Verification-Aided Deep Ensemble Selection

Guy Amir, Tom Zelazny, Guy Katz and Michael Schapira
The Hebrew University of Jerusalem, Jerusalem, Israel
$\{$guyam, tomz, guykatz, schapiram$\}$@cs.huji.ac.il

***Abstract*—Deep neural networks (DNNs) have become the technology of choice for realizing a variety of complex tasks. However, as highlighted by many recent studies, even an imperceptible perturbation to a correctly classified input can lead to misclassification by a DNN. This renders DNNs vulnerable to strategic input manipulations by attackers, and also oversensitive to environmental noise. To mitigate this phenomenon, practitioners apply joint classification by an *ensemble* of DNNs. By aggregating the classification outputs of different individual DNNs for the same input, ensemble-based classification reduces the risk of misclassifications due to the specific realization of the stochastic training process of any single DNN. However, the effectiveness of a DNN ensemble is highly dependent on its members *not simultaneously erring* on many different inputs. In this case study, we harness recent advances in DNN verification to devise a methodology for identifying ensemble compositions that are less prone to simultaneous errors, even when the input is adversarially perturbed — resulting in more *robustly-accurate* ensemble-based classification. Our proposed framework uses a DNN verifier as a backend, and includes heuristics that help reduce the high complexity of directly verifying ensembles. More broadly, our work puts forth a novel universal objective for formal verification that can potentially improve the robustness of real-world, deep-learning-based systems across a variety of application domains.**

I. Introduction

In recent years, deep learning [32] has emerged as the state-of-the-art solution for a myriad of tasks. Through the automated training of *deep neural networks* (*DNNs*), engineers can create systems capable of correctly handling previously unencountered inputs. DNNs excel at tasks ranging from image recognition and natural language processing to game playing and protein folding [2], [20], [37], [47], [75], [76], and are expected to play a key role in various complex systems [14], [43].

Despite their immense success, DNNs suffer from severe vulnerabilities and weaknesses. A prominent example is the sensitivity of DNNs to *adversarial inputs* [33], [48], [81], i.e., slight perturbations of correctly-classified inputs that result in misclassifications. The susceptibility of DNNs to input perturbations involves two risks that limit the applicability of deep learning to mission-critical tasks: (1) falling victim to strategic input manipulations by *attackers*, and (2) failing to *generalize* well in the presence of environmental noise. In light of the above, recent work has focused on enhancing the *robustness* of DNN-based classification to adversarial inputs while preserving *accuracy* [12], [28], [62], [83], [99]. Informally, a classifier is *robustly accurate* (aka *astute* [88]) with respect to a given distribution over inputs, if it continues to correctly classify inputs drawn from this distribution, with high probability, even when these inputs are arbitrarily perturbed (up to some maximally allowed perturbation).

We focus here on a classic technique for improving classification quality [8], [52]: combining the outputs of an *ensemble* [27], [36], [82] of DNN-based classifiers on an input to derive a joint classification decision for that input. By incorporating the outputs of *independently-trained* DNNs, ensembles mitigate the risk of misclassification of a single DNN due to a specific realization of its stochastic training process and the specifics of its training data traversal. For a DNN ensemble to provide a meaningful improvement over utilizing a single DNN, its members should not frequently misclassify *the same* input. Consider, for instance, an extreme example, where an ensemble with $k = 10$ members is used, but for some part of the input space, the 10 DNNs effectively behave identically, making mistakes on the exact same inputs. In this scenario, the ensemble as a whole is no more robust on this input subspace than each of its individual members. Our objective is to demonstrate how recent advances in DNN verification [39], [44] can be harnessed to provide system designers and engineers with the means to avoid such scenarios, by constructing adequately diverse ensembles.

Significant progress has recently been made on formal verification techniques for DNNs [1], [7], [10], [11], [25], [56], [68], [77], [92]. The basic DNN verification query is to determine, given a DNN N, a precondition P, and a postcondition Q, whether there exists an input x such that $P(x)$ and $Q(N(x))$ both hold. Recent verification work has focused on *identifying* adversarial inputs to DNN-based classification, or formally proving that no such inputs exist [29], [34], [58]. We demonstrate the applicability of DNN verification to solving a new kind of queries, pertaining to DNN ensembles, which could significantly boost the robustness of these ensembles (as opposed to just measuring the robustness of individual DNNs). We note that despite great strides in recent years [46], [58], [77], even state-of-the-art DNN verification tools face severe scalability limitations. This renders solving verification queries pertaining to ensembles extremely challenging, since the complexity of this task grows exponentially with the number of ensemble members (see Section III).

In this case-study paper, we propose and evaluate an efficient and scalable approach for verifying that different ensemble members do not tend to err simultaneously. Specifically, our scheme considers *small subsets* of ensemble members,[1] and dispatches verification queries to seek perturbations of

[*] This is an extended version of a paper with the same title that appeared at FMCAD 2022.

[1] While our technique is applicable to subsets of any size, we focused on pairs in our evaluation, as we later elaborate.

inputs for which *all* members in the subset err *simultaneously*. By identifying such inputs, we can assign a *mutual error score* to each subset. Using these mutual error scores, we compute, for each individual ensemble member, a *uniqueness score* that signifies how often it errs simultaneously with other ensemble members. This score can be used to detect the "weakest" ensemble members, i.e. those most prone to erring in parallel to others, and replace them with fresh DNNs — thus enhancing the diversity among the ensemble members, and improving the overall robust accuracy of the ensemble.

To evaluate our scheme, we implemented it as a proof-of-concept tool, and used this tool to conduct extensive experimentation on DNN ensembles for classifying digits and clothing items. Our results demonstrate that by identifying the weakest ensemble members (using verification) and replacing them, the robust accuracy of the ensemble as a whole may be significantly improved. Additional experiments that we conducted also demonstrate that our verification-driven approach affords significant advantages when compared to competing, non-verification-based, methods. Together, these results showcase the potential of our approach. Our code and benchmarks are publicly available online [6].

The rest of the paper is organized as follows. Section II contains background on DNN ensembles and DNN verification. In Section III we present our verification-based methodology for ensemble selection, and then present our case study in Section IV. Next, in Section V we compare our verification-based approach to state-of-the-art, gradient-based, methods. Related work is covered in Section VI, and we conclude and discuss future work in Section VII.

II. Background

Deep Neural Networks. A deep neural network (DNN) [32] is a directed graph, comprised of layers of nodes (also known as *neurons*). In feed-forward DNNs, data flows sequentially from the first (*input*) layer, through a sequence of intermediate (*hidden*) layers, and finally into an *output* layer. The network's output is evaluated by assigning values to the input layer's neurons and computing the value assignment for neurons in each of the following layers, in order, until reaching the output layer and returning its neuron values to the user. In classification networks, which are our subject matter here, each output neuron corresponds to an output *class*; and the output neuron with the highest value represents the class, or label, which the particular input is being classified as.

