-truth label.

[1]Softmax regression (or multinomial logistic regression) is a generalization of logistic regression that maps a P-dimensional vector of arbitrary real values to a P-dimensional vector of real values in the range of $[0, 1)$. The final inference for each input sample can be determined by the output unit that has the largest conditional probability value.

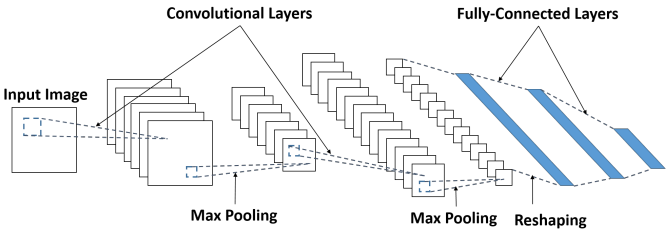


Fig. 1: Schematic depiction of a typical neural network used for computer vision and image processing tasks.

Once the DL network is trained to deliver a desired level of accuracy, the model is employed as a classification oracle in the execution phase (a.k.a., test phase). During the execution phase, the model parameters θ are fixed and prediction is performed through one round of forward propagation for each unknown input sample. Attacks based on adversarial samples target the execution phase of DL networks and do not involve any tampering with the training procedure as will be discussed in Section III.

III. Attack Models

Adversarial machine learning can be cast as a zero-sum Stackelberg game between the machine learning oracle (victim) and the attacker. Depending on the attacker's knowledge, the threat model can be categorized into three classes:

- **White-box attack.** The attacker knows everything about the defender model including the learning algorithm, model topology, and parameters, but has limited or partial access to the training data [12], [12], [13].
- **Gray-box attack.** The attacker only knows the underlying learning algorithm and model topology but has no access to the training data or the trained parameters.
- **Black-box attack.** The attacker knows nothing about the pertinent machine learning algorithm, model, or training data. This attacker only can obtain the corresponding inference label for input samples. In this setting, the adversary can perform a differential attack by observing the output changes with respect to the input variations [30].

A complete taxonomy of adversarial capabilities and goals are provided in [13], [14], [15]. In this paper, we consider the white-box threat model as the most powerful attacker that can appear in real-world machine learning applications.

In an adversarial setting, the attacker aims to find a perturbed adversarial sample (x^a) such that it incurs minimal distance from the source sample (x^s) while its corresponding output is sufficiently different to mislead the victim. Figure 2 illustrates an example, where the image on the left is initially classified correctly as a dog by the victim model while adding a small amount of perturbation to the original image has misled the victim to infer it as a black swan (right image). Clearly, if the source instance is already misclassified by the victim model ($f(x^s, \theta) \neq y^*$), the adversarial problem

becomes trivial. Therefore, we particularly focus on instances x^s that could have been classified correctly by the oracle before adding structured adversarial noises ($f(x^s, \theta) = y^*$).

We define the distance between the machine learning oracle output and the ground-truth label as follows:

$$\mathcal{D}(f(x^a), y^*) = \gamma_0 \mathcal{L}(f(x^a), y^*) - \Sigma_{f(x^a) \neq y^*} \gamma_i \mathcal{L}(f(x^a), y_i), \quad (2)$$

where $\mathcal{L}$ is the loss function used to train the pertinent DL model, $\gamma_0 \geq 0$ is the weight for the ground-truth and $\gamma_i \geq 0$ are the weights for each misleading target. Hence, the adversarial objective is to find the solution to the following:

$$\underset{x^a}{argmax}\ \mathcal{D}(f(x^a), y^*) \quad s.t. \quad \|(x^a - x^s)\|_\infty \leq \epsilon. \quad (3)$$

To solve for the optimal adversarial sample (x^a), the adversary should compute the sensitivity of each DL output with respect to the changes in the input features. The network sensitivity can be computed in several ways. Figure 3 depicts the schematic depiction of adversarial attacks. Below we briefly describe state-of-the-art attack mechanisms against which we evaluate CuRTAIL. Details about each attack algorithm can be found in the corresponding paper.

(a)

(b)

Fig. 2: An example of (a) source input data, and (b) its corresponding adversarial sample. The added noise is visually hard to see but makes the victim misclassify (b).

Fast Gradient Sign (FGS). Authors in [12] suggested the *fast-sign-gradient* method that leverages the sign of the loss function's gradient with respect to each input feature to craft the adversarial samples. In this case, the adversarial sample per input feature (x_i^s) is computed as:

$$x_i^a = x_i^s + \epsilon sign(\frac{\partial \mathcal{L}}{\partial x_i^s}), \quad (4)$$

which guarantees that the overall perturbation ($\|(x_i^a - x_i^s)\|_\infty$) does not exceed the threshold ϵ. The computed gradient in fast-sign-gradient is similar to the cost evaluated in back-propagation step in the training phase with a key difference that the derivative is computed with respect to the input features (x_i^s). Parameter ϵ determines the closeness of the crafted adversarial sample to the legitimate input. Higher ϵ values result in higher attack success rate with the cost of more distinguishable additive noise.

Jacobian Saliency Map Attack (JSMA). A different attack methodology was introduced in [13], which suggested computing the sensitivity map per input feature and iteratively perturbing each input feature (pixel in the case of images) based on the sensitivity map. The attack parameters include: (i) the maximum percentage of the input features that can be perturbed, and (ii) the amount of allowed perturbation for each feature. Please refer to [13] for details about the attack algorithm.

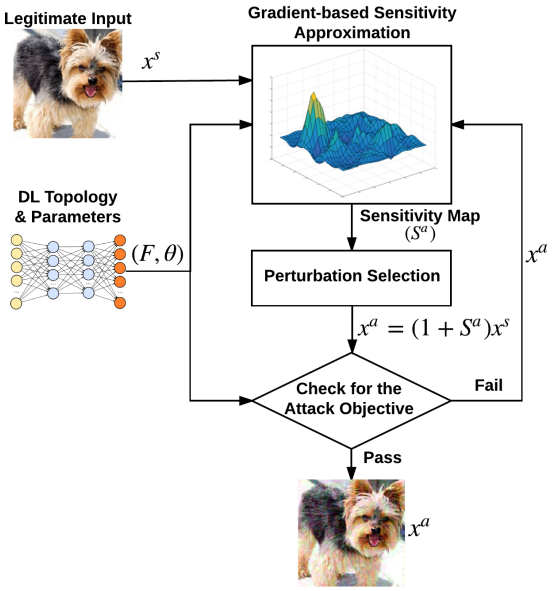


Fig. 3: Overall flow of typical adversarial attack methodologies.

Deepfool. Authors in [27] have introduced another attack heuristic called Deepfool that carefully adds perturbations to minimize the L_2 distance of the adversarial sample and the original data. Deepfool iteratively adds perturbations to all input features based on a certain update rule. The attack parameters include the number of iterative updates. Details about the iterative algorithm can be found in [27].

Carlini&WagnerL2. Another attack [28] is proposed to minimize the L_2 norm of the perturbation. In this attack, authors suggest the use of modified cost function with hyper parameters that allow trade-offs between attack success rate, the confidence of the attacked model on the adversarial samples, and the amount of additive perturbation. During the process of crafting adversarial samples, the hyper-parameters are tuned by binary search to improve the quality of adversarial examples. We refer the reader to the original paper [28] for details about this attack.

