TUM School of Computation, Information and Technology
Technische Universität München

Defending Neural Networks with Activation Analysis

Philip Sperl

Vollständiger Abdruck der von der TUM School of Computation, Information and Technology der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Die Dissertation wurde am 12.04.2023 bei der Technischen Universität München eingereicht und durch die TUM School of Computation, Information and Technology am 09.11.2023 angenommen.

Abstract

Neural networks are vulnerable to adversarial examples. These inputs differ only slightly from their benign counterparts, yet they provoke misbehavior within the attacked models. The perturbations required to craft the examples are humanly imperceptible, making the detection a difficult task. To protect deep learning-based systems from such evasion attacks, several countermeasures have been proposed. Yet a general protection scheme is still missing. Motivated by the wide use of neural networks even in security-critical environments, in this thesis, we propose a set of defense strategies allowing a protected application. Inspired by recent advances in neuron-coverage guided testing, we show that the activation values of neural networks carry robustness-sensitive information. Based on this insight, we first present a generally applicable and modular architecture allowing a reliable detection of adversarial examples. We show that this architecture can further be adapted to be used in anomaly detection. Finally, we present means to leverage an analysis of the models' activations during neural-network training. This allows us to increase robustness and advance neural network defenses.

Zusammenfassung

Neuronale Netze sind anfällig für Adversarial Examples. Diese Eingaben unterscheiden sich nur geringfügig von originalen Eingaben, führen jedoch zu einem Fehlverhalten der angegriffenen Modelle. Die für die Erstellung der Adversarial Examples erforderlichen Eingabeänderungen sind für den Menschen nicht wahrnehmbar, was die Erkennung erschwert. Um Deep-Learning-basierte Systeme vor derartigen Angriffen zu schützen, wurden bereits verschiedene Gegenmaßnahmen vorgeschlagen. Ein allgemeiner Schutz fehlt jedoch noch. Motiviert durch den steigenden Einsatz von neuronalen Netzen auch in sicherheitskritischen Umgebungen, werden in dieser Arbeit eine Reihe von Verteidigungsstrategien vorgeschlagen, die eine geschützte Anwendung ermöglichen. Inspiriert von den jüngsten Fortschritten im Bereich des Testens neuronaler Netze wird gezeigt, dass die Aktivierungswerte der Modelle Informationen relevant für deren Robustheit enthalten. Basierend auf dieser Erkenntnis wird zunächst eine allgemein anwendbare und modulare Architektur vorgestellt, die eine zuverlässige Erkennung von Adversarial Examples ermöglicht. Es wird gezeigt, dass diese Architektur weiter angepasst werden kann, um in der Anomalieerkennung eingesetzt zu werden. Schließlich werden Mittel vorgestellt, um eine Analyse der Modellaktivierungen während des Trainings neuronaler Netze zu nutzen. Dies ermöglicht es, die Robustheit zu erhöhen und Abwehrmechanismen neuronaler Netze zu verbessern.

Contents

List of Figures

Figures 3.2, 3.3, 3.7-3.13, A.1, and A.2 are inspired by the original designs created by Jan-Philipp Schulze in our collaborative publications [1] and [2].

List of Tables

1 Introduction

Machine learning (ML) and especially deep learning (DL) applications transform modern technologies at an impressive pace. Research progress and the availability of high-performance hardware enable the training of increasingly complex models. Such DL models have achieved even super-human results in a broad range of domains: from classical image classification tasks [3], to outplaying humans in Go [4], or even autonomously driving cars [5]. Recently published large language models and their capabilities underline the advancement of the field.

In numerous scenarios the security and safety of these systems is of crucial importance. Errors in the ML processing pipeline can affect our daily routine, lead to severe incidents in the users' health, or threaten future critical infrastructures. Such errors may not only stem from inaccuracies in the training phase, but also from intentionally performed attacks. Hence, the security of systems incorporating DL concepts is a major challenge for engineers, data scientists, and the research community.

Malicious actions aiming at DL models come in two flavors according to their attack timing. *Poisoning* attacks target the training phase, while *evasion* attacks are performed in the test phase. For poisoning attacks the attacker induces changes to the training data set and especially to the labels to provoke misclassifications [6, 7]. As the training data set is typically not available to attackers, the majority of recent work focuses on evasion attacks. Here, the attacker manipulates the behavior of the DL model itself such that intended misclassifications occur. In 2014, Szegedy et al. [8] first demonstrated that small perturbations on images fed to a deep neural network (DNN) can provoke such misclassifications. Since then, new attacks and countermeasures against so-called *adversarial examples* have been introduced at a fast pace without the discovery of a fundamental and general defense strategy, yet.

1.1 Motivation

Current defenses against adversarial examples cannot fully avert successful attacks. State-of-the-art methods either reduce the attack success probability or increase the required effort for the adversary. Therefore, the question of how to protect neural networks (NNs) against adversarial examples is still open. The majority of defenses detect attacks during run-time or modify the training to enable NNs to classify inputs correctly even though the inputs were maliciously altered. Both strategies have downsides and weaknesses which can be exploited. Especially detection approaches can be bypassed if the attackers are aware of the underlying protection scheme. For example, methods which detect attacks using features in the input space are prone to miss more advanced attacks. Here, attackers can easily adapt and incorporate the core ideas of the detection approach in the generation process of the adversarial examples, circumventing exposure.

This leads to the need of new protective measures and generally a better understanding of the influencing factors facilitating robustness. In particular, a thorough analysis of model-centered defenses based on the observation of the NNs' behavior during run-time is currently missing. Such approaches might be less sensitive to attackers' adaptations and therefore provide a more reliable protection compared to input-oriented schemes. Input-independent and task-agnostic defense methods could further facilitate the secure and reliable application of NNs even in exposed systems. Hence, we present methods which protect NNs by analyzing the models' behavior. We achieve this by observing the NNs' activation values while handling benign and adversarial inputs. Based on this analysis of the activations, which was previously done only partially, we aim to close the previously described gap.

1.2 Research Challenge

In this thesis, we investigate the question whether information readily available in NNs can be used to make the models more robust. Our aim is to provide measures which increase the resilience against adversarial examples. This poses a challenging task as methods to generate such samples are highly accessible and current state-of-the-art defenses can oftentimes be bypassed using adaptive attacks. The challenges we are facing throughout this thesis are threefold. This generally applies to all defense strategies for NNs. First, the methods should be applicable for a wide range of architectures and ideally for different types of data. The number of domains and use cases in which NNs are used is ever increasing. Generally applicable defenses making the models robust and ultimately leading to certifiable NN-based systems is of high importance for the research community and model users. Secondly, the computational cost during run-time should be manageable with currently used hardware and should be within typical budget limits. Defenses usually come at a cost. Yet applying reliable systems resilient to attacks should ideally be possible in all scenarios. Thirdly, and most importantly, proposed defenses should be robust against adaptive attacks. A large proportion of methods arguing robustness was not evaluated when facing an omniscient attacker. Inspired by Kerckhoffs's principle, the following should hold true also in the realm of adversarial machine learning:

> *A neural-network-based system should be secure and robust even if everything about the system, the model, and the used countermeasures are public knowledge.*

In summary, when presenting defense strategies for neural networks, the following three challenges arise:

- General applicability to different models and data sets.
- Comparable computational cost compared to unprotected models.

- Robustness against omniscient adversaries.

Throughout the work on this thesis, we tried to provide defense strategies overcoming the aforementioned challenges. As a result, four scientific publications emerged building the foundation of the work at hand [1, 2, 9, 10].

1.3 Research Contribution

In this thesis, we propose modular and effective defense mechanisms against adversarial examples. The methods are designed with respect to the previously introduced research challenges. Furthermore, we aim to provide methods using information readily available in the models to protect. As we have stated above, the methods presented in this thesis aim to first analyze the models during input processing. Then in the second step, the defenses take effect and protect the NNs against adversarial examples. More specifically, based on the analysis of the hidden activations of neural networks we present passive as well as active methods. This results in adversarial example detection schemes as well as a robust training method. Our approaches generalize between a broad range of state-of-the-art attacks and therefore do not only cover contemporary attacks, but will, based on our experiments, also defend against future attacks which follow similar approaches as current ones.

1.4 Outline

This thesis is structured as follows: In Chapter 2 we present background information and summarize state-of-the-art research in adversarial machine learning. This allows us to present two adversarial defense strategies as well as an anomaly detection method in Chapter 3. In this chapter, we show that the hidden activation values can be used to detect attacks using our newly introduced target-alarm architecture. We conclude the chapter by generalizing the findings to the challenging domain of anomaly detection. All three methods are based on passively analyzing the available information in the NNs we want to protect. Complementary, in Chapter 4, we present an active defense method which is applied during the training of NNs. We show that the activation values can be used prior to the deployment to make the models inherently more robust. Finally, in Chapter 6 we summarize our findings, show the connections between our individual methods, and conclude this thesis by presenting open questions for future research.

2 Background

In this chapter, we provide an overview of the background needed throughout this thesis. This includes an introduction to NNs and the implied security risks as well as the most influential findings on defense strategies.

2.1 Neural Networks

We give a brief introduction to NNs, the learning principles, and their inner workings. For the interested reader seeking a detailed overview of DL and NNs in particular we refer to the book by Goodfellow et al. [11].

With NNs we try to approximate complex input-output mappings. Because of the high number of tunable parameters NNs are applicable even in high-dimensional environments like image processing. More formally, NNs approximate the function $f(\mathbf{x}; \boldsymbol{\theta}) = \hat{y}$ with $\boldsymbol{\theta}$ being the parameters adapted during training, $\mathbf{x}$ the input vector, and $\hat{y}$ the output, i.e., the prediction of the model. Commonly used NN architectures consist of multiple layers sequentially processing the inputs. The NN consisting of M layers can therefore be written as $f = f_M \circ f_{M-1} \circ \ldots \circ f_1$. Each layer forwards its processed inputs to the following one with $f_i(\mathbf{x}; \boldsymbol{\theta}) = \mathbf{h}_i$. We denote the output of each layer as $\mathbf{h}_i = \sigma(\mathsf{W}_{i,i-1} \cdot \mathbf{h}_{i-1} + \mathbf{b}_i)$. $\mathsf{W}_{i,j}$ and $\mathbf{b}_i$ build the mapping parameters $\boldsymbol{\theta}$ learned in layer i with respect to layer j. The non-linear activation function $\sigma(\cdot)$ is applied, for which we call the outputs $\mathbf{h}_i$, *activations* of layer i. Currently, one of the most widely used activation functions is ReLU defined as $a(\mathbf{x}) = \max(0, \mathbf{x})$ [12].

During training, the weights $\boldsymbol{\theta}_i$ are adapted using an optimization algorithm. Here, the loss $\mathcal{L}(y, \hat{y})$ is minimized which is defined as the difference between the desired output y and the current predictions $\hat{y}$. Backpropagation allows to adjust the weights to further decrease the measured loss. The gradients are calculated with $\nabla_{\boldsymbol{\theta}} \mathcal{L}(y, \hat{y})$. Throughout this thesis, we mainly consider NNs performing classification tasks. A set of known classes $\mathcal{Y}$ is assumed such that $y \in \{1, 2, \ldots |\mathcal{Y}|\}$.

2.2 Adversarial Machine Learning — Security of Neural Networks

The number of applications and systems based on NNs is ever increasing, so is the interest of potential attackers trying to influence the produced decisions. Methods which threaten the security of NNs can be divided into evasion, poisoning, and extraction attacks. In this thesis, we focus on security measures against evasion attacks which take place during run time and target trained models. Here, the classification output of the model under attack is altered to the attacker's will. This can be achieved using specifically-crafted inputs called adversarial examples. In Section 2.2.2 we present a formal description of such inputs.

Other potential threat vectors out of scope in this thesis are poisoning and information extraction attacks. Poisoning attacks are initiated during training. The attacker poisons the training data, e.g., by changing labels of deliberately chosen samples. As a result, the poisoned models are not able to achieve the same level of accuracy compared to normally trained counterparts. Alternatively, by feeding specifically-crafted triggers to the model, attackers are able to provoke chosen classification outputs to their will. Extraction attacks target either the model weights and architecture or information on the training data set. State-of-the-art neural networks like GPT-3 may contain billions of parameters [13]. Training such models requires large amounts of data and computational power. This results in model parameters being a valuable intellectual property targeted by attackers. Finally, training sample extraction or membership inference threaten highly sensitive data and information of potential users.

In the remainder of this section, we present properties of threat models usually considered during the design of defense methods for NNs. Subsequently, we present the main threat associated with evasion attacks, so-called adversarial examples. Finally, we show current attack strategies as well as the latest findings on countermeasures.

2.2.1 Threat Models in Adversarial Machine Learning

When discussing attack and defense methods, the definition of the considered threat model builds an essential but often neglected part in neural-network robustness analyses. The threat model describes the conditions under which the respective methods were designed and tested. Especially for defense strategies, the definition of the threat models conveys the assumptions under which the method is capable of guaranteeing a certain level of security. As shown by Carlini et al. [14], the definition of the threat models discusses the goals, capabilities, and knowledge of the considered adversary. In the following, we present each aspect in more detail and show the settings typically found in research. Note that we consider evasion attacks against neural networks executing classification tasks. For models performing other tasks such as object detection, partially different threat model settings might be applicable.

1. *Goals of the adversary.* In general, the attacker aims to influence the classification decision process of the attacked NNs, either by influencing and reducing the confidence of the decision or even changing the final class output. In the latter case, the attacker aims to either force the classification to a certain class, i.e., a targeted attack, or to any class different than the original one, i.e., an untargeted attack.

2. *Capabilities of the adversary.* The capabilities of an adversary can range from making changes to the input, to altering the trained network under attack. In evasion attacks, which are exclusively considered in this thesis, solely changes to the input are assumed. Hence, the attacker is capable of inducing local, i.e., patch-based attacks, or global changes to the inputs forming standard adversarial

examples. Induced changes are typically constrained by an l_p norm measuring the distance between the original inputs and the corresponding adversarial examples. In image classification tasks, the l_0, l_2, or l_∞ norms are usually considered. We introduce the standard metrics in more detail in Section 2.2.2.1.

3. *Knowledge of the adversary.* The level of the adversary's knowledge describes the available information on the complete system under attack and influences the selected attack methods. We distinguish between three cases: In black-box settings, the attacker is only aware of the input-output relation of the NN. Opposed to that, in gray-box settings, the attacker has complete knowledge of the NN under attack and can thus access the gradients which is crucial for the generation of powerful adversarial examples. Finally, in white-box scenarios, the attacker is additionally aware of any applied countermeasures and defense methods protecting the NN. Here, the attacker is omniscient and is therefore capable of directly attacking the combined system consisting of defense measure and NN in an adaptive manner. This scenario allows the evaluation of the resulting security level of the protected network without relying upon the defense method being secret.

2.2.2 Adversarial Examples

Adversarial examples are specifically perturbed inputs, which mislead the networks under attack resulting in misclassifications. As the perturbations are humanly imperceptible the adversarial examples look unsuspicious to human observers. Specifically, the changes induced by the attackers do not change the semantics of the input. More formally, adversarial examples can be defined as follows:

Let $f(\mathbf{x};\boldsymbol{\theta})$ be a trained NN used for classification tasks and let $H(\mathbf{x})$ be a human oracle with comparable classification capabilities. Then for a benign input $\mathbf{x}$

$$f(\mathbf{x};\boldsymbol{\theta}) = H(\mathbf{x})$$

holds. Let $\tilde{\mathbf{x}}$ be a slightly perturbed version of $\mathbf{x}$ such that $\|\tilde{\mathbf{x}} - \mathbf{x}\|_p \leqslant \epsilon$ for some small $\epsilon \in \mathbb{R}^+$. Here, $\|\cdot\|_p$ denotes the l_p-norm. Then, $\tilde{\mathbf{x}}$ is an adversarial example if it is misclassified by the NN while being classified correctly by the human oracle:

$$H(\mathbf{x}) = H(\tilde{\mathbf{x}}) \quad \wedge \; f(\mathbf{x};\boldsymbol{\theta}) \neq H(\tilde{\mathbf{x}}).$$

Research on adversarial example generation methods is a rapidly evolving field. The amount of published attack methods exceeds the frame of this thesis. Therefore, in the following, we introduce the most influential and widely-used attack methods. A comprehensive overview of state-of-the-art attack methods is given in the survey by Liu et al. [15].

Before we introduce the most important attack methods in Section 2.2.2.2, we briefly discuss the different distance metrics typically used in research and their impact on the resulting adversarial examples. Note, as the majority of published attacks is evaluated in the image domain, we will introduce the used distance metrics in the context of visual data.

2.2.2.1 Distance Metrics

As we have shown in the general introduction of considered threat models and the formal description of adversarial examples, the distance metrics play an essential role during the execution of attacks. These distance metrics quantify the similarity between the adversarial examples and their benign counterparts.

In Section 2.2.2, we have shown that the distance is typically limited using the parameter ϵ, such that $\|\tilde{\mathbf{x}} - \mathbf{x}\|_p \leqslant \epsilon$. The l_p-norm $\|\cdot\|_p$ used here is defined as:

$$\|z\|_p = (\sum_{i=1}^{n} |z|^p)^{\frac{1}{p}}.$$

In the following, we show the three most widely-used distance metrics.

1. *l_2:* The l_2 norm quantifies the Euclidean distance between adversarial examples and their benign counterparts. Hence, it expresses the similarity by calculating the standard root-mean-square distance. Note, the l_2 norm may be small even if all pixels of an input image were minimally changed during the adversarial example generation process.

2. *l_∞:* The l_∞ norm quantifies the similarity by returning the distance of the maximally changed input pixel with respect to its initial value. Hence, all pixels of the input image can be changed up to the defined limit.

3. *l_0:* The l_0 norm shows the total number of changed input pixels, independently of the level of induced change. This distance is currently the less used one.

2.2.2.2 Adversarial Example Attack Methods

L-BFGS Szegedy et al. [8] first demonstrated the vulnerability of NNs to slightly mutated inputs thereafter known as adversarial examples. The authors described the process of finding adversarial examples with a minimization problem:

$$\min c \cdot \|\mathbf{x} - \tilde{\mathbf{x}}\|_2^2,$$

$$\text{s.t. } f(\tilde{\mathbf{x}}) = l.$$

To solve this problem, the authors used a box-constrained L-BFGS [16]. Formally, the authors iteratively minimize the following function with respect to the added perturbations δ:

$$\mathcal{L}(f(\mathbf{x} + \boldsymbol{\delta}), l) + c \cdot \|\mathbf{x} - \tilde{\mathbf{x}}\|_2^2.$$

Here, $\mathbf{x}$ is the original input. Together with the found perturbations $\boldsymbol{\delta}$, $\tilde{\mathbf{x}}$ forms the newly generated adversarial example such that $\tilde{\mathbf{x}} = \mathbf{x} + \boldsymbol{\delta}$. Using the prediction of the model under attack $f(\mathbf{x}+\boldsymbol{\delta})$ and the desired target class of the attack l, the effectiveness of the current perturbation is quantified using a loss function $\mathcal{L}$, typically the cross-entropy. The second term of the function measures the induced perturbations controlled by the parameter c.

FGSM Shortly after Szegedy et al., Goodfellow et al. [17] presented the fast gradient sign method (FGSM) together with first principles on why adversarial examples exist. This one-step method changes the intensity (sign) of each pixel of the input image in order to find adversarial examples:

$$\tilde{\mathbf{x}} = \mathbf{x} - \epsilon \cdot sign(\nabla \mathcal{L}(f(\mathbf{x}), y)).$$

The loss is calculated with respect to the class output using the original input image and the corresponding class y. As the method solely uses the signs of the gradient of the used loss function, the required computational effort is significantly lower compared to other attacks. The parameter ϵ controls the distance between the original and adversarial examples measured with the l_∞ norm.

BIM Further refining FGSM, Kurakin et al. [18] proposed the basic iterative method (BIM). Instead of performing one single attack step, for BIM the gradient is reevaluated after each iteration to perform smaller steps controlled by α:

$$\tilde{\mathbf{x}}_{\mathbf{i}} = \tilde{\mathbf{x}}_{\mathbf{i-1}} - clip_\epsilon(\alpha \cdot sign(\nabla \mathcal{L}(f(\tilde{\mathbf{x}}_{\mathbf{i-1}}), y))).$$

In each step, the samples $\tilde{\mathbf{x}}_{\mathbf{i-1}}$ are clipped using ϵ to form the resulting adversarial examples $\tilde{\mathbf{x}}$. For the the initial step, the following start point is chosen:

$$\tilde{\mathbf{x}}_{\mathbf{0}} = \mathbf{0}.$$

PGD Similarly to BIM, Madry et al. [19] introduced projected gradient descent (PGD). Instead of performing the clipping operation, the authors project the adversarial examples onto the ϵ-ball around the original input sample:

$$\tilde{\mathbf{x}}_{\mathbf{i}} = \tilde{\mathbf{x}}_{\mathbf{i-1}} - proj_\epsilon(\alpha \cdot sign(\nabla \mathcal{L}(f(\tilde{\mathbf{x}}_{\mathbf{i-1}}), y))).$$

Furthermore, in the initial step, $\tilde{\mathbf{x}}_{\mathbf{0}}$ is randomly selected. The authors argue that their method poses the strongest first order attack.

DeepFool Moosavi-Dezfooli et al. [20] introduced a decision-based attack method called DeepFool. Instead of relying on the gradient, the authors iteratively push the inputs towards the decision boundary of the attacked NN. During this untargeted attack, the NN's decision boundaries are assumed to be linear. With this simplified assumption, the boundaries are modeled as a hyperplane. In each attack step, new perturbations are added to the current adversarial examples, pushing the decision of the attacked NN in this hyperplane towards the desired class.

JSMA The majority of proposed attack algorithms use the l_∞ or l_2 norm. In contrast to that, Papernot et al. [21] proposed the Jacobian-based saliency map attack (JSMA) which considers the l_0 norm. Hence, the attack tries to minimize the amount of changed input pixels rather than measuring the global absolute change. The attack itself can be seen as a greedy algorithm which changes the input pixels with the highest influence on the model's decision. To estimate the impact of pixel changes on the classification output, the authors use the gradients of the models to calculate saliency maps. Higher values in the saliency map indicate a higher influence on the decision.

C&W The optimization-based C&W attack introduced by Carlini and Wagner [22] is currently considered to be the most powerful white-box method. Similarly to L-BFGS, C&W minimizes the distance between the original inputs and the adversarial examples during the generation process. To find minimally perturbed adversarial examples for the l_2 norm, the following function is minimized with respect to the auxiliary variable $\mathbf{w}$ and the target class l:

$$\min_{\mathbf{w}} \left\| \frac{1}{2}(\tanh(\mathbf{w}) + 1) - \mathbf{x} \right\|_2^2 + c \cdot f(\frac{1}{2}(\tanh(\mathbf{w}) + 1)).$$

The objective function $f(\cdot)$ is defined as:

$$f(\tilde{\mathbf{x}}) = \max(\max\{Z(\tilde{\mathbf{x}})_i : i \neq l\} - Z(\tilde{\mathbf{x}})_l, -\kappa)).$$

$Z(\cdot)$ are the logit outputs of the model before applying the softmax function. The variable κ controls the confidence of the found adversarial examples and is typically set to $\kappa = 0$. The C&W method was initially proposed to successfully attack models protected with defensive distillation. Creating imperceptible perturbations with C&W is possible for the l_0, l_2, and l_∞ norm.

Other Attack Methods and Strategies The methods introduced above show the most influential findings in the research field focusing on attacks. Hence, the shown methods do not provide a complete overview. Apart from the selected methods, there following are notable attack approaches.

The one pixel attack introduced by Su et al. [23]. In their publication, the authors showed that adversarial examples successfully fooling image classification NNs can be constructed by solely changing one pixel of the original inputs.

In contrast to the above mentioned methods, Moosavi-Dezfooli et al. [24] presented universal adversarial perturbations. Rather than calculating individual adversarial perturbations added to specific inputs, the authors calculated universally applicable perturbations such that when added to an arbitrary input, the target network is fooled.

When generating adversarial examples, the gradients of the attack NNs are typically leveraged to guide the respective optimization process. In case the gradients are not available, for instance in black-box scenarios or in case the gradients are either hidden or obfuscated [25], other strategies may be used. For example, the boundary attack, presented by Brendel et al. [26] does not require the gradients of the attacked NN, nor its probability distribution. Alternatively, transfer attacks can be used to successfully fool the models under attack. In this setting, adversarial examples are generated for a so-called surrogate model controlled by the attacker. Then, in the second step, the transferability property [8, 17, 27] of the adversarial examples is leveraged to attack the original model under attack [28, 29]. Transfer attacks are typically possible for a wide range of NNs. Yet highly similar surrogate and target models result in a higher probability of successful attacks.

2.2.3 Defense Methods

Due to the ease of use of adversarial example generation methods and especially the vulnerability of NNs, the research community is making great efforts towards finding robust and reliable defense strategies. Based on this necessity, an arms race between research on attack and defense methods has evolved.

In the following, we present major findings and the most relevant publications in research investigating defense strategies. Similarly to research on attack methods, in this thesis we cannot provide a complete overview. For comprehensive summaries we refer to the surveys by Najafirad et al. [29] and Ren et al. [30].

There exist different approaches in categorizing state-of-the-art defense methods, e.g., by distinguishing between reactive or proactive methods [31]. In this thesis, we follow the categorization by Akhtar and Mian [32] using the following three classes: changes to the training procedure or input preprocessing, target model modifications, and the use of an external network add-on. Additionally and out of scope in the context of this thesis, standard security-related strategies can be followed to ensure a secure operation. Such methods are unrelated to the models themselves yet make evasion, poisoning, and extraction attacks less feasible. For example by restricting the access to the model or limiting the number of queries per user, multi-step adversarial example generation methods cannot reach their full potential.

2.2.3.1 Changes to the Training Procedure and Input Preprocessing

Adversarial Training An intuitive and widely performed defense technique is to include adversarial examples in the training phase of the model which is referred to as adversarial training. This is achieved by simply extending the training set with adversarial examples together with their original labels [33]. Adversarial training is often introduced by authors of attacks as the first strategy to prevent a successful attack [8,17,20]. Madry et al. [19] interpreted adversarial training as a robust optimization problem. The authors showed that the PGD attack method provides the most robust NNs when used as the generator in adversarial training. During the iterative and alternating process of adversarial generation and retraining, the following min-max problem is solved:

$$\min_{\boldsymbol{\theta}} \rho(\boldsymbol{\theta}), \text{where } \rho(\boldsymbol{\theta}) = \mathbb{E}_{(\mathbf{x},y)\sim P_{\mathcal{D}}}[\max_{\boldsymbol{\delta}\in\mathcal{S}} \mathcal{L}(f(\mathbf{x}+\boldsymbol{\delta};\boldsymbol{\theta}), y)].$$

$P_{\mathcal{D}}$ denotes the data distribution, $\mathcal{S}$ the perturbation budget, $\mathcal{L}$ the loss function and $\boldsymbol{\theta}$ the trained model parameters. The major downside of adversarial training is the high computational cost. During the training, adversarial examples need to be computed often following a multi-step procedure. In recent years, new methods have been proposed to increase the efficiency by bridging the performance gap between single-step and multi-step adversarial training [34, 35]. Moreover, Tramer et al. [36] showed the vulnerability of adversarially trained models to black-box attacks. Finally, the limited generalization of the approach needs to be emphasized. Adversarial training only allows the robust application of the protected models under the consideration of one particular threat model [14]. For instance, NNs retrained with l_∞-constrained methods might still be vulnerable to l_2 attacks.

Input Preprocessing While the changes to the input data in adversarial training are performed prior to the application of the NNs, the following methods provide robustness by preprocessing the data during inference. The main goal of the approaches is to break the adversarial perturbations which were carefully induced by the attackers. For this purpose, compression algorithms were tested. Dziugaite et al. [37] showed that adversarial examples created with FGSM can be made ineffective if JPG compression is applied. Based on this finding, further insights into JPG-based compression defenses were presented by Guo et al. [38] and Das et al. [39]. Luo et al. [40] proposed to apply NNs only to certain regions of the input images to decrease the influence of the adversarial patterns. Similarly, Wang et al. [41] made use of a separately executed data-transformation module, which partially removes adversarial perturbations.

A set of preprocessing-based defenses further introduced randomized operation steps. With this measure, the methods try to increase the complexity during the generation of adversarial examples. This measure especially tries to make adaptive attacks more difficult. Sharad et al. [42] provide an extensive overview of randomized defense methods together with their method called randomized squeezing. Based on their comparative study, the authors conclude that randomized preprocessing methods increase the robustness of NNs in black-box and gray-box settings, but can be bypassed using adaptive white-box attacks. For instance, by using the expectation over transformation (EOT) approach presented by Athalye et al. [43] the attackers are able to incorporate the randomized process in their adversarial example generation pipeline. By inducing randomized transformations during the calculations, the resulting adversarial examples are more robust and can successfully fool protected NNs. Alternatively, Athalye et al. [25] propose the backward pass differentiable approximation (BPDA). This method approximates non-differentiable functions with differentiable operations to improve the calculation of adversarial examples even for complex preprocessing defenses.

2.2.3.2 Target Model Modifications

An intuitive approach to increase the complexity during the generation of adversarial examples is to limit the attacker's access to the model's gradients referred to as gradient hiding introduced by Tramer et al. [36]. This leads to standard gradient-based attack methods not being applicable to directly attack the models. Note that this technique does not provide protection against black-box or transfer attacks, as shown by Papernot et al. [28]. For the interested reader, we refer to the work by Athalye et al. [25]. Here, the authors show how to attack models for which the gradients are either not available or were influenced by the applied defense strategy. Interestingly, for some defense methods, changes to the gradients are not intended by the authors. Athalye et al. refer to this case as gradient obfuscation which summarizes changes to the gradients, both intentionally and unintentionally. In their evaluation, the authors showed that gradient obfuscation often results in flawed robustness evaluations. This is due to the fact that the models appear robust in white-box settings, while being vulnerable to black-box attacks.

Based on the distillation technique shown by Hinton et al. [44], Papernot et al. [45] presented defensive distillation. Using the original model and its outputted probability vectors during inference, a smaller architecture is trained which is assumed to be less sensitive and therefore robust to adversarial examples. In their seminal work, Carlini and Wagner [22] showed that their C&W attack successfully finds adversarial examples for models protected with defensive distillation.

Finally, related to gradient hiding, Ross and Doshi-Velez [46] introduced gradient regularization. The authors proposed to penalize the degree of variation of the output based on changes in the input.

2.2.3.3 Adversarial Example Detection and External Network Add-Ons

In this subsection, we summarize defense strategies which require an additional model or comparable external components to make the original NNs more robust. As we present a generally applicable and modular adversarial example detection architecture, here we focus on publications proposing methods trying to achieve similar goals.

Adversarial Example Detection Adversarial example detection methods range from simple preprocessing-based approaches, to complex methods investigating the inner workings of the NNs to protect. In this section, we show the most influential findings in this field. For the interested reader, we refer to the survey of Zhang et al. [47] which provides a thorough comparison of state-of-the-art methods.

Hendrycks and Gimpel [48] observed that for adversarial examples, a higher weight is placed on larger principal components in comparison to benign examples. Hence, by first performing a principal component analysis (PCA), followed by training a simple classifier using the constructed features, adversarial examples can be detected. Directly operating in the input domain, Liang et al. [49] modeled adversarial perturbations as noise added to the original inputs. By applying scalar quantization and smoothing filters, the authors showed that attacks can be detected.

In a similar spirit, Gong et al. [50] proposed to train a binary classifier which directly distinguishes between benign and adversarial samples. This simple approach was later shown by Carlini and Wagner [51] to be easily bypassed using other attack methods and data sets.

Metzen et al. [52] further improved upon the findings and proposed to train a secondary classifier using the inner representations of the NN to detect adversarial examples, e.g., the outputs of convolutional layers.

Meng et al. [53] proposed MagNet, which evaluates the original data set and analyzes the manifold of the benign examples. During inference, the samples are compared to the findings about the manifold. This method is shown by Carlini and Wagner [54] to be vulnerable against attacks incorporating larger perturbations.

Similarly, Grosse et al. [55] proposed to use the maximum mean discrepancy test. With this test, the authors analyze whether two sub-data sets were drawn from the same distribution to detect adversarial examples.

Xu et al. [56, 57] introduced feature squeezing to detect adversarial examples which consists of two steps. In the first step, feature squeezing reduces the complexity of the input images by either performing spatial smoothing or color depth reduction. The authors argue that for benign samples, the introduced preprocessing steps do not alter the classification output. In contrast, for adversarial examples, the authors argue that feature squeezing results in different outputs for benign and adversarial inputs. Hence, to leverage this intuition, the authors let the original and squeezed inputs be classified by the target NN. If the distance between the outputs exceeds a predefined threshold, the samples are assumed to be adversarial.

In their work, Lu et al. [58] argued that adversarial examples produce different patterns in ReLU-activated layers in the late stages of NNs compared to benign examples. With their detection method called SafetyNet, the authors leveraged this hypothesis and used a radial basis function support-vector machine (SVM) to distinguish between original and adversarial examples.

Other Add-Ons Ever since the seminal paper by Goodfellow et al. [59], generative adversarial networks (GANs) are widely used and referred to in numerous publications. Some promising defense methods using GANs are presented by Lee at al. [60], Jin et al. [61], and Samangouei et al. [62].

Finally, some methods try to implement previously shown preprocessing steps using external network add-ons as well. Akhtar et al. [63] proposed perturbation rectifying networks (PRN). PRNs are trained individually and preprocess the input data samples fed to the original NN. In this preprocessing step, the perturbations are first rectified to allow the detection of the adversarial examples in a second step.

2.2.3.4 Robustness Certification

In addition to the methods introduced above which provide protection against adversarial examples, certification techniques pose a promising alternative in adversarial machine learning research [64–68]. At the core of the certification process, the methods try to guarantee that no adversarial example can be found for a given perturbation budget. As this thesis introduces adversarial example detection methods, certification approaches are out of scope. Yet their downsides need to be shortly mentioned, to motivate future work and our choice to focus on detection methods for now: The complexity of the methods and the accompanying computational cost limits their use to certain NN architectures. Current methods can only provide certificates which are valid for known samples $\mathbf{x} \in X$ [14]. Finally, similarly to standard defense research, Ghiasi et al. [69] presented an adaptive attack strategy successfully bypassing certified defense methods.

2.2.4 Robustness Quantification

State-of-the-art research still lacks meaningful and generally applicable metrics to assess the robustness of NNs. Strongly related to this lack is the question on how to properly measure the imperceptibility and hence quality of adversarial examples. As we have shown in Section 2.2.2.1, l_p-norms are typically used to calculate the distance between adversarial examples and their benign counterparts. The l_p-norms thus act as a proxy for the quality of the produced adversarial examples. In the majority of cases, this proxy poses a viable solution, yet it has been shown by Sharif et al. [70] and Sen et al. [71] that new metrics are needed to cover a wider variety of possible scenarios. As the two questions introduced above are not yet answered by the research community, in this thesis, we follow the standard approach of measuring the robustness of NNs and potentially accompanying defense strategies introduced below.