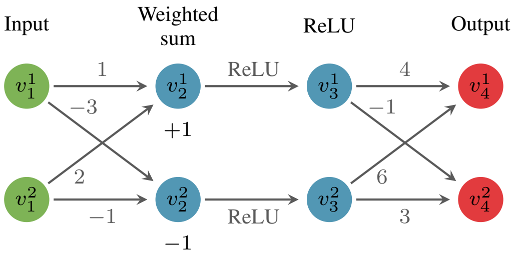

Fig. 1: A toy DNN.

Fig. 1 depicts a toy DNN. It has an input layer with two neurons, followed by a *weighted sum layer*, which computes an affine transformation of values from its preceding layer. For example, for input $V_1 = [1, -5]^T$, the second layer's computed values are $V_2 = [-8, 1]^T$. Next is a ReLU layer, which applies the ReLU function $\text{ReLU}(x) = \max(0, x)$ to each individual neuron, resulting in $V_3 = [0, 1]^T$. Finally, the network's output layer again computes an affine transformation, resulting in the output $V_4 = [6, 3]^T$. Thus, input $[1, -5]^T$ is classified as the label corresponding to neuron v_4^1. For additional details, see [32].

Accuracy, Robustness, and Deep Ensembles. The weights of a DNN are determined through its training process. In supervised learning, we are provided a set of pairs (x_i, l_i) drawn according to some (unknown) distribution D, where x_i is an input point and l_i is a ground-truth label for that input. The goal is to select weights for the DNN N that maximize its *accuracy*, which is defined as: $Pr_{(x,l)\sim D}(N(x) = l)$ (we slightly abuse notation, and use $N(x)$ to denote both the network's output vector, as well as the label it assigns x).

We restrict our attention to the *classification* setting, in which labels are discrete. The training of a DNN-based classifier is typically a stochastic process. This process is affected, for example, by the initial assignment of weights to the DNN, the order in which training data is traversed, and more. A prominent method for avoiding misclassifications originating from the stochastic training of a single DNN is employing *deep ensembles*. A deep ensemble is a set $\mathcal{E} = \{N_1, \ldots, N_k\}$ of k independently-trained DNNs. The ensemble classifies an input by aggregating the individual classification outputs of its members (see Fig. 2). The collective decision is typically achieved by averaging over all members' outputs. Ensembles have been shown to often achieve better accuracy than their individual members [8], [52], [57], [94].

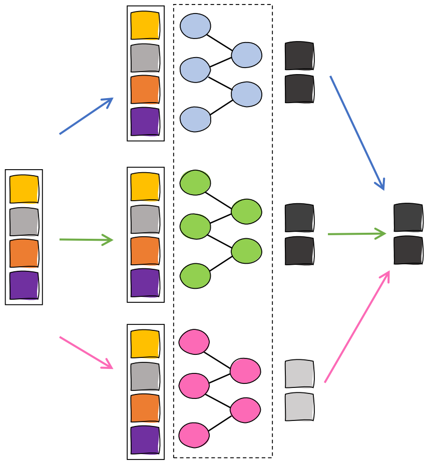

Fig. 2: An ensemble comprising three DNNs. Each input vector is independently classified by all three networks, and the results are aggregated into a final classification.

A critical condition for the success of ensemble-based classifiers is that the ensemble members' misclassifications are not strongly correlated [53], [63], [80]. This key property is crucial in order to avoid a scenario where many different members of the ensemble frequently make mistakes on the same input, causing the ensemble as a whole to also err on that input. Heuristics for achieving diversity across ensemble members include, e.g., training the members simultaneously with diversity-aware loss [42], [52], randomly initializing different weights for the ensemble members [50], and other methods [63], [74].

Since the discovery of adversarial inputs, practitioners have become interested in DNNs that are not only accurate but also *robustly accurate*. We say that a network N is ϵ-robust around the point x if every input point that is at most ϵ away from x receives the same classification as x: $\|x' - x\| \leq \epsilon \Rightarrow N(x) = N(x')$, where $N(x)$ is the label assigned to x; and the definition of accuracy is generalized to ϵ-robust accuracy as follows: $Pr_{(x,l)\sim D}(\|x' - x\| \leq \epsilon \Rightarrow N(x') = l)$. While improvements in accuracy afforded by ensembles are straightforward to measure, this is typically not the case for robust accuracy, as we discuss in Section III.

DNN Verification. Given a DNN N, a verification query on N specifies a precondition P on N's input vector x, and a postcondition Q on N's output vector $N(x)$. A DNN verifier needs to determine whether there exists a concrete input x_0 that satisfies $P(x_0) \wedge Q(N(x_0))$ (the `SAT` case), or not (the `UNSAT` case). Typically, P and Q are expressed in the logic of linear real arithmetic. For instance, the ϵ-robustness of a DNN around a point x can be phrased as a DNN verification query, and then dispatched using existing technology [29], [44], [87]. The DNN verification problem is known to be NP-complete [45].

III. Improving Robust Accuracy using Verification

A. Directly Quantifying Robust Accuracy is Hard

In order to construct a robustly-accurate ensemble $\mathcal{E}$ with k members, we train a set of $n > k$ DNNs and then seek to select a subset of k DNNs that provides high robust accuracy. This method of training multiple models and then discarding a subset thereof is known as *ensemble pruning*, and is a common practice in deep-ensemble training [13], [100]. In our case, a straightforward approach to do so would be to quantify the robust accuracy for all possible k-sized DNN-subsets, and then pick the best one. This, however, is computationally expensive, and requires an accurate estimate of the robust accuracy of an ensemble.

A natural approach for estimating the ϵ-robust accuracy of a DNN is to verify, for many points in the test data, that the DNN yields an accurate label not only on each data point itself, but also on each and every input derived from that data point via an ϵ-perturbation [29]. The fraction of tested points for which this is indeed the case can be used to estimate the accuracy of the classifier on the underlying distribution from which the data is generated.

A similar process can be performed for an ensemble $\mathcal{E} = \{N_1, \ldots, N_k\}$, by first constructing a single, large DNN $N_{\mathcal{E}}$ that aggregates $\mathcal{E}$'s joint classification, and then verifying its robustness on a set of points from the test data (see Section A of the Appendix). However, this approach faces a significant scalability barrier: the DNN ensemble, $N_{\mathcal{E}}$, comprised of all k member-DNNs is (roughly) k times larger than any of the N_i's, and since DNN verification becomes exponentially harder as the DNN size increases, $N_{\mathcal{E}}$'s size might render efficient verification infeasible. As we demonstrate later, this is the case even when the constituent networks themselves are fairly small. Our proposed methodology circumvents this difficulty by only solving verification queries pertaining to *very small* sets of DNNs.

B. Mutual Error Scores and Uniqueness Scores

In general, the less likely it is that members of an ensemble err simultaneously with other members, the more accurate the ensemble is. This motivates our definition of mutual error scores below.

Definition 1 (Agreement Points): Given an ensemble $\mathcal{E} = \{N_1, N_2, \ldots, N_k\}$, we say that an input point x_0 is an *agreement point* for $\mathcal{E}$ if there is some label y_0 such that $N_i(x_0) = y_0$ for all $i \in [k]$. We let $\mathcal{E}(x_0)$ denote the label y_0.