IV. CuRTAIL Global Flow

Figure 4 illustrates the global flow of CuRTAIL framework. CuRTAIL consists of two main phases to analyze the vulnerability of a victim DL model and detect the adversarial samples that might be fed into the underlying network.

(i) Pre-Processing Phase. The pre-processing phase is performed in three successive steps.

1 Sensitivity Analysis. CuRTAIL framework takes a trained DL neural network as its input and primarily performs a thorough sensitivity analysis based on a layer-wise spectrum density evaluation of pertinent model parameters (Section V-A).

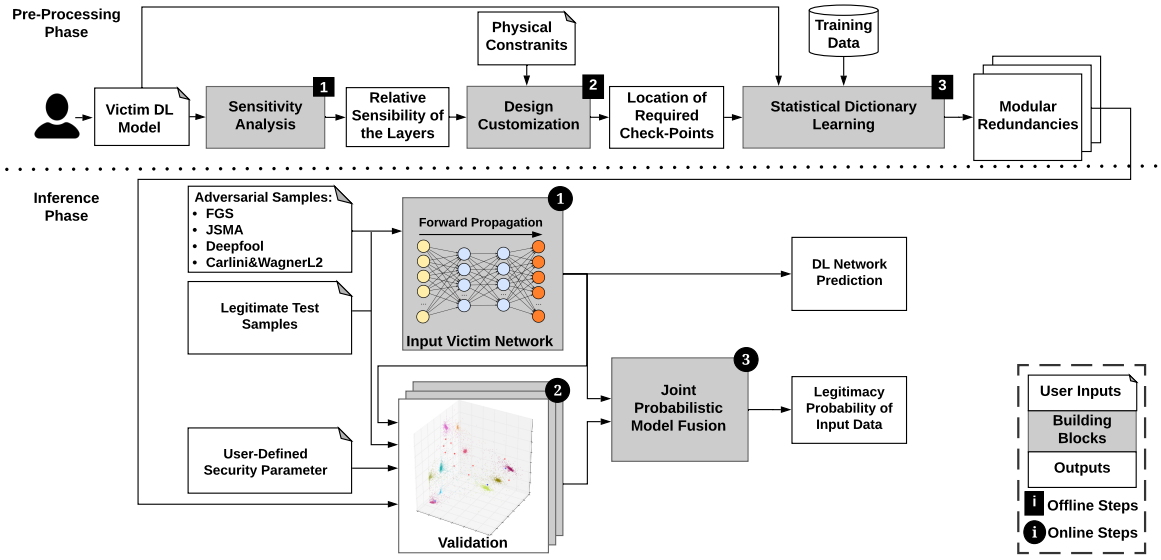


Fig. 4: Global flow of CuRTAIL framework including both off-line (pre-processing) and online (inference) phases. The pre-processing phase includes analyzing the vulnerability of the input neural network (called victim) and learning a set of redundancy modules (defensive dictionaries) to maximally cover the unexplored space in the victim model. The learned dictionaries are used in the inference phase to detect potential adversarial samples fed into the victim DL network.

This information is leveraged to find the vulnerable space of the DL model to the potential adversarial samples and identify the layers in the neural network for which the modular redundancies should be learned.

2 Design Customization. There is a trade-off between the execution runtime and the system reliability in terms of successful detection rate (Section V-E). CuRTAIL takes user-defined physical constraints such as real-time requirements into account and determines the viable number of redundancy modules (checkpoints) and their appropriate locations based on the sensitivity of DL layers computed in step 1.

3 Statistical Dictionary Learning. The key building block of CuRTAIL framework is the dictionary learning unit in which a set of redundancy modules (checkpoints) are learned based on statistical properties of the training data and the victim model parameters. To do so, CuRTAIL first trains a set of complementary neural networks (defenders) with the goal of separating data manifolds in each checkpointing layer by careful realignment of legitimate training data within each class (Section V-B). The complementary networks pose an exact structure of the victim model. The key difference between the defenders and the victim model is the loss function for which each of these models is optimized. Once the complementary neural networks are trained, CuRTAIL uses the probability density function of the acquired features to learn the corresponding dictionary of each checkpoint location.

Note that the pre-processing phase in CuRTAIL framework is an off-line process that is only performed once per DL application. The cost of CuRTAIL pre-processing is amortized over time as the DL network and its associated defensive redundancy modules are employed for online DL inference.

(ii) Inference Phase. In the inference phase, the incoming adversarial samples (Section III) along with the legitimate test data are simultaneously fed to the victim and defender models. CuRTAIL goes through three steps to find the inference label for each incoming data and provides a precise confidence interval for the network prediction.

1 Forward Propagation. The predicted class for each incoming sample is acquired through forward propagation in the main DL network (victim model). CuRTAIL defense mechanism is devised to provide a confidence interval for the network prediction. This confidence interval is used for validation of the input-output pairs in the target application and does not impact the accuracy of the main model.

2 Validation. CuRTAIL leverages the statistical dictionaries learned in the previous steps to validate the legitimacy of the input data and the associated output. In particular, samples that do not lie in the *user-defined* probability interval which we refer to as the *Security Parameter (SP)* are discarded as suspicious samples. SP is a constant number in the range of $[0-100]$ that determines the hardness of adversarial detectors. For applications with excessive security requirements, a high SP value should be set to assure full detection of adversarial samples. A high detection rate (e.g., 100%) may come at the cost of having several false positives in the detection process as will be discussed in Section VI.

3 Joint Probabilistic Model Fusion. The outputs of the redundancy modules (dictionaries) are finally aggregated to compute the legitimacy probability of the input data and its associated inference label (Section V-D).

V. CuRTAIL Methodology

The existence of adversarial samples indicates the presence of small (in the Euclidean distance sense) additive noise patterns in the input space that can make a large impact on the model's output. This phenomenon particularly happens due to two main reasons:

(i) Linear behavior in high-dimensional spaces. Small amounts of perturbation in the input space are exacerbated by successive projections through different layers of a DL network. Mathematically speaking, for each input sample x, the output of a DL network consisting of L successive layers is computed as follows:

$$f(X, \theta) = f_L(f_{L-1}(...f_1(x, \theta_1), \theta_2)..., \theta_L) \tag{5}$$

where f_l is the mapping operator used in layer l and θ_l is the corresponding parameter set of that layer. As such, the overall instability of the DL system is the multiplication aggregate of the instability of each layer [11]. In other words,

$$S = \prod_{l=1}^{L} S_l, \tag{6}$$

where S_l indicates the instability of layer l. In Section V-A, we provide quantitative measures to characterize the instability of each layer normalized to the possible perturbation levels.

(ii) Insufficient regularization in supervised learning. A machine learning model built upon a limited set of labeled data is strongly-robust against adversarial samples if and only if it is trained based on an exact subset of features used by the perfect oracle (e.g., human annotator) [31]. Although DL networks with a more complex topology (e.g., a network with a deeper structure and more number of neurons per layer) can potentially achieve a better accuracy, they are also more vulnerable to adversarial attacks due to the larger space unexplored by such models. Our hypothesis is that a proper feature representation learning is the key to obtain a learning model that is both accurate and robust. As we discuss in Section V-B, adversarial samples can be effectively detected by careful checkpointing in the intermediate stages based on probabilistic dictionaries learned from the data and DL parameter distributions.