For this purpose, we first revisit the attack methods introduced in Section 2.2.2.2 and show a categorization of the approaches. Zhang et al. [47] for example propose to divide the attacks in gradient-based, decision-based, and optimization-based methods. This first categorization approach focuses on the adversarial example generation process itself with respect to the attacked NN and assumed threat model. Alternatively, inspecting the attacks from the adversarial example imperceptibility perspective, the methods can be divided into bounded and unbounded ones [72]: For the first category, the attacker selects a fixed distance ϵ between adversarial and benign samples. As a result, all adversarial examples produced during the attack will have the selected l_p-based distance and hence visual similarity to their benign counterparts. For the second category, the attacks try to generate adversarial examples with minimum perturbations as part of the optimization process. As a result, all adversarial examples produced during the attack will show individual l_p-based distances with respect to their benign counterparts. Based on this distinction of adversarial attack methods, we introduce the following two approaches to measure the robustness of NNs and defense methods. $\tilde{\mathcal{X}}$ denotes the set of adversarial examples fooling the attacked NN. $\mathcal{X}'$ denotes the set of samples which were perturbed by the chosen attack method yet do not successfully fool the NN.

1. *Robustness against bounded attacks:* As the attacker invests a constant amount of perturbation budget in the selected l_p space, the robustness of the NNs is assessed using the attack success rate. For robust models and constant ϵ distances, some of the attacks may not be successful and thus result in examples which are perturbed but still classified correctly by the attacked NN.

$$\text{Attack Success Rate: } ASR := \frac{|\tilde{\mathcal{X}}|}{|\tilde{\mathcal{X}}| + |\mathcal{X}'|}$$

2. *Robustness against unbounded attacks:* As the attacker invests an arbitrary amount of perturbation budget, unbounded attacks should be successful if executed correctly. In the worst case scenario, the attacks result in visually striking differences potentially extending to semantic changes of the input. Due to this property, assessing the attack success rate for unbounded attack does not provide valuable

insights on the robustness of the attacked NNs. Therefore, we quantify the robustness by measuring the required l_p perturbations to successfully fool the attacked NNs. For robust models higher perturbations budgets are required to generate adversarial examples.

$$\text{Mean } l_p \text{ Distortion: } D_{l_p} := \frac{\sum_{i=1}^{|\tilde{\mathcal{X}}|} \|\tilde{\mathbf{x}}_i\|_p}{|\tilde{\mathcal{X}}|}$$

In recent years, progress has been made in establishing new and generally applicable robustness quantification methods. For instance, Wang et al. [73] proposed CLEVER. During the calculation of CLEVER, the local Lipschitz constant is estimated to help to statistically quantify the model's robustness. Note that this method cannot be applied to all NN architectures and is only reliable for Lipschitz-continous functions. Furthermore, in a later evaluation it was shown that CLEVER might overestimate the robustness of NNs and thus does not provide a lower bound [74]. In 2020, Croce and Hein [75] presented the AutoAttack benchmark framework. The authors propose to use an ensemble of parameter-free attacks to empirically estimate the model's robustness. For this purpose, the selected attacks range from white-box attacks to gradient-free approaches. With this selection, the authors try to counteract against overestimated robustness due to incomplete observations. Orthogonal to the findings of Croce and Hein, Pintor et al. [76] present an overview of potential attack failures when empirically evaluating the robustness of NNs. Finally, for the main experimental parts of this thesis, we refer to the valuable guideline by Carlini et al. [14]. Here, the authors show in great detail how to properly assess the security of ML models and discuss potential pitfalls during this process.

3 Neural Network Activation-based Adversarial Example and Anomaly Detection

3.1 Introduction

In this chapter, we present our contributions to the research on methods increasing the protection of NN-based systems against adversarial examples. We show two detection methods each allowing a reliable application of non-robust models even in security-sensitive environments. Furthermore, we show that our modular approaches generalize to different domains, models to protect, and use cases. In the final section of this chapter we take a step back and apply our adversarial example detection methods to the challenging field of anomaly detection.

This chapter is structured as follows: In Section 3.2 we present our modular adversarial example detection method called DLA [9]. DLA analyzes the activation values of the model we try to protect using a secondary NN. Using this modular and easily expandable approach, in Section 3.3 we investigate the question, whether this general structure can be applied using other sources of information available in the models. To this end, we present DA3G [2]. Similarly, DA3G increases the protection of NNs against evasion attacks using a secondary NN detecting attacks. Rather than using the activation values, DA3G operates the gradients of the NNs. With our findings we show that the structure presented in DLA can further be expanded and used in different settings and environments. Finally, in Section 3.4 we leverage our findings to show a generally applicable anomaly detection method based on the analysis of the hidden activations of NNs called A^3 [1]. With our findings we show that the research fields of adversarial examples and anomaly detection can profit from each other and further investigations are highly relevant.

3.2 DLA: Dense-Layer-Analysis for Adversarial Example Detection

In this section, we present our adversarial example detection method DLA. Parts of the section are taken verbatim from the original paper “DLA: Dense-Layer-Analysis for Adversarial Example Detection” published at the 5th IEEE European Symposium on Security and Privacy 2020 (EuroS&P) [9].

3.2.1 Motivation

Currently, evasion attacks against NNs subdue corresponding defense methods. Research in the field of adversarial ML is yet to provide a generally applicable solution to this problem which motivates the work in this thesis. In this section, we present an adversarial example detection method called DLA. Our main idea leading to DLA is based on observing neural activity during classification run-time of the models to protect. We were inspired by recent findings in the field of NN testing and its interesting prospects. Pei et al. [77] introduced the idea of neuron coverage, which serves as a metric to guide testing of NNs. Similar to code coverage in software testing, neuron coverage quantifies the neurons which were activated by a given input. Since then, further coverage metrics have been proposed and various testing techniques have made use of them [78,79]. Odena and Goodfellow [80] reported promising results when applying concepts of coverage-guided fuzzing to NN testing.

These recent findings indicate that the neuron coverage of DL models provides robustness-sensitive information potentially beneficial to the robustness. This led us to the main insight of this section: We empirically show that neuron coverage exhibits a characteristic behavior when the analyzed model processes adversarial examples. In particular, adversarial examples provoke a unique pattern in the coverage such that respective inputs become detectable using a secondary NN. Interestingly, this characteristic is independent of the attack method and underlying data type, as our results strongly indicate. With this observation, we optimistically assume that our approach will also defend against yet unknown attacks.

In summary, in this section we present the following contributions:

- We propose a general and modular end-to-end architecture to detect adversarial examples generated using different state-of-the-art attack methods.
- We successfully detect adversarial examples in image classification, natural language processing (NLP), and DL-based audio processing.
- We implement and evaluate our approach to successfully detect prior unseen adversarial examples using various attack methods.
- We evaluate our method during adaptive attacks.

The rest of this section is structured as follows. In Section 3.2.2, we review related work and extend Section 2.2.3 by focusing on detection methods. In order to fully describe the environment in which we can successfully detect adversarial examples we introduce the threat models we consider throughout this section in Section 3.2.3. We present our main contribution, a novel concept of detecting evasion attacks on NNs, in Section 3.2.4. Section 3.2.5 shows our experiments which we thoroughly evaluate and summarize in Section 3.2.6. To further gain trust in our concept, we perform adaptive white-box attacks in Section 3.2.7. In Section 3.2.8, we discuss the restrictions of our method as well as the real-world applicability, transferability, and generalization to future attacks. Finally, we conclude this section with Section 3.2.9.

3.2.2 Related Work

In 2017, Feinman et al. [81] detected adversarial examples using two features which were extracted from dropout neural networks. With these features, a simple logistic regression is performed building the basis for a binary classifier. The first feature the authors introduced is the density estimate, based on which the distance between a given example and the sub-manifold of a class is quantified. For this purpose, the authors used the feature space of the last hidden layer of the target network. With their second feature, the Bayesian uncertainty estimate, the authors introduced an alternative feature to detect adversarial examples missed by the first feature. Here, points are detected which lie in low-confidence regions of the original input space, indicating an attack.

Similar to our method Ma et al. [82] detect attacks by observing the NN's hidden activations. The authors identify two exploitation channels which form the basis of their detection approach. By extracting provenance and value invariants, attacks are detected using a one-class SVM.

As our concept is closely related to the works by Feinman et al. [81] and Ma et al. [82] we briefly stress the main differences and potential advantages provided by our approach. Both stated frameworks detect adversarial examples by examining the inner processing of the protected NNs. In contrast to our approach, the authors further process the extracted information to craft features enabling a detection. Instead, we propose a method which directly works on the hidden activation values of the dense layers. Opposed to Feinman et al. [81] and Ma et al. [82], we use a secondary NN for the final detection. This poses a more intuitive and easy-to-implement solution. Our detection scheme works out-of-the-box without additional preprocessing or hyperparameter optimization steps. Furthermore, as our system solely comprises NNs, the detection scheme can be easily integrated in existing DL pipelines. An advantage of the concept by Ma et al. [82] is that it does not require adversarial examples during training.

3.2.3 Considered Threat Models

Based on our general introduction to threat models in adversarial machine learning presented in Section 2.2.1, in this section we summarize the attackers' capabilities considered throughout the design and evaluation of DLA. First, we introduce the main threat model usually considered in literature. Then we show our adaptive-attack setting which is used to further analyze the robustness of DLA.

3.2.3.1 Main Threat Model: Gray-Box Attacks

In this scenario, the attacker performs evasion attacks and tries to alter the classification output of our NN in a targeted manner. For this purpose, the attacker uses various state-of-the-art attack algorithms. The added adversarial perturbations are desired to be small enough, so they are imperceptible for a human expert which is consistent with the common definition of adversarial examples shown in Section 2.2.2. Finally, we consider a white-box scenario with respect to the target NN. The attacker performs simple attacks on the target NN only, resulting in an overall gray-box setting. In our main setting,

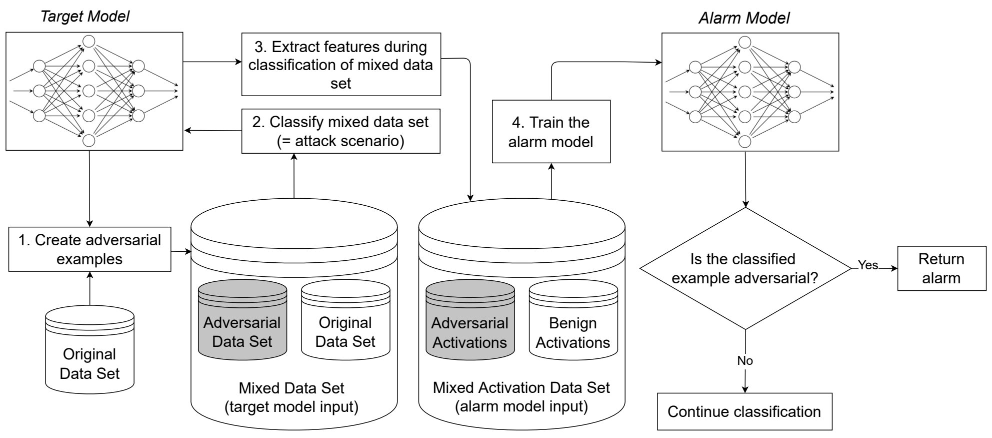

Target Model
3. Extract features during classification of mixed data set
2. Classify mixed data set (= attack scenario)
4. Train the alarm model
Alarm Model
1. Create adversarial examples
Original Data Set
Adversarial Data Set
Original Data Set
Mixed Data Set (target model input)
Adversarial Activations
Benign Activations
Mixed Activation Data Set (alarm model input)
Is the classified example adversarial?
Yes
Return alarm
No
Continue classification

Figure 3.1: Overview of DLA and the required steps to initialize and perform the adversarial example detection.

the attacker is thus not aware of the existence and nature of our proposed defense strategy. With this measure, we provide an unaltered evaluation of the performance of our detection scheme allowing a comparison to related methods.

3.2.3.2 Adaptive-Attack Threat Model: White-Box Attacks

For the adaptive attacks which we describe in more detail in Section 3.2.7, we switch sides and adjust the attacker's knowledge. Here, the adversary is aware of our proposed defense method and mounts an adaptive attack leveraging this knowledge. Therefore, the evaluation of our method under this strict setting allows a profound analysis of DLA's robustness. The level of overall protection does therefore not rely on our method being kept secret. This setting is typically found in real-world applications.

3.2.4 Detecting Adversarial Examples by Analyzing Activations

The core idea leading to the design of DLA originates in our hypothesis as initially shown in Section 3.2.1: Adversarial examples provoke a distinctive behavior of dense-layer neuron activations such that attacks become detectable. We provide a detailed description on how to expand and build upon this idea in the following.

Our method is designed to help developers and maintainers of NNs to secure their models against attacks. Hence, we assume access to the fully trained model as well as read-only access to the benign training data set. We call the model we protect target model. Our aim is to create a secure version of this model by adding our detecting scheme which produces an alarm signal whenever an adversarial example is being processed by the target. The NN which we use for this task is called alarm model. To train the alarm model, we generate adversarial examples and extract the dense-layer neuron coverage of the target model, triggered by benign and adversarial inputs. Using the extracted coverage, we directly train the alarm model in a supervised manner finally

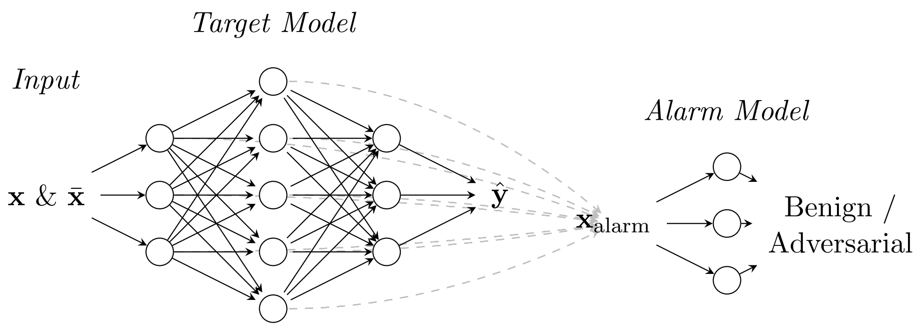


Figure 3.2: Summary of the components of the DLA architecture consisting of a target and an alarm model.

enabling a secure operation of our target model. The individual steps as well as an overview of our concept and the underlying data flow is visualized in Figure 3.1. Joining the four steps shown in the figure provides an end-to-end pipeline for fully automated adversarial example detection. Additionally, for a concise overview on a component-level, in Figure 3.2 we show the DLA architecture consisting of a target and an alarm model. The target model classifies inputs while the alarm model judges if the input $\mathbf{x}$ is benign or adversarial. To perform the attack detection, the alarm model observes the dense-layer activations of the target model triggered by either benign or adversarial inputs. In the next sections we build upon this initial description of DLA and our modular architecture to provide further details on the training and deployment process.

3.2.4.1 Adversarial Example Generation

In the first step of DLA's initialization phase, we generate adversarial examples specifically crafted for our target model. We craft these examples for each class of the data set in a white-box manner by exploiting all available information. Hence, we try to push the generated adversarial examples to be misclassified with an equal distribution among all remaining and therefore false classes. This is a crucial step during the generation phase in order to cover all possible cases which might occur during the application of our method in the field. We add the produced adversarial examples to the corresponding benign samples in a separate data set. For the adversarial example generation, we recommend using a wide range of attack methods, including state-of-the-art techniques. As we discussed in Section 3.2.2, the attacks do not only differ in success rates but also in their detectability. By covering the currently strongest attacks we build an contemporary learning basis for optimizing our detection approach. Moreover, to cover the case of black-box attacks, we recommend using transferred adversarial examples as well. Here, a surrogate target model is used to generate the required adversarial examples. It is important to note that only mutated examples should be considered which lead to misclassifications of the original target model.

3.2.4.2 Dense-Layer Neuron Coverage Extraction

In this step, we observe the target model's behavior while it processes the benign and adversarial training samples. This is visualized in the second and third steps in Figure 3.1. Here, the benign and adversarial samples are fed to the trained target model which performs classifications. Since this feature extraction step is not part of the actual function and objective of the target model, we omit its classification outputs. Instead, we extract the activation values of all available dense layers and concatenate them to one sequence for each input sample. The collected activation values are again stored in a separate data set which we use in the next section to train our alarm model. For further usage, we adopt the labels to distinguish between adversarial and benign samples. Thus, the newly created data set holds the target model's activation value sequences for all benign and adversarial examples for one specific attack method. This is visualized in the fourth step in Figure 3.1. We preserve this separation of the activation value sequences, since we assume the different attack methods to have characteristic impacts on the behavior of the target and the resulting features. This not only enables us to detect the individual attacks, but also to assess the impact of the individual crafting methods. As a result, one might be able to rank the attack methods based on their level of detectability and potentially fine-tune DLA accordingly.

3.2.4.3 Alarm Model Training

The dense-layer neuron coverage we extract in the previous step builds the basis for our core concept to detect adversarial examples. We assume that this coverage contains information about the model, its behavior, and the input. In the following, we present our architecture for automatically analyzing the extracted information.

Previous work by Feinman et al. [81], as discussed in Section 3.2.2, follows a similar idea. The authors try to extract information from neural layers and further process them to detect adversarial images. However, we directly consider the information from all dense layers of the trained model and provide an end-to-end solution without further processing steps. Accordingly, we propose to interpret the analysis of the dense-layer features as a binary classification which generalizes well over different scenarios and model architectures: Instead of performing manual feature-crafting steps and distinguishing between different scenarios, we train an additional NN to perform the required detection which we call alarm model.

To train the alarm model, we use the features stored in the previously created data set for each target model and attack method. Therefore, the network is trained to distinguish between activation values observed during the classification of benign and adversarial features. This concludes the initialization phase. In the final secure operation phase, the alarm model performs a binary classification of newly extracted features provoked by the input samples fed to the target. This enables the adversarial example detection process running alongside the original classification purpose of the target. Hence, DLA does not actively alter the main classification process of the target model.

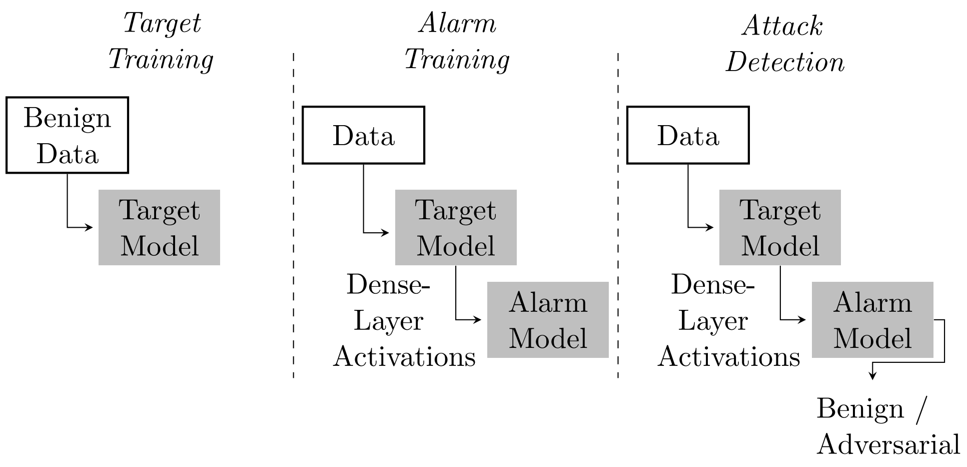


Figure 3.3: Step-by-step overview of DLA's training process in the first two columns and the final attack detection in the right column.

The architecture of the alarm model heavily influences the success of our approach. Different architectures need to be tested against each other to provide a viable and well generalizing solution. In Section 3.2.5, we recommend a specific architecture.

We recommend to create one alarm model for each introduced attack method. The attack methods differ in their approach and complexity and thus influence the neuron activation patterns distinctively. Hence, using a set of different alarm models allows us to detect a broader range of attacks. Furthermore, we are able to evaluate the capability of each alarm model version in detecting different attack methods. This provides information on the applicability of our concept when detecting future attack methods.

3.2.4.4 Concept Overview

Figure 3.3 provides a visual overview of DLA showing the required training steps and the final attack detection provided with our approach. Using a trained target model, the application of our method in a real-world scenario can be divided into two steps: the initialization and secure operation phase.

In the initialization phase, we create adversarial examples and perform the feature extraction steps. With the extracted dense-layer activations, we train our alarm models. We have discussed the importance of using different attack methods to create the adversarial examples. This may ultimately lead to a group of alarm models, each capable of detecting adversarial examples created by one specific attack method.

During the secure operation of the target model, we continuously extract the features during classification of new, unseen samples. The resulting activation sequences are fed to all available alarm models performing binary classifications. If the alarm models' outputs indicate attacks, our framework throws an alarm signal and a human expert is consulted to evaluate the current input. Here, the maintainer chooses if one assumes an attack based on one or more alarm signals, majority votes, or all alarm models synchronously indicating such an event. This use-case-depended choice provides different levels of protection. In Figure 3.3 we show a single-alarm-model setup of DLA. We consider this setup throughout this thesis.

3.2.5 Experimental Setup

In the following, we present details regarding our proof-of-concept implementation and our experimental setup. We evaluate the results in Section 3.2.6.

3.2.5.1 Data Sets

For our main experiments, we considered the MNIST [83] and CIFAR10 [84] image data sets. Both are widely used in research. This allows a comparison of our method to state-of-the art defense techniques. Furthermore, the usage of image data sets enables us to better visualize the adversarial examples and evaluate the performance of different attack methods on a perceptual level. The MNIST data set consists of 70 000 handwritten digits ranging from 0 to 9 of which 60 000 build the training set and 10 000 the test set. Each digit is represented by 28×28 gray-scale pixels. CIFAR10 consists of 60 000 colored images of which again 10 000 images build the test set. Each image is stored using $32 \times 32 \times 3$ pixels, which makes this data set more difficult to classify.

Additionally, we used a natural language processing (NLP) and an audio data set to assess the generalization of DLA to other domains. To analyze detectability of adversarial examples in the NLP context, we used the IMDb data set of movie reviews [85]. Both the train and test set contain 25 000 samples each. For both subsets positive and negative reviews are distributed evenly. The audio examples we considered during our experiments are drawn from the Mozilla Common Voice (CV) data set [86], which contains 803 hours of recorded human sentences. Contrary to the above mentioned data sets, the instances in the CV data set are used for speech-to-text conversions rather than being classified with respect to known classes.

3.2.5.2 Architectural Choices

Target Models Throughout the proof of concept, we used state-of-the-art target models in order to be consistent with setups usually found in research. In Table 3.1 we sum up the used models and show their training and test accuracy as well as a short description of the individual architectures. For MNIST, we chose LeNet5 [87] and a simple Multi-Layer-Perceptron (MLP) [88], we refer to as KerasExM. For CIFAR10 we considered ResNet [89] and a deep CNN [90], we refer to as KerasExC. In later sections we introduce two additional defense methods against adversarial examples. For both associated experiments, we used nearly identical test setups which can be found in Tables 3.8 and 4.1. In order to allow reproduction of the experiments, we added both overviews listing the relevant settings and achieved accuracy scores by the used target models.

In order to evaluate if our method can generally be applied to a wide range of DL architectures, we additionally conducted experiments using the following two examples applied for image processing: We included a Long Short Term Memory (LSTM) based target model. Here, we chose an architecture which achieves on-par results on the MNIST data set compared to CNNs. Moreover, we considered a capsule network we refer to as CapsuleNN. First experiments by Frosst et al. [91] indicate that this type of NN may be more robust to white-box attacks compared to standard CNNs.

For our NLP-based tests, we used an LSTM target model with one embedding layer. Finally, for the audio experiments we chose DeepSpeech (version 0.4.1) [92] which converts speech to text. Since it is pretrained we did not add its training and test accuracy to Table 3.1. CapsuleNN and ResNet are trained using Adam [93] while the remaining models are trained with stochastic gradient descent.

Alarm Model Architecture and Training Throughout this thesis, we used one alarm model architecture. We chose a neural network with seven layers. The input layer accepts the concatenated extracted features which are then flattened. As output we chose a softmax-activated layer consisting of two neurons in order to perform the binary classification. Therefore, throughout this thesis we did not need to optimize a decision threshold which influences the trade-off between precision and recall during detection as can be seen for other methods. Nonetheless, in some cases it might be beneficial to use one single output neuron. This would again allow a use-case specific fine-tuning of DLA's detection performance. As hidden layers, we exclusively chose ReLU-activated dense layers with the following dimensions: 112, 100, 300, 200, 77. We trained each alarm model for ten epochs and a batch size of 100 using Adam with a learning rate of 0.001.

During our proof of concept, we exclusively used this well-generalizing alarm model architecture to detect adversarial examples for MNIST and CIFAR10. With this approach, we restricted the space of tunable hyperparameters to show the generality and simplicity of our concept. Neither the used data set nor the applied target model affected our alarm model architecture. It is worth mentioning that in some cases of our experiments the chosen alarm model suffered from underfitting.

Table 3.1: Target models we used during the proof-of-concept implementation for DLA. We give a short summary of each model's architecture as well as their train and test accuracy.

Data	Model	Architecture	Accuracy
MNIST	LeNet5 [87]	– 2 convolutional layers with filter size 5 – each convolutional layer is followed by a max-pooling layer with size 2 – 2 dense layers after each max-pooling layer	– train: 97.6% – test: 98.7%
	KerasExM [88]	– 2 convolutional layers with filter sizes 32 and 64 – each convolutional layer is followed by 1 max-pooling layer – 1 flatten layer and 1 dropout layer followed by the output dense layer with 10 neurons	– train: 97.2% – test: 98.5%
	CapsuleNN [94]	– 10 capsules each of size 6	– train: 99.2% – test: 99.1%
	LSTM [95]	– 1 LSTM unit followed by two dense layers with 64 and 32 neurons	– train: 97.5% – test: 97.8%
CIFAR10	KerasExC [90]	– 4 convolutional layers with filter of size 3 – each pair of convolutional layers is followed by a max-pooling layer of size 2 – last hidden layer is fully connected with 512 neurons	– train: 85.2% – test: 79.0%
	ResNet [89]	– 3 blocks followed by an average pooling of size 8 – for further details: [89]	– train: 96.1% – test: 79.0%
IMDb	LSTM (for NLP)	– 1 embedding layer – 1 dense layer with 64 neurons	– train: 99.6% – test: 81.0%
Mozilla CV	DeepSpeech [92]	– containing 2 parts: a convolutional and a recurrent neural network – for further details: [92]	– train: – – test: –

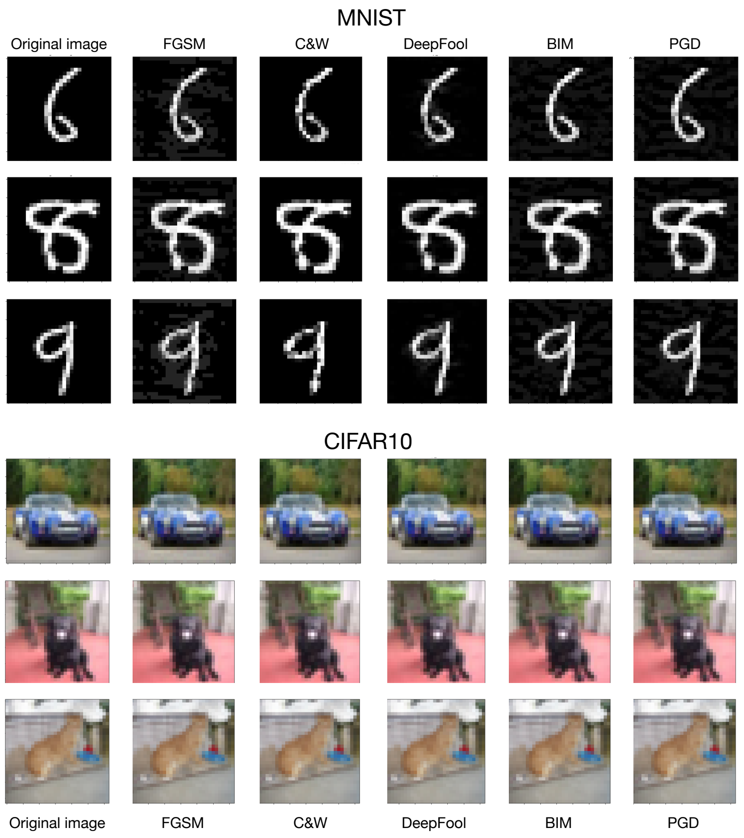


Figure 3.4: Adversarial examples for the MNIST and CIFAR10 data sets created with FGSM, C&W, DF, BIM, and PGD.

3.2.5.3 Attack Methods

We evaluated the detectability of the following attack methods: FGSM, C&W, DeepFool (DF), PGD, and BIM. The motivation to choose these methods originates in their nature and popularity. We aimed to consider a diverse set of attacks such that differences in the basic idea can be seen. For this purpose considered the one-step attack FGSM as well as the remaining multi-step methods. PGD and BIM are gradient-based approaches while C&W and DeepFool are optimization and decision-based, respectively. Moreover, we payed attention to add attacks which differ in strength and complexity. The C&W attack for instance is currently considered to be the most powerful white-box attack. Hence, DLA and future adversarial example detection schemes need to be tested against this method. On the other hand, FGSM was shown to be tractable by multiple defense strategies. In Figure 3.4 we show a series of adversarial images for both data sets crafted with the above mentioned techniques. The top images are based on the MNIST data set and successfully fool the LeNet5 target model. The bottom images are drawn from the CIFAR10 data set and successfully fool the KerasExC target model. For adversarial examples generated for the MNIST data set using FGSM, we already observe apparent perturbations in the input space. Opposed to that, for the stronger C&W attack, no difference between the original and adversarial examples is visible.

Alongside the five stated methods, we additionally considered black-box transfer attacks. Here, we created adversarial images in a white-box setup on a surrogate model and transferred the resulting examples to the actual model under attack. In our evaluations based on MNIST, we used the LeNet5 and KerasExM as target and surrogate models in an alternating manner. Similarly, for CIFAR10 we used KerasExC and ResNet.

Since the implementation of the attacks is not part of our concept, we used the Foolbox framework [96] to craft adversarial examples for MNIST and CIFAR10. To create audio adversarial examples based on the Common Voice data set, we built upon the findings by Carlini and Wagner [97]. We used the code provided by the authors. Finally, for the IMDb data set we created an algorithm to produce adversarial examples, which we briefly describe in Appendix A.1.1.

3.2.5.4 Experiment Overview

Main Test Scenario We assume ourselves in the position of the trained model's maintainer and try to increase its protection against adversarial examples under the consideration of our main threat model shown in Section 3.2.3.1. Each step we present in the following was performed for all introduced attack methods. For the sake of simplicity we show each step only once.

First, we crafted adversarial images using one of the above stated methods. During this process, we payed attention to the way the data sets have been split beforehand. Consequently, we created two separate adversarial data sets, based on the train and test subsets. The samples in the test set simulate inputs fed to the target during an attack while being used in the field. This allows us to assess detectability of adversarial examples which are based on unseen benign inputs to rule out a detection bias. To form the adversarial data sets, we created (60 000, 10 000) adversarial examples for MNIST and (50 000, 10 000) adversarial examples for CIFAR10. We let the target model classify all samples in the resulting four data sets and stored the activation sequences accordingly. Each individual set contains features extracted during the classification of benign and adversarial samples while we preserved the division between test and training samples. This allowed a sound evaluation during the proof of concept.

In the second step, we used data sets containing the activation values based on the benign and adversarial samples to train the alarm model. Hence, for each target model and attack method, we created one specific alarm model. During testing, we let the alarm model classify all samples in the activation-value test data set.

To further show the generality of our concept we performed cross-testing experiments. We therefore evaluated the robustness of our detection approach against new and yet unseen attack methods. We tested one specific alarm model using the features based on a different attack. Consequently, we trained the alarm model with activation values triggered by one specific attack and detected adversarial examples created by another attack method. This setting simulates the scenario in which we encounter a new and yet unknown attack.

Furthermore, we created a combined alarm model. We trained this model using features triggered by a set of multiple attacks. Here, we verified if considering diverse information, based on a wider range of attacks, improves the alarm model's performance and provides a stronger detection capability.

Supplementary Experiments We divided our supplementary analysis into four experiments. This set of experiments aimed to further establish confidence in our approach.

In the first part, we used the previously created adversarial images for the MNIST data set and conducted transfer attacks. Here, we targeted the LSTM neural network and the capsule network. With this experiment, we investigated whether our approach can be applied in the context of different DL architectures or not. Both targets contain dense layers from which we extracted the activation values in order to train our alarm model.

The following two experiments were conducted using the regular target models classifying MNIST and CIFAR10 images. Both experiments tried to rule out a detection bias. With the first test we analyzed the behavior of our concept when the inputs consist of noisy images. Hence, we investigated the possible effect in which DLA may solely be able to distinguish between clean and perturbed images. We therefore answer the question whether adversarial examples are detected due to their impact on the attacked target or due to their nature of containing noise. For this purpose, we created noisy benign images with the same level of distortion compared to their adversarial counterparts. The created samples contain noise while being unaltered on a semantic level and are therefore still classified correctly. We calculated the distances between the original and adversarial images with respect to the used distance metric of the attack. The resulting data sets contain original, adversarial, and benign noisy images. To provide comparability, we preserved the distribution of benign and adversarial examples in this supplementary test set.

In addition to the previous experiment, in this setting we provide evidence for our initial hypothesis presented in Section 3.2.4: We argue that adversarial examples provoke a unique activation pattern in the dense layers which can be exploited to detect attacks. For this purpose, we analyzed the dense layer activation values of the target models during misclassification of original, benign inputs. We therefore extracted the according features and trained an alarm model to detect such incorrectly classified inputs. If our main hypothesis holds true, misclassified original inputs will not be easily detectable with DLA. The behavior of the target model should be similar during correct and incorrect classifications of benign inputs. Solely adversarial examples are assumed to provoke a distinct and detectable behavior of the targets visible in the activation space.

Finally, in the fourth part, we tested our concept in the context of two additional types of data sets. We investigated if we are able to detect adversarial examples in NLP and audio data sets. This test gives first evidence on the applicability in a wider range of use cases based on different DL-based systems. State-of-the-art defense methods mostly focus on image processing target models. Therefore, showing the applicability of our concept in additional types of data sets poses a significant step towards more robust defense methods. We introduce the test environment and results in a stand-alone paragraph in Section 3.2.6.3.

Note that we additionally performed and evaluated adaptive attacks. To emphasize the importance of this final evaluation, we describe the settings in more detail in Section 3.2.7.

Experiment Summary With the following list we summarize our performed experiments and provide an orientation for our subsequent evaluation. In summary, we performed the following experiments:

- Main proof of concept:
  - Detecting one specific attack method
  - Detecting unseen attack methods
  - Detecting multiple attack methods
- Supplementary experiments:
  - LSTM and capsule targets
  - Noisy inputs
  - Misclassified inputs
  - Detecting NLP and audio attacks
- Adaptive attacks

3.2.6 Evaluation

We split the evaluation of DLA's performance under the consideration of our main threat model into three parts. First, we discuss our analysis of the extracted features and show their distribution using a representative example. Secondly, we present the main results accumulated during our experiments shown in Section 3.2.5.4. This includes the performance of the different alarm models while detecting adversarial examples. Thirdly, we present the results of our supplementary experiments according to Section 3.2.5.4.