As we later discuss, the ϵ-neighborhoods of agreement points are natural locations for detecting hidden tendencies of ensemble members to err together.

Definition 2 (Mutual Errors): Let $\mathcal{E}$ be an ensemble, and let x_0 be an agreement point for $\mathcal{E}$. Let $B_{x_0,\epsilon}$ be the ϵ-ball around x_0, $B_{x_0,\epsilon} = \{x \mid \|x - x_0\|_\infty \leq \epsilon\}$. We say that N_1 and N_2 have a *mutual error* in B if there exists a point $x \in B_{x_0,\epsilon}$ such that $N_1(x) \neq \mathcal{E}(x_0)$ and $N_2(x) \neq \mathcal{E}(x_0)$.

Intuitively, if N_1 and N_2 have many mutual errors, incorporating both into an ensemble is a poor choice. This naturally gives rise to the following definition:

Definition 3 (Mutual Error Scores): Let A be a finite set of m agreement points in an ensemble $\mathcal{E}$'s input space, and let $B_1, B_2, \ldots, B_m$ denote the ϵ-balls surrounding the points in A. Let N_1, N_2 denote two members of $\mathcal{E}$. The *mutual error score* of N_1 and N_2 with respect to $\mathcal{E}$ and A is denoted by $\mathrm{ME}_{\mathcal{E},A}(N_1, N_2)$, and defined as:

$$\mathrm{ME}_{\mathcal{E},A}(N_1, N_2) = \frac{|\{i \mid N_1 \text{ and } N_2 \text{ have a mutual error in } B_i\}|}{m}$$

Observe that $\mathrm{ME}_{\mathcal{E},A}(N_1, N_2)$ is always in the range $[0, 1]$. The closer it is to 1, the more mutual errors N_1 and N_2 have, making it unwise to place them in the same ensemble.

Definition 4 (Uniqueness Scores): Given an ensemble $\mathcal{E} = \{N_1, N_2, \ldots, N_n\}$ and a set A of agreement points for $\mathcal{E}$, we define, for each ensemble member N_i, the *uniqueness score* for N_i with respect to $\mathcal{E}$ and A, $\mathrm{US}_{\mathcal{E},A}(N_i)$, as:

$$\mathrm{US}_{\mathcal{E},A}(N_i) = 1 - \frac{\sum_{j \neq i} \mathrm{ME}_{\mathcal{E},A}(N_i, N_j)}{n-1}$$

The uniqueness score (US) of N_i is the complement of its average mutual error score with the other ensemble members.

When this score is close to 0, N_i tends to err simultaneously with other members of the ensemble on points in A. In contrast, the closer the uniqueness score is to 1, the rarer it is for N_i to misclassify the same inputs as other members of the ensemble. Hence, ensemble members with low uniqueness scores are, intuitively, good candidates for replacement.

We point out that our definitions above can naturally be generalized to larger subsets of the ensemble members — thus measuring robust accuracy more precisely, but rendering these measurements more complex to perform in practice.

Computing Mutual Errors. The only computationally complex step in determining the uniqueness scores of individual ensemble members is computing the pairwise mutual errors for the ensemble. To this end, we leverage DNN verification technology. Specifically, given two ensemble members N_1 and N_2, an agreement point a for the ensemble with label l, and $\epsilon > 0$, an appropriate DNN verification query can be formulated as follows. First, we construct from N_1 and N_2 a single, larger DNN N, which captures N_1 and N_2 simultaneously processing a shared input vector, side-by-side. This network N is then passed to a DNN verifier, with the precondition that the input be restricted to B, an ϵ-ball around a, and the postcondition that (1) among N's output neurons that correspond to the outputs of N_1, the neuron representing l not be maximal, and (2) among N's output neurons that correspond to the outputs of N_2, the neuron representing l not be maximal. Such queries are supported by most available DNN verification engines. We note that this encoding (depicted in Figure 3), where two networks and their output constraints are combined into a single query, is crucial for finding inputs on which both DNNs err *simultaneously*. For additional details, see Section B of the Appendix.

C. Ensemble Selection using Uniqueness Scores

An Iterative Scheme. Building on our verification-based method for computing mutual error scores, we propose an iterative scheme for constructing an ensemble. Our scheme consists of the following steps:

1) independently train a set $\mathcal{N}$ of n DNNs, and identify a set A of m agreement points that are *correctly classified* by all n DNNs.[2] This is done by sequentially checking points from the validation dataset;
2) arbitrarily choose an initial candidate ensemble $\mathcal{E}$ of size $k < n$, and compute (using a verification engine backend) all mutual error scores for the DNN members comprising $\mathcal{E}$, with respect to A;
3) compute the uniqueness score for each ensemble member, and identify a DNN member N_l with a low score;
4) identify a fresh DNN N_f, not currently in $\mathcal{E}$, that has a higher uniqueness score than N_l, if one exists, and replace N_l with N_f. Specifically, identify a DNN $N_f \in \mathcal{N} \setminus \mathcal{E}$, such that the uniqueness score of N_f with respect to the ensemble $\mathcal{E} \setminus \{N_l\} \cup \{N_f\}$ and the point set A, namely $\text{US}_{\mathcal{E}\setminus\{N_l\}\cup\{N_f\},A}(N_f)$, is maximal. If this score is greater than $\text{US}_{\mathcal{E},A}(N_l)$, replace N_l with N_f, i.e. set $\mathcal{E} := \mathcal{E} \setminus \{N_l\} \cup \{N_f\}$; and
5) repeat Steps (2) through (4), until no N_f is found or until the user-provided timeout or maximal iteration count are exceeded.

Intuitively, after starting with an arbitrary ensemble, we run multiple iterations, each time trying to improve the ensemble. Specifically, we identify the "weakest" member of the current ensemble, and replace it with a fresh DNN that obtains a higher uniqueness score relevant to the remaining members — thus ensuring that each change that we make improves the overall robust accuracy on the fixed set of agreement points.

The greedy search procedure is repeated for the new candidate ensemble, and so on. The process terminates after a predefined number of iterations is reached, when the process converges (no further improvement is achievable on the fixed set of agreement points), or when a predefined timeout value is exceeded.

On the Importance of Agreement Points. Our iterative scheme for constructing an ensemble starts with an arbitrary selection of k candidate members, and then computes the uniqueness score for each member. As mentioned, the uniqueness scores are computed with respect to a fixed set of agreement points, pre-selected from the validation data (which is labeled data, not used for training the DNNs).

We point out that agreement points are data points on which there is overwhelming consensus among ensemble members, despite the specific realization of the training process of each member. As such, agreement points correspond to data points that are "easy" to label correctly. Consequently, data points in close proximity of an agreement point are rarely classified differently than the agreement point by an individual ensemble member, let alone by multiple members simultaneously. As our objective is to expose implicit tendencies of ensemble members to err together, the close neighborhood of agreement points is a natural area for seeking joint deviations from the consensual label (which are expected to be extremely rare). In our evaluation, we computed uniqueness scores based solely on *correctly-classified* agreement points and ignored any incorrectly-classified agreement points.[3]

As we later demonstrate, a small set of correctly-classified agreement points from the validation set can be used, in practice, to identify ensemble members that tend to err simultaneously on *other* data points. We note that this is also the case even when the chosen agreement points are all identically labeled.