A. *Spectral Analysis of DL Sensitivity*

The perturbation signal in adversarial samples can be cast as an additive noise added to the input data. In particular, for each layer F_l we define the instability as:

$$\underset{r \neq 0}{Sup} \frac{\|f_l(x_l + r_l) - f_l(x_l)\|}{\|r_l\|}, \tag{7}$$

where Sup stands for the supremum value, x_l is the input to the l^{th} layer, and r_l is the additive perturbation propagated to the input of that layer. For instance, for a fully-connected layer with parameter set θ_l, Eq. (7) is equivalent to:

$$\underset{r \neq 0}{Sup} \frac{\|f_l(x_l + r_l) - f_l(x_l)\|}{\|r_l\|} = \frac{\|\theta_l \times r_l\|}{\|r_l\|} \leq \|\theta_l\|, \tag{8}$$

where the last step is obtained by applying the Cauchy-Schwarz inequality. The same equation applies to convolutional layers for which the Tensor kernels should be vectorized to form θ_l.

The instability of core computation layers (e.g., fully-connected and convolution layers) is bounded by the spectrum of their underlying parameters. Considering the principal spectrum of the parameter set θ_l, the upper bound in Equation (8) is achieved if and only if the perturbation vector r is aligned with the main Eigenvector of the underlying parameters. To quantify and compare the instability of various layers in a DL network, we suggest using the *Spectral Energy Factor (SEF)* preserved by the first Eigenvalue as a measurement to identify most sensitive intermediate layers. The SEF metric is computed as:

$$SEF(\theta_l) = \frac{|e_1|}{\sum_{i=1}^{min(N_{l-1}, N_l)} |e_i|}, \tag{9}$$

where $|e_i|$ is the absolute value of the i^{th} Eigenvalue and N_l is the number of neurons (units) in the layer l. The sensitivity of specific non-linearity layers outlined in Table I is upper bounded by their Lipschitz constant as shown in [11].

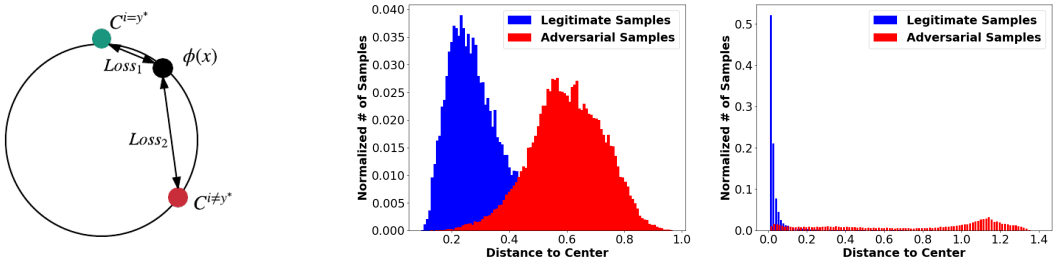
B. *Training Redundancy Modules For Intermediate Layers*

The goal of each intermidiate defender (checkpointing) module is to learn the pdf of the explored sub-spaces in a particular DL feature map. The learned density function is then used to identify the rarely observed regions. We consider a Gaussian Mixture Model (GMM) as the prior probability to characterize the data distribution at each checkpoint location. We emphasize that our proposed approach is rather generic and is not restricted to the GMM distribution. The GMM distribution can be replaced with any other prior depending on the application.

To effectively characterize the explored sub-space as a GMM distribution, one is required to minimize the entanglement between each two Gaussian distribution (corresponding to every two different classes) while decreasing the inner-class diversity. Training a defender module is a one-time offline process and is performed in three steps:

Step I. Replicating the victim neural network and all its feature maps. An L_2 normalization layer is inserted in the desired checkpoint location. The normalization layer maps the latent feature variables, $f(x)$, into the Euclidean space such that the acquired data embeddings live in a d-dimensional hyper-sphere, i.e., $\|f(x)\|_2 = 1$. This normalization is crucial as it partially removes the effect of over-fitting to particular data samples that are highly correlated with the underlying DL parameters. The L_2 norm is selected to be consistent with our assumption of GMM prior distribution. This norm can be easily replaced by an arbitrarily user-defined norm through our accompanying API.

Step II. Fine-tuning the replicated network to enforce disentanglement of data features (at a particular checkpoint location). To do so, we optimize the defender module by



(a) (b) (c)

Fig. 5: (a) Illustration of the optimization objective in each defender module. (b) The distance of legitimate (blue) and adversarial (red) samples from the corresponding centers C^i before, and (c) after realignment of data samples. In this example, we consider the LeNet3 model ([32]) trained on MNIST dataset (the checkpoint is inserted in the second-to-last layer) and adversarial samples are generated by FGS attack with different perturbation levels.

incorporating the following loss function with the conventional cross entropy loss:

$$\mathcal{L} += \gamma \; [\; \underbrace{\|C^{y^*} - f(x)\|_2^2}_{loss_1} \; - \; \underbrace{\Sigma_{i \neq y^*} \|C^i - f(x)\|_2^2}_{loss_2} \; + \; \underbrace{\Sigma_i (\|C^i\|_2 - 1)^2}_{loss_3} \;]. \quad (10)$$

Here, γ is a trade-off parameter that specifies the contribution of the additive loss term, $f(x)$ is the corresponding feature vector of input sample x at the checkpoint location, y^* is the ground-truth label, and C^i denotes the center of all data abstractions ($f(x)$) corresponding to class i. The center values C^i and intermediate feature vectors $f(x)$ are trainable variables that are learned by fine-tuning the defender module. In our experiments, we set the parameter γ to 0.01 and retrain the defender model with the same optimizer used for training the victim model. The learning rate of the optimizer is set to $\frac{1}{10}$ of that of the victim model as the model is already in a relatively good local minima.

Figure 5a illustrates the optimization goal of each defender module per Eq. (10). The first term ($loss_1$) in Eq. (10) aims to condense latent data features $f(x)$ that belong to the same class. Reducing the inner-class diversity, in turn, yields a sharper Gaussian distribution per class. The second term ($loss_2$) intends to increase the intra-class distance between different categories and promote separability. The composition of the first two terms in Eq. (10) can be arbitrarily small by pushing the centers to ($C^i \leftarrow \pm\infty$). We add the term, $loss_3$, to ensure that the underlying centers lie on a unit d-dimensional hyper-sphere and avoid divergence in training the defender modules.

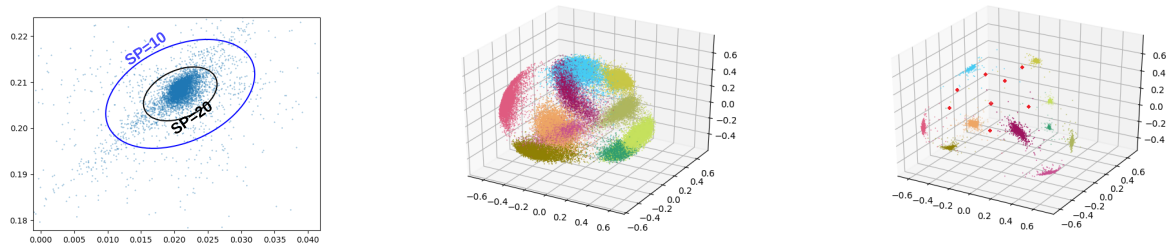
Figures 5b and 5c demonstrate the distance of legitimate (blue) and adversarial (red) samples from the corresponding centers C^i in a checkpoint module before and after retraining.[2] As shown, fine-tuning the defender module with proposed objective function can effectively separate the distribution of legitimate samples from malicious data points. Note that training the defender module is carried out in an unsupervised setting, meaning that no adversarial sample is included in the training phase.