3.2.6.1 Dense-Layer Activation Analysis

As the extracted dense-layer activations are the core of our hypothesis and concept, we illustrate the major findings during our analysis. In Figure 3.5 we show neuron activation sequences for the LeNet5 target model and all five attack methods. For better visualization we reduced the dimensionality of the data using PCA and t-distributed stochastic neighbor embedding (t-SNE). The sub-figures show the neuron coverage of the dense layers during the classification of benign and adversarial images. Gray dots represent

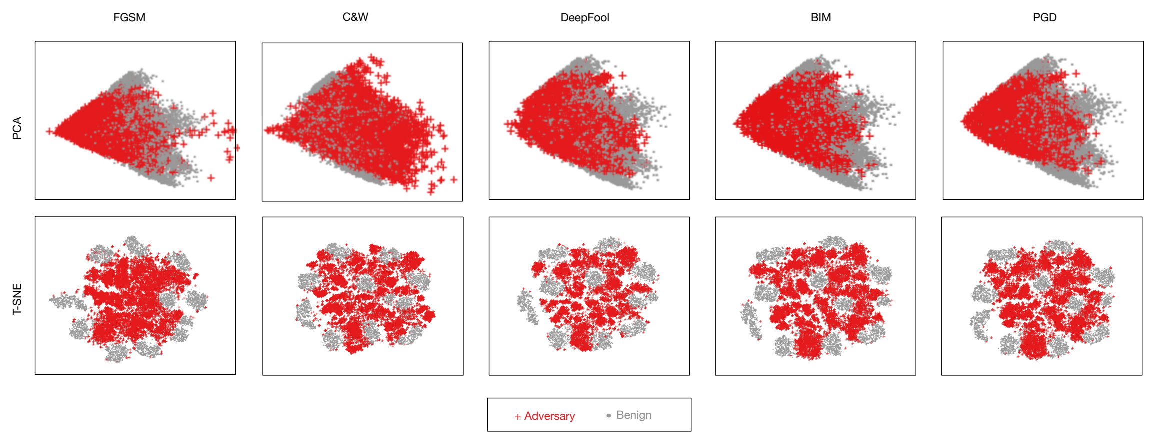


Figure 3.5: Visualization of the extracted dense-layer activations after dimensionality reduction using PCA and t-SNE.

benign instances and red crosses indicate adversarial ones. Each column shows the hidden activations for one attack method. We can clearly see a difference in the dense-layer activation patterns with respect to the nature of the inputs. Especially for the visualization using t-SNE. Here, we can see clusters indicating the ten classes of the MNIST data set. As expected, the adversarial examples fill the gaps between the class-based clusters. Figuratively speaking, we see the adversarial examples crossing the decisions boundaries of the attacked neural network. This finding gives first evidence on the verity of our initial hypothesis that the activations carry important information on the behavior of the target. Furthermore, we can provide a first estimation of the complexity and detectability of the individual attack methods. The PCA data points of the C&W-based activation sequences overlap to a higher extent than for the remaining methods. This suggests a more challenging detection of the adversarial examples confirming the current assumption of the attack being the most efficient white-box method.

With this initial experiment we argue that the activation values are indeed useful to detect attacks. Even this simple visualization might already be used to accomplish this task. Yet, since we provide an end-to-end framework to detect adversarial examples we directly use the raw extracted activations. We assume a higher level of generalization if we train our alarm neural networks to perform the detection.

3.2.6.2 Results of the Main Experiments

We quantify the detection capability of DLA using the F1 score. This is due to the fact that our alarm models contain output layers with two neurons. Hence, the decision of the alarm models does not depend on a detection threshold. Due to potential imbalances in the test data sets, we chose the F1 score instead of the standard accuracy as our main evaluation metric. Furthermore, we present the mean false positive and false negative rates. These rates can further be optimized according to the underlying use case. As we have mentioned in Section 3.2.5.2, by changing the alarm model's architecture to solely

Table 3.2: F1 scores of the individual DLA alarm models when trained and tested with the activations extracted using the corresponding attack method. Main results of the proof-of-concept experiments for DLA.

Data	**Target**	**F1 Score of the Alarm Models Tested with the According Attacks:**				
		FGSM	***C&W***	***DF***	***PGD***	***BIM***
MNIST	LeNet5	0.99	0.98	0.98	0.99	0.99
	KerasExM	0.99	0.98	0.98	0.99	0.99
CIFAR10	KerasExC	0.85	0.73	0.86	0.84	0.85
	ResNet	0.82	0.73	0.83	0.83	0.83

Table 3.3: Precision and recall of the individual DLA alarm models. Each alarm model is trained and tested with the features extracted for one attack method. Supporting the main results of the proof-of-concept evaluation.

Data	**Target**	**Precision and Recall of the Alarm Models Tested with the According Attacks: (prec.; rec.)**				
		FGSM	***C&W***	***DF***	***PGD***	***BIM***
MNIST	LeNet5	1.00; 0.98	0.98; 0.97	0.99; 0.97	1.00; 0.99	1.00; 1.00
	KerasExM	1.00; 0.99	0.98; 0.97	0.99; 0.98	1.00; 0.99	0.99; 0.99
CIFAR10	KerasExC	0.87; 0.82	0.72; 0.75	0.87; 0.84	0.85; 0.83	0.86; 0.85
	ResNet	0.86; 0.77	0.78; 0.68	0.89; 0.78	0.87; 0.80	0.88; 0.79

contain one neuron in the output layer, an adaptable threshold can be used to influence the sensitivity of the detection.

Detecting One Specific Attack Method In Table 3.2, we list the F1 scores of the individual alarm models when tested against their dedicated attack method. We see a near perfect detection capability for all attack methods and target models with the MNIST data set. The respective F1 scores range above 0.9. For the CIFAR10 data set we report competitive results as well. Here, our framework detects the majority of attacks, posing a viable solution for real-world applications. Due to the higher complexity of the CIFAR10 images, we report a lower mean performance compared to the MNIST experiments. In Table 3.3, we show the achieved precision and recall values. We report a balanced performance of DLA for the individual experiments. A more detailed evaluation of the error rates is shown below.

Detecting Unseen Attack Methods With our cross-testing experiments we show that our concept is capable of detecting new, unseen attacks. We trained the alarm model with features provoked by one attack method and tested it with features based on multiple attacks. In summary, for both data sets and for each target model we tested six alarm models. Each alarm model was tested against six attack methods simultaneously.

Table 3.4: F1 scores of the individual DLA alarm models when detecting all attack methods simultaneously. Each alarm model was trained with activations provoked by one attack and tested with a combined feature set including unseen attack methods.

Data	**Target**	**F1 Score of the Alarm Models Tested with all Attacks:**					
		FGSM	***C&W***	***DF***	***PGD***	***BIM***	***Transf.***
MNIST	LeNet5	0.94	0.96	0.96	0.92	0.92	0.96
	KerasExM	0.93	0.97	0.96	0.92	0.92	0.97
CIFAR10	KerasExC	0.78	0.66	0.79	0.78	0.79	0.74
	ResNet	0.76	0.72	0.78	0.77	0.77	0.57

Table 3.5: F1 scores of the combined DLA alarm models when tested against each attack separately and all attacks at once.

Data	**Target**	**F1 Score of the Combined Alarm Model Tested with Individual Attacks:**						
		FGSM	***C&W***	***DF***	***PGD***	***BIM***	***Transf.***	***Comb.***
MNIST	LeNet5	0.98	0.97	0.97	0.98	0.98	0.93	0.98
	KerasExM	0.98	0.97	0.98	0.98	0.98	0.95	0.98
CIFAR10	KerasExC	0.77	0.74	0.77	0.77	0.77	0.74	0.77
	ResNet	0.81	0.69	0.83	0.82	0.82	0.75	0.80

This includes the five attack methods as well as transferred adversarial examples. Five of the six alarm models we used are trained on the attack methods FGSM, C&W, DF, PGD, and BIM. The additional alarm models are trained using features extracted during transfer attacks. In Table 3.4 we summarize the performance of our approach during this experiment. As an example, for LeNet5 DLA scored an F1 score of 0.94 when trained with FGSM, yet tested against FGSM, C&W, DF, PGD, BIM, and transferred adversarial examples. With the results we report a successful detection of adversarial examples, for which the underlying method has not been known beforehand. Interestingly, the alarm models trained with transferred adversarial examples yield competitive results. Especially for MNIST, DLA reached an F1 score of 0.96 and 0.97 for the two target models. For ResNet the performance dropped significantly to 0.57 in this case. We therefore recommend using specifically crafted adversarial examples targeting the model one tries to protect.

Detecting Multiple Attack Methods To evaluate if training on a combination of features based on multiple attack methods improves the performance of our method, we created a combined alarm model for each data set. For each target model, the combined alarm model is trained with features extracted during the evaluation of all attack methods. Table 3.5 provides an overview of this experiment. The F1 scores show that the combined alarm models are able to detect all tested adversarial attack methods. Comparing the results to Table 3.2, we can report similar results with the combined models. As expected, the performance decreased slightly across the experiments. Due to the combination of attacks, the alarm models are optimized to generalize across a diverse set of training samples. The advantage of this settings is the fact, that solely one alarm model needs to be trained, reducing the computational overhead of DLA. In Table 3.5 we additionally show the F1 scores when detecting all attacks simultaneously. Hence, the right-most column shows the performance of DLA when trained and tested with a highly diverse feature set. Again, we report competitive results. Finally, regarding the trade-off between robustness against multiple attacks and the resulting detection capability we can recommend using a combined model during the application in the field.

Error Rates During the detection of adversarial examples it is important to evaluate the error rates which heavily influence the real-world applicability of the system. Therefore, we performed detection runs on multiple non-overlapping batches using our test data sets. For MNIST, the mean false positive and false negative rates are 0.01 and 0.02, respectively. Similarly, for CIFAR10 we report mean error rates of 0.14 and 0.21, respectively. For both settings, our method did not miss a disproportionate amount of adversarial examples. This is an important finding with regard to the applicability in a real-world setup. Still further use-case and data-set-depended optimizations are required. For the interested reader we show all individual results of this experiment including the mean error rates with the according standard deviations in Table A.1 (Appendix A.1.2).

3.2.6.3 Results of the Supplementary Experiments

LSTM and Capsule Targets During our tests with an LSTM and capsule target network, we were able to detect adversarial images based on the MNIST data set with F1 scores of 0.93 and 0.94, respectively. The positive results emphasize the applicability of our concept for a wide range of neural-network architectures using dense layers. Furthermore, with this experiment we make first steps towards applying our method to dense-layer-free target models. Our LSTM target model incorporates two dense layers after the LSTM unit. Even though the main learning power resides in the LSTM unit, our framework can leverage the information from the additional dense layers. This suggests that adding dense layers to dense-layer-free target models again enables the application of our concept.

Noisy Inputs In Table 3.6, we show the results of the tests containing noisy images. We performed the experiments for two target models and the FGSM attack method. The performance of the individual alarm models was decreased by 10% in the worst case. Even though our method still detected a fair amount of attacks, noise noticeably reduced the performance of the system. Still, we conclude that DLA does not solely detect perturbed inputs. This provides evidence that the activations allow an analysis of the observed target model's behavior.

Table 3.6: Performance of DLA when detecting adversarial examples among clean and noisy benign images.

Data	Target	Attack	F1 Score
MNIST	LeNet5	FGSM	0.90
CIFAR10	ResNet	FGSM	0.79

Misclassified Inputs In this experiment we evaluated if our approach allows detection of original but incorrectly classified inputs. We argue that adversarial examples provoke a distinct pattern in the activation values enabling detection. Hence, successfully training an alarm model which is designed to detect misclassified inputs would contradict our main hypothesis. During our experiments with the MNIST and CIFAR10 data sets, we were not able to train such an alarm model. This indicates that our intuition holds true. Contrary to adversarial examples, misclassified original inputs provoke a behavior of the target model similar to the scenario in which correctly classified inputs are handled. Therefore, DLA does not flag incorrectly classified benign samples as adversarial.

Detecting NLP and Audio Attacks With the following experiments, we evaluated the general applicability of DLA in different application domains. During this analysis, we crafted adversarial examples based on an NLP and audio data set. As previously introduced, we used the IMDb and Mozilla Common Voice data sets.

The process of generating adversarial examples for the two data sets is not part of the contribution of this thesis. Nevertheless, some basic notes are worth mentioning. With the IMDb data set, we used Algorithm 1 (Appendix A.1.1) to generate misclassified movie reviews. Instead of adding or deleting words, we chose to replace words in the individual instances. With this approach, we preserved the lengths of the classified sentences and reduced the distance between benign and adversarial examples. Here, after training DLA, we were able to reliable detect adversarial examples. DLA reached an F1 score of 0.97.

For the audio data set, we used Carlini and Wagner's approach to create adversarial examples [97]. Extracting the features of the audio files leads to neuron activation sequences of different lengths. This is the result of various sampling rates during the recording of the original audio samples in the data set. To be able to perform binary classifications using all feature instances, we used a different alarm model architecture here. The alarm model contains one LSTM unit followed by one output layer with two neurons to enable the detection of attacks. For this data set we were able to detect adversarial examples with an F1 score of 0.86.

3.2.7 Robustness Against Adaptive Attacks

In this section we evaluate adaptive white-box attacks in which the attacker has perfect knowledge of the target model and our detection method. For this purpose we assume the second threat model presented in Section 3.2.3.2. As shown by Carlini and Wagner [51], the majority of proposed detection methods can easily be bypassed by an adaptive attack. Therefore, this evaluation should play a major role during the presentation of new defense strategies. We emphasize the importance by showing the experimental setup and the according evaluation in this separate section.

3.2.7.1 Experimental Setup for Adaptive Attacks

To perform an adaptive attack against DLA, a few adaptations need to be introduced: First, we changed the alarm model architecture such that one single linear output neuron is used. Note that we did not optimize the detection threshold for this output neuron which might influence the robustness against adaptive attacks. Secondly, we combined our target and alarm models to one overall network. We use the following function $G(x)$ to represent this combination of classifier and detector:

$$G(x)_i = \begin{cases} Z_F(x)_i, & \text{if } i \leq N \\ A_D(x) \cdot \max_j Z_F(x)_j, & \text{if } i = N+1 \end{cases}$$

where $A_D(x) = (2(\max_j Z_F(x)_j > 0) - 1) \cdot Z_D(x) + 1$ decides if our secured model will output an alarm signal or not. The overall system now contains eleven output logits for which the first ten logits represent the classification output, while the last logit shows the output of the detector. This formula allows the first ten logits to be negative.

With $G(x)$ we generated adversarial examples for the complete system using the C&W method. Here, we used the code published together with the paper [98]. We performed this attack on our four main target models: LeNet5 and KerasExM for MNIST as well as KerasExC and ResNet for CIFAR10.

3.2.7.2 Evaluation of Adaptive Attacks

To evaluate the robustness of our method against adaptive attacks, we used the mean l_2-distance between adversarial and benign images to either fool the simple target model or the DLA-based system as well as the attack success rates as metrics. We generated adversarial images for both systems using the same attack parameters to preserve comparability. This allows an estimation of the persisting vulnerability to attacks after the addition of DLA. The parameters listed in Table A.2 (Appendix A.1.3) are chosen such that the attacks reached a 100% success rate when targeting the unprotected target models building the baseline protection level. In Table 3.7 we summarize our results. The four secured target models required a significantly higher l_2-distortion compared to their unsecured counterparts.

For the MNIST-based LeNet5 target model, our defense method more than doubled the required mean l_2-distance compared to the unsecured models. Furthermore, we reduced the attack success rate to 44.6%. Similarly, for KerasExM we increased the mean l_2-distance to fool our secured model by 80%. We report an attack success rate of 97.6%.

For CIFAR10 and the KerasExC target model we achieved similar results. When attacking the simple target model, the mean l_2-distance is 0.43. In contrast to that, when

Table 3.7: Results of the adaptive attacks on DLA.

Data	**Target**	**Attack Success Rate**		**Mean l_2 Distortion**	
		unsecured	***secured***	***unsecured***	***secured***
MNIST	LeNet5	100.0%	44.6%	2.17	4.42
	KerasExM	100.0%	97.6%	1.51	2.72
CIFAR10	KerasExC	100.0%	53.3%	0.43	0.85
	ResNet	100.0%	99.7%	0.13	0.38

attacking DLA, the adversarial images show a mean l_2-distance of 0.85 with respect to their benign counterparts. Moreover, only 53.3% of adversarial examples of the adaptive attack were successful. Concerning our ResNet target model we are able to report a minor improvement of its protection. This is due to its complexity and size of the resulting feature space. As mentioned above, we used the same alarm model architecture for all test scenarios to show the simplicity of our approach. In this case, a bigger alarm model may be required to cope with the extracted features more efficiently.

With our numerical study we show the robustness of our concept against adaptive attacks. To support our conclusion, we show the adversarial examples which are able to fool our secured KerasExM and KerasExC target models in Figure 3.6. For both data sets we show the original images in the first and the adversarial counterparts in the second row. We report a significant difference between benign and adversarial images. The perturbations required to fool our secured systems are clearly visible which enables a human expert to identify the adversarial examples. This becomes even more clear when comparing the images to the previously successful adversarial examples in Figure 3.4. With respect to our introduced threat models, we emphasize that the resulting adversarial attacks may not be considered successful. In our threat model we restricted the capabilities of the attacker to mutate the inputs of our target models, such that the changes are not easily visible to a human expert. This is not fulfilled here. Hence, we can report a significant improvement in our target models' protection, even during white-box adaptive attacks.

3.2.8 Discussion

With the in-depth experiments in this section we show the importance of the dense layers analysis in future NN defense strategies. In Section 3.2.6, we sum up the most important results to underline our conclusion. Nonetheless, some aspects regarding detection performance, transferability, and real-world applications, as well as a comparison to related work require further discussion.

First, we want to discuss the trade-off between false positive and false negative errors during detection. Without optimizing the detection threshold of the alarm models we report the false negative rate to be higher than the false positive rate. Hence, we recommend further use-case specific adaptations before deploying our method in security-

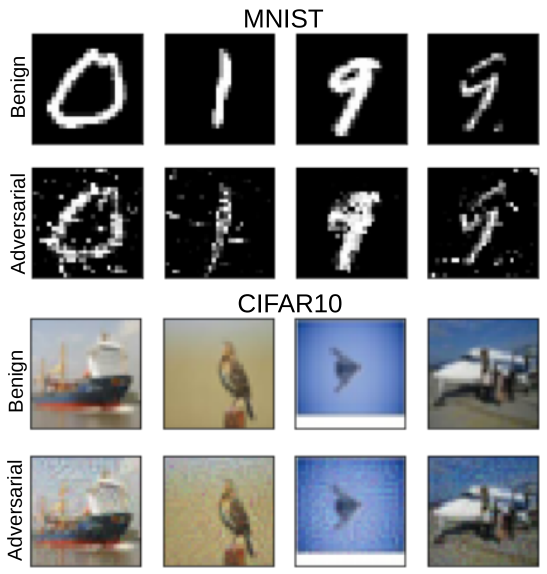


Figure 3.6: C&W-based adversarial examples during adaptive attacks on DLA. The first two rows show MNIST-based examples for the KerasExM target model. The lower two rows are based on CIFAR10 and KerasExC.

sensitive setups. Here, the base-rate fallacy is worth mentioning. The number of false alarms is often a crucial performance characteristic of attack detection systems [99] and should not be neglected when aiming to reduce the false negative rate.

Throughout our cross-testing experiments, we evaluated the generality of our method. We give evidence for the distinct behavior of NNs when confronted with adversarial examples, independently of the used attack method. This allows two main conclusions: Future, yet unknown attacks which follow similar approaches as current ones seem detectable using our concept. Furthermore, it allows a ranking of attack methods. This ranking is based on two findings. First, the difficulty in detecting each attack, expressed by the F1 score of the respective alarm model. Secondly, by the performance of the alarm model created for the attack itself when detecting examples crafted with other attack methods. As an example, if we focus on the C&W attack on the ResNet target model: We can clearly see that this attack method can only be detected by the alarm model specifically created for this purpose. Hence, we deduce the C&W attack being the most powerful method used here. This correlates with current findings in the field of adversarial attacks and defense strategies. As discussed in Section 3.2.2, the C&W attack is currently considered to be the most powerful white-box attack.

A possible restriction our approach may suffer from is the architecture of the model to protect. One could argue that we are not able to detect attacks if the target model does not incorporate dense layers. This can be ruled out considering the following three concepts: The maintainer of the target model creates a substitute model which performs the same task as the target model itself, achieving a similar accuracy. If this substitute model uses dense layers, we are again able to apply our concept. The positive results during our experiments regarding transfer attacks indicate the practicality of this idea. Alternatively, additional dense layers may be added to the original target model. Our experiments with LSTM target models indicate the practicability of this idea as well. The target model contains dense layers after the LSTM unit in which the main learning power resides. A thorough evaluation of both solutions is out of scope in the context of this thesis. The third approach to allow the application of DLA together with a wider range of target model architectures would be the analysis of all activation values. Instead of specifically selecting the dense layers from which the activations are drawn, training an alarm model with the activation values of all hidden layers may lead to similar results as shown in this section. We build upon this idea in Section 3.4.

Finally, we want to emphasize the simplicity of our approach. During our research on related work, we noticed the defense strategies to be rather counterintuitive and having several sources of errors when not applied correctly. Our method, in contrast, is easy to use and seems intuitively reasonable. In addition, our method does not decrease the accuracy of the model to protect when tested against benign images, which is the case for some state-of-the-art defense strategies. One important aspect when comparing our method to related techniques is worth mentioning: We did not optimize our framework to its full detection capability. Note that we solely used one alarm model architecture during our main experiments on MNIST and CIFAR10. Furthermore, we did not tune the training process of the alarm models in relation to each target model. We regard this as out of scope for this study. Moreover, it shows the simplicity and generality of our approach. Even with a generic setting, promising and highly competitive results were achieved. If the user further tunes the settings in a use-case-specific manner, superior results are achievable. The drawback of this approach is the limited possibility of comparing our method directly to related work. As related detection schemes are often optimally adapted to the underlying use case and data set, a direct comparison would not lead to meaningful conclusions. Therefore, we solely compared our method to state-of-the-art techniques on a conceptual level.

3.2.9 Conclusion

In this section, we introduce a general end-to-end framework to detect adversarial examples during classification time called DLA. Our approach consists of two phases.

First, in the training phase we observe the dense-layer activation patterns of the model to protect. For this purpose, we extract the neuron coverage of the target model to directly train a secondary NN we call alarm model. This alarm model distinguishes between activations sequences extracted during the classification of benign and adversarial inputs. This approach is motivated by our main hypothesis that the dense layers of the target model carry robustness-sensitive information. Thus, the alarm model is trained to detect malicious activity patterns triggered by adversarial examples during classification time.

In the second phase, the target model runs in secure operation mode, which is enabled by enhancing it with our trained alarm model. When the target model classifies new, unseen inputs, the alarm model runs in parallel and produces an alarm if an adversarial example is being processed by the target. This approach leaves all parameters — especially the accuracy — of the target model untouched, while improving overall application robustness significantly. In our proof-of-concept implementation, we show the extensive capability of our approach to detect adversarial examples in image, NLP, and audio data sets. The evaluation results strongly indicate that we can not only defend with high accuracy against state-of-the-art adversarial examples, but also against future, yet unknown attacks which follow similar approaches as current ones. Finally, with adaptive attacks we show that an attacker needs to induce significantly more adversarial perturbations to attack our detection-enhanced system, compared to attacking the unsecured target model.

With the insights of this section at hand, two open questions arise which we answer throughout the remainder of this thesis: **Q1**: *Is our presented target-alarm structure a generally applicable architecture to increase the level of protection of NNs against adversarial examples?* We have seen that our architecture allows the detection of attacks based on the analysis of the activations of the NNs. In the following, we present a method to protect NNs even if the activation values are not available by leveraging a different source of information.

Q2: *How can the insight that the activation values of NNs carry robustness-sensitive information be leveraged and further be used in the context of robustness and security in general?* We have seen that the activations values can be used to detect attacks in a hostile environment where attackers are present. In the following, we leave this environment and deploy our method in the realm of anomaly detection.

Q1 is answered in Section 3.3.

Q2 is answered in Section 3.4.

3.3 DA3G: Detecting Adversarial Attacks by Analyzing Gradients

In this section, we present an additional application mode of our previously introduced target-alarm architecture. Parts of the section are taken verbatim from the original paper "DA3G: Detecting Adversarial Attacks by Analysing[1] Gradients" published at the 26th European Symposium on Research in Computer Security (ESORICS) 2021 [2]. This is collaborative work with the co-first author Jan-Philipp Schulze.

3.3.1 Motivation

In Section 3.2 we introduce our adversarial example detection method DLA. We show that by using our modular structure containing the model to protect (target model) and an auxiliary detection model (alarm model) we are able to successfully detect evasion attacks. The main source of information on the current behavior of the target model are its activation values, automatically analyzed by the alarm model.

In this section, we analyze whether our modular approach can be applied even though the activation values of the target model are not available. Specifically, we consider **Q1** which arose in the previous section: Is our target-alarm structure a generally applicable architecture to increase the level of protection of NNs against adversarial examples? To answer this question, we use DLA's architecture, yet provide it with an alternative source of information. Instead of analyzing the dense-layer activations, we use the gradient of the target models caused by the input samples. With our evaluation we show that our architecture can indeed be used as a powerful detection approach, either with the dense-layer activations or the gradient of the target model. Based on this change of architecture, we call the gradient-driven DLA-based adversarial example detection method DA3G: detecting adversarial attacks by analyzing gradients. In summary, in this section we present the following contributions:

- We show that our target-alarm architecture is useful to detect adversarial examples even when the target's activations are not available.
- We compare the performance of the gradient-driven DLA version to the original DLA approach and another state-of-the-art detection method.
- We thoroughly investigate adaptive white-box attacks and find a significant increase of protection compared to unprotected models.

3.3.2 Related Work

With DA3G, we show that our target-alarm structure is a versatile tool allowing a reliable adversarial example detection. This holds true even if we replace the analysis of

[1]Note that the original paper was written and published in British English. Yet the contributions in this thesis are presented in American English.

the activations by observing the target's gradient. In the following, we introduce most related recent findings in this area in more detail.

ML-LOO is currently considered the most accurate detection method presented by Yang et al. [100]. The authors build upon methods from the field of explainable AI, namely leave-one-out (LOO). Hence, ML-LOO is tested for the image processing domain. The importance of each input pixel is quantified by erasing it and measuring the change in the output prediction of the target model. ML-LOO uses the interquartile range of the LOO distribution as decision variable. The authors discovered that benign inputs have a significantly narrower LOO distribution. This translates to the observation that for benign inputs only a few input pixels are important for the output.

Closely related to our approach of using the gradient within our target-alarm architecture are the methods Gradient Similarity by Dhaliwal and Shintre [101] and GraN by Lust and Condurache [102]. Both detection methods use the gradient of a target model to calculate a set of metrics which are used as input to a logistic regression classifier. This classifier then acts as the adversarial example detection instance. In GraN, the authors calculate the layer-wise l_1-norm. The authors of Gradient Similarity use the l_2-norm enriched by the cosine similarity to certain training samples. Both sets of metrics are valuable sources of information to detect known and unknown adversarial attacks which the authors show with their evaluations. Opposed to Gradient Similarity and GraN, with DA3G we directly use the raw gradient of the target model instead of performing a manual feature extracting step. By directly using the raw gradients without the calculation of predefined metrics, our method uses the entire information available. Thanks to our target-alarm architecture, DA3G is more flexible and scalable as it readily integrates with complex NN architectures. Furthermore, in contrast to the aforementioned research, we consider an adaptive-attack threat model throughout our evaluation.

Finally, the most related method to DA3G is DLA introduced in Section 3.2. We build upon the target-alarm structure and instead of using the activation values of the target we analyze its gradient. To this end, we show that our approach works for a wide range of data sets and target models even if the activation values are not available. Similarly to DLA, our evaluation of adaptive white-box attacks shows that DA3G increases the required resources of attackers to successfully fool the protected models.

3.3.3 Considered Threat Models

Based on our general introduction to threat models in adversarial machine learning presented in Section 2.2.1, in this section we summarize the considered capabilities of the attackers for the introduction and evaluation of DA3G.

3.3.3.1 Main Threat Model: Gray-Box Attacks

During gray-box attacks, adversaries have access to the parameters of the NNs and hence are capable of using the model's gradients. In our main threat model, the attackers alter the classification output of the attacked NNs in an untargeted manner. The produced adversarial examples are bounded by the l_2 and l_∞ norm depending on the chosen attack method. Here, the attackers are not aware of the defense method, however, they have unrestricted access to the NN under attack, i.e., our target model. We use untargeted attacks as these pose the less challenging task for the attacker: The target model suggesting any other class than the original one is considered as successful attack. This setting evaluates DA3G's detection performance prior to adaptive attacks and allows comparison to our later introduced baseline methods.

3.3.3.2 Adaptive-Attack Threat Model: White-Box Attacks

In white-box settings, the adversaries are aware of all parameters of the system containing the NN under attack and the used countermeasures. Thus, attackers aim to bypass the defense while steering the decision to their desire. These adaptive attacks are known to circumvent many past defense methods [14, 51, 72] and are therefore a main evaluation point for DA3G. We use l_2 and l_∞ bounded attacks and consider targeted attacks using the least-likely target class [56] similar to Carlini et al. in [51].

3.3.4 Detecting Adversarial Examples using Gradient-driven DLA

In Section 3.2, we have shown that this discrepancy between the adversarial input which is close to a benign class, and its output which the attacker shifted to a wrong class, is measurable in the dense-layer activations. Using our DLA architecture we are able to leverage this information. Complementary, in this section we argue that our architecture provides means to detect adversarial examples even if the activations are not available. In this section, we exemplary use the gradient of the attacked NN. Again, instead of manually defining features we leverage DLA's architecture and directly analyze the available information in the gradient using a secondary NN, the alarm model. During training, the true labels of the inputs are available which directly allows the calculation of the gradients. This does not hold true for the testing phase. Here, the labels of the inputs are not known. The gradients in this phase are therefore calculated with respect to the output neuron with the highest activation.

3.3.4.1 DA3G Architecture

As we directly use our DLA architecture, DA3G consists of two NNs: the pretrained target model f_t and the alarm model f_a. The alarm model analyzes the target model's gradient caused by new yet unseen inputs. As output, it returns a score which increases with the probability of the input being adversarial. During supervised training, the alarm model learns to generalize the gradient of the known benign and adversarial samples to yet unseen inputs. We give an overview of our architecture in Figure 3.7. Note that solely the information provided to the alarm model differs from our visualization of DLA in Figure 3.2.

Previous work by Dhaliwal and Shintre [101] and Lust and Condurache [102] has shown that there are measurable differences in the gradient for benign and adversarial inputs. In this section, we use this insight to test if our modular architecture provides

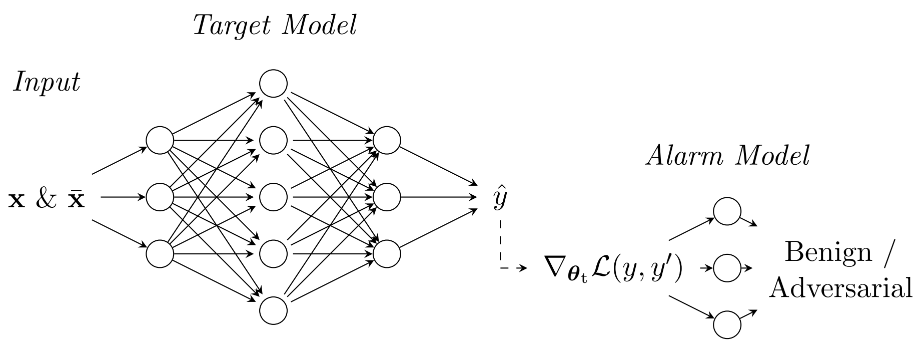

Target Model
Input
$\mathbf{x}$ & $\bar{\mathbf{x}}$
$\hat{y}$
Alarm Model
$\nabla_{\boldsymbol{\theta}_t}\mathcal{L}(y, y')$
Benign / Adversarial

Figure 3.7: Summary of the components of the adapted DLA architecture using gradients as inputs, here called DA3G, consisting of a target and an alarm model.

means to detect adversarial examples even if the activation values are not available. Therefore, in the following we give a brief introduction to gradients and how we use them within our architecture. An NN's gradient is based on its loss function $\mathcal{L}(y, y')$, which allows the quantification of the discrepancy between the current output y to the expected output y'. During inference in which DA3G protects the target model, y' is not known. Instead, we use the estimated output $\hat{y} = \hat{f}(\mathbf{x}; \boldsymbol{\theta})$, which is always available for the pretrained target model. In other words, we compare the predicted output to the most likely output class.

3.3.4.2 DA3G's Alarm Model Training

The training procedure is closely related to the one of DLA. In Figure 3.8 we visualize the required steps to first train the target and alarm models to finally allow the adversarial example detection. With DA3G, we protect a target model which was pretrained using the benign training data set. In the initialization phase, the alarm model analyzes the target's gradient calculated using benign and specifically crafted adversarial inputs. Using the gradient, the alarm model is trained in a supervised manner finally being able to distinguish between benign and adversarial inputs processed by the target model. In the secure operation phase, the target performs standard classifications while the alarm model observers the target's gradient and produces an alarm once attacks are detected.

3.3.5 Experimental Setup

In the following, we describe the settings, which we carefully evaluated DA3G in. This serves as basis for our evaluation in Section 3.3.6.

3.3.5.1 Data Sets

For easier comparison to prior work, we chose the publicly available and commonly applied [47] data sets MNIST [83], Fashion-MNIST [103], and CIFAR10 [84]. All three data sets contain images, differing in complexity and size. Fashion-MNIST is a drop-in

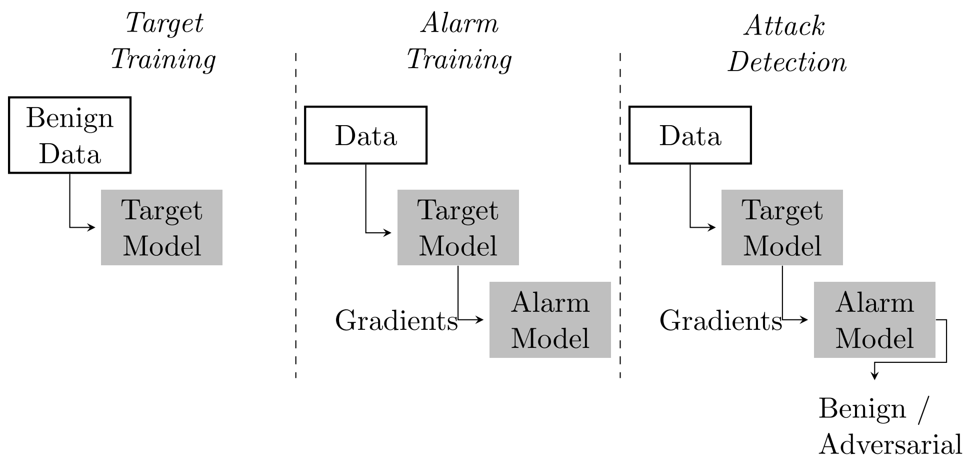


Figure 3.8: Step-by-step overview of DA3G's training process in the first two columns and the final attack detection in the right column.

replacement for MNIST and shows more complex items like clothes and shoes. The data sets comprise a predefined train and test split, which we used throughout our evaluation. During the training of DA3G, we introduced a validation set for which we drew 20% from the training data. As preprocessing step, we scaled input data to the range $[0, 1]$.