Monotonicity and Convergence. Using our approach, an ensemble member is replaced with a fresh DNN only if this replacement leads to *strictly* fewer joint errors with the *remaining members* on the fixed set of agreement points. Thus, the total number of joint errors decreases with every replacement; and, as this number is trivially lower-bounded by 0, this ("potential-function" style) argument establishes the process's monotonicity and convergence.

[2] In our experiments, we arbitrarily chose $k = 5$, $n = 10$ and $m = 200$.

[3] For example, in our MNIST experiments 99.7% of the agreement points were correctly classified by all individual DNNs, and by the ensemble as a whole.

By iteratively reducing the number of joint errors across all pairs of chosen ensemble members, our iterative process improves the robust accuracy of the resulting ensemble on the fixed set of agreement points. This, however, does not guarantee improved robust accuracy over the entire input domain. Nonetheless, we show in Section IV that such an improvement does typically occur in practice, even on randomly sampled subsets of input points (which are not necessarily agreement points).

IV. Case Study: MNIST and Fashion-MNIST

Below, we present the evaluation of our methodology on two datasets: the MNIST dataset for handwritten digit recognition [51], and the Fashion-MNIST dataset for clothing classification [93]. Our results for both datasets demonstrate that our technique facilitates choosing ensembles that provide high robust accuracy via relatively few, efficient verification queries.

The considered datasets are conducive for our purposes since they allow attaining high accuracy using fairly small DNNs, which enables us to *directly quantify* the robust accuracy of an entire ensemble, by dispatching verification queries that would otherwise be intractable (see Section III-A). This provides the ground truth required for assessing the benefits of our approach. The scalability afforded by our approach is crucial even for handling the relatively modest-sized DNNs considered: on the MNIST data, for instance, mutual-error verification queries for two ensemble members typically took a few seconds, whereas verification queries involving the full ensemble of five networks often timed out (35% of the queries on the MNIST data timed out after 24 hours, versus only roughly 1% of the pairwise mutual-error queries). As constituent DNN sizes and ensemble sizes increase, this gap in scalability is expected to become even more significant.

Our verification queries were dispatched using the Marabou verification engine [46] (although other engines could also be used). Additional details regarding the encoding of the verification queries appear in Sections A and B of the Appendix, and additional details about the experimental results appear in Section C therein. We have publicly released our code, as well as all benchmarks and experimental data, within an artifact accompanying this paper [6].

MNIST. For this part of our evaluation, we trained 10 independent DNNs $\{N_1, \ldots, N_{10}\}$ over the MNIST dataset [51], which includes 28×28 grayscale images of 10 handwritten digits (from "0" to "9"). Each of these networks had the same architecture: an input layer of 784 neurons, followed by a fully-connected layer with 30 neurons, a ReLU layer, another fully-connected layer with 10 neurons, and a final softmax layer with 10 output neurons, corresponding to the 10 possible digit labels.[4] All networks achieved high accuracy rates of $96.29\% - 96.57\%$ (see Table I).

After training, we arbitrarily constructed two distinct ensembles with five DNN members each: $\mathcal{E}_1 = \{N_1, \ldots, N_5\}$ and $\mathcal{E}_2 = \{N_6, \ldots, N_{10}\}$, with an accuracy of 97.8% and 97.3%, respectively. Notice that the ensembles achieve a higher accuracy over the test set than their individual members.

We then applied our method in an attempt to improve the robust accuracy of $\mathcal{E}_1$. We began by searching the validation set, and identifying 200 agreement points (the set A),[5] all correctly labeled as "0" by all 10 networks.[6] Using the 200 agreement points and 6 different perturbation sizes[7] $\epsilon \in \{0.01, 0.02, 0.03, 0.04, 0.05, 0.06\}$, we constructed 1200 ϵ-balls around the selected agreement points; and then, for every ball B and for every pair $N_i, N_j \in \mathcal{E}_1$, we encoded a verification query to check whether N_i and N_j have a mutual error in B (see example in Fig. 3). This resulted in $\binom{5}{2} \cdot 200 \cdot 6 = 12000$ verification queries, which we dispatched using the Marabou DNN verifier [46] (each query ran with a 2-hour timeout limit). Finally, we used the results to compute the uniqueness score for each network in $\mathcal{E}_1$; these results, which appear briefly in Table I (for $\epsilon = 0.02$) and appear in full in Table V of the Appendix, clearly show that two of the members, N_2 and N_5, are each relatively prone to erring simultaneously with the remaining four members of $\mathcal{E}_1$.

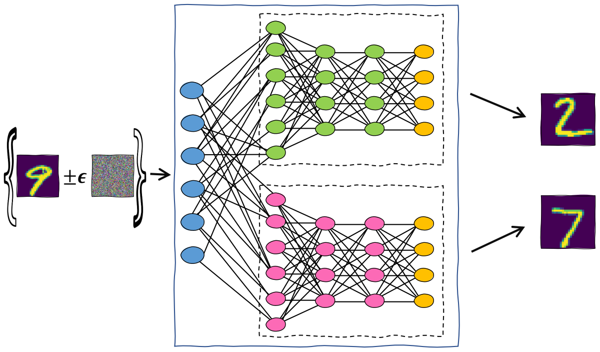

Fig. 3: Checking whether two MNIST digit recognition networks have a mutual error around an agreement point labeled "9". In this case, the same perturbation causes one network to output the incorrect label "2", and the other network to output the incorrect label "7".

Next, we began searching among the remaining networks, $N_6, \ldots, N_{10}$, for good replacements for N_2 and N_5. Specifically, we searched for networks that obtained higher US scores than N_2 and N_5. To achieve this, we began modifying $\mathcal{E}_1$, each time removing either N_2 or N_5, replacing them with one of the remaining networks, and computing the uniqueness scores for the new members (with respect to the four remaining original networks). We observed that for both N_2 and N_5, network N_9

[4] Although the DNNs all have the same size and architecture, common ensemble training processes randomly initialize their weights, and also randomly pick samples from the same training set (see [50]). This is the cause for diversity among ensemble members, which our algorithm later detects.

[5] In our experiments, we empirically selected 200 agreement points in order to balance between precision (a higher number of points) and verification speed (a smaller number of points). This selection is based on a user's available computing power.

[6] The "0" label is the label with the highest accuracy among the trained ensemble members, and thus "0"-labeled agreement points represent areas in the input space with extremely high consensus.

[7] ϵ values which are too small, or too large, render the queries trivial. Thus, we found it to be useful to use a varied selection of ϵ values.

TABLE I: Accuracy and uniqueness scores for the MNIST networks. Uniqueness scores are measured with respect to the ensemble (either $\mathcal{E}_1$ or $\mathcal{E}_2$).