[2]The centers C^i before fine-tuning the checkpoint (defender) module are equivalent to the mean of the data points in each class.

Step III. High dimensional real-world datasets can be represented as an ensemble of lower dimensional sub-spaces ([33], [34], [35]). As discussed in ([33]), under a GMM distribution assumption, the data points belonging to each class can be characterized as a spherical density in two sub-spaces: (i) The sub-space where the data actually lives and (ii) its orthogonal complementary space. We leverage High Dimensional Discriminant Analysis (HDDA) algorithm ([33]) to learn the mean and the conditional covariance of each class as a composition of lower dimensional sub-spaces.

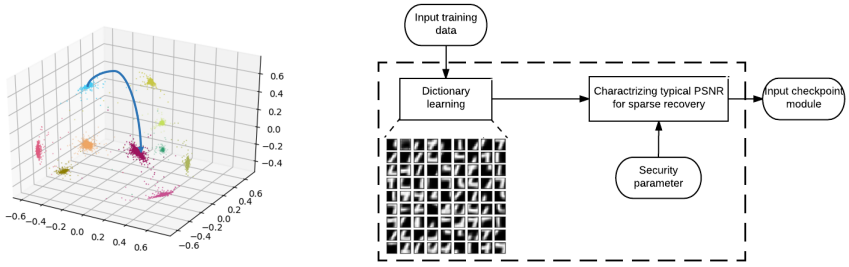
The learned pdf variables (i.e., mean and conditional covariance) are used to compute the probability of a feature point $\phi(x)$ coming from a specific class. In particular, for each incoming test sample x, the probability $p(f(x)|y^i)$ is evaluated where y^i is the predicted class (output of the victim neural network) and $f(x)$ is the corresponding data abstraction at the checkpoint location. The acquired likelihood is then compared against a user-defined *cut-off threshold* which we refer to as the *security parameter*. The Security Parameter (SP) is a constant number in the range of $[0\% - 100\%]$ that determines the hardness of defender modules. Figure 6 illustrates how the SP can control the hardness of the pertinent decision boundaries. In this example, we have depicted the latent features of one category that are projected into the first two Principal Component Analysis (PCA) components in the Euclidean space (each point corresponds to a single input image). The blue and black contours correspond to security parameters of 10% and 20%, respectively. For example, 10% of the legitimate training samples lie outside the contour specified with $SP = 10\%$.

One may speculate that an adversary can add a structured noise to a legitimate sample such that the data point is moved from one cluster to the center of the other clusters; thus fooling the defender modules (Figure 7a). The risk of such attack approach is significantly reduced in our proposed MRR countermeasure due to three main reasons: (i) Use of parallel checkpointing modules; the attacker requires to simultaneously deceive all the defender models in order to succeed. (ii) Increasing intra-class distances in each checkpointing module; The latent defender modules are trained such that not only the



(a) (b) (c)

Fig. 6: (a) Illustration of the effect of security parameter (SP) on the detection policy. A high SP leads to a tight boundary which treats most samples as adversarial examples. (b) Example feature samples in the second-to-last layer of LeNet3 trained for classifying MNIST data. (c) Latent feature samples of the same layer in the defender module after data realignment. The majority of adversarial samples (e.g., the red dot points) reside in the regions with low density of training samples.



(a) (b)

Fig. 7: An input defender module is devised based on robust dictionary learning techniques to automatically filter out test samples that highly deviate from the typical PSNR of data points within the corresponding predicted class.

inner-class diversity is decreased, but also the distance between each pair of different classes is increased (see Equation (10)). (iii) Learning a separate defender module in the input space to validate the Peak Signal-to-Noise Ratio (PSNR) level of the incoming samples as discussed in Section V-C. In the remainder of the paper, we refer to the defender modules operating on the input space as the *input defender*. MRR modules that checkpoint the intermediate data features within the DL network are referred as *latent defender*.

C. *Training Redundancy Modules For The Input Space*

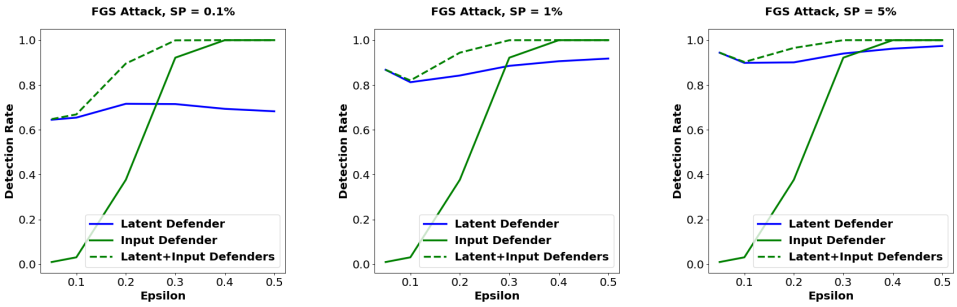
We leverage dictionary learning and sparse signal recovery techniques to measure the PSNR of each incoming sample and automatically filter out atypical samples in the input space. Figure 7b illustrates the high-level block diagram of an input defender module. Devising an input checkpoint model is performed in two main steps: (i) dictionary learning, and (ii) characterizing the typical PSNR per class after sparse recovery. **(I)** Dictionary learning; we learn a separate dictionary for each class of data by solving:

$$\underset{D^i}{argmin}\ \frac{1}{2}\|Z^i - D^i V^i\|_2^2 + \beta\|V^i\|_1 \quad s.t. \quad \|D_K^i\| = 1, \quad 0 \leq K \leq K_{max}, \tag{11}$$

Here, Z^i is a matrix whose columns are pixels extracted from different regions of input images belonging to category i. For instance, if we consider 8×8 patches of pixels, each column of Z^i would be a vector of 64 elements. The goal of dictionary learning is to find matrix D^i that best represents the distribution of pixel patches from images belonging to class i. We denote the number of columns in D^i by k_{max}. For a certain D^i, the image patches Z^i are represented with a sparse matrix V^i, and $D^i V^i$ is the reconstructed patches. We leverage Least Angle Regression (LAR) method to solve the Lasso problem defined in Eq. (11).

For an incoming sample, during the execution phase, the input defender module takes the output of the victim DL model (e.g., predicted class i) and uses Orthogonal Matching Pursuit (OMP) routine ([36]) to sparsely reconstruct the input data with the corresponding dictionary D^i. The dictionary matrix D^i contains a set of samples that commonly appear in the training data belonging to class i; As such, the input sample classified as class i should be well-reconstructed as $D^i V^*$ with a high PSNR value, where V^* is the optimal solution obtained by the OMP routine. During the execution phase, all of the non-overlapping patches within the image are denoised by the dictionary to form the reconstructed image.