3.3.5.2 Architectural Choices

Target Models As target models, we evaluated a set of popular NN architectures, each serving as classifier for the respective data set. For MNIST and Fashion-MNIST, we chose the well established and tested LeNet5 architecture together with an example model provided by Keras which we call KerasExM. Similarly, we trained a ResNet model and again a Keras example architecture for CIFAR10, called KerasExC. We give an in-depth overview of the respective architectures and parameters in Table 3.8. During our experiments with DLA, we used similar target architectures. We therefore use the same model names. Note that we did not use the identical models and used newly trained ones throughout this section.

Alarm Models To show the general applicability of DA3G, we used one common architecture for the alarm model and all three data sets. Across all experiments, we chose a simple, fully connected, and SELU-activated [104] NN with the following layer dimensions: 100, 50, 10. We trained the alarm model for 500 epochs using Adam with a learning rate of $5 \cdot 10^{-5}$. The detection decision of the alarm model is produced by one single linear output neuron which can be controlled by setting a respective threshold. In order to lower the resource demands of DA3G especially during the training process of the alarm models, we limited our analysis to the gradient based on the last two target model layers. We did not perform an evaluation, which layers result in the best detection

Table 3.8: Target models we used during the proof-of-concept implementation for DA3G. We give a short summary of each model's architecture as well as the used training settings and resulting train and test accuracy.

Data	Model	Architecture	Settings
(Fashion-) MNIST	LeNet5 [87]	– 2 convolutional layers with filter sizes 6 and 16 – each convolutional layer is followed by a average-pooling layer – finally, 1 flatten layer and 3 dense layers with 120, 84, and 10 neurons each	– learning rate: 0.001 – epochs = 20 – batch size: 128 – test accuracy MNIST: 98.9% – test accuracy F-MNIST: 90.8%
	KerasExM [88]	– 2 convolutional layers with filter sizes 32 and 64 – each convolutional layer is followed by 1 max-pooling layer – finally, a flatten layer, a 0.5 dropout layer, and a dense layers 10 neurons	– learning rate: 0.001 – epochs: 12 – batch size: 128 – test accuracy MNIST: 99.3% – test accuracy F-MNIST: 90.8%
CIFAR10	KerasExC [90]	– 4 convolutional layers, the first pair with filter size 32, the second pair with filter size 64 – max-pooling after each pair – 0.25 and 0.5 dropout, flatten layer, and 2 dense layers with 512 and 10 neurons	– learning rate: 0.0005 – epochs: 100 – batch size: 32 – test accuracy: 85.0%
	ResNet [89]	– ResNet20, version 1 – for further details: [89]	– epochs: 200 – batch size: 32 – test accuracy: 91.8%

performance, but expect the last two layers to influence the output the most. Future work might investigate this setting in more detail.

3.3.5.3 Attack Methods

In accordance with common guidelines presented by Carlini et al. [14], we evaluated DA3G using three attack methods of different types: PGD [19], DeepFool (DF) [20], and the C&W [22] attack. Because of similarities among attacks of the same type, our evaluation contains a sufficiently diverse set of methods. We profited from the attack categorization introduced by Zhang et al. [47]. Here, the authors divided the attacks into gradient-based, decision-based, and optimization-based methods. PGD represents gradient-based multi-step attacks, DF decision-based, and C&W optimization-based approaches. For PGD and DF, we considered both the l_2 and l_∞ versions of the attacks, while concentrating on the l_2 version of the C&W attack. DF and C&W generate adversarial examples in an unbounded manner. Hence, the attacks try to find minimally altered samples fooling the attacked NN within the given perturbation budget ϵ. PGD, on the other hand, is a bounded method and therefore generates adversarial examples $\tilde{\mathbf{x}}$ with the property $\|\tilde{\mathbf{x}} - \mathbf{x}\| \approx \epsilon$. For MNIST and Fashion-MNIST, we set ϵ to 0.3 and 4.0 for the l_2 and l_∞ attacks, respectively. Similarly, for CIFAR10 we set ϵ to 0.03 and 0.9. For all attacks, we selected the parameters such that a 100% success rate is reached when attacking the unprotected targets.

3.3.5.4 Baseline Methods

During our experiments, we compared the performance of DA3G to our adversarial example detection method DLA, and to the state-of-the-art method ML-LOO introduced by Yang et al. [100].

We selected ML-LOO based on the thorough evaluation presented in the survey on adversarial example detection by Zhang et al. [47]. In this survey, the authors compared five different methods using three data sets. The authors concluded that ML-LOO provided the best detection capabilities, outperforming the remaining four methods in the majority of the evaluated test cases. A drawback of ML-LOO is its required computation time during training and inference. To mitigate this problem, we concentrated on the last two layers of each target model as source of information as also done in DA3G. We trained ML-LOO's logistic regression models for 1000 iterations each.

As DA3G uses the target-alarm architecture we introduced in Section 3.2, we slightly adapted DLA and added it to this experiment pipeline. By comparing DA3G to DLA, we can directly assess the impact of using the gradients instead of the dense-layer activations as source of information. Therefore, we are able to investigate and argue the general applicability of our modular target-alarm structure consisting of NNs only. In our experiments, we used the same alarm models for DLA and DA3G. We trained DLA's alarm models for 50 epochs each, again using Adam with a learning rate of $5 \cdot 10^{-5}$. In contrast to the original implementation of DLA's alarm models containing two output neurons, in this section we used single output-neuron models. This allows a direct comparison of our two detection methods without the need to individually set detection thresholds or translating results to a common base.

3.3.5.5 Experiment Overview

We carefully designed a set of experiments to evaluate the performance of DA3G. Each one considers a different attack surface, where the attacker's abilities and knowledge increase. If not stated otherwise, we used 40 000 adversarial examples to train DA3G and the two baseline methods. In our evaluation, we considered the following four experiments. For the first three settings, we assumed the gray-box threat model (Section 3.3.3.1), while we used the white-box threat model (Section 3.3.3.2) for the final experiment.

1. *Basic Gray-Box Detection.* Detection of the same attack type that DA3G was trained on. We compared DA3G's performance against ML-LOO and DLA as examples of state-of-the-art adversarial detection methods.

2. *Combined Gray-Box Detection.* Detection of multiple attack types that DA3G was trained on. Additionally, we evaluated the influence of the number of known adversarial examples to simulate environments, where it is infeasible to generate a large amount of training samples.

3. *Leave-One-Out Gray-Box Detection.* Detection of other kinds of attack methods than the ones known during training. This measures the protection against yet unknown attacks.

4. *Adaptive White-Box Attacks.* Protection against adversaries that are aware of DA3G and adapt their attack accordingly. This simulates a setting with an omniscient attacker, who knows the countermeasures used.

Adaptive Attacks We based our adaptive attacks on the findings of Carlini et al. [51] and on our attack strategy against DLA which we presented in Section 3.2.7.2. Let $Z_\mathrm{t}(\cdot)$ and $Z_\mathrm{a}(\cdot)$ be the output logits of the target and alarm models, respectively. And let N be the number of output classes, then an adversary attacks the combined function $G(\cdot)$:

$$G(\mathbf{x})_i = \begin{cases} Z_\mathrm{t}(\mathbf{x})_i, & \text{if } i \leq N \\ (Z_\mathrm{a}(\mathbf{x}) + 1) \cdot \max_j Z_\mathrm{t}(\mathbf{x})_j, & \text{if } i = N + 1 \end{cases}$$

We distinguish between inputs considered benign by the alarm network, i.e., $Z_\mathrm{a}(\mathbf{x}) < 0$, and malicious inputs, i.e., $Z_\mathrm{a}(\mathbf{x}) > 0$. In the respective cases, DA3G's decision becomes:

$$\operatorname*{argmax}_i(G(\mathbf{x})_i) = \begin{cases} \operatorname*{argmax}_i(Z_\mathrm{t}(\mathbf{x})_i), & Z_\mathrm{a}(\mathbf{x}) < 0 \\ N + 1, & Z_\mathrm{a}(\mathbf{x}) > 0 \end{cases}$$

The argmax$(\cdot)$ function returns the argument of the maximum within a given domain. In contrast to our gray-box experiments, here a threshold-independent evaluation is not possible. Using the formulae introduced above to describe the combination of target and alarm model results in a threshold of value 0. Naturally, the global detection threshold of 0 may not be the optimal decision boundary for DA3G. Similarly to our evaluation of DLA against adaptive attacks, a fine-tuned detection threshold when applying DA3G in the field may increase its robustness. As we performed targeted white-box attacks, we omitted DF and evaluated the robustness of DA3G against PGD and the C&W attack. The alarm models were trained on 40 000 adversarial examples generated with C&W and both versions of PGD simultaneously using a combined training set.

3.3.6 Evaluation

In the following, we present the detection performance of DA3G. For the first three experiments, we show the mean area under the Receiver Operating Characteristic (ROC) curve for three runs of each experiments. The ROC-curve visualizes the true positive and false positive rates for a given detector and all possible thresholds. Therefore, the area under the ROC-curve (AUC) measures the performance independently of an output threshold, thus allows the general detection capabilities to be judged. For the adaptive attacks using PGD, we visualize the attack success as a function of the the attack steps. In the case of adaptive adversaries using C&W, we visualize the required perturbation budget. We introduced both metrics in relation to the analyzed attack method in Section 2.2.4.

3.3.6.1 Basic Gray-Box Detection

We started our evaluation with the simplest and most widely considered scenario, the detection of known attack types. Here, we show that the raw gradient can indeed be used as main source of information fed to our target-alarm system. Furthermore, we compare DA3G to our method, DLA, and to another state-of-the-art approach, ML-LOO. The results are summarized in Table 3.9. We report near perfect results on all experiments except for PGD on MNIST. Indeed, for DF and C&W, DA3G was the best performing method across all three data sets beating both baseline methods. The performance of DA3G was especially striking on F-MNIST, where our method was the only one achieving a 100% detection score for all three attack types.

Motivated by these results, we further analyzed why DA3G's performance was lower for PGD on MNIST. Looking at the gradients, we saw nearly vanishing values when the PGD-generated adversarial examples were processed by the targets. This was in stark contrast to the adversarial examples generated with DF and C&W, where the gradient values showed more variance and higher magnitudes in general. We believe this phenomenon is similar to the findings of Tramer et al. [72], who showed that PGD allows the generation of adversarial examples with high confidence when fooling the attacked models. DA3G uses the loss between the output probabilities and the maximum output — hence, the loss and thus the gradients vanish for outputs of high confidence, i.e. close to the arguments of the maxima. MNIST promotes high confidence outputs due to its low complexity and the fact that the classes are already well separated. Each individual sample of MNIST is well preprocessed unlike real-world inputs. As we can see in the results, more realistic and thus complex data sets like F-MNIST and CIFAR10 do not show the same performance drop for PGD. We conclude that the low detection rate for PGD is likely inherent to MNIST and not a general problem of DA3G.

The results of our first experiment indicate the validity of our core assumption: Indeed, the raw gradient can be used as main source of information fed to our target-alarm system to distinguish between benign inputs and adversarial examples. We observed a high detection performance across all three attack types especially for more complex data sets. With the results in this section we already see a high potential for our target-alarm

Table 3.9: Results for experiment 1 testing DA3G: mean AUC with the according standard deviation after three runs for the basic gray-box detection. We highlighted the best result for each experiment.

Data	Target	Method	$\mathbf{PGD}_\infty$	$\mathbf{PGD}_2$	$\mathbf{DF}_\infty$	$\mathbf{DF}_2$	$\mathbf{C\&W}_2$
MNIST	LeNet5	DA3G	$.94 \pm 0.00$	0.93 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
		ML-LOO	1.00 ± 0.00	1.00 ± 0.00	0.98 ± 0.00	0.91 ± 0.00	0.91 ± 0.00
		DLA	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
	KerasExM	DA3G	0.94 ± 0.00	0.93 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
		ML-LOO	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	0.99 ± 0.00	0.99 ± 0.00
		DLA	0.99 ± 0.00	0.98 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
F-MNIST	LeNet5	DA3G	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
		ML-LOO	1.00 ± 0.00	0.99 ± 0.00	0.86 ± 0.00	0.82 ± 0.00	0.82 ± 0.00
		DLA	1.00 ± 0.00	1.00 ± 0.00	0.99 ± 0.00	0.99 ± 0.00	0.99 ± 0.00
	KerasExM	DA3G	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
		ML-LOO	1.00 ± 0.00	1.00 ± 0.00	0.95 ± 0.00	0.93 ± 0.00	0.94 ± 0.00
		DLA	10.0 ± 0.00	1.00 ± 0.00	0.99 ± 0.00	0.99 ± 0.00	0.99 ± 0.00
CIFAR10	KerasExC	DA3G	0.98 ± 0.00	0.97 ± 0.00	0.98 ± 0.00	0.98 ± 0.00	0.98 ± 0.00
		ML-LOO	1.00 ± 0.00	1.00 ± 0.00	0.81 ± 0.00	0.81 ± 0.00	0.82 ± 0.00
		DLA	1.00 ± 0.00	1.00 ± 0.00	0.97 ± 0.00	0.97 ± 0.00	0.97 ± 0.00
	ResNet	DA3G	0.99 ± 0.00	0.99 ± 0.00	0.99 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
		ML-LOO	1.00 ± 0.00	1.00 ± 0.00	0.74 ± 0.00	0.73 ± 0.00	0.78 ± 0.00
		DLA	1.00 ± 0.00	1.00 ± 0.00	0.98 ± 0.00	0.98 ± 0.00	0.98 ± 0.00

structure even when using other sources of information than the dense-layer activations. Without model-specific adaptations to our initial architecture, replacing the dense-layer activations with the available gradient information allows a comparable detection of adversarial examples.

3.3.6.2 Combined Gray-Box Detection

Motivated by our first experiment, we increased the difficulty during the detection of attacks. With our second experiment, we show the performance of DA3G when confronted with all three attack types at once. We randomly sampled adversarial examples generated by the different attack methods to train our alarm models. Additionally, to further investigate the real-world applicability, we show the detection performance as function of the number of adversarial training samples. For large data sets, it may be infeasible to generate large amounts of adversarial examples under limited hardware resources. We summarize the results in Table 3.10.

DA3G followed an intuitive behavior: the more adversarial examples were available during training, the better the detection performance. Even for 1000 adversarial examples we saw promising results. We report that DA3G's detection performance remained strong when trained on multiple attacks at once. Indeed, we saw near perfect results of

Table 3.10: Results for experiment 2 testing DA3G: mean AUC with the according standard deviation after three runs for the combined gray-box detection using different amounts of adversarial examples during training.

Data	Target	100	1000	10000	40000
MNIST	LeNet5	0.65 ± 0.01	0.84 ± 0.01	0.94 ± 0.00	0.96 ± 0.00
	KerasExM	0.56 ± 0.04	0.89 ± 0.01	0.98 ± 0.00	0.99 ± 0.00
F-MNIST	F-LeNet5	0.62 ± 0.01	0.87 ± 0.01	0.98 ± 0.00	0.99 ± 0.00
	F-KerasExM	0.72 ± 0.02	0.87 ± 0.00	0.98 ± 0.00	0.99 ± 0.00
CIFAR10	KerasExC	0.64 ± 0.01	0.79 ± 0.01	0.89 ± 0.00	0.93 ± 0.00
	ResNet	0.67 ± 0.01	0.87 ± 0.00	0.97 ± 0.00	0.98 ± 0.00

more than 98% for the more complex target networks across all data sets. This experiment shows that DA3G reliably detects known adversarial attacks of multiple types.

3.3.6.3 Leave-One-Out Gray-Box Detection

In this experiment, we expanded our evaluation to the detection of yet unknown attacks. We evaluated DA3G in multiple leave-one-out settings. Here, DA3G was trained on two attack types and subsequently evaluated on adversarial examples produced by the left-out type. This simulates the real-world scenario in which attackers leverage new attack strategies. In Table 3.11, we summarize the leave-one-out experiments for all possible combinations.

Again, we report near perfect results for the majority of combinations. As seen in our first experiment, DA3G performed especially well when detecting the decision- and optimization-based attacks DF and C&W. Indeed, the only combination resulting in inferior performance was the detection of PGD without known examples in the training data. We believe this is due to the aforementioned observation that PGD-generated adversarial examples have nearly vanishing gradients. Nonetheless, our evaluation showed that DA3G's detection performance significantly increased when PGD samples were part of the training set — without any degradation for the left-out attack type. We conclude that a gradient-based method should be combined with either a decision- or an optimization-based attack during training. Then, DA3G allows reliable detection, even of yet unknown attacks. This again reflects our findings using DLA. For both methods, our target-alarm architecture provides means to protect NNs against attacks. Our experiments suggest that this is possible even if the adversary uses attack methods not known during the training of out detectors.

3.3.6.4 Adaptive White-Box Attacks

In our final experiment, we evaluated the robustness increase due to DA3G against an omniscient adversary. This scenario is especially challenging as both the target model and our defense method were attacked simultaneously. We closely followed the available

Table 3.11: Results for experiment 3 testing DA3G: mean AUC with the according standard deviation after three runs for the leave-one-out gray-box detection.

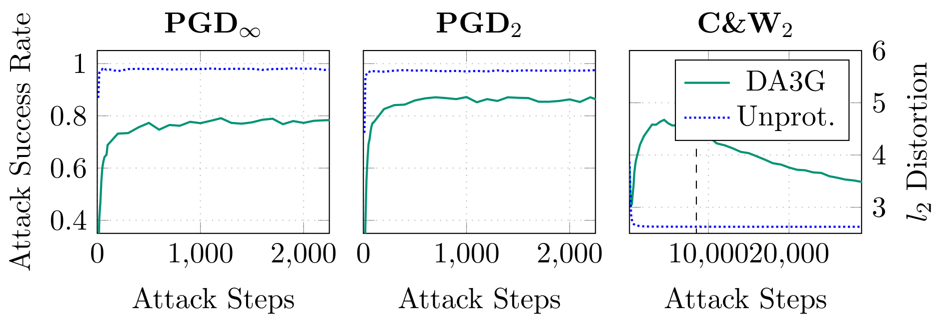
Data	**Target**	**Trained With**	**Tested With**		
			PGD$_{\infty,2}$	**DF**$_{\infty,2}$	**C&W**$_2$
MNIST	LeNet5	DF$_{\infty,2}$, C&W$_2$	0.66 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
		PGD$_{\infty,2}$, C&W$_2$	0.93 ± 0.00	0.99 ± 0.00	1.00 ± 0.00
		PGD$_{\infty,2}$, DF$_{\infty,2}$	0.92 ± 0.00	0.99 ± 0.00	0.99 ± 0.00
	KerasExM	DF$_{\infty,2}$, C&W$_2$	0.45 ± 0.02	1.00 ± 0.00	1.00 ± 0.00
		PGD$_{\infty,2}$, C&W$_2$	0.98 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
		PGD$_{\infty,2}$, DF$_{\infty,2}$	0.98 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
F-MNIST	LeNet5	DF$_{\infty,2}$, C&W$_2$	0.56 ± 0.01	1.00 ± 0.00	1.00 ± 0.00
		PGD$_{\infty,2}$, C&W$_2$	0.99 ± 0.00	0.99 ± 0.00	0.99 ± 0.00
		PGD$_{\infty,2}$, DF$_{\infty,2}$	0.99 ± 0.00	0.99 ± 0.00	0.99 ± 0.00
	KerasExM	DF$_{\infty,2}$, C&W$_2$	0.48 ± 0.02	1.00 ± 0.00	1.00 ± 0.00
		PGD$_{\infty,2}$, C&W$_2$	0.99 ± 0.00	0.98 ± 0.00	0.98 ± 0.00
		PGD$_{\infty,2}$, DF$_{\infty,2}$	0.99 ± 0.00	0.99 ± 0.00	0.99 ± 0.00
CIFAR10	KerasExC	DF$_{\infty,2}$, C&W$_2$	0.59 ± 0.07	0.98 ± 0.00	0.98 ± 0.00
		PGD$_{\infty,2}$, C&W$_2$	0.93 ± 0.00	0.91 ± 0.01	0.92 ± 0.00
		PGD$_{\infty,2}$, DF$_{\infty,2}$	0.93 ± 0.00	0.94 ± 0.00	0.93 ± 0.00
	ResNet	DF$_{\infty,2}$, C&W$_2$	0.63 ± 0.02	0.99 ± 0.00	1.00 ± 0.00
		PGD$_{\infty,2}$, C&W$_2$	0.98 ± 0.00	0.97 ± 0.00	0.99 ± 0.00
		PGD$_{\infty,2}$, DF$_{\infty,2}$	0.96 ± 0.01	0.99 ± 0.00	0.99 ± 0.00

guidelines in research [14, 51, 72]. The function expressing DA3G's application was introduced in Section 3.3.5.5. We used this function and performed PGD$_\infty$, PGD$_2$, and C&W$_2$ attacks against it. For a strong adversary, we took special care to optimize the attack hyperparameters. Previous work has shown that especially the attack step size may further increase the attack success of PGD [75]. Therefore, we evaluated multiple hyperparameters to increase the strength of the according adaptive attacks. In Table 3.12, we list the PGD step sizes chosen for the final models. For the gray-box experiments we used Foolbox's default values relative to ϵ. We show our results on MNIST, CIFAR10, and Fashion-MNIST in Figures 3.9–3.11. The robustness against the bounded PGD attack is measured using the attack success rate for a fixed perturbation budget. For the unbounded C&W attack we show the l_2 perturbation required for successful attacks. As general and natural observation we see all adaptive attacks bypassing DA3G after a certain level of attack effort. Hence, our evaluation does not seem to overestimate the robustness due to obfuscated gradients [25] or similar effects. We therefore argue that our attack settings and training parameters are indeed correctly chosen.

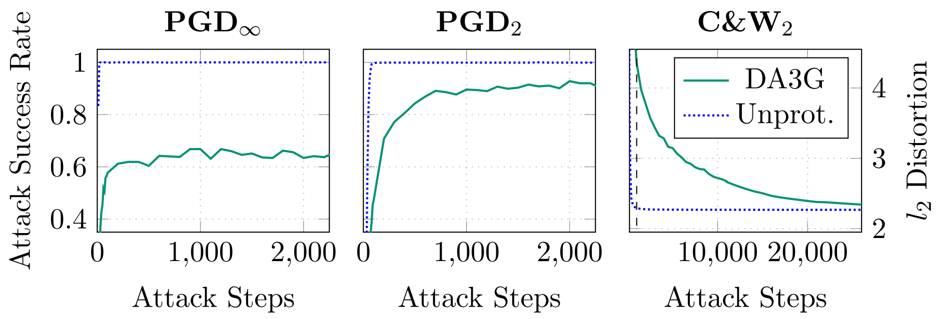
Robustness against PGD adaptive attacks As expected, all unprotected target models could be attacked within the minimum number of steps. The DA3G-protected models

Table 3.12: PGD step sizes during the adaptive attacks on DA3G.

Data	Target	PGD$_2$	PGD$_\infty$
MNIST	LeNet5	$0.500/\epsilon$	$0.040/\epsilon$
	KerasExM	$0.075/\epsilon$	$0.025/\epsilon$
F-MNIST	F-LeNet5	$0.300/\epsilon$	$0.015/\epsilon$
	F-KerasExM	$0.200/\epsilon$	$0.015/\epsilon$
CIFAR10	KerasExC	$0.150/\epsilon$	$0.0015/\epsilon$
	ResNet	$0.050/\epsilon$	$0.001/\epsilon$



(a) MNIST, LeNet5

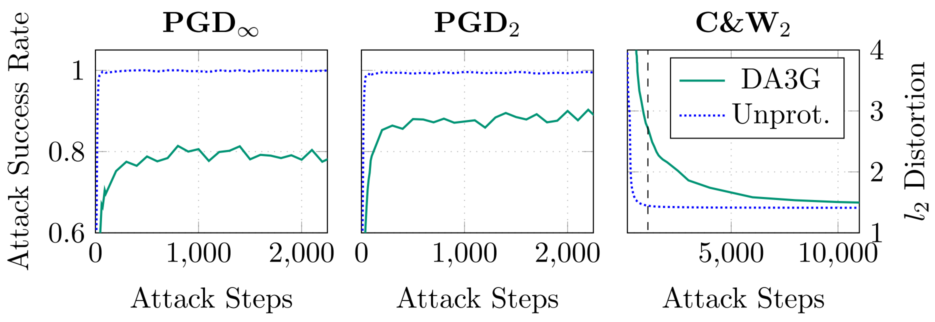


(b) MNIST, KerasExM

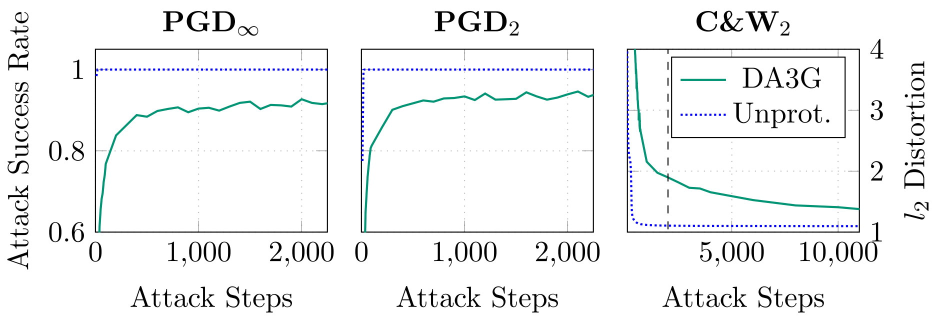
Figure 3.9: Adaptive attacks on the MNIST models protected by DA3G. The dashed lines show the number of C&W steps required such that at least 80% of all attacks on the protected model were successful.

showed more resilience, either by increasing the attack effort or by generally lowering the attack success rate even after convergence.

Especially on MNIST and Fashion-MNIST, the differences were striking. For PGD$_\infty$, DA3G could lower the attack success to less than 90% for MNIST under increased attack effort. Our results suggest that the target network architecture has an impact on the vulnerability of the entire system. For KerasExM, the attack success rate increased slowly, then converged to 66.8% after 1000 steps. In contrast, the attacker achieved an



(a) F-MNIST, LeNet5



(b) F-MNIST, KerasExM

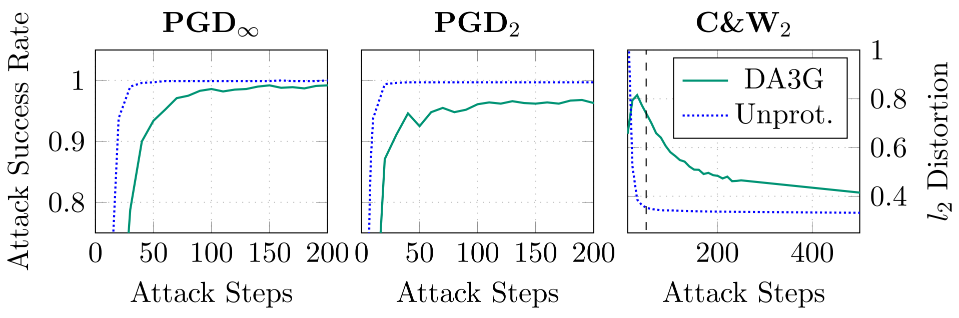
Figure 3.10: Adaptive attacks on the Fashion-MNIST models protected by DA3G. The dashed lines show the number of C&W steps required such that at least 80% of all attacks on the protected model were successful.

attack success rate of around 80% on LeNet5. After the attacks have converged, even doubling the number of steps did not further increase the success rate. We report similar observations for Fashion-MNIST. Although improving on the unprotected MNIST models, DA3G seemed more vulnerable to PGD$_2$ attacks. For Fashion-MNIST we report a similar level of robustness against both PGD versions.

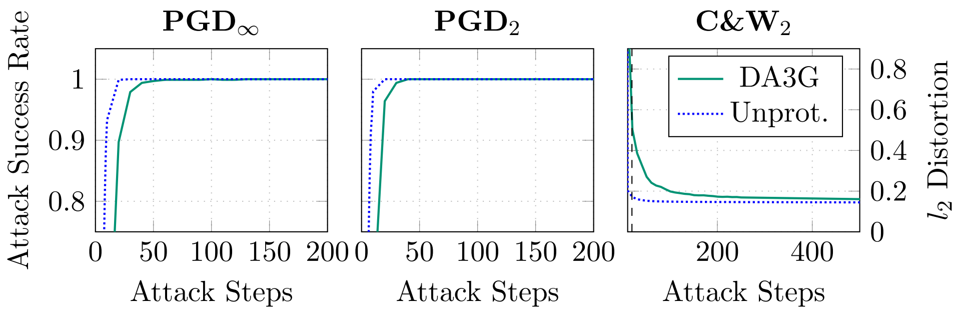
For ResNet on CIFAR10, both PGD$_2$ and PGD$_\infty$ reached 100% attack success rates. In the more resilient case, attacks on KerasExC did not reach perfect success rates. Here, PGD$_2$ converged to 96.1% after 100 iterations.

Throughout the experiments, strictly more attack steps were needed to achieve successful attacks compared to the unprotected case. We conclude that DA3G successfully increased the level of protection of the NNs and often prevents a notable fraction of the performed attacks. Furthermore, the adversaries' attack effort increased considerably compared to the unprotected case.

Robustness against C&W adaptive attacks For the unbounded C&W attack, we expressed the level of protection by measuring the required perturbation budget. As seen in Figures 3.9–3.11 severe levels of perturbation were needed for a successful attack even after thousands of attack steps. During our evaluation, we saw that the attack did not



(a) CIFAR10, KerasExC



(b) CIFAR10, ResNet

Figure 3.11: Adaptive attacks on the CIFAR10 models protected by DA3G. The dashed lines show the number of C&W steps required such that at least 80% of all attacks on the protected model were successful.

reach a perfect success rate immediately, even though the C&W method is unbounded and thus could alter the input by any extent. In such cases, we calculated the l_2 perturbations using the successful adversarial attacks only. We saw similar results across all data sets: Whereas the unprotected model was easily attacked, the distortion needed for the DA3G-protected model nearly doubled. Depending on the properties of the data and underlying use case, the increased amount of required perturbations may prevent the attacker from successfully crafting humanly imperceptible adversarial examples. We conclude that DA3G improves the robustness against optimization-based attacks and forces adversaries to induce more changes to the inputs.

3.3.7 Discussion

With DA3G, we present a new end-to-end method that reliably detects adversarial examples under challenging threat models. Summarizing our evaluation, we report that DA3G forces potential adversaries to invest more computational power to fool our protected NNs. For some attacks, even considerable attack effort did not result in overall attack success. Interestingly, we report a notable difference in the level of protection against adversaries using either PGD$_\infty$ or PGD$_2$. For bounded attacks, the reported success rates depend on the fixed values of ϵ, quantifying the allowed perturbation budgets. In real-world applications, ϵ highly depends on the underlying use case and definition of the imperceptibility of adversarial examples defined by the assumed threat model. Hence, we argue that even though DA3G may seem to provide a limited robustness towards PGD in our experiments, it may find its application in real-world scenarios in which adversaries face a more limited perturbation space. In our evaluation, we applied DA3G to a wide range of target networks indicating the applicability independently of the NN architecture. We envision the detection of, e.g., audio adversarial examples using sequential target models.

With DA3G, we analyzed whether our target-alarm system can be used to detect adversarial examples when provided with raw gradients instead of activations. Our empirical evaluation showed that our architecture is indeed suitable for this task when using the gradient as main source of information.

3.3.8 Conclusion

In this section, we present an extension to our DLA architecture and provide means to reliably detect adversarial examples. We replace the analysis of the dense-layer activations by automatically leveraging the information in the gradient of the network under test for potential attacks. With our evaluation under challenging threat models, we show that our architecture allows reliable detection of known and yet unknown attacks even if the dense-layer activations are not available. We show that adaptive adversaries are forced to invest significantly more computational effort to successfully craft adversarial examples when attacking protected NNs using the gradient-driven DLA version. With our findings, we contribute to the research on adversarial example detection and show the following two insights: First, using our target-alarm architecture presented in Section 3.2 is indeed usable even if the dense-layer activations of the NNs are not available. This directly answers **Q1** presented in Section 3.2.9. Secondly, not only the activation values of NNs carry robustness-sensitive information, but other sources, in this case the gradients, can be used in a similar manner.

3.4 Activation Anomaly Analysis

In this section, we present an additional field of application of our previously introduced target-alarm architecture. Parts of the section are taken verbatim from the original paper "Activation Anomaly Analysis" published at the Joint European Conference on Machine Learning and Knowledge Discovery in Databases (ECML PKDD) 2020 [1]. This is collaborative work with the co-first author Jan-Philipp Schulze.

3.4.1 Motivation

In Section 3.3 we investigate whether our target-alarm architecture presented in Section 3.2 is applicable even if the dense-layer activations of the NNs are not available and other sources of information are required. To this end, we present DA3G and show that by analyzing NNs' gradients and training an auxiliary model the detection of adversarial examples is possible. In this section, we present our findings on the second question **Q2** which arose at the end of Section 3.2.9: Can the activation values of NNs be further leveraged and used in the context of robustness and security in general? We show that the contained information indeed can be utilized in the more general and broader field of anomaly detection.

Anomaly detection is the task of identifying data points that differ from samples considered normal. Reliable anomaly detection is of great interest especially in the context of security-sensitive systems. Here, anomalies can indicate attacks on the infrastructure, fraudulent behavior, or general points of interest. DL methods allow to analyze highly complex data in an automated manner. Therefore, in recent years, the number of ML applications using DL concepts to detect anomalies has steadily grown. Yet, anomaly detection tasks are especially challenging for DL methods due to the inherent class imbalance. Usually, a higher number of samples considered normal is available during training. In our work, we develop a new DL-based anomaly detection method showing superior results compared to state-of-the-art methods. We simulate real-world settings and only use a handful of anomalous samples.

In DL-based anomaly detection, a popular idea is to use an autoencoder (AE) to preprocess or reconstruct the data points. By only using normal samples during the training of the AE, the reconstruction error when processing anomalous samples is assumed to be higher. This allows the detection of anomalies by evaluating this reconstruction error, e.g., by using manually set thresholds. In this section, we show that our findings from our two previous sections can be used to improve state-of-the-art anomaly detection using AEs. We show that the hidden activations of AEs, but also other types of NN architectures, are useful to judge if the current input is normal or anomalous. By using our target-alarm architecture and combining the information of two interrelated NNs, we achieve strong detection results.

During the conceptual phase, we were again inspired by coverage-guided NN testing methods. In this promising new research direction, software testing concepts build the foundation of methods to test DL models. The goal is to identify faulty regions in NNs responsible for unusual behavior, or errors during run-time. Pei et al. [77] first introduced the idea of neuron coverage to guide a testing process. Since then, further improvements and modifications have been proposed, e.g., by Ma et al. [78] and Sun et al. [79]. In this section, we build upon recent findings and our insights presented in Section 3.2 to further generalize the concept by adapting it to the constraints of anomaly detection. Whereas a high number of samples for benign as well as adversarial inputs are available in adversarial ML, anomaly detection is a semi-supervised setting with only a few anomaly-related data points available. Similar to the case of adversarial attacks, we see that target models behave in a distinguishable manner when processing anomalous inputs. We show that this behavior is again detectable by analyzing the activation values during run-time and training an alarm model optimized for this task.