	$\mathcal{E}_1$					$\mathcal{E}_2$				
	N_1	N_2	N_3	N_4	N_5	N_6	N_7	N_8	N_9	N_{10}
Accuracy	96.42%	96.55%	96.40%	96.46%	96.29%	96.44%	96.48%	96.57%	96.51%	96.46%
US	90.75%	**88.38%**	90.63%	92.13%	**88.63%**	97.38%	96.75%	97.5%	**98.88%**	**97.75%**

was a good replacement, obtaining very high US values. For additional details, see Section C of the Appendix.

Finally, to evaluate the effect of our changes to $\mathcal{E}_1$, we constructed the two new ensembles, $\mathcal{E}_1^{2\to 9} = \{N_1, N_9, N_3, N_4, N_5\}$ and $\mathcal{E}_1^{5\to 9} = \{N_1, N_2, N_3, N_4, N_9\}$. Computing the new ensembles' robust accuracy over the entire test set is computationally expensive, and thus we sampled 200 random points from the test set (these did not necessarily have the same label, nor were they required to be agreement points for the ensemble). For each sample, we created a verification query to check the robust accuracy of the new ensembles around the point, compared to the original ensemble. The results are plotted in Fig. 4, and indicate that the new ensembles demonstrated *significantly higher* robust accuracy on the tested points. These results validate our claim that a scoring metric based on agreement points is useful in improving the ensemble's robustness also on other, "harder", input points. Our analysis also indicates that the improved robustness results originated not only from ϵ-balls around inputs labeled as "0", but from other labels as well. In fact, the gain in robustness was not just in quantity, but also in quality: for almost all cases, whenever $\mathcal{E}_1$ proved robust around an input, so did $\mathcal{E}_1^{2\to 9}$ and $\mathcal{E}_1^{5\to 9}$. This indicates that the improved robustness originated from inputs on which $\mathcal{E}_1$ was prone to err.

Next, we turned our attention to $\mathcal{E}_2$, and computed the uniqueness scores for each of its members (see Table I). This time we conducted a "reverse" experiment: we identified the two *best* members of $\mathcal{E}_2$, i.e. the two networks that had the highest uniqueness scores, and were consequently the least prone to err simultaneously. These turned out to be networks N_9 and N_{10}. Next, we replaced each of these networks with each of the networks $\{N_1, \ldots, N_5\}$, in order to identify a network that, when inserted into $\mathcal{E}_2$, achieved a lower score than N_9 and N_{10}. N_4 turned out to be such a network. We created the two modified ensembles, $\mathcal{E}_2^{9\to 4} = \{N_6, N_7, N_8, N_4, N_{10}\}$ and $\mathcal{E}_2^{10\to 4} = \{N_6, N_7, N_8, N_9, N_4\}$, and compared their robust accuracy to that of $\mathcal{E}_2$ on 200 random points from the test set. The results, depicted in Fig. 4, indicate that the ensemble's robust accuracy decreased significantly, as expected.

In both aforementioned experiments, we also computed the *accuracy* (as opposed to *robust accuracy*) of the new ensembles, by evaluating them over the test set. All new ensembles had an accuracy that was on par with that of the original ensembles — specifically, within a range of $\pm 0.2\%$ from the original ensembles' accuracy.

Fashion-MNIST. For the second part of our evaluation, we trained 10 independent DNNs $\{N_{11}, \ldots, N_{20}\}$ over the Fashion-MNIST dataset [93], which includes 28×28 grayscale images of 10 clothing categories ("Coat", "Dress", etc.), and is considered more complex than the MNIST dataset. Each DNN had the same architecture as the MNIST-trained DNNs, and achieved an accuracy of 87.05%–87.53% (see Table II). We arbitrarily constructed two distinct ensembles, $\mathcal{E}_3 = \{N_{11}, \ldots, N_{15}\}$ and $\mathcal{E}_4 = \{N_{16}, \ldots, N_{20}\}$, with an accuracy of 88.22% and 88.48%, respectively.

Next, we again computed the US values of each of the networks. The results, which appear in full in Tables XI and XIII of the Appendix, indicate a high variance among the uniqueness scores of the members of $\mathcal{E}_4$, as compared to the relatively similar scores of $\mathcal{E}_3$'s members. We thus chose to focus on $\mathcal{E}_4$. Based on the computed US values, we identified N_{20} as its least unique DNN; and, by replacing N_{20} with each of the five networks not currently in $\mathcal{E}_4$, identified that N_{15} is a good candidate for replacing N_{20}. Performing our validation step over $\mathcal{E}_4^{20\to 15}$ revealed that its robust accuracy has indeed increased. Running the "reverse" experiment, in which $\mathcal{E}_4$'s most unique member is replaced with a worse candidate, led us to consider the ensemble $\mathcal{E}_4^{18\to 13}$, which indeed demonstrated lower robust accuracy than the original ensemble. For additional details, see Section C of the Appendix.

For the final step of our experiment, we used our approach to iteratively switch two members of an ensemble. Specifically, after creating $\mathcal{E}_4^{20\to 15}$, which had higher robust accuracy than $\mathcal{E}_4$, we re-computed the US scores of its members, and identified again the least unique member — in this case, N_{16}. Per our computation, the best candidate for replacing it was N_{12}. The resulting ensemble, namely $\mathcal{E}_4^{20\to 15, 16\to 12}$, indeed demonstrated higher robust accuracy than both its predecessors. Performing another iteration of the "reverse" experiment yielded ensemble $\mathcal{E}_4^{18\to 13, 17\to 11}$, with poorer robust accuracy. The results appear in Fig. 5. We note that the only discrepancy, namely the robust accuracy of $\mathcal{E}_4^{20\to 15}$ being lower than that of $\mathcal{E}_4$ for $\epsilon = 0.04$, is due to timeouts.

Similarly to the MNIST case, the new ensembles in the Fashion-MNIST experiments obtained an accuracy that was on par with that of the original ensembles — specifically, within a range of $\pm 0.17\%$ from the original ensemble's accuracy.

V. Comparison to Gradient-Based Attacks

Current state-of-the-art approaches for assessing a network's robustness and robust accuracy rely on *gradient-based attacks* — a popular class of algorithms that, like verification methods, are capable of finding adversarial examples for a given neural network. In this section we compare our verification-based approach to these methods.