(II) Characterizing typical PSNR in each category; we profile the PSNR of legitimate samples within each class and find a threshold that covers all legitimate training samples. If an incoming sample has a PSNR lower than the threshold (i.e., high perturbation after reconstruction by the corresponding dictionary), it will be regarded as a malicious data point. In



(a) (b) (c)

Fig. 8: Adversarial detection rate of the latent and input defender modules as a function of the perturbation level for (a) $SP = 0.1\%$, (b) $SP = 1\%$, and (c) $SP = 5\%$. In this experiment, the FGS attack is used to generate adversarial samples and the perturbation is adjusted by changing its specific attack parameter ϵ.

particular, PSNR is defined as:

$$PSNR = 20log_{10}(MAX_I) - 10log_{10}(MSE), \qquad (12)$$

where the mean square error (MSE) is defined as the L_2 difference of the input image and the reconstructed image based on the corresponding dictionary. The MAX_I is the maximum possible pixel value of the image (e.g., 255).

Figure 8 demonstrates the impact of perturbation level on the pertinent adversarial detection rate for three different security parameters (cut-off thresholds). In this experiment, we have considered the FGS attack with different ϵ values on the MNIST benchmark. Table II summarizes the DL model topology used in each benchmark. The latent defender module (checkpoint) is inserted at the second-to-last layers. As shown, the use of input dictionaries facilitate automated detection of adversarial samples with relatively high perturbation (e.g., $\epsilon > 0.25$) while the latent defender module is sufficient to effectively distinguish malicious samples even with very small perturbations. We extensively evaluate the impact of security parameter on the ultimate system performance for various benchmarks in Section VI.

D. *Joint Probabilistic Model Aggregation*

The output of redundancy modules in CuRTAIL framework are combined in a weighted aggregation setting. The weight of each redundancy module can be easily set by users in our provided API to account for prior knowledge about the testing environments of the underlying application. In the default setup of our API, the contribution (weight) of each latent defender is set in accordance with the sensitivity of the corresponding DL layer. Let us denote the contribution of the i^{th} latent dictionary by ϕ^i. As such, ϕ^i is primary set to $SEF(\theta_i)$ where $SEF(\theta_i)$ is defined in Equation (9). The weights of the latent dictionaries are then scaled such that:

$$\sum_{i=1}^{N_{MRR}} \phi^i = 1, \qquad (13)$$

where N_{MRR} is the total number of modular redundancies.

The aggregated results of the latent dictionaries for each sample is used along with the output of the input dictionary to determine the legitimacy level of an input-output pair in CuRTAIL framework. In our experiments in Section VI, we assume an equal contribution for the aggregated latent feature dictionaries and the input dictionary.

E. *Complexity and Reliability Trade-off*

There is a trade-off between the computational complexity (e.g., runtime overhead) of the modular redundancies and the reliability of the overall system. On the one hand, a high number of validation checkpoints increases the reliability of the systems, but it also increases the computational load as each input sample should be validated by more defender networks. On the other hand, a small number of checkpoints degrades the defense mechanism performance by treating adversarial samples as legitimate ones. The execution complexity of each latent dictionary is equivalent to the cost of one forward propagation in the target model. This is because each complementary network has a similar topology as the victim model. For the input dictionary detector, the complexity of computing the sparse representation for a single patch of pixels is $\mathcal{O}(m \times K \times K_{max})$, where K_{max} is the number of samples in the pertinent dictionary, m is the number of input features (i.e., pixels in the patch), and K is the desired number of nonzeros in the sparse representation. Figure 9 demonstrates the utility and reliability trade-off for analyzing MNIST dataset on LeNet DL model. The runtime is normalized with respect to the cost of one forward propagation in the target neural network. In this experiment, the DL execution is performed *sequentially* on an NVIDIA Geforce 980 GPU hosted by an Intel Core-i7 CPU. We emphasize that MRR computations (different colored bars in Figure 9) can be run in parallel to minimize the runtime overhead.

CuRTAIL takes the user-defined runtime constraint for DL execution of one sample in the target application as its input. Our framework provides automated subroutines to perform platform profiling on various CPU and CPU-GPU hardware. The platform profiling is a one-time process and takes $10 - 100$ $msec$ depending on the hardware platform. CuRTAIL adjusts the number of viable checkpoints according to the user-specific runtime constraint and available compu-

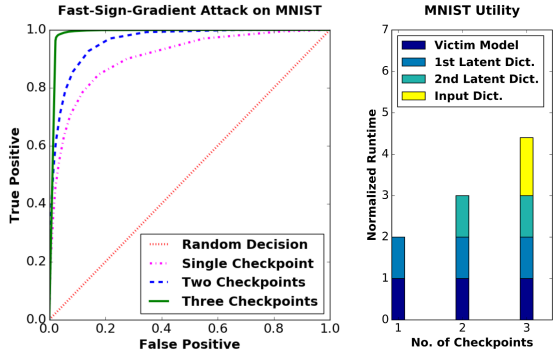


Fig. 9: Complexity and reliability trade-off for analyzing MNIST dataset on LeNet DL model.

tational resource provisioning. The intermediate DL layers are prioritized based on the instability analysis outlined in Section V-A. Due to the space limits, we focus our experiments in Section VI on using only two checkpoints, one in the input data space and one in the last hidden layer of the neural network prior to the output layer. This setup corresponds to the minimum overhead for the proposed defense mechanism (twice the execution time in the primary network). As shown in Figure 9, intermediate checkpoints can be used to effectively improve the detection performance in the target application.

VI. Evaluations

We evaluate CuRTAIL framework on three canonical machine learning datasets: MNIST [24], CIFAR10 [25], and ImageNet [26]. Figure 10 illustrates several representative samples from each dataset.

Fig. 10: Example legitimate samples in each benchmark dataset. These samples are randomly selected from each of the target classes in the MNIST (top row), CIFAR10 (middle row), and ImageNet (bottom row) datasets.

MNIST Benchmark. The MNIST data is a collection of 70000 black and white images of handwritten digits where the goal is to classify the written digits $(0-9)$ into one of the potential 10 classes. The images are normalized such that each pixel takes a real value in the range of 0 to 1. The original data is split into 60000 training samples and 10000 test samples. In our experiments for this dataset, we train and use the DL topology proposed in [32] which is also available in Table II.

CIFAR10 Benchmark. The CIFAR10 data [25] is a collection of 60000 color images of size 32×32 that are classified in 10 categories: Airplane, Car, Bird, Cat, Deer, Dog, Frog, Horse, Ship, and Truck. The images are represented in three (red, green, blue) channels and are normalized such that each pixel takes a value in the $[0-1]$ range. We split the data samples into a set of 50000 training data and a set of 10000 test data. In our experiments, we train and use the state-of-the-art DL topology proposed in [37] for the CIFAR10 dataset. Details about the architecture are available in Table II.

TABLE II: Baseline (victim) network architectures for evaluated benchmarks. Here, $\mathbf{128C3(2)}$ denotes a convolutional layer with 128 maps and $\mathbf{3 \times 3}$ filters applied with a stride of 2, $\mathbf{MP3(2)}$ indicates a max-pooling layer over regions of size $\mathbf{3 \times 3}$ and stride of 2, and $\mathbf{300FC}$ is a fully-connected layer consisting of 300 neurons. All convolution and fully connected layers (except the last layer) are followed by ReLU activation functions. A Softmax activation function is applied to the last layer of each network.