Applying our concept, we empirically show that anomalous samples cause different hidden activations compared to normal ones. We profit from the target-alarm architecture originally presented in Section 3.2 and analyze all hidden layers of the target network using an auxiliary alarm model. In our evaluation we show that our concept reliably detects anomalies from different data sets and we report superior performance compared to common baseline methods. In summary, in this section we present the following contributions:

- We show that our target-alarm architecture is useful in the field of anomaly detection. Based on the analysis of the hidden activations of NNs using our architecture, we propose a purely data-driven semi-supervised anomaly detection method we call A^3.
- With our thorough evaluation, we show that the patterns in the activations generalize to new anomaly types even when only a few anomaly examples are available during training.
- We show that our target-alarm architecture generalizes to different target NN architectures.

3.4.2 Related Work

Anomaly detection is a topic of active research with a wide range of use cases and methods. For instance, in network intrusion detection [105], power grids [106], or industrial control systems [107], automated mechanisms improve the security of the overall system. Thorough overviews of current advances in anomaly detection in general and DL-based systems in particular can be found in the surveys by Chandola et al. [108] and Chalapathy and Chawla [109], respectively. In this section we present our method A^3, which we designed for a semi-supervised setting. For this purpose, we follow the surveys' definitions on this term and the considered scenario. Thus in semi-supervised settings, some knowledge on anomalous samples is given and we assume the training data set to consist of normal samples only. In contrast, in unsupervised settings no knowledge on the nature of any sample is given. This might lead to the occurrence of anomalous samples even in the training data set. Well-known unsupervised methods are OC-SVM by Schölkopf et al. [110], or Isolation Forest by Liu et al. [111].

In DL-based anomaly detection systems, a popular choice are architectures incorporating AEs, often used as feature extractors. Here, AEs are then combined with classical ML classifiers like k-nearest neighbor [112], OC-SVMs [113], NN-based classifiers [114], or Gaussian Mixture Models [115]. Similarly, other feature extraction networks like recurrent NNs have been evaluated [116–118]. To detect anomalies, AEs may also be used in their purest form: restoring the input under the constraint of small hidden layers, similar to classical dimensionality reduction methods like PCA. The reconstruction error is used to discriminate anomalous samples from normal data points [112, 119]. Research has further analyzed how to improve the anomaly detection results, e.g., by iteratively adding human feedback [120, 121].

Over the past decade, computing power and data storage have steadily risen. DL methods profit from the increased amount of training samples. Recent research [122–124] has studied ways to incorporate known anomalies into DL-based anomaly detection. They show that even a few anomalous samples, which are usually available in practice, improve the overall detection performance. In Section 3.4.4.2, we discuss state-of-the-art frameworks to which we compare A^3. With our work, we show that the activations of NNs differ for normal and anomalous samples. Using our method, we detect known and even new, yet unseen types of anomalies in different scenarios with high confidence. With our method, we significantly improve NN-based semi-supervised anomaly detection systems.

3.4.3 Detecting Anomalies by Analyzing Activations

We based our anomaly detection method A^3 on the same hypothesis as the one presented in Section 3.2. The activation values of NNs carry robustness-sensitive information and allow an assessment of the behavior of the analyzed model. With this observation, we empirically showed that the detection of attacks against the observed model is possible. In the following, we reformulate this initial hypothesis and apply it to the task of detecting anomalies. We again assume a pretrained neural network. It is fair to expect that this model was trained using normal and known data only which are part of the training data set $\mathcal{D}_{\text{train}}$. In an anomaly detection task, unknown test samples are presented to the NN which we summarize by $\mathcal{D}_{\text{test}}$. Normal samples of this test data set follow a similar distribution and are of comparable nature as the samples found in $\mathcal{D}_{\text{train}}$. In contrast, anomalies are defined as samples which differs from normal data in some unspecified and even unknown manner. Hence, we assume that if the pretrained NN processes anomalies, a distinguishable behavior of the model can be observed by analyzing its activations compared to the case in which it processes normal samples.

3.4.3.1 A^3 Architecture

Based on our hypothesis, here we present our new anomaly detection method A^3 which consists of two parts.

The target model performs a task unrelated to anomaly detection. In accordance to our assumption, the classes the target was trained on are considered normal. Several

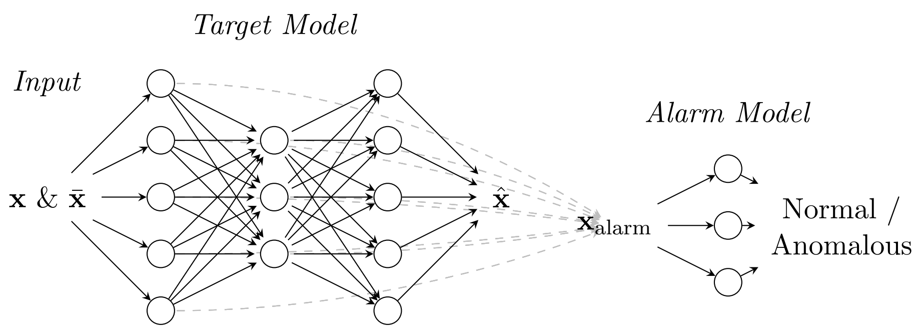


Figure 3.12: Summary of the components of the adapted DLA architecture to detect anomalies, here called A^3, consisting of a target and an alarm model.

NN architectures can be used to implement the target. As we have seen in Section 3.2 classifiers based on CNNs, LSTMs, and Capsule networks can be protected by analyzing their activations. Here, we evaluate fully-connected as well as convolutional autoencoders and classifiers to act as the target in our system.

The alarm model evaluates the activation values of the target model. Using the available information in the hidden activations of the target, the alarm model decides if the given sample is normal or anomalous.

Both models are combined to one connected architecture. In the scope of this section, we fix the target model to its pretrained state and solely train the alarm model to detect anomalies. Our assumption is that the activations caused by the input show particular patterns for samples the target model was trained on (i.e., normal samples), and for other samples (i.e., anomalous samples). The alarm model analyzes the anomaly-related patterns in the target network's activations. A high-level overview is given in Figure 3.12.

3.4.3.2 Training and Prediction Objectives

In the following we describe the objectives followed during training, ultimately allowing the detection of anomalies in the prediction phase. We present a combined overview of the target training, alarm training, and final anomaly detection in Figure 3.13.

Target Model Training The target model performs a task unrelated to anomaly detection using input $\mathbf{x}$. We propose autoencoders and classifiers to act as targets. Their original tasks, as well as their objectives during training can be summarized as follows: Autoencoders try to reconstruct a given input under the constraint of small hidden representations such that $f_{\text{target}} : \mathbf{x} \mapsto \hat{\mathbf{x}}$. Classifiers predict the class of a given input sample with respect to known classes such that $f_{\text{target}} : \mathbf{x} \mapsto \hat{y} \in \{1, \dots, |\mathcal{Y}|\}$. According to our fundamental assumption, all samples used during training are considered normal. As we use pretrained target models, standard training procedures can be followed and are thus not presented in more detail in this section.

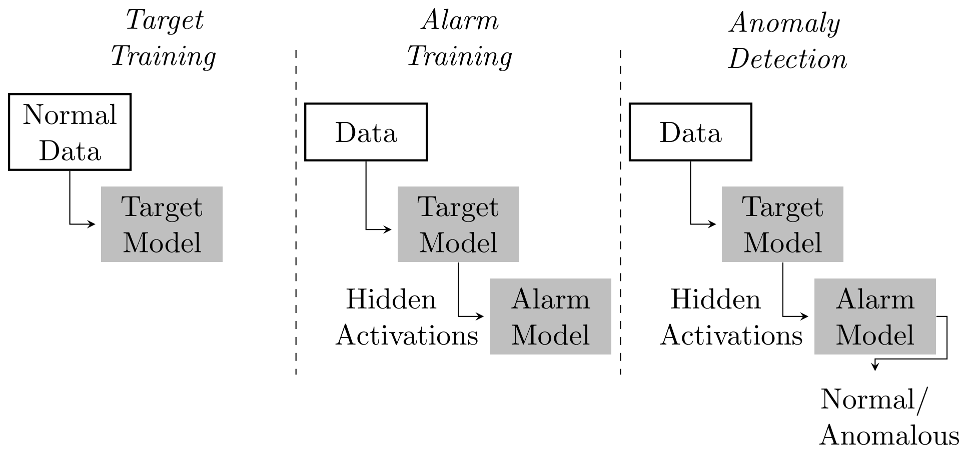


Figure 3.13: Step-by-step overview of A^3's training process in the first two columns and the final anomaly detection in the right column.

Alarm Model Training The alarm model maps the original input sample to an anomaly score. Following our original target-alarm architecture, it does not operate on the input directly, but observes the target's activations caused by the input. Hence, the alarm's output $\hat{y}$ is implicitly dependent on the input, the trained weights, and behavior of the target. The function approximated by the alarm model can be formalized as follows:

$$f_{\text{alarm}} : [\mathbf{h}_{\text{target},1}(\mathbf{x}; \boldsymbol{\theta}_{\text{target}}), \dots, \mathbf{h}_{\text{target},L-1}(\mathbf{x}; \boldsymbol{\theta}_{\text{target}})] \mapsto \hat{y} \in [0,1].$$

Note that only the alarm network's weights $\boldsymbol{\theta}_{\text{alarm}}$ are adapted while the target network's weights $\boldsymbol{\theta}^{\star}_{\text{target}}$ remain unchanged. With $\mathcal{L}_x(y, \hat{y})$ being the binary cross-entropy loss, the overall objective of the A^3 training process is defined as:

$$\mathcal{L}(\mathbf{x}, y) = \mathcal{L}_x(y, \hat{y})|_{\hat{y}=f_{\text{alarm}}(\mathbf{x}; \boldsymbol{\theta}^{\star}_{\text{target}}, \boldsymbol{\theta}_{\text{alarm}})} + \lambda \cdot \mathcal{L}_x(1, \hat{y})|_{\hat{y}=f_{\text{alarm}}(\bar{\mathbf{x}}; \boldsymbol{\theta}^{\star}_{\text{target}}, \boldsymbol{\theta}_{\text{alarm}})}.$$

The left term expresses the loss during training if the current sample is normal. Complementary, the right term quantifies the loss based on an anomalous training sample. The weighting factor λ might be used to counteract the class imbalance apparent during training. Usually a higher number of normal training samples are available.

Prediction Phase During the prediction phase, the target and the alarm models act as one combined system, mapping the input to an anomaly score: $f_{\text{detect}} : \mathbf{x} \mapsto \hat{y} \in [0,1]$. The input is processed using the target-alarm pipeline in an automated end-to-end manner: $\mathbf{x}$ is fed to the target model triggering the activation values $\mathbf{h}_{\text{target},i}(\mathbf{x}; \boldsymbol{\theta}_{\text{target}})$. Based on the target's activations, the alarm model decides whether the input is more likely to be normal or anomalous.

3.4.4 Experimental Setup

In the following, we present the details characterizing our experiments and the implementation of A^3.

3.4.4.1 Data Sets

We evaluated the performance of A^3 on five publicly available and commonly used data sets. Our selection allowed us to compare the performance to related work and motivates that A^3 performs well even in complex scenarios.

1. *MNIST* [83]: common image data set for ML problems with 70 000 images showing ten handwritten digits.
2. *EMNIST* [125]: extension to MNIST with handwritten letters.
3. *NSL-KDD* [126]: common intrusion detection data set with around 150 000 samples. We use *KDDTest*$^+$ for testing containing new anomalies not present in the training set.
4. *Credit Card Transactions* [127]: anonymized credit card data of around 285 000 transactions of which 492 are fraudulent.
5. *CSE-CIC-IDS2018* [128]: large network data set. We omit the DDoS data due to the high resource demands. Around five million samples remain.

We limited the preprocessing to minmax-scaling of numerical data and 1-Hot-encoding of categorical data. Samples still containing non-numerical values were omitted. In the IDS data set, we discarded the IP addresses as well as the flow IDs. Generally, we took 80% of the data for training, 5% for validation, and 15% for testing. If a test set is given, we used it instead.

3.4.4.2 Baseline Methods

We compared the performance of A^3 to four common baseline methods. Note that we only considered baseline methods that scale to the large amount of data.

1. *Autoencoder Reconstruction Threshold:* When trained on normal data only, there is a measurable difference in the reconstruction quality compared to anomalous samples. We use the mean squared error, i.e., $\mathcal{L}(\mathbf{x}, \hat{\mathbf{x}}) = \|\hat{\mathbf{x}} - \mathbf{x}\|_2^2$, to quantify this difference. Using a preset threshold on this error, the detection of anomalous inputs is possible. In the settings in which we used AEs as targets, we used the very same models as baseline.
2. *Isolation Forest (IF):* IF is a common unsupervised anomaly detection method by Liu et al. [111]. Based on the given data, an ensemble of random trees is built. The average path length is used as anomaly score. We used the implementation provided by scikit-learn along with the default parameters [129].

Table 3.13: Dimensionality of the layers used in the target and alarm models for A^3. All hidden layers are activated by ReLU.

Data Set	Target Architecture	Alarm Architecture
MNIST	according to [132]	1000, 500, 200, 75, 1
NSL-KDD	200, 100, 50, 25, 50, 100, 200	1000, 500, 200, 75, 1
IDS	150, 80, 40, 20, 40, 80, 150	1000, 500, 200, 75, 1
CreditCard	50, 25, 10, 5, 10, 25, 50	1000, 500, 200, 75, 1
MNIST & EMNIST	according to [133]	1000, 500, 200, 75, 1

3. *Deep Autoencoding Gaussian Mixture Model (DAGMM):* DAGMM is a state-of-the-art unsupervised DL-based anomaly detection method by Zong et al. [115]. The authors combine information of an AE with a Gaussian mixture model. For all experiments, we used the implementation by Nakae with the architecture recommended for the KDDCUP data set [130].

4. *Deviation Networks (DevNet):* DevNet is a state-of-the-art semi-supervised DL-based anomaly detection method by Pang et al. [122]. The anomaly detection is split between a feature learner and an anomaly score learner, both implemented with a NN. The anomaly scorer sets normal samples close to a Gaussian prior distribution and enforces a minimum distance to anomalous samples. As recommended by the authors, we used their default architecture.

3.4.4.3 Implementation Details

An overview of the architectures used for each experiment is given in Table 3.13. For the AE target models, we chose the first layer to be slightly larger than the dimension of the input vectors, whereas the hidden representation is smaller. For the sake of simplicity, we used a common alarm model architecture throughout our experiments. Note that for the MNIST-related experiments, we considered two publicly available architectures from Keras [131], a convolutional AE [132] extended by a dropout layer, as well as a CNN [133]. This underlines the generality of our method. All hidden layers are activated by ReLU.

Parameter Choices Based on a non-exhaustive parameter search on MNIST, we chose the following global optimizer settings. We used Adam with a learning rate of $1 \cdot 10^{-3}$ for the target models, and $1 \cdot 10^{-5}$ for the alarm models. The training was stopped after 30 and 60 epochs, respectively. No other regularizer than 10% dropout [134] before the last layer was used. We ran our experiments on an Intel Xeon E5-2640 v4 server accelerated by an NVIDIA Titan X GPU. To support further research, we open-sourced our code[2].

[2]Code available at: `https://github.com/Fraunhofer-AISEC/A3`.

3.4.4.4 Experiment Overview

Anomaly Detection Constraints In the setting of anomaly detection, special constraints apply in real-world scenarios. We present the three main challenges based on which we designed our experimental setup to test the performance of A^3. With our experiments, we show that our method performs well even if considering all constraints simultaneously.

1. *Scarcity of Anomaly Samples.* Generally, a high number of training samples are required to create well-performing DL systems. However, there is a natural imbalance in anomaly detection scenarios: Most data samples are normal, only a few examples of anomalies are usually found manually. We show that A^3 performs well in this semi-supervised setting.
2. *Variable Extent of Abnormality.* Anomalous samples are not bound by a common behavior or magnitude of abnormality. By definition, the only difference is that anomalies do not resemble normal samples. We show the transferability of A^3, i.e., using known anomalies during training also allows us to detect unknown anomalies during testing.
3. *Driven by Data, not Expert Knowledge.* Anomaly detection algorithms should be applicable in multiple scenarios, even when no expert knowledge is available. Performance that is only achievable using domain knowledge may result in inferior results in real-world applications. We show that A^3 generalizes to other settings, i.e., uses the data itself to distinguish between normal and anomalous behavior and does not depend on expert knowledge during run time.

Summary of the Performed Experiments We designed three experiments to show that A^3 works well under all three constraints. An overview is given in Table 3.14.

As an initial proof-of-concept experiment, we first tested our hypothesis that the hidden activations of the targets can indeed be used to detect anomalous samples. This is not part of the main experiments of this evaluation. Here, we used all available anomalies during training and only detected known anomaly types during testing. Hence, the detection of new, yet unseen types of anomalies is not considered in this experiment.

In the following, we describe the three main experiments in more detail. For all three experiments we used 100 known anomalies during training. This simulates a real-world application as close as possible. Hence, constraint 1 is considered in all three main experiments of this evaluation. For all three experiments we also compared our results to baseline methods, if applicable.

1. *Experiment 1: Detection of Known Anomalies.* Considering constraint 1, we evaluated the fundamental assumption our method is based on: The activation values of the target model contain information useful to distinguish between normal and anomalous samples. It is important to remember that only the alarm network, not the target, is trained on the anomaly detection task. We limited the anomaly

Table 3.14: A^3 experiments exploring the detection of known and unknown types of anomalies. The three experiments are divided with respect to the used data sets.

Exp.	Data	Normal	Train Anomaly	$\subseteq$	Test Anomaly
1a	MNIST	0, . . . , 5	6, 7		6, 7
1b	MNIST	4, . . . , 9	0, 1		0, 1
1c	NSL-KDD	Normal	DoS, Probe		DoS, Probe
1d	NSL-KDD	Normal	R2L, U2R		R2L, U2R
1e	IDS	Benign	BF, Web, DoS, Infil.		BF, Web, DoS, Infil.
1f	IDS	Benign	Bot, Infil., Web, DoS		Bot, Infil., Web, DoS
1g	CC	Normal	Fraudulent		Fraudulent
2a	MNIST	0, . . . , 5	6, 7		6, 7, 8, 9
2b	MNIST	4, . . . , 9	0, 1		0, 1, 2, 3
2c	NSL-KDD	Normal	DoS, Probe		DoS, Probe, R2L, U2R
2d	NSL-KDD	Normal	R2L, U2R		R2L, U2R, DoS, Probe
2e	IDS	Benign	BF, Web, DoS, Infil.		BF, Web, DoS, Infil., Bot
2f	IDS	Benign	Bot, Infil., Web, DoS		Bot, Infil., Web, DoS, BF
3a	(E-)MNIST	0, . . . , 9	A, B, C, D, E		A, . . . , E
3b	(E-)MNIST	0, . . . , 9	A, B, C, D, E		A, . . . , E, V, . . . , Z
3c	(E-)MNIST	0, . . . , 9	V, W, X, Y, Z		V,. . . , Z
3d	(E-)MNIST	0, . . . , 9	V, W, X, Y, Z		A, . . . , E, V, . . . , Z

samples to 100 randomly selected instances during training in accordance with the semi-supervised setting. This limitation may cause some classes to not be present during training.

2. *Experiment 2: Transferability to Unknown Anomalies.* Considering constraints 1 and 2, we evaluated the transferability of our fundamental assumption: The activation values of the target model are inherently different for normal and anomalous samples. Similar to experiment 1, we evaluated the detection performance bound by the scarcity of anomaly labels. Furthermore, the test data set contained more anomaly classes than the alarm network has been trained on. With this experiment, we tried to find anomalies that follow a different nature and data distribution than the few known samples.

3. *Experiment 3: Generality of the Method.* Considering constraints 1, 2, and 3, we evaluated the generality of our fundamental assumption: The activation values of any type of target network contain information that allows samples to be distinguished between normal and anomalous samples. We used a publicly available classifier as target, extracted the activation values, and tested whether these can be used to detect known as well as unknown anomalies. Hence, we evaluated

Table 3.15: Proof-of-concept results: Anomaly detection test scores for A^3 after ten runs given all normal and all anomalous samples during training.

Exp.	AUC	AP	Exp.	AUC	AP	Exp.	AUC	AP
1a	1.00±0.00	1.00±0.00	2a	0.93±0.01	0.92±0.01	3a	1.00±0.00	1.00±0.00
1b	1.00±0.00	1.00±0.00	2b	0.93±0.02	0.92±0.01	3b	1.00±0.00	1.00±0.00
1c	0.98±0.01	0.98±0.01	2c	0.93±0.02	0.95±0.01	3c	1.00±0.00	1.00±0.00
1d	0.85±0.06	0.80±0.05	2d	0.75±0.09	0.82±0.05	3d	1.00±0.00	1.00±0.00
1e	0.98±0.00	0.96±0.00	2e	0.89±0.02	0.85±0.02	-	-	-
1f	0.98±0.00	0.95±0.00	2f	0.95±0.01	0.93±0.01	-	-	-
1g	0.98±0.01	.80±0.04	-	-	-	-	-	-

whether our anomaly detection mechanism can be applied in combination with already existing target networks deployed in environments of any type.

3.4.5 Evaluation

In the following, we present and evaluate the results measured during our experiments. As metrics, we chose the average precision (AP) and the area under the ROC curve (AUC) to be consistent with related work [122]. Whereas the AP quantifies the trade-off between precision and recall, the AUC measures the trade-off between the TPR and FPR. Both metrics are independent of a detection threshold, and thus give a good overview of the general detection performance. The scores range between 0 and 1, higher values indicate a more accurate detection.

3.4.5.1 Proof-Of-Concept Results

In Table 3.15 we show the results of our proof-of-concept experiment. We evaluated three experimental setups and used all available anomalous samples. For a more compact visualization in the table, we present the results using three columns for each experimental setup. In experiment 1 and 2, A^3 achieved perfect results in a wide range of settings. Even in the most challenging setting of experiment 2 in which unknown types of anomalies need to be detected, A^3 scored an AUC of above 0.9 in four of the six settings. With these first results at hand, we confirm our hypothesis: The activation values indeed carry generally information which can be used to detect anomalous data points. This observation directly extends our findings on adversarial example detection. Additionally, not only classifiers can act as target models, but also autoencoders which are designed to reconstruct a given input.

The amount of used anomalies during training in our proof of concept exceeds the usually available number of samples. Therefore, for the first two experimental setups using the MNIST data set we re-run our evaluation while limiting the available anomalous samples during training. In Table 3.16 we summarize the results. We see that the

Table 3.16: Test results for experiments 1a and 2a using the MNIST data set after ten runs given all normal samples. In the individual experiments, different amounts of known anomalies were used during training. This table shows how different amounts of anomalies influence the performance of A^3.

Exp.	#Anomalies	AUC	AP
1a	100	0.99±0.00	0.99±0.00
1a	1000	1.00±0.00	1.00±0.00
1a	5000	1.00±0.00	1.00±0.00
1a	11576 (all)	1.00±0.00	1.00±0.00
2a	100	0.88±0.02	0.87±0.02
2a	1000	0.90±0.01	0.90±0.01
2a	5000	0.92±0.01	0.91±0.01
2a	11576 (all)	0.93±0.01	0.92±0.01

detection performance degrades with a decreasing number of anomalies used in training. This effect is especially visible for the second experiment in which we detected unknown types of anomalies. Nonetheless, for this first experiment, we see that using 100 anomalies during training still allows for a reliable detection. We argue that this amount of labeled anomalies provides an experimental setup close to real-world settings.

3.4.5.2 Main Results

In Table 3.17 we show the main results of our evaluation. This includes the detection scores for all three experiments and five data sets. We summarize the results for A^3 and the baseline methods averaged over ten passes using the test data sets. To simulate real-life conditions, we limited the amount of available anomalies to 100 randomly chosen samples of the training set. We find that A^3 performs well even under this strict setting.

In experiment 1 we detected known types of anomalies using autoencoder target models. Here, A^3 outperforms the baseline methods in the majority of test cases. We report a mean AUC and AP of 0.94 and 0.90, respectively. Comparing the results to the other semi-supervised detection method DevNet, we make the following observations: While we surpassed DevNet's results in six of the seven cases, we see similar mean detection scores. Our mean AUC score is 1% lower, while we achieved a 3% higher mean AP compared to DevNet. Assessing A^3's results in more detail, we report a comparably low performance for experiment 1d. A limited diversity of the training set or anomalous training samples not representative for the test set might have contributed to the performance drop in this case.

In experiment 2 we detected known and unknown types of anomalies using autoencoder target models. As expected, in this highly competitive case, A^3 was outperformed in the majority of cases. Yet comparing to DevNet, we see only a slightly lower detection

Table 3.17: A^3 anomaly detection test results after ten runs given all normal and 100 anomalous samples. Black digits indicate the highest scores in each experiment.

	A^3		**AE**		**IF**		**DAGMM**		**DevNet**	
Epx.	AUC	AP	AUC	AP	AUC	AP	AUC	AP	AUC	AP
1a	0.99±0.00	0.99±0.00	0.70±0.03	0.44±0.04	0.57±0.02	0.28±0.02	0.85±0.02	0.70±0.02	0.98±0.00	0.95±0.01
1b	1.00±0.00	1.00±0.00	0.39±0.04	0.25±0.03	0.53±0.01	0.41±0.02	0.63±0.04	0.34±0.03	0.99±0.00	0.98±0.00
1c	0.96±0.01	0.97±0.01	0.96±0.00	0.96±0.01	0.97±0.00	0.98±0.00	0.94±0.01	0.91±0.04	0.95±0.02	0.95±0.02
1d	0.80±0.01	0.75±0.03	0.82±0.03	0.55±0.06	0.84±0.00	0.49±0.02	0.91±0.01	0.59±0.03	0.91±0.03	0.79±0.02
1e	0.94±0.01	0.91±0.01	0.88±0.08	0.75±0.12	0.47±0.04	0.17±0.01	0.90±0.02	0.68±0.06	0.93±0.00	0.90±0.01
1f	0.93±0.00	0.90±0.00	0.86±0.07	0.65±0.13	0.36±0.03	0.14±0.01	0.68±0.12	0.34±0.18	0.90±0.01	0.74±0.04
1g	0.97±0.01	0.78±0.04	0.97±0.01	0.54±0.09	0.97±0.01	0.14±0.04	0.96±0.02	0.35±0.23	0.97±0.01	0.77±0.04
2a	0.88±0.02	0.87±0.02	0.72±0.02	0.63±0.02	0.54±0.01	0.40±0.01	0.74±0.02	0.69±0.02	0.84±0.04	.84±0.03
2b	0.87±0.02	0.87±0.02	0.64±0.03	0.62±0.03	.64±0.01	0.60±0.01	0.71±0.02	0.61±0.01	0.90±0.03	0.90±0.02
2c	0.87±0.03	0.92±0.01	0.93±0.01	0.94±0.01	0.94±0.00	0.96±0.00	0.93±0.01	0.92±0.03	0.90±0.02	0.93±0.01
2d	0.68±0.14	0.77±0.09	0.94±0.00	0.94±0.01	0.94±0.00	0.96±0.00	0.93±0.01	.92±0.03	0.88±0.02	0.89±0.02
2e	0.87±0.04	0.83±0.03	0.92±0.02	0.82±0.07	0.45±0.03	0.20±0.01	0.76±0.08	0.52±0.10	0.90±0.00	0.83±0.01
2f	0.90±0.02	0.88±0.02	0.89±0.06	0.79±0.10	0.45±0.03	0.20±0.01	0.80±0.04	0.56±0.02	0.92±0.00	0.81±0.02
3a	0.99±0.00	0.99±0.00	-	-	0.93±0.00	0.85±0.01	0.93±0.01	0.84±0.03	0.99±0.00	0.98±0.00
3b	0.96±0.01	0.97±0.01	-	-	0.91±0.01	0.89±0.01	0.95±0.00	0.93±0.01	0.97±0.01	0.97±0.01
3c	0.99±0.00	0.99±0.00	-	-	0.89±0.01	0.79±0.02	0.96±0.00	0.91±0.01	0.98±0.01	0.97±0.01
3d	0.96±0.01	0.96±0.01	-	-	0.91±0.01	0.89±0.01	0.95±0.00	0.93±0.01	0.96±0.02	0.96±0.02

reliability. A^3 achieved a 5% and 1% lower mean AUC and AP score, respectively. Again with the exception of experiment 2d were parts of the anomalies from experiment 1d were used we report strong results. With our results we argue that A^3 is able to act as a viable solution in revealing new types of anomalies.

In experiment 3 we detected known and unknown types of anomalies using classifier target models. Interestingly, in this setting we achieved nearly perfect results even when detecting unknown types of anomalies. We surpassed all unsupervised baseline methods. Solely DevNet achieved similarly high results compared to A^3.

Finally, we conclude that our fundamental assumption that the hidden activations of NNs carry information useful to anomaly detection, is well supported. The test results from experiment 1 show that our method identifies suitable patterns for this task very well. These patterns generalize well to yet unseen patterns as shown in experiment 2. Although only parts of the test anomalies are known during training, strong results are achieved. Whereas we used autoencoders as targets for the aforementioned experiments, we generalized the setting to classifiers in experiment 3. A reliable anomaly detection was achieved. We conclude that A^3 is able to detect known and yet unknown anomalies with high confidence, and is flexible enough to adapt to a wide range of environments.

3.4.6 Discussion

In this section, we present A^3, an anomaly detection method that analyzes the activations of NNs. In A^3, the alarm model observes the behavior of a target model during run time. A^3 shows strong anomaly detection results for all five analyzed data sets across all experiments. We empirically show that the activation analysis generalizes to yet unseen anomalies across different network architectures. With the modularity of our concept, various architectures may be used as target and alarm models covering numerous types of data and use cases. We emphasize the real-life applicability by limiting the amount of anomaly samples during training. In practice, often a few known anomalies are available, e.g., by manual exploration or unsupervised methods.

For the interested reader we refer to the Appendix of this thesis for the remaining contributions of our original publication [1]. In Appendix A.2.1 we show an improved A^3 architecture allowing even higher anomaly detection scores. As we have shown with the results of experiment 2, a slight degradation of the performance occurs if new types of anomalies should be detected. By adding a third component to our target-alarm architecture we boosted the performance, especially in this demanding setting. A so-called anomaly network creates artificial anomalies during the training of the alarm model. With this process, the training set is enriched with valuable counterexamples. Using this improved architecture we achieved state-of-the-art results across all experiments and surpassed the baseline methods in the majority of test cases. Finally, we show first steps towards the applicability of the improved A^3 architecture in a fully unsupervised setting.

3.4.7 Conclusion

We introduce a novel approach for anomaly detection called A^3 based on the analysis of the hidden activation patterns of NNs. This finding directly answers **Q2** presented in Section 3.2.9. Our architecture comprises two parts: a target model unrelated to the anomaly detection task and an alarm model analyzing the resulting activations of the target. Our framework works under common assumptions and constraints typically found in anomaly detection tasks. We assume that anomalous training samples are scarce, and new types of anomalies exist during deployment. With our evaluation, we provide strong evidence that our method works on different target network architectures, and generalizes to yet unseen anomalous samples. Furthermore, we detect anomalies across different data types with a limited number of labeled anomalies available during training. We present a valuable semi-supervised DL-based anomaly detection framework providing a purely data-driven solution for a variety of use cases.

With the insights of this section at hand, an open questions arises which we answer throughout the remainder of this thesis: **Q3**: *How can the observation that the activation values of NNs carry information useful to passively detect adversarial examples and anomalies be further used in an active manner to increase the robustness of NNs?* Our presented passively operating methods allow a reliable detection of attacks, yet require an initial training phase prior to their deployment. Furthermore, in the case of a successfully detected attack, external measures are required to again allow a correct classification of the input. A solution here would be the deployment of an active defense method based on the already presented insights of this thesis.

Q3 is answered in Section 4.4.

4 Activation-Based Training Towards Robust Neural Networks

In this chapter, we present our robust training method for neural networks called entropic retraining. Parts of the section are taken verbatim from the original paper "Optimizing Information Loss Towards Robust Neural Networks" published in Dynamic and Novel Advances in Machine Learning and Intelligent Cyber Security (DYNAMICS) 2020 co-located with the Annual Computer Security Applications Conference (ACSAC) [10].

4.1 Motivation

In the previous chapter we introduced our adversarial example detection method DLA based on the analysis of the hidden activations of NNs. Our modular approach enabled us to further refine and extend the initial ideas to introduce DA3G, an adversarial example detection technique using the analysis of gradients, as well as A^3, a generally applicable anomaly detection method. In this chapter, we revisit our findings on the analysis of the hidden activations of NNs and change perspectives: Instead of passively making pretrained NNs more robust by detecting attacks, we now focus on the training process itself. With the concepts presented in this chapter we answer **Q3**: How can the analysis of NN activations be used in an active manner complementary to our findings shown in Chapter 3. Based on state-of-the-art research in adversarial training and new advances in information-theoretic methods to analyze NNs we present our training approach that we call entropic retraining.

Adversarial training is still considered to be the most effective measure to enhance the robustness of NNs in an active manner. Here, the training set is extended by adversarial examples. First findings by Madry et al. [19] suggest that using PGD to generate samples for training produces the most robust models. Yet, additional research is required to further improve adversarial training and explore approaches to overcome its natural downsides, especially the intensive process of generating additional training samples and the negative effects on the resulting performance of the hardened NNs. Important first steps in this direction were presented by Shafahi et al. [135] and Wong et al. [34]. Shafahi et al. [135] showed that high levels of robustness can be achieved with adversarial-example-free training procedures, while Wong et al. [34] revisited adversarial training using FGSM as the generating method. Both methods shed more light on the well known robust training method and further try to increase the efficiency and thus reduce the computational cost of making NNs more secure. In a similar spirit, we present a new information-theoretic-inspired robust training approach for NNs.

Motivated by our findings on the hidden activations of NNs presented in Chapter 3, we analyze whether similar strategies may lead to new findings on robust training methods. We have seen that the activation values of NNs carry robustness-sensitive information which were useful in the detection of anomalous data samples and even adversarial examples. Therefore, in this chapter, we first shed light on adversarial training and analyze its impacts on NNs. We fuse insights from two intriguing research directions which gained attention in recent years: activation analysis [1,9] and information-theoretic concepts to understand NNs [136–140]. Combining these two worlds reveals new insights in how NNs behave under different circumstances and types of inputs. This fusion leads to a new training approach, which we call entropic retraining (ER). We empirically show that this training procedure improves the robustness of NNs against gradient-based attacks compared to standard training. Interestingly, this approach does not require adversarial examples and works for various NN architectures and data sets using the original input data only.

In summary, in this chapter we present the following contributions:

- We analyze the impacts of adversarial training on NNs using information-theoretic-inspired approaches.
- We leverage the knowledge gained during this analysis and present a new training method that we call entropic retraining.
- We implement and evaluate our new training method and show a significant improvement in robustness for various NN architectures and data sets.

This chapter is structured as follows: In Section 4.2 we present related work and the fundamental background information needed in this chapter. This includes an overview of information-theoretic tools used in DL, as well as adversarial training and the notion of activation analysis introduced in the previous chapter. In Section 4.4 we present entropic retraining, our robust training method inspired by recent advances in making adversarial training more efficient and less computationally expensive. We show the considered threat models in Section 4.3, the experimental setup in Section 4.5, and finally our thorough evaluation using different NN models and data sets in Section 4.6. We further investigate the impacts of Entropic Retraining on NNs and discuss potential pitfalls in Section 4.7. Here, we shed more light on the inner workings of our robust training approach and show that further research is required in the field of information-theoretic approaches for analyzing NNs.