Gradient-based attacks generate adversarial examples by optimizing (via various techniques) a loss metric over the network's output, relative to its input. This allows these methods to effectively search the local surroundings of a

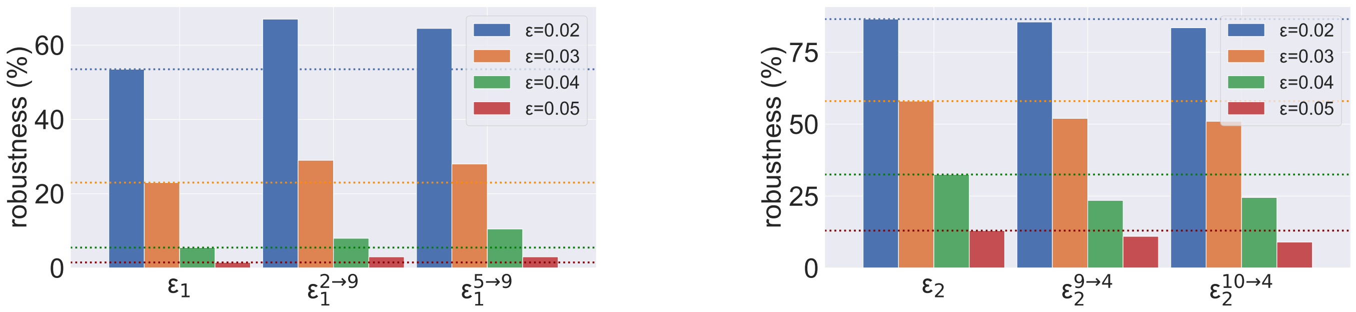

Fig. 4: The average robust accuracy scores for our original and modified ensembles. The results for $\epsilon = 0.01$ and $\epsilon = 0.06$ are trivial (the ensembles achieve near-perfect or near-zero robustness), and are omitted to reduce clutter.

TABLE II: Accuracy and uniqueness scores for the Fashion-MNIST networks. Uniqueness scores are measured with respect to the ensemble (either $\mathcal{E}_3$ or $\mathcal{E}_4$).

	$\mathcal{E}_3$					$\mathcal{E}_4$				
	N_{11}	N_{12}	N_{13}	N_{14}	N_{15}	N_{16}	N_{17}	N_{18}	N_{19}	N_{20}
Accuracy	87.14%	87.13%	87.53%	87.34%	87.3%	87.05%	87.32%	87.35%	87.34%	87.11%
US	70.63%	71.5%	**69.75%**	70.88%	**73.25%**	67.38%	72.38%	**80.13%**	71.38%	**66.75%**

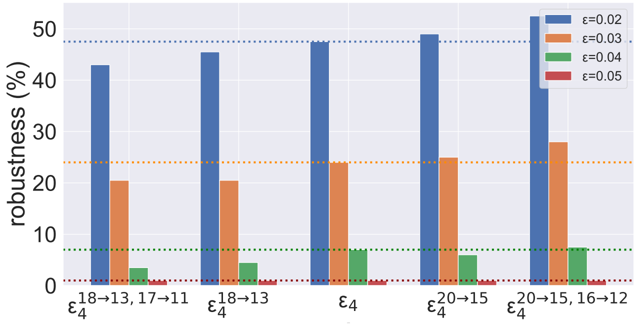

Fig. 5: The original ensemble $\mathcal{E}_4$ (center), ensembles modified to gain robust accuracy (right), and ensembles modified to reduce robust accuracy (left).

fixed input point for local optima, which often constitute adversarial inputs. Gradient-based methods, such as the *fast-gradient sign method* (FGSM) [38], *projected gradient descent* (PGD) [60], and others [48], [59], are in widespread use due to their scalability and relative ease of use. However, as we demonstrate here, they are often unsuitable in our setting.

In order to evaluate the effectiveness of gradient-based methods for measuring the robust accuracy of ensembles, we modified the common FGSM [38] and I-FGSM [48] ("Iterative FGSM") methods, thus extending them into three novel attacks aimed at finding adversarial examples that can fool multiple ensemble members simultaneously. We refer to these attacks as *Gradient Attack (G.A.) 1, 2, and 3*. For a thorough explanation of these attacks, as well as information about their design and implementation, see Section D of the Appendix.

Next, we used our three attacks to search for mutual errors of DNN pairs — i.e., adversarial examples that simultaneously affect a pair of DNNs. Specifically, we applied the attacks on both datasets (MNSIT and Fashion-MNIST), and searched for adversarial examples within various ϵ-balls around the same set of agreement points used in our previous experiments. This allowed us to subsequently compute, via gradient attacks, the mutual error scores of DNN pairs, and consequently, the uniqueness scores of each constituent ensemble member. The results of the total number of adversarial inputs found (`SAT` queries) are summarized in Table III. Each gradient attack typically took a few seconds to run. We also provide further details regarding the uniqueness scores computed by the three gradient-based methods in Section D of the Appendix, and in our accompanying artifact [6].

TABLE III: The number of `SAT` queries discovered when searching for an adversarial attack, using the three gradient attack methods ($G.A.$ 1, 2 and 3), and our verification approach.

Experiment	G.A. 1	G.A. 2	G.A. 3	verification
MNIST	1,333	3,886	5,574	**16,826**
Fashion-MNIST	17,190	21,245	22,129	**33,152**
Total	18,523	25,131	27,703	**49,978**

The results in Table III include a total of 108000 experiments, on all ensemble pairs.[8] In these experiments, our verification-based approach returned 49978 `SAT` results, while the strongest gradient-based method (gradient attack number 3) returned only 27703 `SAT` results — a 44% decrease in the number of counterexamples found. This discrepancy is on par with previous research [91], which indicates that gradient-based methods may err significantly when used for adversarial robustness analysis. This phenomenon manifests strongly in

[8] The 108000 experiments consist of $\binom{10}{2}$ pairs, times 200 agreement points, times 6 perturbation sizes, times 2 datasets.

our setting, which involves many small and medium-sized perturbations that gradient-based approaches struggle with [23].

The reduced precision afforded by gradient-based approaches can, in some cases, lead to sub-optimal ensemble selection choices when compared to our verification-based approaches. Specifically, even if a gradient-based approach produces a uniqueness score ranking that coincides with the one produced using verification, the dramatic decrease in the number of `SAT` queries leads to much smaller mutual error scores, and consequently — to uniqueness score values that are overly optimistic, and less capable of distinguishing between poor and superior robust accuracy results.

For example, when observing the first two arbitrary ensembles on the MNIST dataset, $\mathcal{E}_1$ and $\mathcal{E}_2$, the three gradient approaches (G.A. 1, 2 and 3) respectively assign average uniqueness scores of $\langle 95.4\%, 97.8\%\rangle$, $\langle 87.5\%, 94.5\%\rangle$ and $\langle 83.1\%, 92.5\%\rangle$ to the two ensembles (when averaging the US over all ensemble members and all perturbations). This indicates that the robust accuracy of the two ensembles is fairly similar (see Tables in section D of the appendix). In contrast, when using the more sensitive, verification-based approach, we find a substantially higher number of mutual errors (see Table III), and consequently, detect a much larger gap between the uniqueness scores of the two ensembles: 55% and 77%.

Another example that demonstrates the increased sensitivity of our method, when compared to gradient-based approaches, is obtained by observing the average uniqueness score of $\mathcal{E}_3$ and $\mathcal{E}_4$ on the Fashion-MNIST dataset. The strongest gradient attack that we used assigned almost identical average uniqueness scores to both ensembles (up to a difference of 0.01%), while our approach was sensitive enough to find a 2% difference between the average US of the two ensembles.

Finally, we note that, unlike verification-based approaches, gradient attacks are incomplete, and are consequently unable to return `UNSAT`. This makes them less suitable for assessing any additional uniqueness metrics based on robust ϵ-balls. We thus argue that, although gradient-based methods are faster and more scalable than verification, our results showcase the benefits of using verification-based approaches for assessing uniqueness scores and for ensemble selection.