Benchmark	Architecture
MNIST	784-300FC-100FC-10FC
CIFAR10	$3 \times 32 \times 32 - 300C3(1) - MP2(2) - 300C2(1) - MP2(2) - 300C3(1) - MP2(2) - 300FC - 100FC - 10FC$
ImageNet	$3 \times 224 \times 224 - 96C11(4) - 256C5(1) - MP3(2) - 128C3(1) - MP3(2) - 128C3(1) - 128C3(1) - MP3(2) - 1024FC - 1024FC - 10FC$

ImageNet Benchmark. ImageNet [26] is a large database consisting of over 15 million data samples. The images are collected from the web and human-labeled using Amazon's Mechanical Turk tool. Typically, a subset of images belonging to 1000 different categories is used by the research community for learning evaluation of ImageNet data [6]. In our experiments, we train and use a DL architecture inspired by the well-known AlexNet [6] DL topology for ImageNet classification. Details about the trained model are available in Table II. We down-sample ImageNet classes by a factor of 100 for execution efficiency purposes. The selected classes include Tinca Tinca fish (a.k.a., Tench), black swan, Chrysanthemum dog, tiger beetle, academic gown and robe, cliff dwelling, hook and claw, paper towel, one-armed bandit, and water tower.

A. *CuRTAIL Open-Source API*

We provide an accompanying end-to-end API to ensure easy adoption of CuRTAIL framework by data scientists and engineers[3]. Our implementations for multi-core CPU and CPU-GPU platforms are built to work with the highly popular DL library known as TensorFlow [38]. CuRTAIL API takes user-specific variables such as DL model and training data, the desired security parameter SP (which, in turn, defines the hardness of defensive redundancy modules), and the underlying physical constraints in terms of runtime budget for DL execution of one data sample in the target application (Section V-E). The accompanying API automates the sensitivity analysis, design customization, and dictionary learning processes in CuRTAIL framework. The current realization of CuRTAIL framework provides support for DL sensitivity analysis against state-of-the-art attacks including fast-sign-gradient (FSG) [12], Jacobian Saliency Map attack

[3]Codes are available at https://github.com/Bitadr/CurTAIL

(JSMA) [13], Deepfool [27], and Carlini&WagnerL2 [28]. Our API is implemented based on an object-oriented design. As such, new modular redundancies and/or new attack models can be easily incorporated in our API by simply defining the desired functionality as a new class.

B. *CuRTAIL Performance Against Adversarial Attacks*

To illustrate the effect of our defense algorithm, we create adversarial samples using the four attack scenarios described in Section III. The parameters for each attack algorithm is outlined in Table III. We used the open source library[4], which is provided by [39], for implementation of the attack algorithms. The JSMA attack was too slow on the Imagenet task, thus, we did not include the results in this study. We define the *False Positive (FP)* rate as the ratio of legitimate test samples that are mistaken for adversarial samples by CuRTAIL. The *True Positive (TP)* rate is defined as the ratio of adversarial samples detected by CuRTAIL.

TABLE III: Details of attack algorithms for each evaluated application. The FGS method [12] is characterized with a single ϵ parameter. The JSMA attack [13] has two parameters: γ specifies the maximum percentage of perturbed features and θ denotes the value added to each selected feature. The Deepfool attack [27] is characterized by the number of iterative updates, which we denote by n_{iters} in this table. For the Carlini&WagnerL2 attack [28], "C" denotes the confidence, "LR" is the learning rate, "steps" is the number of binary search steps, and "iterations" stands for the maximum number of iterations.

Application	Attack	Attack Parameters
MNIST	FGS	$\epsilon \in \{0.05, 0.1, 0.2, 0.3, 0.4, 0.5\}$
	JSMA	$\gamma = 5\%, \theta \in \{0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1\}$
	Deepfool	$n_{iters} \in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$
	Carlini&WagnerL2	$C \in \{0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.99\}$ LR = 0.1, steps = 20, iterations = 500
CIFAR	FGS	$\epsilon \in \{0.05, 0.1, 0.2, 0.3, 0.4, 0.5\}$
	JSMA	$\gamma = 5\%, \theta \in \{0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1\}$
	Deepfool	$n_{iters} \in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$
	Carlini&WagnerL2	$C \in \{0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.99\}$ LR = 0.1, steps = 20, iterations = 500
ImageNet	FGS	$\epsilon \in \{0.01, 0.05\}$
	JSMA	Attack not successful
	Deepfool	$n_{iters} \in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$
	Carlini&WagnerL2	$C \in \{0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.99\}$ LR = 0.1, steps = 20, iterations = 500

CuRTAIL Input Dictionaries. The input dictionaries of CuRTAIL are trained as follows: for each dataset, we learn separate dictionaries for each class. to learn each dictionary, we first extract small patches of pictures from the data. For each input image, we randomly subsample 30 patches and create a training set of these patches from all training images. The size of the patches was 7×7 for MNIST, 8×8 for CIFAR-10, and 16×16 for Imagenet. We set the number of columns in each dictionary to 225. The dictionaries are learned as described in Section V-C. Once the dictionaries are learned, we can run the OMP algorithm [36] to denoise each input sample using its corresponding dictionary. The PSNR of the original sample is then computed as in Eq. 12, and compared against a cut-off threshold to raise alarms for high distortion values. We set the cut-off threshold value of input defender such that all the training data are considered legitimate samples.

CuRTAIL Latent Dictionaries. For each application, we train a complementary neural network (see Section V-B) with the checkpoint placed at the second-to-last layer. More specifically, we initialize the weights of the complementary network using those of the victim model, then retrain the defender model by adding the extra term to the loss function with parameter γ set to 0.01 for all applications (see Eq. 10). For each application, we retrain the model with the same optimizer used for training the victim model, and we set the learning rate of the optimizer to $\frac{1}{10}$ of that of the victim model. Once the defender modules are trained, the low dimensional PCA features and the corresponding Gaussian PDF estimators are constructed as discussed in Section V-B.

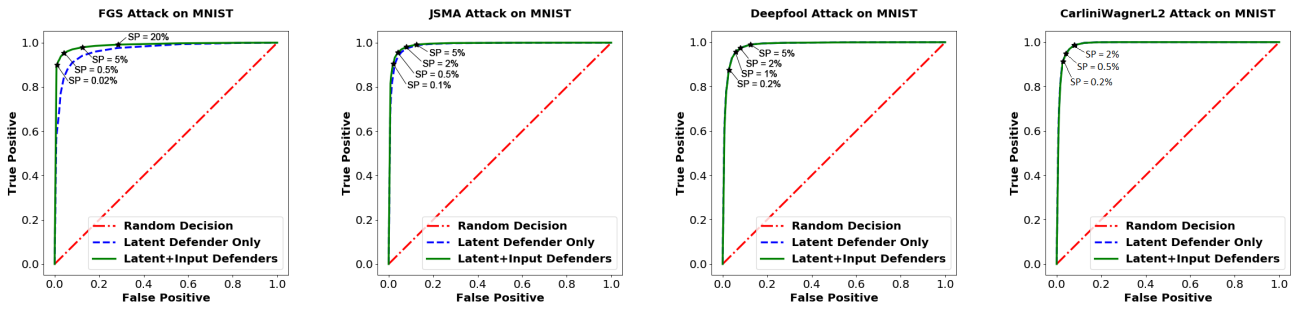
To consider the joint decision metric for each application and attack model, we evaluate the false positive and true positive rates for different applications and attack algorithms. Figure 11 presents the pertinent Receiver Operating Characteristic (ROC) curves for the MNIST application. The ROC curves are established as follows: first, we consider a latent defender and change the security parameter (SP) in the range of $[0\% - 100\%]$ and evaluate the FP and TP rates for each security parameter, which gives us the dashed blue ROC curves. Next, we consider an input defender and modify the detection policy: a sample is considered to be malicious if either of the input or latent defenders raise an alarm flag. The ROC curve for this joint defense policy is shown as the green curves in Figure 11. The gap between the dashed blue curve and the green curve indicates the effect of the input defender on the overall decision policy; as can be seen, the input defender has more impact for the FGS attack. This is compatible with our intuition since, compared to the other three attack methods, the FGS algorithm induces more perturbation to generate adversarial samples.