4.2 Related Work

Robust Training Approaches in Deep Learning

In Section 2.2.3.1 we introduced adversarial training as the current state-of-the-art defense strategy to make NNs more robust. Madry et al. [19] further improved initial findings and formalized adversarial training using their PGD attack as a min-max problem. Even though adversarial training is well researched and evaluated, some aspects still remain unclear: for instance, the full impact on the protected NNs and how to further improve the concept of adversarial training in general. The trade-off between the resulting level of robustness and the final accuracy of the model is known, but yet to be fully understood [141]. Furthermore, the adequate setting of hyperparameters poses an uncertainty for developers trying to protect their NNs using adversarial training. Here, the number of epochs, learning rate, or attack parameters need to be defined and are often set following a laborious empirical approach. Recent work by Shafahi et al. [135] showed that adversarial training can be further improved and even performed at reduced cost. The authors reuse the gradient information available during the updates of the model parameters in normal training. With this approach, the authors achieve a similar level of robustness compared to PGD-based adversarial training, without the generation of adversarial examples. Similarly, Wong et al. [34] revisited adversarial training using FGSM as the generating method. The authors were able to achieve similar levels of robustness compared to PGD-based training yet profiting from the significantly faster adversarial example generation. Still, to shed more light on adversarial training, we inspect NNs using information-theoretic-inspired approaches at different robustness levels.

Information-Theoretic Approaches in Deep Learning

In this paragraph, we briefly present the most related publications on information-theoretic approaches to study NNs. We focus on work improving the understanding of NNs in general and the robustness against adversarial examples in particular. In 2000, Tishby et al. [142] introduced the information bottleneck (IB) method solving the problem of distributional clustering. The findings were further refined, by, among others, Tishby et al. [136], and used to analyze the training of NNs. Shwartz-Ziv and Tishby [137] introduced their so-called information planes as a metric describing the information flow during NN training. The authors analyze the mutual information between the layers and make the following observations: First, NNs perform a compression of the inputs to a compact representation. Then, by neglecting or *forgetting* parts of the compressed information, NNs achieve better generalization and overall performance. In 2019, Achille et al. [143] investigated NNs using information-theoretic approaches as well and made similar observations. The authors find that models which generalize well, often show a low level of information.

Based on these findings and concepts, we investigate the impacts of adversarial training on NNs. Closely related in this regard are the contributions by Terzi et al. [140] published in 2020, showing that adversarial training reduces information in NNs. Using this insight, the authors empirically demonstrate that robust models show a better transferability characteristic compared to their unsecured counterparts for the CIFAR10 and CIFAR100 data sets. In this chapter, we profit from this finding and present a new training approach that we call entropic retraining.

4.3 Considered Threat Model

Based on our general introduction to threat models in adversarial machine learning presented in Section 2.2.1, in this section we summarize the considered attacker capabilities for this chapter. We generally assume untargeted attacks. From a defender-perspective, this poses a more challenging task compared to targeted attacks. In general, untargeted attacks should strictly be easier compared to selecting one specific target class the decision of the NN should be altered to [14]. Therefore, by assuming this type of attack, we expect that our experiments as well as the results can be directly transferred to the targeted attack setting. Furthermore, we solely utilize attack methods which are bounded by the l_2 and l_∞ norm. Finally, we assume attackers with white-box access to the model under attack. Hence, we use specifically crafted adversarial examples to provoke our models to wrongly classify the inputs. Even though we assume the attackers to be aware of our modified training method, further attack methods targeting the information-theoretic properties of the models under attack are out of scope in this thesis and left for future work.

4.4 Entropic Retraining: Activation-Based Training Towards Robust Neural Networks

In this section, we introduce our main contribution of this chapter: a modified and robust training approach we call entropic retraining. Based on our adapted loss function and the optimization goal we show in Section 4.4.3, ER does not rely on the time consuming and computationally expensive process of generating adversarial examples.

First, we evaluate adversarial training and the behavior of the resulting robust NNs. For this purpose, in Section 4.4.1 we introduce our evaluation metrics inspired by information theory and perform a preliminary analysis. We then transfer our findings to an adversarial-example-free case and introduce ER.

4.4.1 Concept and Preliminary Analysis

We assess the impact of adversarial and entropic training using two information-theoretic-inspired values. Mainly, we were inspired by the classical Shannon entropy [144] defined as:

$$H(\mathbf{x}) = -\sum_i p(x_i)\log(p(x_i)). \tag{4.1}$$

In this paper, we directly use the hidden activation values of each layer of the NN. More precisely, we extract the activation values $\mathbf{h}_i$ of each layer $i \in 1,..,M$ and calculate $E(\mathbf{h}_i)$ as follows:

$$E(\mathbf{h}_i) = -\sum_{j=1}^{N_i} h_j \log(h_j), \tag{4.2}$$

with N_i being the number of neurons in layer i. To finally calculate the network-entropy-related value E_N we first normalize $E(\mathbf{h}_i)$ with respect to the number of neurons in the layer and then average the result with respect to the number of layers in the NN:

$$E_N = \frac{1}{M}\sum_{i=1}^{M}\frac{1}{N_i}E(\mathbf{h}_i). \tag{4.3}$$

Moreover, we introduce and evaluate E_T, which layer-wise quantifies the differences in $E(\mathbf{h}_i)$ and, thus, allows an evaluation of the information-related flow in the NN. More formally, we define E_T as follows:

$$\begin{aligned} E_T &= \frac{1}{M-1}\sum_{i=1}^{M-1} E(\mathbf{h}_i) - E(\mathbf{h}_{i+1}) \\ &= \frac{1}{M-1}(E(\mathbf{h}_1) - E(\mathbf{h}_M)) \end{aligned} \tag{4.4}$$

Note that we do not normalize $E(\mathbf{h}_i)$ to calculate E_T. We assume that non-normalized values provide a more useful picture of the information-related flow in this case. This assumption is based on the following intuition: During the adversarial training process and while solving the min-max problem introduced in Section 2.2.3.1, some of the weights might be set to negligible magnitudes. This would lead to neurons with insignificant impacts on the level of activation in the analyzed layer. Hence, normalizing $E(\mathbf{h}_i)$ would hide this effect. Our intuition is inspired by the contribution of Ilyas et al. [145]. The authors state that models are required to *forget* some of the non-robust features to be more robust. This correlates to neglecting the influence of the neurons which are trained to process these non-robust features. To capture this effect and the overall impact of adversarial training on the NNs, we do not normalize the components of E_T.

In summary, with our newly introduced values E_N and E_T at hand we evaluate the information-theoretic-related properties of the NNs during input processing.

4.4.2 Impacts of Adversarial Training

To visualize the impact of adversarial training, we considered the following settings and model under test (MUT). We used a pretrained LeNet5 CNN [87] model. This architecture is designed to classify the simple and widely used MNIST data set. For the sake of readability, we introduce the architecture and data set in more detail in Section 4.5.2.

After every fifth epoch of adversarial training using the PGD attack, we performed forward passes using original inputs drawn from the test set and calculated E_N and E_T. This allows us to visualize the evolution of both E_N and E_T, based on changes made to the NN due to adversarial training. We adversarially trained the MUT for 200 epochs in total.

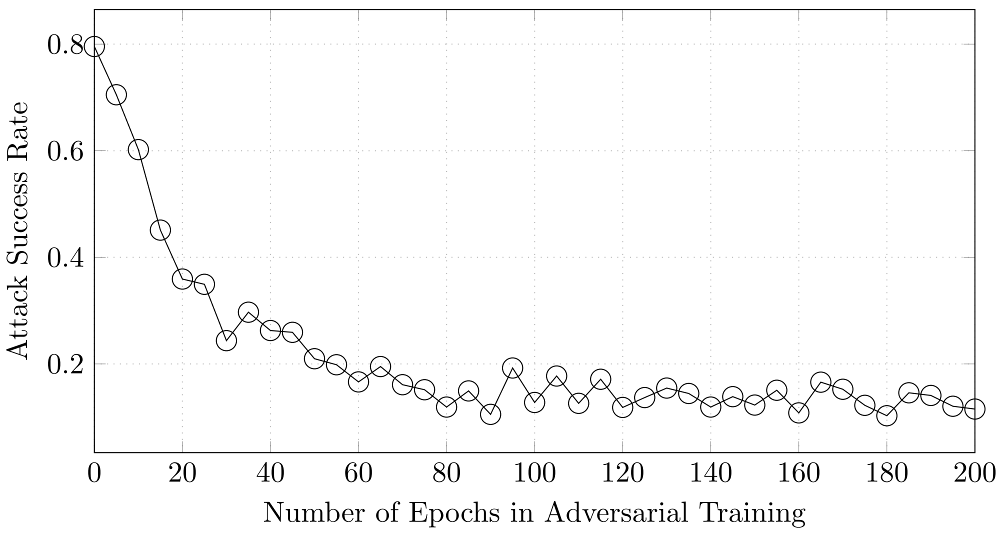


Figure 4.1: Evolution of the robustness during PGD-based adversarial training of the LeNet5 model classifying MNIST when attacked using FGSM.

In Figure 4.1, we show the resulting attack success rate for each instance of the MUT. To quantify the model's level of robustness, we generated l_∞-bounded ($\epsilon = 0.3$) adversarial examples using FGSM with images randomly drawn from the test set. As expected, for a higher number of epochs in adversarial training we see a decrease in the attack success rate, indicating a robustification of the MUT. With Figures 4.2 and 4.3, we visualize the impact of adversarial training on the same instances of the MUT using the previously introduced E_N and E_T. In Figure 4.2, we see a decrease in E_N allowing the following conclusion which strongly corresponds to the contribution of Terzi et al. [140]:

Adversarial training decreases E_N.

This effect is already visible after five epochs of adversarial training. Furthermore, a strong correlation between the evolution of the attack success rate and E_N is given. As stated before and corresponding to previous work by Ilyas et al. [145] and Terzi et al. [140], adversarial training increases the model's robustness by boosting the generality of the MUT at the cost of its initial accuracy shown by Zhang et al. [146].

With Figure 4.3, we report a similar effect when observing E_T. The plot shows that E_T increases with a higher level of robustness. Together with the previous observation of decreasing attack success rates, this suggests a negative correlation between E_T and the level of robustness achieved during adversarial training. This effect is even more visible for robust models, which brings us to our second observation:

Adversarial training increases E_T.

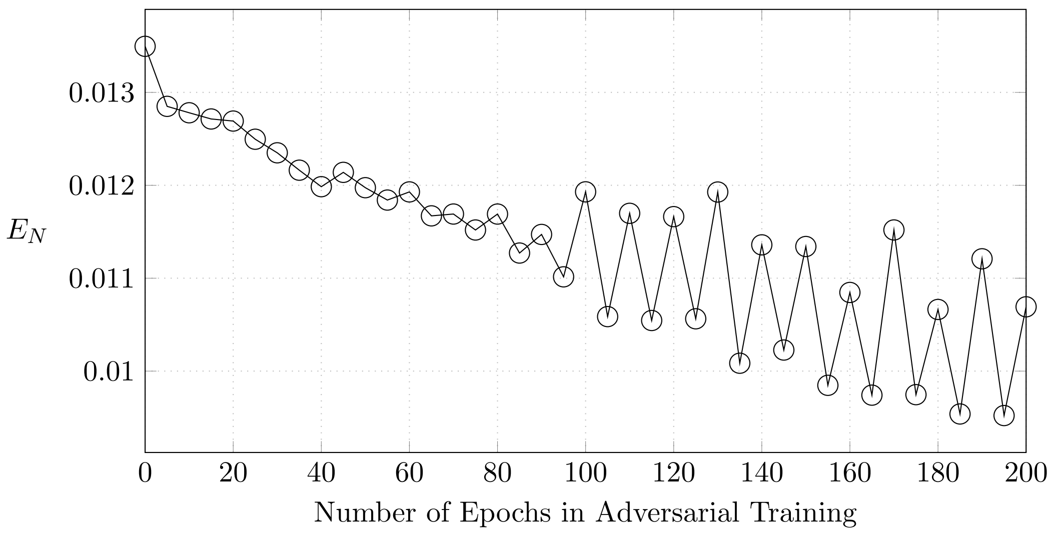


Figure 4.2: E_N for the LeNet5 model classifying the MNIST data set during PGD-based adversarial training.

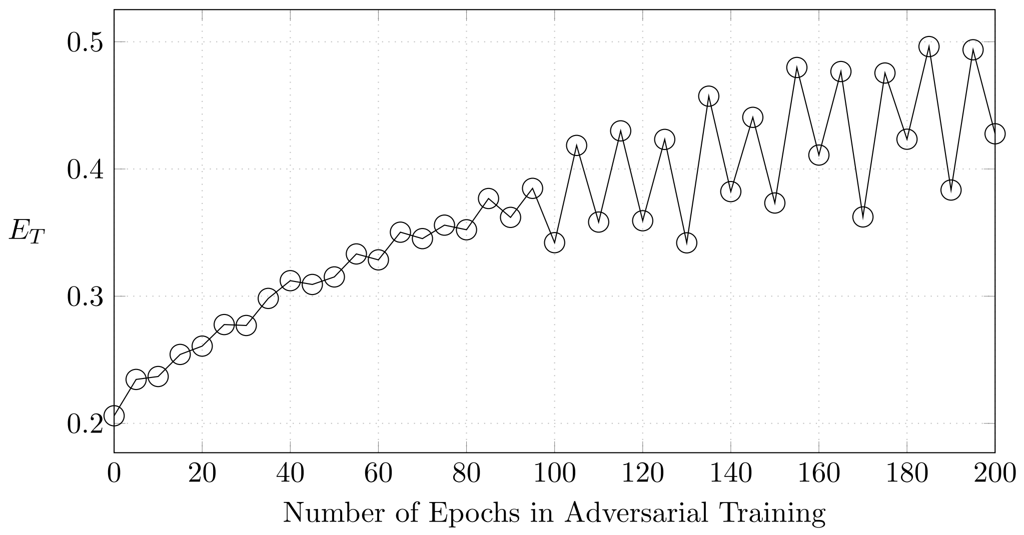


Figure 4.3: E_T for the LeNet5 model classifying the MNIST data set during PGD-based adversarial training.

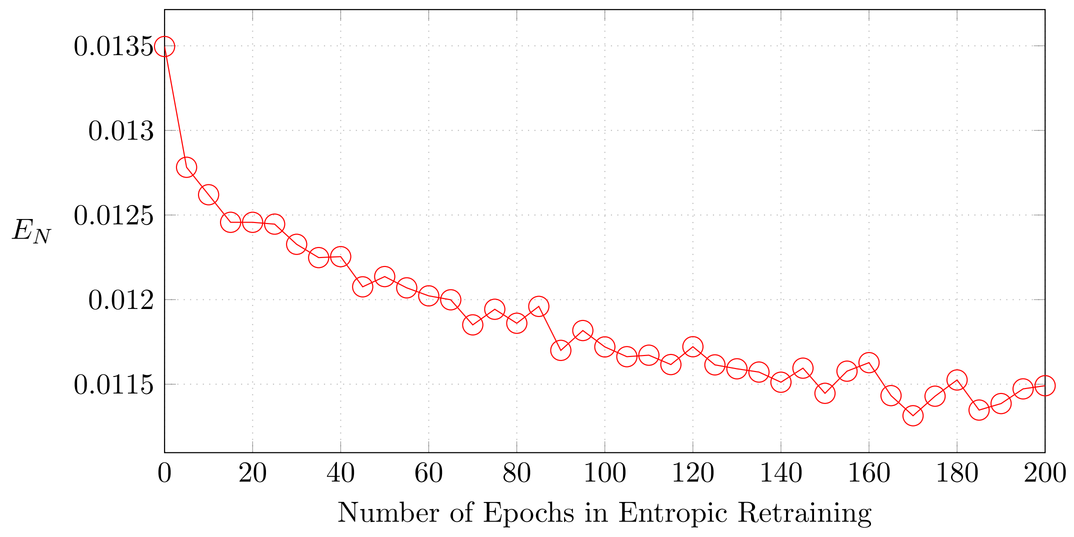


Figure 4.4: E_N for the LeNet5 model classifying the MNIST data set during entropic retraining.

4.4.3 Entropic Retraining Objectives

With the above figures and observations at hand, we present the concept of ER. Above, we observed that adversarial training yields NNs with lower levels of E_N and higher levels of E_T compared to their normally trained initial instances. Following this insight, we incorporate E_T in the training process and present an adapted loss function. Thus, during entropic retraining, we optimize the performance of the model while simultaneously increasing E_T. Formally, we define our adapted loss function as follows:

$$\mathcal{L}_{tot} = \mathcal{L}_{scce}(\mathbf{y}, \hat{\mathbf{y}}) + \lambda \cdot \mathcal{L}_T(E_T, \hat{E}_T) \tag{4.5}$$

with

$$\mathcal{L}_{scce}(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{i=1}^{N} y_i log(p_{\hat{y}_i}) \tag{4.6}$$

being the standard sparse categorical cross-entropy and

$$\mathcal{L}_T(E_T, \hat{E}_T) = (\hat{E}_T - \sigma E_T)^2 \tag{4.7}$$

being our newly introduced optimization term. Here, E_T is the value of the original pretrained model, while $\hat{E}_T$ is calculated during entropic retraining. The factors $\lambda, \sigma \in \mathbb{R}^+$ are positive hyperparameters set in the optimization process before training. With our simple extension of the original loss function, ER can be easily performed using standard NN training approaches. As in standard training, we aim to decrease the current loss by adapting the weights of the NN accordingly.

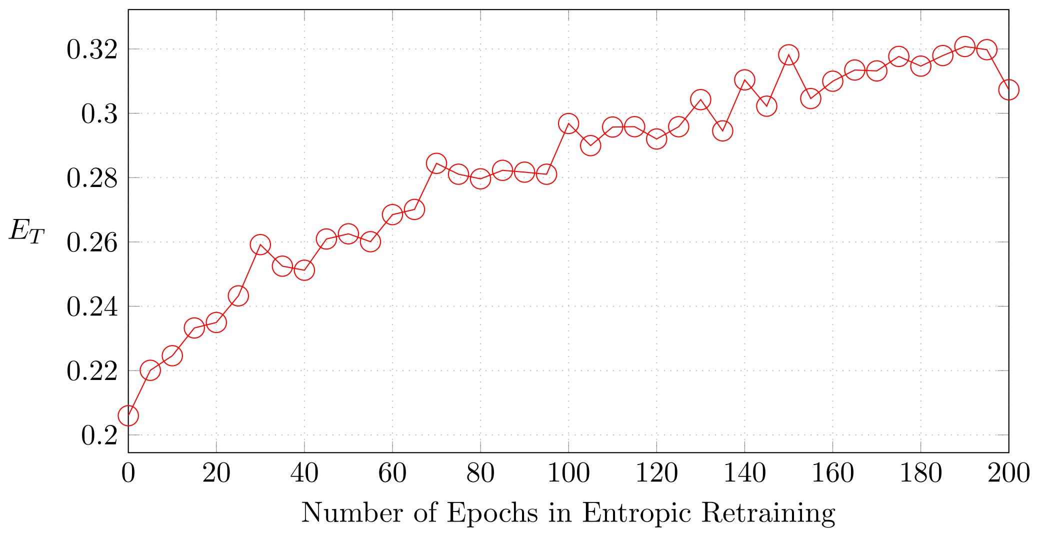


Figure 4.5: E_T for the LeNet5 model classifying the MNIST data set during entropic retraining.

4.4.4 Impacts of Entropic Retraining

In this section, we show the impacts of ER with the following experiment: We performed ER with the same pretrained MUT from above, based on the LeNet5 architecture. During this process, we calculated E_N and E_T. To assess the achieved level of robustness after every fifth epoch, we attacked the resulting instances of the MUT using FGSM and measured the achieved attack success rates. In Figures 4.4 and 4.5, we show the evolution of E_N and E_T, respectively. Compared to adversarial training, we see the same effects for the resulting model instances. With our experiments we make the following observation, which is closely related to adversarial training:

Entropic retraining decreases E_N.

Moreover:

Entropic retraining increases E_T.

This observation suggests that ER has similar effects on E_N and E_T as adversarial training. Remarkably, we achieve this solely using the original inputs without the laborious generation of adversarial examples.

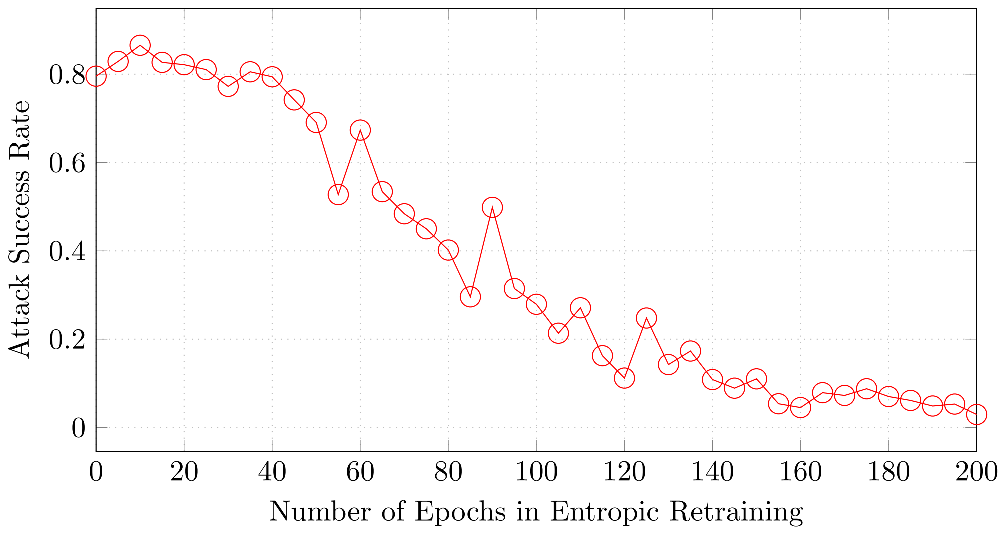


Figure 4.6: Evolution of the robustness during entropic retraining of the LeNet5 model classifying MNIST when attacked using FGSM.

In Figure 4.6, we show the attack success rates for the model during ER. Again, the same effect as in the previously performed adversarial training experiment is visible. The attack success rate decreases with the number of performed ER epochs. With this observation, we provide first strong indications that an analysis of the NN activations in an active manner provides means to increase the robustness of the models. More specifically, investigating the information-theoretic behavior of NNs gives insights on their level of robustness. By simultaneously optimizing for accuracy and a higher level of E_T, we achieve a robustification of the MUT.

4.5 Experimental Setup

In the following, we empirically confirm our initial observation by extending the proof of concept with a thorough evaluation. Hence, we introduce details on our implementation, considered data sets, and evaluated NN architectures. For our prototype implementation, we used Keras [131] and ran the training procedures on an Intel Xeon E5-2640 v4 server with an NVIDIA Titan X GPU.

4.5.1 Data Sets

We used the MNIST [83], Fashion-MNIST [103], and CIFAR10 [84] image data sets. Further information on the data sets can be found in Section 3.2.5.1.

4.5.2 Target Models

Throughout our evaluation we considered four different NN architectures. We are able to show a robustness increase for all of them. For the sake of comparability and in order to ease reproducibility, we utilized publicly available and thoroughly evaluated architectures. For MNIST, we considered the LeNet5 architecture [87] and an example model provided by Keras [88]. Similarly, for Fashion-MNIST and CIFAR10, we used an example model provided by Keras [90, 147] for each of the data sets. In Table 4.1, we summarize the considered NNs and show details regarding their architectures and the executed training processes. During our experiments with DLA and DA3G, we used similar target architectures. We therefore use the same model names. Note that we did not use the identical models and used newly trained ones throughout this section. For the Keras example models classifying MNIST and CIFAR10 images, we slightly changed the architectures and neglected the drop-out layers.

4.5.3 Attack Methods

In accordance with common guidelines by Carlini et al. [14] and our remarks on attack methods presented in Section 3.3.5.3, during the evaluation of entropic retraining we used the following methods: PGD [19], DeepFool (DF) [20], and C&W [22]. For PGD and DF, we considered both the l_2 and l_∞ versions of the attacks. For the C&W attack we solely used the l_2 version. We motivate our choice of attacks based on their performance and, as stated above, based on their underlying principles. This provides a diversely executed assessment of the quality of our proposed countermeasure. In summary, we again considered bounded gradient-based multi-step attacks (PGD), decision-based (DF), as well as optimization-based (C&W) approaches. We used the Foolbox framework [96] to generate the adversarial examples. For MNIST, we set ϵ to 4.5 and 0.3 for l_2 and l_∞ bounded attacks, respectively, while we set the values to 1.5 and 0.1 for the Fashion-MNIST data set. Similarly, for CIFAR10 we limited ϵ to 0.95 and 0.03 for l_2 and l_∞ bounded attacks, respectively.

4.5.4 Entropic Retraining Settings

In Table 4.2, we summarize the settings we chose for entropic retraining of the four MUTs. Setting the parameters, we followed an intuitive approach and did not perform an extensive search. Hereby, we want to show the simplicity of our approach underlining the ease of use. We used the Adam optimizer to retrain the four models and chose similar settings in each case. As the gradient calculation during entropic retraining poses a rather complex task, we used small batches. With this, we aim to avoid sharper minimizers during training, which would lead to a decrease in generalization as shown by Keskar et al. [148]. For the MNIST and Fashion-MNIST models we chose a batch size of four, while the CIFAR10 model was trained with a batch size of 64. Accordingly, we set the number of epochs to 200 and 500 for the (Fashion-)MNIST and CIFAR10 experiments, respectively. Finally, we set the weighting factor σ based on the observations shown in Section 4.4.2. As we want to increase E_T, we chose $\sigma > 1$ for all experiments.

Table 4.1: Target models we used during the proof-of-concept implementation for entropic retraining. We give a short summary of each model's architecture as well as the used training settings and resulting test accuracy.

	Model	Model Details	Training Settings
MNIST	LeNet5 [87]	– 2 convolutional layers with filter sizes 5 and 16 – each convolutional layer is followed by an average-pooling layer – finally, 3 dense layers with 120, 84, and 10 neurons each – no drop-out layers	– opt.: Adam – l.r.: 0.001 – epochs: 24 – batch size: 64 – test acc.: 99.1%
MNIST	KerasExM [88]	– 2 convolutional layers with filter sizes 32 and 64 – followed by 1 max-pooling layer – finally, 3 dense layers with 512, 84, and 10 neurons each – no drop-out layers	– opt.: Adam – l.r.: 0.001 – epochs: 12 – batch size: 128 – test acc.: 99.1%
Fashion-MNIST	KerasExF [147]	– 2 convolutional layers with filter sizes 32 and 64 – followed by 1 max-pooling layer – finally, 2 dense layers with 256 and 10 neurons each – no drop-out layers	– opt.: Adam – l.r.: 0.001 – epochs: 10 – batch size: 64 – test acc.: 91.7%
CIFAR10	KerasExC [90]	– 4 convolutional layers, the first two with filter of size 32, the second pair with filter size of 64 – each pair of convolutional layers is followed by a max-pooling layer – finally, 6 dense layers with 512, 256, 128, 128, 84, and 10 neurons each – no drop-out layers	- opt.: RMSprop – l.r.: 0.0001 – epochs: 50 – batch size: 32 – using data augmentation – test acc.: 81.7%

4.5.5 Experiment Overview

In summary, we performed the following experiments to present the performance of our new robust training method.

First, as a baseline we show the properties of our normally trained models presented in Table 4.1 in more detail in Section 4.6.1. This includes an analysis of the attack-free case when classifying unaltered benign inputs and noisy inputs. Furthermore, we show the robustness of the baseline models against our selection of attack methods. With these results at hand, in the subsequent sections we are able to quantify the impacts

Table 4.2: Entropic retraining settings for the considered neural networks.

Data	**Model**	**Entropic Retraining Settings**
MNIST	LeNet5	– optimizer: Adam – learning rate: 0.0009 – epochs: 200 – batch size: 4 – λ: 1; σ: 1.65
	KerasExM	– optimizer: Adam – learning rate: 0.001 – epochs: 200 – batch size: 4 – λ: 1; σ: 1.65
Fashion-MNIST	KerasExF	– optimizer: Adam – learning rate: 0.001 – epochs: 200 – batch size: 4 – λ: 1; σ: 1.5
CIFAR10	KerasExC	– optimizer: Adam – learning rate: 0.0005 – epochs; 500 – batch size: 64 – λ: 1; σ: 1.01

of entropic retraining more accurately. To create the noisy input images we again used the Foolbox framework and generated two sets of test images. For the three data sets, Foolbox generated noise which is added to the original input images. Here, we chose the noise to be either l_2 or l_∞ constrained. This allows a direct comparison to the performance of the NNs when classifying adversarial examples bounded by the same distance metrics. Hence, in the MNIST setting, for the l_2 and l_∞ constrained noise generation we set ϵ to 4.5 and 0.3, respectively. For Fashion-MNIST ϵ was set 1.5 and 0.1. Similarly, in the CIFAR10 setting we chose ϵ to be 0.95 and 0.03.

In the second part of the experiment summarized in Section 4.6.2, we first show the standard performance, i.e., the accuracy of our models after entropic retraining, in Section 4.6.2.1. Here, we investigate whether the use of our robust training approach degrades the accuracy and generalization of the MUTs. Finally, in Section 4.6.2.2 we show the robustness of the MUTs after entropic retraining. This is the main experiment of this chapter. Throughout this experiment, we did not change any parameters unrelated to entropic retraining compared to the baseline experiments using the original models. We divided this experiment into two parts and first assessed the performance of NNs with softmax-activated outputs before we collected the results for the models with linear outputs. In our original paper [10], the robustness of the neural networks was exclusively shown for NNs with softmax-activated outputs. As the calculation of

Table 4.3: Standard accuracy scores of the original neural networks. Here, identical results are achieved for softmax-activated and linear outputs.

Data	**Model**	**Tested with:**		
		Original	l_2***Noise***	l_∞***Noise***
MNIST	LeNet5	99.1%	98.9%	98.8%
	KerasExM	99.1%	98.7%	98.6%
F.-MNIST	KerasExF	91.7%	90.0%	89.6%
CIFAR10	KerasExC	81.7%	80.9%	80.8%

the gradients needed during the attacks may be critical for softmax-activated outputs, in this thesis, we additionally show the robustness for linear outputs. This provides a clearer picture of our method's performance and, for future work, allows an easier comparison to new methods.

4.6 Evaluation

In this section we present the results of our experiments using entropic retraining. We divide the evaluation into two parts according to our experiment overview from Section 4.5.5. First, we show the results for the baseline models before we present the results for the models after ER.

4.6.1 Baseline Model Properties

For the following results, we used the pretrained models introduced in Table 4.1.

4.6.1.1 Performance of the Baseline Models

We observed the models' performance classifying the benign original data, as well as noisy input images. With this experiment we evaluated whether entropic retraining increases the sensitivity of the MUTs towards slight but unintentional perturbations. In Table 4.3 we summarize the resulting accuracy values. For the three data sets and four MUTs, we observed that noisy images were classified with a similar accuracy compared to their original counterparts. The impact of noise on the performance of all MUTs was negligible as it decreased the accuracy scores by less than 5%. We conclude that the performance of the originally pretrained MUTs is robust towards noise. This suggests that the considered target models are a viable choice for our experiments.

4.6.1.2 Robustness of the Baseline Models

Next, we evaluated the robustness of the pretrained MUTs by performing attacks with the three introduced methods. As shown in Section 2.2.4, we quantify the NNs' robustness by evaluating the attack success rate for PGD while showing the required mean

Table 4.4: Attack success rates for PGD and mean l_p-distances for DF and C&W when attacking the original neural networks with softmax-activated outputs.

Data	**Model**	**Attack Success Rate or Mean l_p-Distance:**				
		$\boldsymbol{PGD_\infty}$	$\boldsymbol{PGD_2}$	$\boldsymbol{DF_\infty}$	$\boldsymbol{DF_2}$	$\boldsymbol{C\&W_2}$
MNIST	LeNet5	100.0%	99.2%	0.29	4.31	1.72
	KerasExM	99.8%	99.5%	0.29	4.28	1.63
Fashion-MNIST	KerasExF	100.0%	99.6%	0.07	1.04	0.38
CIFAR10	KerasExC	100.0%	99.8%	0.02	0.50	0.21

Table 4.5: Attack success rates for PGD and mean l_p-distances for DF and C&W when attacking the original neural networks with linear outputs.

Data	**Model**	**Attack Success Rate or Mean l_p-Distance:**				
		$\boldsymbol{PGD_\infty}$	$\boldsymbol{PGD_2}$	$\boldsymbol{DF_\infty}$	$\boldsymbol{DF_2}$	$\boldsymbol{C\&W_2}$
MNIST	LeNet5	99.9%	99.1%	0.13	1.63	1.27
	KerasExM	99.8%	99.5%	0.14	1.63	1.24
Fashion-MNIST	KerasExF	100.0%	99.6%	0.02	0.35	0.31
CIFAR10	KerasExC	100.0%	99.7%	0.01	0.21	0.19

distortion for DF and C&W. We solely used images which are originally classified correctly by the MUTs to generate adversarial examples. In the Tables 4.4 and 4.5, we summarize the results for the models with softmax-activated and linear outputs, respectively. With our experiments, and as also shown in previous work, we see that our unprotected models were highly vulnerable. For the four MUTs, PGD achieved nearly perfect scores for all experiments indicating the low level of robustness in this case. For both, the models with softmax outputs as well as with linear outputs, we see minor differences in the success of the PGD attack. All attacks achieved a success rate of over 99%. For the unbounded DF and C&W attacks, we report a notable difference in achieved attack success depending on the output type. In our experiments we saw that attacking the models with softmax-activated outputs requires significantly higher levels of perturbations to fool the targets. This effect is especially striking for the DF_2 attack. For the remainder of this chapter, we therefore focus on the robustness of the linearly outputting models. The interested reader may find the results for the models with softmax outputs in Appendix A.3.1.

Table 4.6: Standard accuracy scores of the neural networks (linear outputs) after entropic retraining.

Data set	**Model**	**Tested with:**		
		Original	l_2–***Noise***	l_∞–***Noise***
MNIST	LeNet5	97.7%	95.8%	95.7%
	KerasExM	99.1%	98.4%	98.3%
F.-MNIST	KerasExF	90.9%	88.9%	89.0%
CIFAR10	KerasExC	85.0%	84.1%	84.2%

4.6.2 Model Properties after Entropic Retraining

In the following we show the properties of our models after entropic retraining.

4.6.2.1 Performance of the Models after Entropic Retraining

Table 4.6 displays the accuracy scores of the four MUTs when classifying original as well as noisy inputs. This table complements the results summarized in Table 4.3 which focuses on the baseline models. LeNet5's performance decreased by 1% when classifying the original inputs. When classifying noisy inputs, the accuracy scores decreased by 3% for both, the l_2 and l_∞ constrained cases. For both Keras example models, operating on the MNIST and Fashion-MNIST data sets, entropic retraining had a negligible impact on the standard accuracy scores. In summary, for all three test cases and each model, the accuracy decreased by less than 1%. Interestingly, for the CIFAR10 MUT we report an increase in accuracy of 4% when classifying the original inputs. The same increase can be seen for noisy input samples. Here, again, the model performed 4% better for the l_2 and l_∞ constrained examples.