VI. Related Work

Due to its pervasiveness, the phenomenon of adversarial inputs has received a significant amount of attention [26], [33], [61], [66], [67], [81], [101]. More specifically, the machine learning community has put a great deal of effort into measuring and improving the robustness of networks [17]–[19], [28], [35], [54], [60], [69], [72], [73], [89], [96]. The formal methods community has also been looking into the problem, by devising scalable DNN verification, optimization and monitoring techniques [1], [5], [7], [9]–[11], [15], [25], [40], [41], [55], [56], [65], [68], [71], [77], [92], [98]. To the best of our knowledge, ours is the first attempt to apply DNN verification to the setting of DNN ensembles. We note that our approach uses a DNN verifier strictly as a black-box backend, and so its scalability will improve as DNN verifiers become more scalable.

Obtaining DNN specifications to be verified is a difficult problem. While some studies have successfully applied verification to properties formulated by domain-specific experts [3], [4], [21], [24], [44], [79], most research has been focusing on *universal properties*, which pertain to every DNN-based system; specifically, local adversarial robustness [16], [34], [58], [77], fairness properties [85], network simplification [30] and modification [22], [31], [70], [78], [86], [95], and watermark resilience [31].

VII. Conclusion and Future Work

In this case-study paper, we demonstrate a novel technique for assessing a deep ensemble's robust accuracy through the use of DNN verification. To mitigate the difficulty inherent to verifying large ensembles, our approach considers pairs of networks, and computes for each ensemble member a score that indicates its tendency to make the same errors as other ensemble members. These scores allow us to iteratively improve the robust accuracy of the ensemble, by replacing weaker networks with stronger ones. Our empiric evaluation indicates the high practical potential of our approach; and, more broadly, we view this work as a part of the ongoing endeavor for demonstrating the real-world usefulness of DNN verification, by identifying additional, universal, DNN specifications.

Moving forward, we plan to tackle the natural open questions raised by our work; specifically, how our methodology for selecting robustly accurate ensembles can be extended beyond the current greedy search heuristic, as well as how ensembles should be selected in the context of other performance objectives, beyond robust accuracy. We also plan on experimenting with multiple stopping conditions for the ensemble member replacement process; as well as explore potential synergies between our verification-based approach and gradient-based approaches for computing mutual error scores. In addition, we note that we are currently extending our approach to regression learning ensembles and deep reinforcement learning ensembles. Finally, we are in the process of optimizing our approach by using lighter-weight, incomplete verification tools (e.g., [77], [90], [97]), which afford better scalability, and also support parallelization. This will hopefully allow us to handle significantly larger DNNs and more complex datasets.

Acknowledgements. We thank Haoze Wu for his contribution to this project. The first three authors were partially supported by the Israel Science Foundation (grant number 683/18). The first author was partially supported by the Center for Interdisciplinary Data Science Research at The Hebrew University of Jerusalem. The fourth author was partially supported by funding from Huawei.

References

[1] P. Alamdari, G. Avni, T. Henzinger, and A. Lukina. Formal Methods with a Touch of Magic. In *Proc. 20th Int. Conf. on Formal Methods in Computer-Aided Design (FMCAD)*, pages 138–147, 2020.

[2] M. AlQuraishi. AlphaFold at CASP13. *Bioinformatics*, 35(22):4862–4865, 2019.

[3] G. Amir, D. Corsi, R. Yerushalmi, L. Marzari, D. Harel, A. Farinelli, and G. Katz. Verifying Learning-Based Robotic Navigation Systems, 2022. Technical Report. https://arxiv.org/abs/2205.13536.

[4] G. Amir, M. Schapira, and G. Katz. Towards Scalable Verification of Deep Reinforcement Learning. In *Proc. 21st Int. Conf. on Formal Methods in Computer-Aided Design (FMCAD)*, pages 193–203, 2021.

[5] G. Amir, H. Wu, C. Barrett, and G. Katz. An SMT-Based Approach for Verifying Binarized Neural Networks. In *Proc. 27th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 203–222, 2021.

[6] G. Amir, T. Zelazny, G. Katz, and M. Schapira. Supplementary Artifact, 2022. https://zenodo.org/record/6557083.

[7] G. Anderson, S. Pailoor, I. Dillig, and S. Chaudhuri. Optimization and Abstraction: a Synergistic Approach for Analyzing Neural Network Robustness. In *Proc. 40th ACM SIGPLAN Conf. on Programming Languages Design and Implementations (PLDI)*, pages 731–744, 2019.

[8] O. Araque, I. Corcuera-Platas, J. Sánchez-Rada, and C. Iglesias. Enhancing Deep Learning Sentiment Analysis with Ensemble Techniques in Social Applications. *Expert Systems with Applications*, 77:236–246, 2017.

[9] P. Ashok, V. Hashemi, J. Kretinsky, and S. Mohr. DeepAbstract: Neural Network Abstraction for Accelerating Verification. In *Proc. 18th Int. Symp. on Automated Technology for Verification and Analysis (ATVA)*, pages 92–107, 2020.

[10] G. Avni, R. Bloem, K. Chatterjee, H. T., B. Konighofer, and S. Pranger. Run-Time Optimization for Learned Controllers through Quantitative Games. In *Proc. 31st Int. Conf. on Computer Aided Verification (CAV)*, pages 630–649, 2019.

[11] T. Baluta, S. Shen, S. Shinde, K. Meel, and P. Saxena. Quantitative Verification of Neural Networks and its Security Applications. In *Proc. ACM SIGSAC Conf. on Computer and Communications Security (CCS)*, pages 1249–1264, 2019.

[12] R. Bhattacharjee, S. Jha, and K. Chaudhuri. Sample Complexity of Robust Linear Classification on Separated Data. In *Proc. 38th Int. Conf. on Machine Learning (ICML)*, pages 884–893, 2021.

[13] Y. Bian, Y. Wang, Y. Yao, and H. Chen. Ensemble Pruning Based on Objection Maximization With a General Distributed Framework. *IEEE Transactions on Neural Networks and Learning Systems*, 31(9):3766–3774, 2019.

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

[15] R. Bunel, I. Turkaslan, P. Torr, P. Kohli, and P. Mudigonda. A Unified View of Piecewise Linear Neural Network Verification. In *Proc. 32nd Conf. on Neural Information Processing Systems (NeurIPS)*, pages 4795–4804, 2018.

[16] N. Carlini, G. Katz, C. Barrett, and D. Dill. Provably Minimally-Distorted Adversarial Examples, 2017. Technical Report. https://arxiv.org/abs/1709.10207.

[17] M. Casadio, E. Komendantskaya, M. Daggitt, W. Kokke, G. Katz, G. Amir, and I. Refaeli. Neural Network Robustness as a Verification Property: A Principled Case Study. In *Proc. 34th Int. Conf. on Computer Aided Verification (CAV)*, 2022.