We summarize the performance of the CuRTAIL methodology against each of the FGS, JSMA, Deepfool, and Carlini&WagnerL2 attacks for MNIST, CIFAR10, and ImageNet in Table IV. The reported numbers in this table are gathered as follows: we consider a few points on the green ROC curve (marked on Figure 11), which correspond to certain TP rates (i.e., 90%, 95%, 98%, and 99%), then report the FP rates for these points. In all our experiments, the use of only one latent defender module to checkpoint the second-to-last layer of the pertinent victim model was enough to prevent adversarial samples generated by the existing state-of-the-art attacks.

C. *Discussion on Transferability of Adversarial Samples*

Figure 12 demonstrates an example of the adversarial confusion matrices for victim neural networks with and without using parallel checkpointing learners. In this example, we set the security parameter to only 1%. As shown, the adversarial sample generated for the victim model **are not transferred** to the checkpointing modules. In fact, the proposed approach

[4] https://github.com/tensorflow/cleverhans



(a) (b) (c) (d)

Fig. 11: ROC performance curve of CuRTAIL methodology against FGS, JSMA, Deepfool, and Carlini&WagnerL2 attacks. The diagonal line indicates the trajectory obtained by a random prediction.

TABLE IV: CuRTAIL performance against different attack methodologies for MNIST, CIFAR10, and ImageNet benchmarks. The reported numbers correspond to the pertinent false positives for achieving particular detection rates in each scenario. The JSMA attack for the ImageNet benchmark is computationally expensive (e.g., it took more than 20 minutes to generate one adversarial sample on an NVIDIA TITAN Xp GPU). As such, we could not generate the adversarial samples of this attack using the JSMA library provided by [39].

Benchmark	**MNIST**				**CIFAR10**				**ImageNet**			
Attack \ Detection Rate	90%	95%	98%	99%	90%	95%	98%	99%	90%	95%	98%	99%
FGS	1.1%	4.2%	12.4%	2.84%	8.1%	21.1%	62.9%	62.9%	14.2%	26.8%	60.7%	60.7%
JSMA	2.1%	4.2%	8.0%	12.4%	8.1%	14.9%	21.1%	33.0%	-	-	-	-
Deepfool	2.8%	5.9%	8.0%	12.4%	12.0%	17.9%	33.0%	40.8%	8.1%	8.1%	14.2%	21.5%
Carlini&WagnerL2	2.8%	4.2%	8.0%	8.0%	12.0%	17.9%	33.1%	40.8%	7.9%	7.9%	14.0%	21.3%

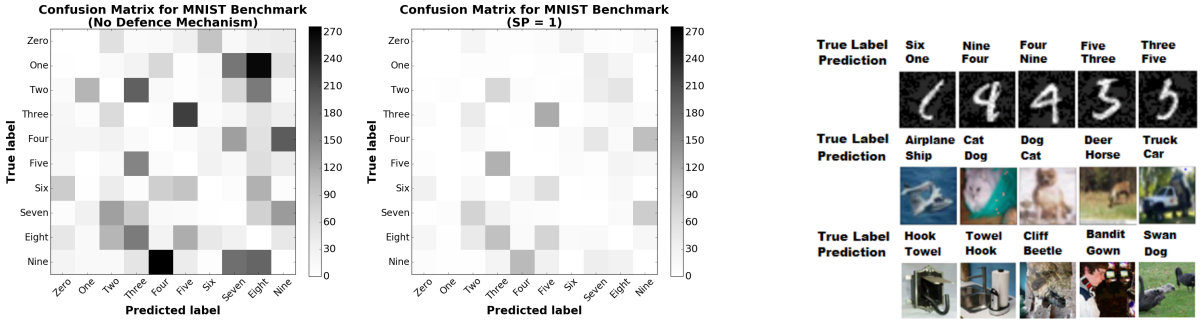
can effectively remove/detect adversarial samples by characterizing the rarely explored sub-spaces and looking into the statistical density of data points in the pertinent space.

Note that the remaining adversarial samples that are not detected in this experiment are crafted from legitimate samples that are inherently hard to classify even by a human observer due to the closeness of decision boundaries corresponding to such classes. For instance, in the MNIST application, such adversarial samples mostly belong to class 5 that is misclassified to class 3 or class 4 misclassified as 9. Such misclassifications are indeed the model approximation error which is well-understood to the statistical nature of the models. As such, a more precise definition of adversarial samples is extremely required to distinguish malicious samples form those that simply lie near the decision boundaries.

VII. Related Work

Securing machine learning models against adversarial samples is an important step towards building intelligent and autonomous systems that are reliable and trustworthy [1]. The existence of adversarial samples and their severe impact on the integrity of autonomous systems have been shown in the literature for both shallow [14], [40], [41], [42], [43] and deep [11], [12], [13], [44] learning models. Depending on the adversary goals, the adversarial attacks are performed either in the training process of a model (e.g., [45], [46]) or during the execution time where an already trained model is leveraged for data inference (e.g., [12], [40]).

In response to the various adversarial attack methodologies proposed in the literature (e.g., [12], [13], [27], [28]), several research attempts have been made to design DL strategies that are more robust in the face of adversarial examples. The existing countermeasures can be classified into two categories: **(i)** Supervised strategies which aim to improve the generalization of the learning models by incorporating the noise-corrupted version of inputs as training samples ([17], [16]) and/or injecting adversarial examples generated by different attacks into the DL training phase [18], [19], [12], [11]. The proposed defense mechanisms in this category are particularly tailored for specific perturbation patterns and can only partially evade adversarial samples generated by other attack scenarios (with different perturbation distributions) from being effective as shown in [16].
(ii) Unsupervised approaches which aim to smooth out the underlying gradient space (decision boundaries) by incorporating a smoothness penalty [47], [28] as a regularization term in the loss function or compressing the neural network by removing the nuisance variables [21]. These set of works have been mainly remained oblivious to the pertinent data density in the latent space. In particular, these works have been developed based on an implicit assumption that the existence of adversarial samples is due to the piece-wise linear behavior of decision boundaries (obtained by gradient descent) in the high-dimensional space. As such, their integrity can be jeopardized by considering different perturbations at the input space and evaluating the same attack on various perturbed data points to even pass the smoothed decision boundaries



(a) (b) (c)

Fig. 12: Example adversarial confusion matrix (a) without MRR defense mechanism, and (b) with MRR defense and a security parameter of (1%). (c) Example adversarial samples for which accurate detection is hard due to the closeness of decision boundaries for the corresponding classes.

as demonstrated in [22].