In summary, we conclude that entropic retraining did not decrease the natural performance of the models and in some cases even allowed higher accuracy values. Furthermore, the robustness against noise did not decrease and the MUTs achieved similar levels of reliability compared to their initial versions.

4.6.2.2 Robustness of the Models after Entropic Retraining

In the following we present our results on the achieved levels of robustness for our four MUTs after entropic retraining. As mentioned above, we focused on the models with linear outputs and show the according results in Table 4.7. This consists of the attack success rates and mean distortions when attacking our protected models. Additionally, in Table 4.8 we show the differences of the robustness metrics between the unprotected models as well as the models after entropic retraining. This allows a direct assessment of the impact of entropic retraining and emphasizes potential strengths and weaknesses of our method. Again, for the interested reader, we show the according results for softmax NNs in Appendix A.3.1.

Table 4.7: Attack success rates for PGD and mean l_p-distances for DF and C&W when attacking the neural networks after entropic retraining with linear outputs.

Data	**Model**	**Attack Success Rate or Mean l_p–Distance:**				
		$\boldsymbol{PGD_\infty}$	$\boldsymbol{PGD_2}$	$\boldsymbol{DF_\infty}$	$\boldsymbol{DF_2}$	$\boldsymbol{C\&W_2}$
MNIST	LeNet5	18.1%	8.3%	0.09	0.57	0.43
	KerasExM	2.4%	1.8%	0.09	1.01	0.79
Fashion-MNIST	KerasExF	24.8%	16.5%	0.02	0.21	0.18
CIFAR10	KerasExC	85.2%	73.5%	0.01	0.22	0.17

Table 4.8: Relative change in robustness after entropic retraining for the models with linear outputs. Black and gray digits indicate a positive or negative impact on the robustness, respectively.

Data	**Model**	**Attack Success Rate or Mean l_p–Distance:**				
		$\boldsymbol{PGD_\infty}$	$\boldsymbol{PGD_2}$	$\boldsymbol{DF_\infty}$	$\boldsymbol{DF_2}$	$\boldsymbol{C\&W_2}$
MNIST	LeNet5	−82%	−92%	−29%	−65%	−66%
	KerasExM	−98%	−98%	−36%	−37%	−36%
Fashion-MNIST	KerasExF	−75%	−83%	−17%	−39%	−42%
CIFAR10	KerasExC	−15%	−26%	17%	6%	−11%

Robustness against PGD We observed an increase in robustness for all data sets and MUTs when attacked with PGD. For MNIST, only PGD_∞ achieved a success rate greater than 10% when attacking LeNet5. For the remaining scenarios, we saw success rates of well below 10%. In the case of the MUT classifying Fashion-MNIST examples, we also report a high level of robustness against PGD. Here, PGD_∞ and PGD_2 reached a success rate of 24.8% and 16.5%, respectively. Evaluating the CIFAR10 data set we see a decrease in the attack success rates as well. PGD_∞ and PGD_2 achieved success rates of 85.2% and 73.5%, respectively. This is a decrease in success of 15% for PGD_∞ and of 27% for PGD_2.

Robustness against DeepFool and the C&W attack For MNIST and Fashion-MNIST we observed a lower level of robustness against DF and C&W after entropic retraining. In this setting, both unbounded attacks required less perturbations to produce adversarial examples. The smallest degradation was observed for KerasExF when attacked with DF_∞. A 17% lower perturbation budget was needed to find adversarial examples compared to the normally trained models. We observed the highest negative impact for LeNet5 when attacked with C&W. Adversarial examples required 66% less perturbations to successfully fool the model. Interestingly, solely for our CIFAR10 model we observed a higher level of robustness against DF. Entropic retraining forced the attacker to add 17% and 6% more perturbations for DF_∞ and DF_2, respectively. For the C&W attack we saw a slight degradation in the robustness of 11%.

Summary In summary, with our evaluation we report an increase in robustness against the gradient-based PGD attack without loosing the capability of reliably classifying original as well as noisy inputs. In contrast to that, we report that entropic retraining did not increase the robustness of the models against decision-based and optimization-based attacks especially for MNIST and Fashion-MNIST. For CIFAR10 on the other hand, we report an increase of robustness against the gradient-based and decision-based attacks. Solely for the optimization-based C&W attack, a slight degradation of the robustness was observed.

4.7 Supplementary Analysis of Entropic Retraining

In Section 4.4.4, we show first intuitions on the impact of entropic retraining using E_N and E_T. With our thorough experiments and the accompanying evaluation in Sections 4.5 and 4.6, we empirically show the effects of entropic retraining on the robustness and overall performance of the protected NNs. In this section, we extend our analysis and further investigate the characteristics of the NNs during and after entropic retraining.

For this purpose, we revisit our experiments presented in Section 4.4.4 for the LeNet5 model classifying MNIST images. Here, we show the mean activation variance of the NN during forward passes using the test data, while neglecting the input layer, expressed with:

$$V_N = \frac{1}{M-1}\sum_{i=2}^{M} Var(\mathbf{h}_i). \tag{4.8}$$

Hence, V_N quantifies the mean layer-wise variance of the observed activation values and visualizes the activity of the NN during classification. We visualize V_N for the different instances of LeNet5 when classifying original MNIST images in Figure 4.7.

We see an increase of V_N during entropic retraining. This suggests that entropic retraining increases the mean variance in the layers' activations during input handling. To further explore this effect and to analyze the cause of the increased network variance, we introduce the activation heatmap of LeNet5 in Figure 4.8. Here, we analyzed the 84-neuron dense-layer of the MUT during input handling by calculating the mean activation values in each neuron for different levels of robustness. For this purpose, we used images drawn from MNIST with label 5 and again performed forward passes. The mean activation of each neuron of the analyzed layer is color-encoded visualizing its intensity. Brighter colors indicate a higher mean activation of the respective neuron. We found that entropic retraining increased the activation values of dense layers in the NN. Furthermore, the activity of a small number of neurons increased significantly more, compared to the remaining neurons.

For reference, in Figure 4.9, we show the activation heatmap for the same model and data during normal training. Here, we observe that the activations did not reach the same intensity as during entropic retraining. Moreover, we see a higher number of neurons with relatively high activations compared to the case of entropic retraining.

Combining the insights on the network variance and the activation heatmaps, we argue that fewer neurons in the layers contributed more to the final decision of our robust NNs

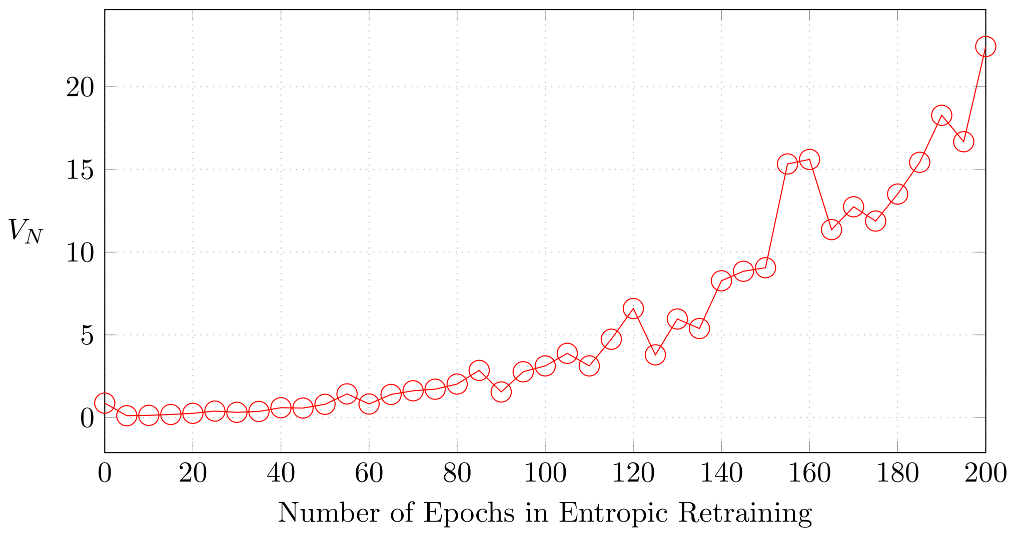


Figure 4.7: Evolution of the network variance during entropic retraining of the LeNet5 model classifying MNIST.

compared to their unprotected counterparts. In the unprotected case, the activation values were more evenly distributed and, therefore, contributed to a similar extent to the decision process. This suggests, that our robust models worked in a less sensitive manner compared to their unsecured counterparts.

4.8 Discussion

Our adapted loss function allows an information-theoretic lossy but partially robust training process for NNs. We show that entropic retraining generalizes well and achieves promising results when protecting against the bounded and gradient-based PGD attack for three data sets and different NN architectures. Here, entropic retraining reduced the achieved attack success rates without the use of adversarial examples during training. First experiments suggest that our robust NNs work in a less sensitive manner compared to their unprotected counterparts. Here, additional experiments revealing the full nature of the neglected information may contribute to the understanding of robust feature extraction. Especially investigating the question whether entropic retraining is conducive to obfuscated gradients poses an intriguing question for future work. Our results and the discrepancy between the protection against gradient-based attacks on the one side, and optimization and decision-based approaches on the other side points to this effect.

Throughout our experiments, we kept the parameter optimization process as short as possible. We used similar settings for all four MUTs to show the simplicity of our approach. A deliberate choice of parameters with respect to the chosen architecture and use case may further improve the results. For example, the number of epochs heavily influences the trade-off between robustness and accuracy.

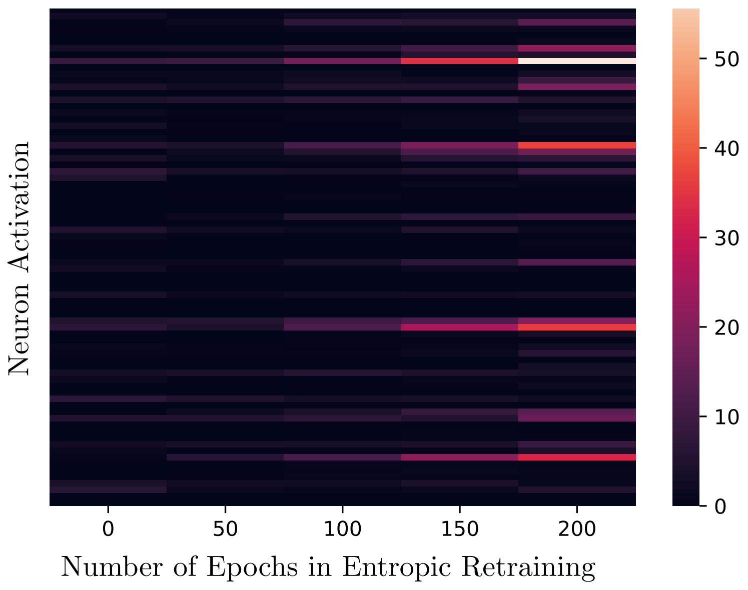


Figure 4.8: Visualization of the activation values of the LeNet5 model during entropic retraining. The activations where triggered by MNIST images with label 5.

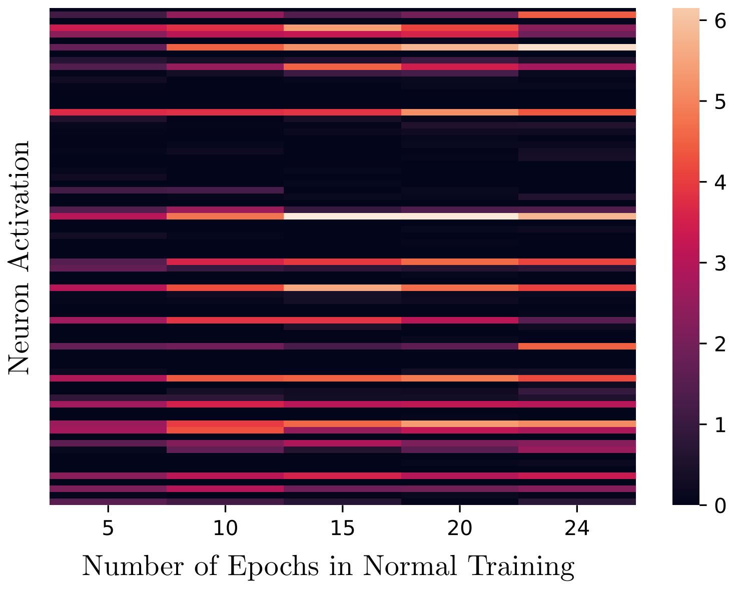


Figure 4.9: Visualization of the activation values of the LeNet5 model during normal training. The activations where triggered by MNIST images with label 5.

4.9 Conclusion

In this chapter we present, implement, and evaluate a new training approach for NNs that we call entropic retraining. Based on our information-theoretic-inspired analysis of adversarial training and the observed activation patterns, we gain new insights into the information flow of robust models. By adapting and expanding currently used loss functions accordingly, in entropic retraining, we optimize for accuracy and robustness simultaneously. With this approach, we train partially robust neural networks without the laborious generation of adversarial examples. Our evaluation clearly indicates the effectiveness of entropic retraining against PGD for multiple data sets and NN architectures. Furthermore, we answer **Q3** presented in Section 3.4.7 and show that our findings from Section 3.2 that the activation values of NNs carry robustness-sensitive information, can also be leveraged in an active manner.

5 Future Work

In Chapters 3 and 4 we presented our passive and active defense methods for NNs as well as an anomaly detection approach. After the presentation of each individual method we formulated local research questions which we answered in the subsequent chapters. These questions posed the next logical steps after the individual contributions and their impact in the field. In this chapter, we revisit open questions from a global perspective with respect to the findings of this thesis. This includes practical aspects as well as questions which partially exceed the research area of adversarial ML. We hope that this thesis sparks interest and paves the ground for future work on the analysis of the activations of NNs.

5.1 Selecting Information Sources

In our passive detection schemes the alarm model is responsible for assessing the behavior of the target model. This is done by either analyzing the target's activations or gradients. In this thesis, we present practical suggestions on how to select the layers based on which the information relevant for the alarm model is extracted. For this selection we consider the resulting performance of the detection schemes as well as computational resources available to the defender. Future work may build upon this and answer the question on how to select the layers the information should be extracted from, potentially from a general perspective. This may also include an extension of our target-alarm architecture. More complex systems consisting of a set of models require a deliberate choice of activation sources to be assessed, in order to allow an efficient detection of attacks or anomalies.

5.2 Understanding Influencing Factors for Robustness

We empirically show that the activation values of NNs carry robustness-sensitive information. Exceeding the benefits of passively analyzing the activation values in our detection schemes as well as actively influencing them in our presented training approach we see the following open question: How can the activation values, or more generally, all information available in the NNs be used to assess the robustness of the models themselves? This specifically includes, next to increasing robustness, measuring the level of protection against adversarial examples. As we have shown throughout this thesis, current methods to assess the robustness of NN-based systems consist of observing the attack success rates or the required perturbation budgets. New methods based on an analysis of the models themselves, e.g., by observing their activations values could provide means to quantify the robustness of NNs independently of specific attacks and according implementations.

5.3 Designing Trustworthy Neural-Network-based Systems

Finally, we broaden our view and discuss open questions from a more holistic perspective. NNs build the foundation of numerous systems based on ML. The characteristics of the deployed NNs are of crucial importance for the trustworthiness of the overall systems. Next to the robustness against adversarial examples, properties like fairness, privacy, or explainability need to be ensured. Through the insights gained in this thesis, we might be able to advance research in other domains beyond adversarial ML. By facilitating research on new methods enhancing the aforementioned properties, we believe that our contributions might have a positive impact on the trustworthiness of NNs and systems relying on those.

6 Conclusion

State-of-the-art neural networks are able to solve complex tasks in a wide range of domains. Typical applications range from object detection in the image domain to text-to-speech synthesis in the audio domain. The performance of the models is the main driving force behind the increasing number of systems based on neural networks. Due to their high number of parameters, approximating functions for high-dimensional data inputs becomes feasible. The availability of high-performance hardware as well as online training services further contributes to this effect. Despite their recent success and widespread use, neural networks are vulnerable to adversarial examples. During evasion attacks, adversaries induce slight changes to the inputs which alter the classification output of the attacked system. As the changes are humanly imperceptible, identifying adversarial examples as such is a challenging task. Furthermore, there is still no definitive answer to why adversarial examples exist and how to effectively protect models against attacks. Still, due to their performance, the number of security-sensitive systems relying on neural networks may significantly increase in the near future.

This motivated the work presented in this thesis resulting in the following main insight: The activations readily available during the input processing of neural networks carry robustness-sensitive information. Based on an analysis of the activations, we design passive as well as active defense strategies. This consists of attack detection methods shown in Chapter 3 and an activation-based training approach shown in Chapter 4.

Leveraging our main insight, we present a generally applicable and modular architecture allowing a reliable detection of adversarial examples. This architecture consists of two parts: the target neural network we try to protect and an alarm neural network performing the attack detection. In Section 3.2 we show that the alarm model trained on the target's activations is able to identify adversarial examples with high confidence. We show that our approach is applicable in image, NLP, and audio processing settings. Furthermore, our results strongly suggest that we cannot only detect state-of-the-art attacks, but also adversarial examples generated with future, yet unknown methods based on currently known paradigms.

Following the design of our modular approach, we answer the question whether this architecture can further be leveraged to contribute towards robust neural networks. To this end, in Section 3.3 we show that our architecture is useful not only using the models' activations but also using the targets' gradients. Here, we use the very same architecture yet replace the analysis of the activations with an assessment of the gradient. We therefore show that our architecture is useful even if a model's activations are not available. This shows the applicability of our concept for a wide range of neural-network architectures and setups.

Concluding our investigation of passive activation-based methods, in Section 3.4 we transfer our findings on adversarial example detection to the field of anomaly detection. Again using our modular architecture and deliberately choosing appropriate target models we are able to reliably detect anomalies while outperforming state-of-the-art methods. With this finding, we propose a new neural-network-based approach supporting data-processing systems by detecting anomalous samples. By reliably detecting anomalies, potential malfunctions of the system or even intrusions can be identified.

Complementary to our detection methods, in Section 4.4, we present an activation-based training method for neural networks. We analyze the activation values exhibited during training and incorporate the insights into the optimization process of the models. This active defense strategy may act as an additional layer of protection for our target-alarm architecture and can be applied independently of any detection method already in use. By presenting active as well as passive defenses we underline the importance and benefits of leveraging the information available in the models' activations.

Based on our results in Chapters 3 and Chapter 4, we show that the activation values are highly relevant in the context of AI-based systems. With the findings and methods individually presented in each chapter we contribute to the research on robust neural networks. By combining the methods in this dissertation, we further show the potential prospects of automatically analyzing neural networks' activations during run time and training. Building upon our simple target-alarm structure may lead to further advances in adversarial machine learning and anomaly detection. We hope that this thesis inspires further research investigating our initial findings in the promising field of *activation analysis*.

Bibliography

[1] P. Sperl, J. P. Schulze, and K. Böttinger. Activation Anomaly Analysis. In *Machine Learning and Knowledge Discovery in Databases*, pages 69–84, 2021. `doi:10.1007/978-3-030-67661-2_5`.

[2] J.-P. Schulze, P. Sperl, and K. Böttinger. DA3G: Detecting Adversarial Attacks by Analysing Gradients. In *Computer Security – ESORICS 2021*, pages 563–583. Springer International Publishing, 2021. `doi:10.1007/978-3-030-88418-5_27`.

[3] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the Inception Architecture for Computer Vision. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2826, 2016. `doi:10.1109/CVPR.2016.308`.

[4] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529:484–503, 2016. URL: `https://www.nature.com/articles/nature16961`, `doi:10.1038/nature16961`.

[5] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba. End to End Learning for Self-Driving Cars. *arXiv preprint arXiv:1604.07316*, 2016. URL: `http://arxiv.org/abs/1604.07316`.

[6] M. Mozaffari-Kermani, S. Sur-Kolay, A. Raghunathan, and N. K. Jha. Systematic Poisoning Attacks on and Defenses for Machine Learning in Healthcare. *IEEE Journal of Biomedical and Health Informatics*, 19(6):1893–1905, 2015. `doi:10.1109/JBHI.2014.2344095`.

[7] B. Nelson, M. Barreno, F. J. Chi, A. D. Joseph, B. I. P. Rubinstein, U. Saini, C. Sutton, J. D. Tygar, and K. Xia. Exploiting Machine Learning to Subvert Your Spam Filter. In *Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats (LEET'08)*, USA, 2008. USENIX Association.

[8] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations*, 2014. `arXiv:1312.6199`.

[9] P. Sperl, C. Y. Kao, P. Chen, X. Lei, and K. Böttinger. DLA: Dense-Layer-Analysis for Adversarial Example Detection. In *Proceedings - 5th IEEE European Symposium on Security and Privacy*, pages 198–215, 2020. `doi:10.1109/EuroSP48549.2020.00021`.

[10] P. Sperl and K. Böttinger. Optimizing Information Loss Towards Robust Neural Networks. In *Proceedings of DYnamic and Novel Advances in Machine Learning and Intelligent Cyber Security (DYNAMICS)*, 2020. `doi:10.1145/3477997.3478016`.

[11] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning.* MIT Press, 2016. URL: `http://www.deeplearningbook.org`.

[12] A. L. Maas, A. Y. Hannun, and A. Y. Ng. Rectifier Nonlinearities Improve Neural Network Acoustic Models. In *ICML Workshop on Deep Learning for Audio, Speech and Language Processing*, volume 28, 2013. URL: `https://ai.stanford.edu/~amaas/papers/relu_hybrid_icml2013_final.pdf`.

[13] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems*, 2020. URL: `https://papers.nips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf`.

[14] N. Carlini, A. Athalye, N. Papernot, W. Brendel, J. Rauber, D. Tsipras, I. Goodfellow, A. Madry, and A. Kurakin. On Evaluating Adversarial Robustness. *arXiv preprint arXiv:1902.06705*, 2019. URL: `http://arxiv.org/abs/1902.06705`.

[15] X. Liu, L. Xie, Y. Wang, J. Zou, J. Xiong, Z. Ying, and A. V. Vasilakos. Privacy and Security Issues in Deep Learning: A Survey. *IEEE Access*, 9, 2020. `doi:10.1109/ACCESS.2020.3045078`.

[16] R. Fletcher. *Practical Methods of Optimization.* John Wiley & Sons, 2013. `doi:10.1002/9781118723203`.

[17] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and Harnessing Adversarial Examples. In *International Conference on Learning Representations*, 2015. URL: `http://arxiv.org/abs/1412.6572`.

[18] A. Kurakin, I. Goodfellow, and S. Bengio. Adversarial Examples in the Physical World. *International Conference on Learning Representations*, pages 99–112, 2016. `doi:10.1201/9781351251389-8`.

[19] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. Towards Deep Learning Models Resistant to Adversarial Attacks. In *International Conference on Learning Representations*, 2018. URL: `https://openreview.net/forum?id=rJzIBfZAb`.

[20] S. Moosavi-Dezfooli, A. Fawzi, and P. Frossard. DeepFool: A Simple and Accurate Method to Fool Deep Neural Networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2574–2582, 2016. `doi:10.1109/CVPR.2016.282`.

[21] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami. The Limitations of Deep Learning in Adversarial Settings. In *2016 IEEE European Symposium on Security and Privacy*, pages 372–387, 2016. `doi:10.1109/EuroSP.2016.36`.

[22] N. Carlini and D. Wagner. Towards Evaluating the Robustness of Neural Networks. In *IEEE Symposium on Security and Privacy*, pages 39–57. IEEE, 2017. `doi:10.1109/SP.2017.49`.

[23] J. Su, D. V. Vargas, and K. Sakurai. One Pixel Attack for Fooling Deep Neural Networks. *IEEE Transactions on Evolutionary Computation*, 23(5):828–841, 2019. `doi:10.1109/TEVC.2019.2890858`.

[24] S. M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard. Universal Adversarial Perturbations. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 86–94, 2017. `doi:10.1109/CVPR.2017.17`.

[25] A. Athalye, N. Carlini, and D. Wagner. Obfuscated Gradients Give a False Sense of Security: Circumventing Defenses to Adversarial Examples. In *International Conference on Machine Learning*, volume 80, pages 274–283, 2018. URL: `http://proceedings.mlr.press/v80/athalye18a.html`.

[26] W. Brendel, J. Rauber, and M. Bethge. Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. In *International Conference on Learning Representations*, 2018. URL: `https://arxiv.org/abs/1712.04248`.

[27] Y. Liu, X. Chen, C. Liu, and D. Song. Delving into Transferable Adversarial Examples and Black-box Attacks. In *The International Conference on Learning Representations*, 2017. URL: `https://openreview.net/forum?id=Sys6GJqxl`.

[28] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami. Practical Black-Box Attacks Against Machine Learning. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, ASIA CCS '17, pages 506–519, New York, NY, USA, 2017. ACM. URL: `http://doi.acm.org/10.1145/3052973.3053009`, `doi:10.1145/3052973.3053009`.

[29] P. Najafirad and S. H. Silva. Opportunities and Challenges in Deep Learning Adversarial Robustness: A Survey. *arXiv preprint arXiv:2007.00753*, 2020. URL: `https://arxiv.org/abs/2007.00753`.

[30] K. Ren, T. Zheng, Z. Qin, and X. Liu. Adversarial Attacks and Defenses in Deep Learning. *Engineering*, 6(3):346–360, 2020. URL: `https://doi.org/10.1016/j.eng.2019.12.012`, `doi:10.1016/j.eng.2019.12.012`.

[31] M. Barreno, B. Nelson, R. Sears, A. D. Joseph, and J. D. Tygar. Can machine learning be secure? In *Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security*, volume 2006, pages 16–25, 2006. URL: `https://doi.org/10.1145/1128817.1128824`, `doi:10.1145/1128817.1128824`.

[32] N. Akhtar and A. Mian. Threat of Adversarial Attacks on Deep Learning in Computer Vision: A Survey. *IEEE Access*, 6:14410–14430, 2018. `doi:10.1109/ACCESS.2018.2807385`.

[33] A. Kurakin, I. Goodfellow, and S. Bengio. Adversarial Machine Learning at Scale. *International Conference on Learning Representations*, 2016. URL: `https://openreview.net/forum?id=BJm4T4Kgx`.

[34] E. Wong, L. Rice, and J. Z. Kolter. Fast Is Better Than Free: Revisiting Adversarial Training. In *International Conference on Learning Representations*, 2020. URL: `https://openreview.net/forum?id=BJx040EFvH`.

[35] G. Sriramanan, S. Addepalli, A. Baburaj, and V. B. R. Towards Efficient and Effective Adversarial Training. In *Advances in Neural Information Processing Systems*, 2021. URL: `https://proceedings.neurips.cc/paper/2021/file/62889e73828c756c961c5a6d6c01a463-Paper.pdf`.

[36] F. Tramèr, A. Kurakin, N. Papernot, I. Goodfellow, D. Boneh, and P. McDaniel. Ensemble Adversarial Training: Attacks and Defenses. In *International Conference on Learning Representations*, 2018. URL: `https://openreview.net/pdf?id=rkZvSe-RZ`.

[37] G. K. Dziugaite, Z. Ghahramani, and D. M. Roy. A study of the effect of JPG compression on adversarial images. *arXiv preprint arXiv:1608.00853*, 2016. URL: `https://arxiv.org/abs/1608.00853`.

[38] C. Guo, M. Rana, M. Cisse, and L. van der Maaten. Countering Adversarial Images using Input Transformations. In *The International Conference on Learning Representations*, 2017. URL: `https://openreview.net/forum?id=SyJ7ClWCb`.

[39] N. Das, M. Shanbhogue, S.-T. Chen, F. Hohman, L. Chen, M. E. Kounavis, and D. H. Chau. Keeping the Bad Guys Out: Protecting and Vaccinating Deep Learning with JPEG Compression. *arXiv preprint arXiv:1705.02900*, 2017. URL: `http://arxiv.org/abs/1705.02900`.

[40] Y. Luo, X. Boix, G. Roig, T. Poggio, and Q. Zhao. Foveation-based Mechanisms Alleviate Adversarial Examples. *arXiv preprint arXiv:1511.06292*, 2015. URL: `http://arxiv.org/abs/1511.06292`.

[41] Q. Wang, W. Guo, K. Zhang, I. I. Ororbia, G. Alexander, X. Xing, X. Liu, and C. L. Giles. Learning Adversary-Resistant Deep Neural Networks. *arXiv preprint arXiv:1612.01401*, 2016. URL: `https://arxiv.org/abs/1612.01401`.

[42] K. Sharad, G. A. Marson, H. T. T. Truong, and G. Karame. On the Security of Randomized Defenses against Adversarial Samples. *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security, ASIA CCS 2020*, pages 381–393, 2020. `doi:10.1145/3320269.3384751`.

[43] A. Athalye, L. Engstrom, A. Ilyas, and K. Kevin. Synthesizing Robust Adversarial Examples. In *International Conference on Machine Learning*, volume 80, pages 284–293, 2018. URL: `http://proceedings.mlr.press/v80/athalye18b.html`.

[44] G. Hinton, O. Vinyals, and J. Dean. Distilling the Knowledge in a Neural Network. In *NIPS Deep Learning and Representation Learning Workshop*, 2015. URL: `http://arxiv.org/abs/1503.02531`.

[45] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami. Distillation as a Defense to Adversarial Perturbations Against Deep Neural Networks. In *2016 IEEE Symposium on Security and Privacy*, pages 582–597, 2016. `doi:10.1109/SP.2016.41`.

[46] A. S. Ross and F. Doshi-Velez. Improving the Adversarial Robustness and Interpretability of Deep Neural Networks by Regularizing Their Input Gradients. In *Thirty-second AAAI Conference on Artificial Intelligence*, pages 1660–1669, 2018. `doi:10.1609/aaai.v32i1.11504`.

[47] S. Zhang, S. Chen, X. Liu, C. Hua, W. Wang, K. Chen, J. Zhang, and J. Wang. Detecting Adversarial Samples for Deep Learning Models: A Comparative Study. *IEEE Transactions on Network Science and Engineering*, 4697(c):1–1, 2021. `doi:10.1109/tnse.2021.3057071`.

[48] D. Hendrycks and K. Gimpel. Early Methods for Detecting Adversarial Images. In *International Conference on Learning Representations, Workshop Track*, 2017. URL: `https://openreview.net/forum?id=B1dexpDug`.

[49] B. Liang, H. Li, M. Su, X. Li, W. Shi, and X. F. Wang. Detecting Adversarial Image Examples in Deep Neural Networks with Adaptive Noise Reduction. *IEEE Transactions on Dependable and Secure Computing*, 18:72–85, 2018. `doi:10.1109/TDSC.2018.2874243`.

[50] Z. Gong, W. Wang, and W. S. Ku. Adversarial and Clean Data Are Not Twins. *arXiv preprint arXiv:1704.04960*, 2017. URL: `https://arxiv.org/abs/1704.04960`.

[51] N. Carlini and D. Wagner. Adversarial Examples Are Not Easily Detected: Bypassing Ten Detection Methods. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, AISec '17, pages 3–14, New York, NY, USA, 2017. ACM. URL: `http://doi.acm.org/10.1145/3128572.3140444`, `doi:10.1145/3128572.3140444`.

[52] J. H. Metzen, T. Genewein, V. Fischer, and B. Bischoff. On Detecting Adversarial Perturbations. In *International Conference on Learning Representations*, 2017. URL: `https://openreview.net/forum?id=SJzCSf9xg`.

[53] D. Meng and H. Chen. MagNet: A Two-Pronged Defense against Adversarial Examples. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 135–147, 2017. `doi:10.1145/3133956.3134057`.

[54] N. Carlini and D. Wagner. MagNet and "Efficient Defenses Against Adversarial Attacks" are Not Robust to Adversarial Examples. *arXiv preprint arXiv:1711.08478*, 2017. URL: `https://arxiv.org/abs/1711.08478`.

[55] K. Grosse, P. Manoharan, N. Papernot, M. Backes, and P. McDaniel. On the (Statistical) Detection of Adversarial Examples. *arXiv preprint arXiv:1702.06280*, 2017. URL: `https://arxiv.org/abs/1702.06280`.

[56] W. Xu, D. Evans, and Y. Qi. Feature Squeezing: Detecting Adversarial Examples in Deep Neural Networks. In *Network and Distributed System Security Symposium, NDSS*, 2018. `doi:10.14722/ndss.2018.23198`.

[57] W. Xu, D. Evans, and Y. Qi. Feature Squeezing Mitigates and Detects Carlini/Wagner Adversarial Examples. *arXiv preprint arXiv:1705.10686*, 2017. URL: `https://arxiv.org/abs/1705.10686`.

[58] J. Lu, T. Issaranon, and D. Forsyth. SafetyNet: Detecting and Rejecting Adversarial Examples Robustly. In *IEEE International Conference on Computer Vision (ICCV)*, pages 446–454, 2017. `doi:10.1109/ICCV.2017.56`.

[59] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative Adversarial Nets. In *Advances in Neural Information Processing Systems*, pages 2672–2680. Curran Associates, Inc., 2014. URL: `http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf`.

[60] H. Lee, S. Han, and J. Lee. Generative Adversarial Trainer: Defense to Adversarial Perturbations with GAN. *arXiv preprint arXiv:1705.03387*, 2017. URL: `https://arxiv.org/abs/1705.03387`.

[61] G. Jin, S. Shen, D. Zhang, F. Dai, and Y. Zhang. APE-GAN: Adversarial Perturbation Elimination with GAN. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3842–3846, 2019. `doi:10.1109/ICASSP.2019.8683044`.

[62] P. Samangouei, M. Kabkab, and R. Chellappa. Defense-GAN: Protecting Classifiers Against Adversarial Attacks Using Generative Models. In *The International Conference on Learning Representations*, 2018. URL: `https://openreview.net/forum?id=BkJ3ibb0-`.

[63] N. Akhtar, J. Liu, and A. Mian. Defense Against Universal Adversarial Perturbations. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3389–3398, 2018. `doi:10.1109/CVPR.2018.00357`.

[64] K. D. Dvijotham, S. Gowal, R. Stanforth, R. Arandjelović, B. O'Donoghue, P. Kohli, and J. Uesato. Training Verified Learners with Learned Verifiers. *arXiv preprint arXiv:1805.10265*, 2018. URL: `https://arxiv.org/abs/1805.10265`.

[65] M. Hein and M. Andriushchenko. Formal guarantees on the robustness of a classifier against adversarial manipulation. In *Advances in Neural Information Processing Systems*, 2017. URL: `https://proceedings.neurips.cc/paper/2017/file/e077e1a544eec4f0307cf5c3c721d944-Paper.pdf`.

[66] A. Raghunathan, J. Steinhardt, and P. Liang. Certified Defenses against Adversarial Examples. In *International Conference on Learning Representations*, 2018. URL: `https://openreview.net/forum?id=Bys4ob-Rb`.

[67] A. Raghunathan, J. Steinhardt, and P. Liang. Semidefinite relaxations for certifying robustness to adversarial examples. In *Advances in Neural Information Processing Systems*, 2018. URL: `https://proceedings.neurips.cc/paper/2018/file/29c0605a3bab4229e46723f89cf59d83-Paper.pdf`.

[68] E. Wong and J. Zico Kolter. Provable Defenses against Adversarial Examples via the Convex Outer Adversarial Polytope. In *Proceedings of the 35th International Conference on Machine Learning*, pages 5286–5295, 2018. URL: `http://proceedings.mlr.press/v80/wong18a/wong18a.pdf`.

[69] A. Ghiasi, A. Shafahi, and T. Goldstein. Breaking Certified Defenses: Semantic Adversarial Examples With Spoofed Robustness Certificates. In *International Conference on Learning Representations*, 2020. URL: `https://openreview.net/forum?id=HJxdTxHYvB`.

[70] M. Sharif, L. Bauer, and M. K. Reiter. On the Suitability of Lp-Norms for Creating and Preventing Adversarial Examples. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 1686–16868, 2018. `doi:10.1109/CVPRW.2018.00211`.

[71] S. Sen, B. Ravindran, and A. Raghunathan. EMPIR: Ensembles of Mixed Precision Deep Networks for Increased Robustness Against Adversarial Attacks. In *The International Conference on Learning Representations (ICLR)*, 2019. URL: `https://openreview.net/forum?id=HJem3yHKwH`.

[72] F. Tramèr, N. Carlini, W. Brendel, and A. Madry. On Adaptive Attacks to Adversarial Example Defenses. In *Advances in Neural Information Processing Systems*, pages 1633—-1645, 2020. URL: `https://proceedings.neurips.cc/paper/2020/file/11f38f8ecd71867b42433548d1078e38-Paper.pdf`.

[73] T.-W. Weng, H. Zhang, P.-Y. Chen, J. Yi, D. Su, Y. Gao, C.-J. Hsieh, and L. Daniel. Evaluating the Robustness of Neural Networks: An Extreme Value Theory Approach. In *International Conference on Learning Representations*, 2018. URL: `https://openreview.net/forum?id=BkUHlMZ0b`.

[74] I. Goodfellow. Gradient Masking Causes CLEVER to Overestimate Adversarial Perturbation Size. *arXiv:1804.07870*, 2018. URL: `http://arxiv.org/abs/1804.07870`.

[75] F. Croce and M. Hein. Reliable Evaluation of Adversarial Robustness with an Ensemble of Diverse Parameter-free Attacks. In *Proceedings of Machine Learning Research*, pages 2206–2216. PMLR, 2020. URL: `http://proceedings.mlr.press/v119/croce20b/croce20b.pdf`.

[76] M. Pintor, L. Demetrio, A. Sotgiu, G. Manca, A. Demontis, N. Carlini, B. Biggio, and F. Roli. Indicators of Attack Failure: Debugging and Improving Optimization of Adversarial Examples. *arXiv:1804.07870*, 2021. URL: `http://arxiv.org/abs/2106.09947`.

[77] K. Pei, Y. Cao, J. Yang, and S. Jana. DeepXplore: automated whitebox testing of deep learning systems. *Commun. ACM*, 62(11):137–145, oct 2019. `doi:10.1145/3361566`.

[78] L. Ma, Y. Liu, J. Zhao, Y. Wang, F. Juefei-Xu, F. Zhang, J. Sun, M. Xue, B. Li, C. Chen, T. Su, and L. Li. DeepGauge: Multi-granularity Testing Criteria for Deep Learning Systems. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, pages 120–131, 2018. `doi:10.1145/3238147.3238202`.

[79] Y. Sun, X. Huang, and D. Kroening. Testing Deep Neural Networks. *arXiv preprint arXiv:1803.04792*, 2018. URL: `https://arxiv.org/abs/1803.04792`.

[80] A. Odena and I. Goodfellow. TensorFuzz: Debugging Neural Networks with Coverage-Guided Fuzzing. *Proceedings of the 36th International Conference on Machine Learning*, pages 4901—-4911, 2019. URL: `https://proceedings.mlr.press/v97/odena19a.html`.

[81] R. Feinman, R. R. Curtin, S. Shintre, and A. B. Gardner. Detecting Adversarial Samples from Artifacts. *arXiv preprint arXiv:1703.00410*, 2017. URL: `https://arxiv.org/abs/1703.00410`.

[82] S. Ma, Y. Liu, G. Tao, W.-C. Lee, and X. Zhang. NIC: Detecting Adversarial Samples with Neural Network Invariant Checking. In *Network and Distributed System Security Symposium (NDSS)*, 2019. URL: `https://www.ndss-symposium.org/wp-content/uploads/2019/02/ndss2019_03A-4_Ma_paper.pdf`, `doi:10.14722/ndss.2019.23415`.

[83] Y. LeCun, C. Cortes, and C. Burges. Mnist handwritten digit database. *ATT Labs*, 2010. Accessed: 2023-03-01. URL: `http://yann.lecun.com/exdb/mnist`.

[84] A. Krizhevsky. Learning Multiple Layers of Features from Tiny Images. *University of Toronto*, 2009. URL: `https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf`.

[85] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts. Learning Word Vectors for Sentiment Analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA, 2011. Association for Computational Linguistics. URL: `http://www.aclweb.org/anthology/P11-1015`.

[86] Mozilla Common Voice dataset. Accessed: 2023-03-01. URL: `https://voice.mozilla.org/datasets`.

[87] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2323, 1998. `doi:10.1109/5.726791`.

[88] F. Chollet. Simple MNIST convnet, 2015. Accessed: 2023-03-01. URL: `https://keras.io/examples/vision/mnist_convnet/`.

[89] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016. `doi:10.1109/CVPR.2016.90`.

[90] Keras. CIFAR-10 CNN, 2020. Accessed: 2020-04-23. URL: `https://keras.io/examples/cifar10_cnn/`.

[91] N. Frosst, S. Sabour, and G. Hinton. DARCCC: Detecting Adversaries by Reconstruction from Class Conditional Capsules. *arXiv preprint arXiv:1811.06969*, 2018. URL: `http://arxiv.org/abs/1811.06969`.

[92] Mozilla Project DeepSpeech. Accessed: 2019-05-20. URL: `https://github.com/mozilla/DeepSpeech`.

[93] D. P. Kingma, D. J. Rezende, S. Mohamed, and M. Welling. Semi-supervised Learning with Deep Generative Models. In *Proceedings of the 27th International Conference on Neural Information Processing Systems*, pages 3581–3589, 2014.

[94] S. Sabour, N. Frosst, and G. E. Hinton. Dynamic Routing Between Capsules. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 3859–3869, 2017. URL: `http://papers.nips.cc/paper/6975-dynamic-routing-between-capsules.pdf`.

[95] S. Hochreiter and J. Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 1997. `doi:10.1162/neco.1997.9.8.1735`.

[96] J. Rauber, W. Brendel, and M. Bethge. Foolbox: A python toolbox to benchmark the robustness of machine learning models. In *Reliable Machine Learning in the Wild Workshop, 34th International Conference on Machine Learning*, 2017. URL: `http://arxiv.org/abs/1707.04131`.

[97] N. Carlini and D. Wagner. Audio Adversarial Examples: Targeted Attacks on Speech-to-Text. In *2018 IEEE Security and Privacy Workshops*, pages 1–7, 2018. `doi:10.1109/SPW.2018.00009`.

[98] Nicholas Carlini: Breaking Neural Network Detection Schemes. Accessed: 2019-08-21. URL: `https://nicholas.carlini.com/code/nn_breaking_detection`.

[99] S. Axelsson. The Base-Rate Fallacy and the Difficulty of Intrusion Detection. *ACM Transactions on Information and System Security*, 3(3):186–205, 2000. `doi:10.1145/357830.357849`.

[100] P. Yang, J. Chen, C. J. Hsieh, J. L. Wang, and M. I. Jordan. ML-LOO: Detecting adversarial examples with feature attribution. *AAAI Conference on Artificial Intelligence*, 2020. `doi:10.1609/aaai.v34i04.6140`.

[101] J. Dhaliwal and S. Shintre. Gradient Similarity: An Explainable Approach to Detect Adversarial Attacks against Deep Learning. *arXiv preprint arXiv:1806.10707*, 2018. URL: `https://arxiv.org/abs/1806.10707`.

[102] J. Lust and A. P. Condurache. GraN: An Efficient Gradient-Norm Based Detector for Adversarial and Misclassified Examples. In *28th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, 2020. URL: `https://www.esann.org/sites/default/files/proceedings/2020/ES2020-159.pdf`.

[103] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. *arXiv preprint arXiv:1708.07747*, 2017. URL: `https://arxiv.org/abs/1708.07747`.

[104] G. Klambauer, T. Unterthiner, and A. Mayr. Self-Normalizing Neural Networks. In *NIPS'17: Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 972–981, 2017. URL: `https://dl.acm.org/doi/pdf/10.5555/3294771.3294864`.

[105] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai. Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection. In *Network and Distributed Systems Security (NDSS)*, 2018. doi:10.14722/ndss.2018.23204.

[106] T. Shekari, C. Bayens, M. Cohen, L. Graber, and R. Beyah. RFDIDS: Radio Frequency-based Distributed Intrusion Detection System for the Power Grid. In *Network and Distributed Systems Security (NDSS)*, 2019. doi:10.14722/ndss.2019.23462.

[107] C. Feng, V. R. Palleti, A. Mathur, and D. Chana. A Systematic Framework to Generate Invariants for Anomaly Detection in Industrial Control Systems. In *Network and Distributed Systems Security (NDSS)*, 2019. doi:10.14722/ndss.2019.23265.

[108] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM Computing Surveys*, 41(3):1–58, 2009. doi:10.1145/1541880.1541882.

[109] R. Chalapathy and S. Chawla. Deep Learning for Anomaly Detection: A Survey. *arXiv preprint arXiv:1901.03407*, 2019. URL: http://arxiv.org/abs/1901.03407.

[110] B. Schölkopf, R. Williamson, A. Smola, J. Shawe-Taylor, and J. Piatt. Support Vector Method for Novelty Detection. *Proceedings of the 12th International Conference on Neural Information Processing Systems*, pages 582–588, 1999. URL: https://papers.nips.cc/paper/1723-support-vector-method-for-novelty-detection.pdf.

[111] F. T. Liu, K. M. Ting, and Z. H. Zhou. Isolation Forest. In *IEEE International Conference on Data Mining*, pages 413–422, 2008. doi:10.1109/ICDM.2008.17.

[112] M. Yousefi-Azar, V. Varadharajan, L. Hamey, and U. Tupakula. Autoencoder-based feature learning for cyber security applications. In *Proceedings of the International Joint Conference on Neural Networks*, pages 3854–3861, 2017. doi:10.1109/IJCNN.2017.7966342.

[113] J. T. A. Andrews, E. J. Morton, and L. D. Griffin. Detecting Anomalous Data Using Auto-Encoders. *International Journal of Machine Learning and Computing*, 6(1):21–26, 2016. URL: http://ijmlc.org/vol6/565-L009.pdf, doi:10.18178/ijmlc.2016.6.1.565.

[114] A. S. Qureshi, A. Khan, N. Shamim, and M. H. Durad. Intrusion detection using deep sparse auto-encoder and self-taught learning. In *Neural Computing and Applications*, pages 1–13, 2019. doi:10.1007/s00521-019-04152-6.

[115] B. Zong, Q. Song, M. R. Min, W. Cheng, C. Lumezanu, D. Cho, and H. Chen. Deep Autoencoding Gaussian Mixture Model for Unsupervised Anomaly Detection. In *International Conference on Learning Representations (ICLR)*, 2018. URL: https://openreview.net/forum?id=BJJLHbb0-.

[116] T. D. Nguyen, S. Marchal, M. Miettinen, H. Fereidooni, N. Asokan, and A.-R. Sadeghi. DÏoT: A Federated Self-learning Anomaly Detection System for IoT. In *Proceedings of the 39th IEEE International Conference on Distributed Computing Systems (ICDCS)*, Dallas, USA, 2019. doi:10.1109/ICDCS.2019.00080.

[117] P. Malhotra, A. Ramakrishnan, G. Anand, L. Vig, P. Agarwal, and G. Shroff. LSTM-based Encoder-Decoder for Multi-sensor Anomaly Detection. In *ICML 2016 Anomaly Detection Workshop*, 2016. URL: http://arxiv.org/abs/1607.00148.

[118] J.-P. Schulze, A. Mrowca, E. Ren, H.-A. Loeliger, and K. Böttinger. Context by Proxy: Identifying Contextual Anomalies Using an Output Proxy. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining - KDD '19*, pages 2059–2068, 2019. doi:10.1145/3292500.3330780.

[119] C. Zhou and R. C. Paffenroth. Anomaly Detection with Robust Deep Autoencoders. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017. doi:10.1145/3097983.3098052.

[120] K. Veeramachaneni, I. Arnaldo, V. Korrapati, C. Bassias, and K. Li. AI2: Training a Big Data Machine to Defend. In *IEEE 2nd International Conference on Big Data Security on Cloud (BigDataSecurity), IEEE International Conference on High Performance and Smart Computing (HPSC), and IEEE International Conference on Intelligent Data and Security (IDS)*, pages 49–54, 2016. doi:10.1109/BigDataSecurity-HPSC-IDS.2016.79.

[121] S. Das, W.-K. Wong, T. Dietterich, A. Fern, and A. Emmott. Incorporating Expert Feedback into Active Anomaly Discovery. In *IEEE 16th International Conference on Data Mining (ICDM)*, pages 853–858, 2017. doi:10.1109/icdm.2016.0102.

[122] G. Pang, C. Shen, and A. van den Hengel. Deep Anomaly Detection with Deviation Networks. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 353–362, 2019. doi:10.1145/3292500.3330871.

[123] L. Ruff, R. A. Vandermeulen, N. Görnitz, A. Binder, E. Müller, K.-R. Müller, and M. Kloft. Deep Semi-Supervised Anomaly Detection. In *International Conference on Learning Representations (ICLR)*, 2020. URL: https://openreview.net/forum?id=HkgH0TEYwH.

[124] G. Pang, C. Shen, H. Jin, and A. van den Hengel. Deep Weakly-supervised Anomaly Detection. *arXiv preprint arXiv:1910.13601*, 2019. URL: http://arxiv.org/abs/1910.13601.

[125] G. Cohen, S. Afshar, J. Tapson, and A. Van Schaik. EMNIST: Extending MNIST to handwritten letters. In *International Joint Conference on Neural Networks (IJCNN)*, pages 2921–2926, 2017. doi:10.1109/IJCNN.2017.7966217.

[126] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani. A detailed analysis of the KDD CUP 99 data set. In *IEEE Symposium on Computational Intelligence for Security and Defense Applications*, pages 1–6, 2009. doi:10.1109/CISDA.2009.5356528.

[127] A. D. Pozzolo, O. Caelen, R. A. Johnson, and G. Bontempi. Calibrating Probability with Undersampling for Unbalanced Classification. In *2015 IEEE Symposium Series on Computational Intelligence*, pages 159–166. IEEE, 2015. doi:10.1109/SSCI.2015.33.

[128] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani. Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization. In *ICISSP 2018 - Proceedings of the 4th International Conference on Information Systems Security and Privacy*, pages 108–116, 2018. doi:10.5220/0006639801080116.

[129] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[130] T. Nakae. DAGMM TF Implementation. Accessed: 2023-03-01. URL: https://github.com/tnakae/DAGMM.

[131] F. Chollet et al. Keras, 2015. URL: https://keras.io.

[132] Keras. Building Autoencoders in Keras, 2016. Accessed: 2023-03-01. URL: https://blog.keras.io/building-autoencoders-in-keras.html.

[133] Keras. MNIST CNN, 2015. Accessed: 2020-01-13. URL: https://keras.io/examples/mnist_cnn/.

[134] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014. URL: https://dl.acm.org/doi/abs/10.5555/2627435.2670313.

[135] A. Shafahi, M. Najibi, A. Ghiasi, Z. Xu, J. Dickerson, C. Studer, L. S. Davis, G. Taylor, and T. Goldstein. Adversarial Training for Free! In *Advances in Neural Information Processing Systems*, 2019. URL: https://proceedings.neurips.cc/paper/2019/file/7503cfacd12053d309b6bed5c89de212-Paper.pdf.

[136] N. Tishby and N. Zaslavsky. Deep learning and the information bottleneck principle. In *IEEE Information Theory Workshop (ITW)*, pages 1–5, 2015. doi:10.1109/ITW.2015.7133169.

[137] R. Shwartz-Ziv and N. Tishby. Opening the Black Box of Deep Neural Networks via Information. *arXiv preprint arXiv:1703.00810*, 2017. URL: http://arxiv.org/abs/1703.00810.

[138] M. Gabrie, A. Manoel, C. Luneau, J. Barbier, N. Macris, F. Krzakala, and L. Zdeborová. Entropy and mutual information in models of deep neural networks. In *Advances in Neural Information Processing Systems*, volume 31, 2018. URL: `https://papers.nips.cc/paper/7453-entropy-and-mutual-information-in-models-of-deep-neural-networks.pdf`.

[139] M. Amirian, F. Schwenker, and T. Stadelmann. Trace and Detect Adversarial Attacks on CNNs using Feature Response Maps. In *Artificial Neural Networks in Pattern Recognition*, pages 346—-358, 2018. URL: `https://core.ac.uk/download/pdf/159488771.pdf`.

[140] M. Terzi, A. Achille, M. Maggipinto, and G. A. Susto. Adversarial Training Reduces Information and Improves Transferability. *arXiv preprint arXiv:2007.11259*, 2020. URL: `http://arxiv.org/abs/2007.11259`.

[141] D. Tsipras, S. Santurkar, L. Engstrom, A. Turner, and A. Madry. Robustness May Be at Odds with Accuracy. In *International Conference on Learning Representations*, 2018. URL: `https://openreview.net/forum?id=SyxAb30cY7`.

[142] N. Tishby, F. C. Pereira, and W. Bialek. The information bottleneck method. In *Proceedings of the 37-th Annual Allerton Conference on Communication, Control and Computing*, 1999.

[143] A. Achille, G. Paolini, and S. Soatto. Where is the Information in a Deep Neural Network? *arXiv preprint arXiv:1905.12213*, 2019. URL: `http://arxiv.org/abs/1905.12213`.

[144] C. E. Shannon. A Mathematical Theory of Communication. *Bell System Technical Journal*, 27(4):623–656, 1948. URL: `http://people.math.harvard.edu/~ctm/home/text/others/shannon/entropy/entropy.pdf`, `doi:10.1002/j.1538-7305.1948.tb00917.x`.

[145] A. Ilyas, S. Santurkar, D. Tsipras, L. Engstrom, B. Tran, and A. Madry. Adversarial Examples Are Not Bugs, They Are Features. In *Advances in Neural Information Processing Systems*, volume 32, 2019. URL: `https://proceedings.neurips.cc/paper/2019/file/e2c420d928d4bf8ce0ff2ec19b371514-Paper.pdf`.

[146] H. Zhang, Y. Yu, J. Jiao, E. P. Xing, L. E. Ghaoui, and M. I. Jordan. Theoretically Principled Trade-off between Robustness and Accuracy. *Proceedings of the 36th International Conference on Machine Learning*, pages 7472—-7482, 2019. URL: `http://proceedings.mlr.press/v97/zhang19p.html`.

[147] Keras. Keras Convolutional Neural Network for Fashion-MNIST, 2019. Accessed: 2019-05-20. URL: `https://blog.tensorflow.org/2018/04/fashion-mnist-with-tfkeras.html`.

[148] N. S. Keskar, J. Nocedal, P. T. P. Tang, D. Mudigere, and M. Smelyanskiy. On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima. *International Conference on Learning Representations*, 2016. URL: `https://openreview.net/forum?id=H1oyRlYgg`.

[149] D. P. Kingma and M. Welling. Auto-Encoding Variational Bayes. *arXiv preprint arXiv:1312.6114*, 2013. URL: `https://arxiv.org/abs/1312.6114`.

A Appendix

A.1 DLA Appendix

In this section we show supplemental material for Section 3.2 based on our publication "DLA: Dense-Layer-Analysis for Adversarial Example Detection" [9].

A.1.1 NLP Adversarial Example Generation

With Algorithm 1 we generated adversarial examples for our NLP target model classifying IMDb reviews. The target model performs a binary classification and distinguishes between positive and negative reviews.

```
Data: IMDb reviews
Result: Adversarial IMDb reviews
Train a Word2Vec model with all reviews;
Randomly pick one word to start;
while not at the end of this document do
    Find $N$ most similar words of current word using Word2Vec;
    for substitute ← next most similar word do
        Replace the current word with the substitute;
        Predict and calculate the margin;
        if margin decrease then
            break
        end
    end
    if $margin < MarginThreshold$ then
        break
    else
        Recover the current word to the original word;
        Move to next word;
    end
end
```

Algorithm 1: Generation of adversarial examples in the IMDb data set containing movie reviews.

A.1.2 Confusion Matrix Values

In Table A.1 we show the confusion matrix values of each performed test. Note that we tested the alarm models against multiple non-overlapping batches of test data to show the standard deviation of the according results.

Table A.1: Confusion matrix values including error rates for all data sets, target models, and attack methods. Each result corresponds to the detection of adversarial attack methods with the specified target model.

Data	**Target**	**Attack**	**Performance of the Alarm Models Tested with the According Attacks:**			
			TPR	***TNR***	***FPR***	***FNR***
MNIST	LeNet	FGSM	0.98±0.00	1.00±0.00	0.01±0.00	0.02±0.00
		C&W	0.97±0.01	0.98±0.00	0.02±0.00	0.03±0.01
		DF	0.97±0.01	0.99±0.00	0.01±0.00	0.03±0.01
		PGD	0.99±0.00	1.00±0.00	0.01±0.00	0.01±0.00
		BIM	0.99±0.00	0.99±0.00	0.01±0.00	0.01±0.00
	kerasExM	FGSM	0.99±0.00	1.00±0.00	0.01±0.00	0.01±0.00
		C&W	0.97±0.01	0.98±0.00	0.02±0.00	0.03±0.01
		DF	0.98±0.01	0.99±0.00	0.01±0.00	0.03±0.01
		PGD	0.99±0.00	1.00±0.00	0.01±0.00	0.01±0.00
		BIM	0.99±0.00	0.99±0.00	0.01±0.00	0.01±0.00
	LSTM	Transfer	0.89±0.01	0.97±0.00	0.03±0.01	0.11±0.01
	CapsuleNN	Transfer	0.95±0.02	0.93±0.04	0.07±0.04	0.05±0.02
CIFAR10	kerasExC	FGSM	0.82±0.01	0.86±0.02	0.14±0.02	0.18±0.01
		C&W	0.75±0.02	0.73±0.01	0.27±0.01	0.25±0.02
		DF	0.84±0.01	0.86±0.01	0.14±0.01	0.16±0.01
		PGD	0.83±0.01	0.85±0.01	0.15±0.01	0.17±0.01
		BIM	0.84±0.01	0.86±0.02	0.14±0.02	0.16±0.01
	ResNet	FGSM	0.77±0.02	0.85±0.01	0.15±0.01	0.23±0.02
		C&W	0.68±0.02	0.74±0.01	0.26±0.01	0.32±0.02
		DF	0.78±0.02	0.88±0.01	0.13±0.01	0.22±0.02
		PGD	0.80±0.01	0.86±0.01	0.14±0.01	0.20±0.01
		BIM	0.79±0.01	0.87±0.01	0.14±0.01	0.21±0.01
NLP and Audio	LSTM (NLP)	Custom	0.97±0.03	0.96±0.03	0.04±0.03	0.03±0.03
	DeepSpeech	Carlini	1.00±0.00	0.67±0.07	0.33±0.07	0.00±0.00

A.1.3 C&W Adaptive Attack Parameters

Table A.2 shows the parameters of the C&W attack during the adaptive white-box attacks for the MNIST and CIFAR10 data sets.

Table A.2: C&W attack parameters during the adaptive attack for MNIST and CIFAR10.

Parameter	**Value** (MNIST)	**Value** (CIFAR10)
max-iteration	2000	200
batch-size	100	100
learning-rate	0.1	0.01
binary-search-steps	5	3

A.2 A^3 Appendix

In this section we show supplemental material for Section 3.4 based on our publication "Activation Anomaly Analysis" [1].

A.2.1 Improved A^3 Architecture

In our original publication, A^3 consists of three components. Additionally to our target-alarm structure, we propose the use of an anomaly network. This generative model creates artificial anomalies useful during the training of our detector.

Combining the three used models we present an overview of the improved A^3 architecture in Figure A.1.

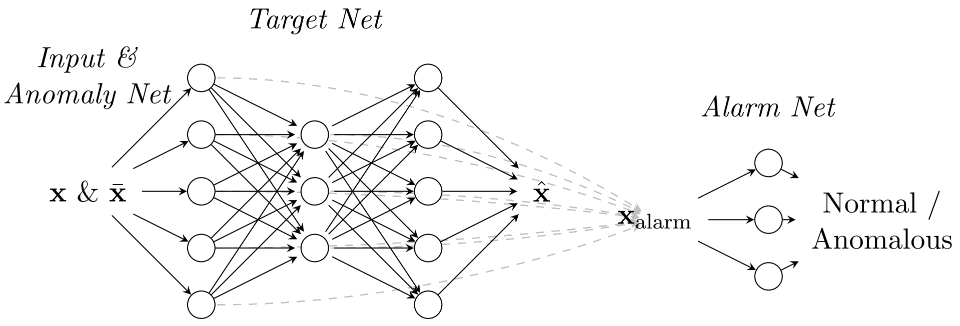


Figure A.1: Improved A^3 consists of two connected parts: 1) a target network unrelated to anomaly detection (e.g., an autoencoder), 2) the anomaly network providing anomalous samples $\bar{\mathbf{x}}$, and 3) the alarm network judging if the input $\mathbf{x}$ is normal.

The training process and the final deployment of our detector can again be divided into three parts. We show an overview of the required steps in Figure A.2.

A.2.2 Evaluation of the Improved A^3 Architecture

A.2.2.1 Experimental Setup

To evaluate the improved A^3 architecture we used the very same experiments as shown in Table 3.14. As an addition to the three experimental setups, we briefly introduce

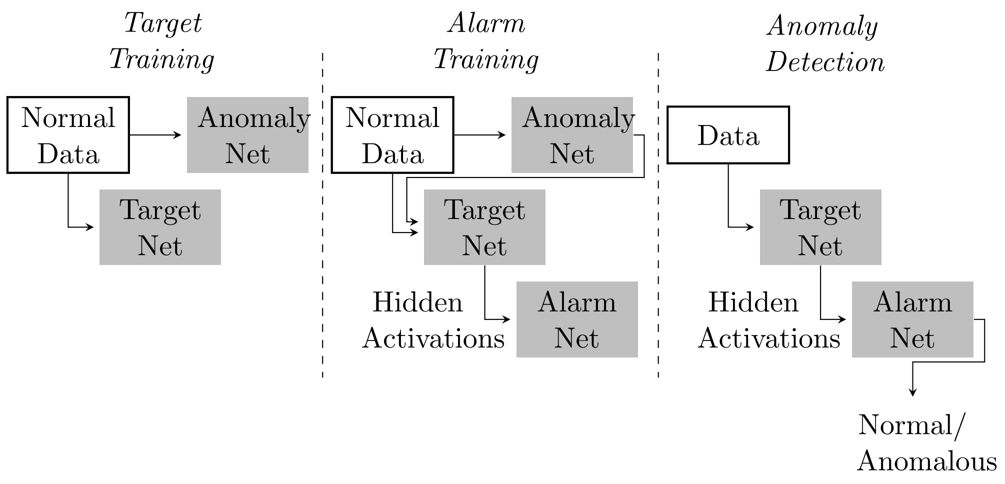


Figure A.2: Improved A^3. Step-by-step overview of the training process including the anomaly network in the first two columns and the final anomaly detection in the right column.

a fourth setting. An overview is given in Table A.3. Considering the extreme case of constraint 1, we conducted first evaluations of the detection performance when no labeled anomalies are available during training. We used normal samples, as well as the output of a generative anomaly model. This does not include labeled anomalous samples.

Table A.3: A^3 experiments exploring the detection of known and unknown anomalies in an unsupervised setting.

Exp.	**Data**	**Normal**	**Train Anomaly**	$\subseteq$	**Test Anomaly**
4a	MNIST	0, ..., 5	6, 7		6, 7
4c	MNIST	4, ..., 9	0, 1		0, 1
4b	MNIST	0, ..., 5	6, 7		6, 7, 8, 9
4d	MNIST	4, ..., 9	0, 1		0, 1, 2, 3

A.2.2.2 Anomaly Network Settings

For experiments 1, 2, and 3 we used a simple Gaussian noise generator as anomaly network. As all inputs are within $[0, 1]$, we fixed the noise parameters to $\mathcal{N}(.5, 1)$. For experiment 4, we chose a variational autoencoder (VAE) [149] with the dense hidden layers $800, 400, 100, 25, 100, 400, 800$. During the training of the target model, it was adapted to reconstruct the normal samples.

A.2.2.3 Results: Experiments 1 – 3

In Table A.4, we summarize the results for the improved A^3 architecture and the baseline methods averaged over five passes using the test data sets. Note that not all baseline methods finished five runs on the IDS data set. To simulate real-life conditions, we limited the amount of available anomalies to 100 randomly chosen samples of the training set.

Table A.4: Improved A^3 test results after ten runs given all normal, and 100 anomaly samples using the anomaly network.

	A^3		**AE**		**IF**		**DAGMM**		**DevNet**	
Epx.	AUC	AP	AUC	AP	AUC	AP	AUC	AP	AUC	AP
1a	0.99±0.00	0.99±0.00	0.70±0.03	0.44±0.04	0.57±0.02	0.28±0.02	0.85±0.02	0.70±0.02	0.98±0.00	0.95±0.01
1b	1.00±0.00	1.00±0.00	0.39±0.04	0.25±0.03	0.53±0.01	0.41±0.02	0.63±0.04	0.34±0.03	0.99±0.00	0.98±0.00
1c	0.97±0.01	0.97±0.01	0.96±0.00	0.96±0.01	0.97±0.00	0.98±0.00	0.94±0.01	0.91±0.04	0.95±0.02	0.95±0.02
1d	0.91±0.03	0.71±0.08	0.82±0.03	0.55±0.06	0.84±0.00	0.49±0.02	0.91±0.01	0.59±0.03	0.91±0.03	0.79±0.02
1e	0.95±0.01	0.92±0.01	0.88±0.08	0.75±0.12	0.47±0.04	0.17±0.01	0.90±0.02	0.68±0.06	0.93±0.00	0.90±0.01
1f	0.94±0.01	0.91±0.01	0.86±0.07	0.65±0.13	0.36±0.03	0.14±0.01	0.68±0.12	0.34±0.18	0.90±0.01	0.74±0.04
1g	0.97±0.01	0.77±0.04	0.97±0.01	0.54±0.09	0.97±0.01	0.14±0.04	0.96±0.02	0.35±0.23	0.97±0.01	0.77±0.04
2a	0.88±0.01	0.88±0.01	0.72±0.02	0.63±0.02	0.54±0.01	0.40±0.01	0.74±0.02	0.69±0.02	0.84±0.04	0.84±0.03
2b	0.92±0.02	0.91±0.02	0.64±0.03	0.62±0.03	0.64±0.01	0.60±0.01	0.71±0.02	0.61±0.01	0.90±0.03	0.90±0.02
2c	0.95±0.01	0.96±0.01	0.93±0.01	0.94±0.01	0.94±0.00	0.96±0.00	0.93±0.01	0.92±0.03	0.90±0.02	0.93±0.01
2d	0.94±0.01	0.94±0.03	0.94±0.00	0.94±0.01	.94±0.00	0.96±0.00	0.93±0.01	0.92±0.03	0.88±0.02	0.89±0.02
2e	0.93±0.02	0.88±0.03	0.92±0.02	0.82±0.07	0.45±0.03	0.20±0.01	0.76±0.08	0.52±0.10	0.90±0.00	0.83±0.01
2f	0.95±0.01	0.93±0.01	0.89±0.06	0.79±0.10	0.45±0.03	0.20±0.01	0.80±0.04	0.56±0.02	0.92±0.00	0.81±0.02
3a	0.99±0.00	0.99±0.00	-	-	0.93±0.00	0.85±0.01	0.93±0.01	0.84±0.03	0.99±0.00	0.98±0.00
3b	0.96±0.00	0.97±0.00	-	-	0.91±0.01	0.89±0.01	0.95±0.00	0.93±0.01	0.97±0.01	0.97±0.01
3c	0.99±0.01	0.99±0.01	-	-	0.89±0.01	0.79±0.02	0.96±0.00	0.91±0.01	0.98±0.01	0.97±0.01
3d	0.97±0.01	0.97±0.01	-	-	0.91±0.01	0.89±0.01	0.95±0.00	0.93±0.01	0.96±0.02	0.96±0.02

A.2.2.4 Results: Outlook to Unsupervised Anomaly Detection – Experiment 4

Table A.5 summarizes the results for this experiment.

Table A.5: Improved A^3 architecture. Test result for experiment 4, where no anomaly samples were used to train A^3.

4a-AUC	**4a**-AP	**4b**-AUC	**4b**-AP	**4c**-AUC	**4c**-AP	**4d**-AUC	**4d**-AP
0.97±0.01	0.94±0.03	0.91±0.06	0.90±0.06	0.68±0.05	0.49±0.03	0.64±0.04	0.61±0.05

A.3 Entropic Retraining Appendix

In this section we show supplemental material for Chapter 4 based on our publication "Optimizing Information Loss Towards Robust Neural Networks" [10].

A.3.1 Robustness of Softmax-Activated Output Neural Networks after Entropic Retraining

Table A.6 shows the robustness of the tested NNs with softmax outputs after entropic retraining.

Table A.6: Attack success rates for PGD and mean l_p-distances for DF and C&W when attacking the neural networks after entropic retraining with softmax-activated outputs.

Data	**Model**	**Attack Success Rate or Mean l_p–Distance:**				
		$\boldsymbol{PGD_\infty}$	$\boldsymbol{PGD_2}$	$\boldsymbol{DF_\infty}$	$\boldsymbol{DF_2}$	$\boldsymbol{C\&W_2}$
MNIST	LeNet5	18.0%	8.1%	0.19	1.75	4.33
	KerasExM	2.6%	1.8%	0.05	0.23	2.24
Fashion-MNIST	KerasExF	24.3%	16.2%	0.05	0.44	0.04
CIFAR10	KerasExC	84.9%	73.2%	0.02	0.64	0.15

Table A.7 shows the differences of the robustness metrics between the unprotected models as well as the models after entropic retraining. Both sets of models have softmax outputs.

Table A.7: Relative change in robustness after entropic retraining for the models with softmax outputs. Black and gray digits indicate a positive or negative impact on the robustness, respectively.

Data	**Model**	**Attack Success Rate or Mean l_p–Distance:**				
		$\boldsymbol{PGD_\infty}$	$\boldsymbol{PGD_2}$	$\boldsymbol{DF_\infty}$	$\boldsymbol{DF_2}$	$\boldsymbol{C\&W_2}$
MNIST	LeNet5	−82%	−92%	−35%	−59%	152%
	KerasExM	−97%	−98%	−83%	−94%	37%
Fashion-MNIST	KerasExF	−76%	−84%	−38%	−58%	−91%
CIFAR10	KerasExC	−15%	−27%	24%	28%	−30%