To the best of our knowledge, CuRTAIL is the first *unsupervised* learning framework developed based upon probabilistic density analysis and dictionary learning to effectively characterize and thwart adversarial samples. As corroborated in Section VI, CuRTAIL is capable of accurate detection of the state-of-the-art attack models including fast-sign-gradient [12], Jacobian Saliency Map Attack [13], Deepfool [27], and Carlini&WagnerL2 [28]. We emphasize that the MRR methodology provides a rather generic approach that can be adapted/modified against potential new attacks that might be proposed in future.

VIII. Conclusion

This paper proposes CuRTAIL, a novel end-to-end framework for characterizing and thwarting adversarial DL space. We propose a set of novel quantitative measurements to assess and compare the vulnerability of various DL topologies. CuRTAIL introduces the concept of Modular Robust Redundancy as a viable countermeasure to maximally cover the space unexplored by the victim DL mode thus reducing the risk of integrity attacks. The MRR methodology explicitly characterizes statistical properties of the features within different layers of the neural network by learning a set of complementary dictionaries and corresponding probability density functions. CuRTAIL effectiveness is evaluated against the various attack models including fast-sign-gradient, Jacobian Saliency Map Attack, Deepfool, and Carlini&WagnerL2. Proof-of-concept experiments for analyzing various data collections including MNIST, CIFAR10, and ImageNet datasets corroborate successful detection of adversarial samples with small false-positive rates. Our proposed framework is devised with an accompanying automated API to ensure ease of use for deployment of an arbitrarily DL application.

References

[1] P. McDaniel, N. Papernot, and Z. B. Celik, "Machine learning in adversarial settings," *IEEE Security & Privacy*, vol. 14, no. 3, pp. 68–72, 2016.

[2] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[3] L. Deng, D. Yu *et al.*, "Deep learning: methods and applications," *Foundations and Trends® in Signal Processing*, vol. 7, no. 3–4, pp. 197–387, 2014.

[4] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[5] G. E. Dahl, D. Yu, L. Deng, and A. Acero, "Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 20, no. 1, pp. 30–42, 2012.

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," pp. 1097–1105, 2012.

[7] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "Deepface: Closing the gap to human-level performance in face verification," pp. 1701–1708, 2014.

[8] E. Knorr, "How paypal beats the bad guys with machine learning," 2015.

[9] G. E. Dahl, J. W. Stokes, L. Deng, and D. Yu, "Large-scale malware classification using random projections and neural networks," pp. 3422–3426, 2013.

[10] Z. Yuan, Y. Lu, Z. Wang, and Y. Xue, "Droid-sec: deep learning in android malware detection," vol. 44, no. 4, pp. 371–372, 2014.

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

[12] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

[13] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," pp. 372–387, 2016.

[14] L. Huang, A. D. Joseph, B. Nelson, B. I. Rubinstein, and J. Tygar, "Adversarial machine learning," pp. 43–58, 2011.

[15] M. Barreno, B. Nelson, R. Sears, A. D. Joseph, and J. D. Tygar, "Can machine learning be secure?" pp. 16–25, 2006.

[16] S. Gu and L. Rigazio, "Towards deep neural network architectures robust to adversarial examples," *arXiv preprint arXiv:1412.5068*, 2014.

[17] J. Jin, A. Dundar, and E. Culurciello, "Robust convolutional neural networks under adversarial noise," *arXiv preprint arXiv:1511.06306*, 2015.

[18] R. Huang, B. Xu, D. Schuurmans, and C. Szepesvári, "Learning with a strong adversary," *arXiv preprint arXiv:1511.03034*, 2015.

[19] U. Shaham, Y. Yamada, and S. Negahban, "Understanding adversarial training: Increasing local stability of neural nets through robust optimization," *arXiv preprint arXiv:1511.05432*, 2015.

[20] S. Zheng, Y. Song, T. Leung, and I. Goodfellow, "Improving the robustness of deep neural networks via stability training," pp. 4480–4488, 2016.

[21] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami, "Distillation as a defense to adversarial perturbations against deep neural networks," pp. 582–597, 2016.

[22] N. Carlini and D. Wagner, "Defensive distillation is not robust to adversarial examples," *arXiv preprint*, 2016.

[23] J. Mairal, J. Ponce, G. Sapiro, A. Zisserman, and F. R. Bach, "Supervised dictionary learning," pp. 1033–1040, 2009.

[24] Y. LeCun, C. Cortes, and C. J. Burges, "The mnist database of handwritten digits," 1998.

[25] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," 2009.

[26] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," 2009.

[27] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "Deepfool: a simple and accurate method to fool deep neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2574–2582.

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

[29] L. Bottou, "Large-scale machine learning with stochastic gradient descent," pp. 177–186, 2010.

[30] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against deep learning systems using adversarial examples," *arXiv preprint arXiv:1602.02697*, 2016.

[31] B. Wang, J. Gao, and Y. Qi, "A theoretical framework for robustness of (deep) classifiers under adversarial noise," *arXiv preprint arXiv:1612.00334*, 2016.

[32] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[33] C. Bouveyron, S. Girard, and C. Schmid, "High-dimensional discriminant analysis," *Communications in StatisticsTheory and Methods*, vol. 36, no. 14, pp. 2607–2623, 2007.

[34] B. D. Rouhani, A. Mirhoseini, and F. Koushanfar, "Delight: Adding energy dimension to deep neural networks," pp. 112–117, 2016.

[35] ——, "Deep3: Leveraging three levels of parallelism for efficient deep learning," in *Proceedings of the 54th Annual Design Automation Conference 2017*. ACM, 2017, p. 61.

[36] J. Tropp, A. C. Gilbert *et al.*, "Signal recovery from random measurements via orthogonal matching pursuit," *IEEE Transactions on Information Theory*, vol. 53, no. 12, pp. 4655–4666, 2007.

[37] D. Ciregan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," pp. 3642–3649, 2012.

[38] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[39] I. G. R. F. F. F. A. M. K. H. Y.-L. J. A. K. R. S. A. G. Y.-C. L. Nicolas Papernot, Nicholas Carlini, "cleverhans v2.0.0: an adversarial machine learning library," *arXiv preprint arXiv:1610.00768*, 2017.

[40] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Šrndić, P. Laskov, G. Giacinto, and F. Roli, "Evasion attacks against machine learning at test time," pp. 387–402, 2013.

[41] B. Biggio, G. Fumera, and F. Roli, "Pattern recognition systems under attack: Design issues and research challenges," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 28, no. 07, p. 1460002, 2014.

[42] A. Anjos and S. Marcel, "Counter-measures to photo attacks in face recognition: a public database and a baseline," pp. 1–7, 2011.

[43] P. Fogla and W. Lee, "Evading network anomaly detection systems: formal reasoning and practical techniques," pp. 59–68, 2006.

[44] J. Kos, I. Fischer, and D. Song, "Adversarial examples for generative models," *arXiv preprint arXiv:1702.06832*, 2017.

[45] B. Biggio, B. Nelson, and P. Laskov, "Support vector machines under adversarial label noise." *ACML*, vol. 20, pp. 97–112, 2011.

[46] ——, "Poisoning attacks against support vector machines," *arXiv preprint arXiv:1206.6389*, 2012.

[47] T. Miyato, S.-i. Maeda, M. Koyama, K. Nakae, and S. Ishii, "Distributional smoothing with virtual adversarial training," *arXiv preprint arXiv:1507.00677*, 2015.