[18] M. Cisse, P. Bojanowski, E. Grave, Y. Dauphin, and N. Usunier. Parseval Networks: Improving Robustness to Adversarial Examples. In *Proc. 34th Int. Conf. on Machine Learning (ICML)*, pages 854–863, 2017.

[19] J. Cohen, E. Rosenfeld, and Z. Kolter. Certified Adversarial Robustness via Randomized Smoothing. In *Proc. 36th Int. Conf. on Machine Learning (ICML)*, pages 1310–1320, 2019.

[20] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. Natural Language Processing (Almost) from Scratch. *Journal of Machine Learning Research (JMLR)*, 12:2493–2537, 2011.

[21] D. Corsi, R. Yerushalmi, G. Amir, A. Farinelli, D. Harel, and G. Katz. Constrained Reinforcement Learning for Robotics via Scenario-Based Programming, 2022. Technical Report. https://arxiv.org/abs/2206.09603.

[22] G. Dong, J. Sun, J. Wang, X. Wang, and T. Dai. Towards Repairing Neural Networks Correctly, 2020. Technical Report. http://arxiv.org/abs/2012.01872.

[23] Y. Dong, F. Liao, T. Pang, H. Su, J. Zhu, X. Hu, and J. Li. Boosting Adversarial Attacks with Momentum. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 9185–9193, 2018.

[24] S. Dutta, X. Chen, and S. Sankaranarayanan. Reachability Analysis for Neural Feedback Systems using Regressive Polynomial Rule Inference. In *Proc. 22nd ACM Int. Conf. on Hybrid Systems: Computation and Control (HSCC)*, 2019.

[25] R. Ehlers. Formal Verification of Piece-Wise Linear Feed-Forward Neural Networks. In *Proc. 15th Int. Symp. on Automated Technology for Verification and Analysis (ATVA)*, pages 269–286, 2017.

[26] H. Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller. Adversarial Attacks on Deep Neural Networks for Time Series Classification. In *Proc. Int. Joint Conf. on Neural Networks (IJCNN)*, pages 1–8, 2019.

[27] S. Fort, H. Hu, and B. Lakshminarayanan. Deep Ensembles: A Loss Landscape Perspective, 2019. Technical Report. http://arxiv.org/abs/1912.02757.

[28] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, and V. Lempitsky. Domain-Adversarial Training of Neural Networks. *Journal of Machine Learning Research (JMLR)*, 17(1):2096–2030, 2016.

[29] T. Gehr, M. Mirman, D. Drachsler-Cohen, E. Tsankov, S. Chaudhuri, and M. Vechev. AI2: Safety and Robustness Certification of Neural Networks with Abstract Interpretation. In *Proc. 39th IEEE Symposium on Security and Privacy (S&P)*, 2018.

[30] S. Gokulanathan, A. Feldsher, A. Malca, C. Barrett, and G. Katz. Simplifying Neural Networks using Formal Verification. In *Proc. 12th NASA Formal Methods Symposium (NFM)*, pages 85–93, 2020.

[31] B. Goldberger, Y. Adi, J. Keshet, and G. Katz. Minimal Modifications of Deep Neural Networks using Verification. In *Proc. 23rd Int. Conf. on Logic for Programming, Artificial Intelligence and Reasoning (LPAR)*, pages 260–278, 2020.

[32] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.

[33] I. Goodfellow, J. Shlens, and C. Szegedy. Explaining and Harnessing Adversarial Examples, 2014. Technical Report. http://arxiv.org/abs/1412.6572.

[34] D. Gopinath, G. Katz, C. Păsăreanu, and C. Barrett. DeepSafe: A Data-driven Approach for Checking Adversarial Robustness in Neural Networks. In *Proc. 16th. Int. Symp. on on Automated Technology for Verification and Analysis (ATVA)*, pages 3–19, 2018.

[35] B. Han, Q. Yao, X. Yu, G. Niu, M. Xu, W. Hu, I. Tsang, and M. Sugiyama. Co-teaching: Robust Training of Deep Neural Networks with Extremely Noisy Labels, 2018. Technical Report. http://arxiv.org/abs/1804.06872.

[36] L. Hansen and P. Salamon. Neural Network Ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(10):993–1001, 1990.

[37] G. Hinton, L. Deng, D. Yu, G. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. Sainath, and B. Kingsbury. Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

[38] S. Huang, N. Papernot, I. Goodfellow, Y. Duan, and P. Abbeel. Adversarial Attacks on Neural Network Policies, 2017. Technical Report. https://arxiv.org/abs/1702.02284.

[39] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu. Safety Verification of Deep Neural Networks. In *Proc. 29th Int. Conf. on Computer Aided Verification (CAV)*, pages 3–29, 2017.

[40] O. Isac, C. Barrett, M. Zhang, and G. Katz. Neural Network Verification with Proof Production. In *Proc. 22nd Int. Conf. on Formal Methods in Computer-Aided Design (FMCAD)*, 2022.

[41] Y. Jacoby, C. Barrett, and G. Katz. Verifying Recurrent Neural Networks using Invariant Inference. In *Proc. 18th Int. Symposium on Automated Technology for Verification and Analysis (ATVA)*, pages 57–74, 2020.

[42] S. Jain, G. Liu, J. Mueller, and D. Gifford. Maximizing Overall Diversity for Improved Uncertainty Estimates in Deep Ensembles. In *Proc. 34th AAAI Conf. on Artificial Intelligence (AAAI)*, pages 4264–4271, 2020.

[43] K. Julian, J. Lopez, J. Brush, M. Owen, and M. Kochenderfer. Policy Compression for Aircraft Collision Avoidance Systems. In *Proc. 35th Digital Avionics Systems Conf. (DASC)*, pages 1–10, 2016.

[44] G. Katz, C. Barrett, D. Dill, K. Julian, and M. Kochenderfer. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. In *Proc. 29th Int. Conf. on Computer Aided Verification (CAV)*, pages 97–117, 2017.

[45] G. Katz, C. Barrett, D. Dill, K. Julian, and M. Kochenderfer. Reluplex: a Calculus for Reasoning about Deep Neural Networks. *Formal Methods in System Design (FMSD)*, 2021.

[46] G. Katz, D. Huang, D. Ibeling, K. Julian, C. Lazarus, R. Lim, P. Shah, S. Thakoor, H. Wu, A. Zeljić, D. Dill, M. Kochenderfer, and C. Barrett. The Marabou Framework for Verification and Analysis of Deep Neural

Networks. In *Proc. 31st Int. Conf. on Computer Aided Verification (CAV)*, pages 443–452, 2019.

[47] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.

[48] A. Kurakin, I. Goodfellow, and S. Bengio. Adversarial Examples in the Physical World, 2016. Technical Report. http://arxiv.org/abs/1607.02533.

[49] O. Lahav and G. Katz. Pruning and Slicing Neural Networks using Formal Verification. In *Proc. 21st Int. Conf. on Formal Methods in Computer-Aided Design (FMCAD)*, pages 183–192, 2021.

[50] B. Lakshminarayanan, A. Pritzel, and C. Blundell. Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles, 2016. Technical Report. https://arxiv.org/abs/1612.01474.

[51] Y. LeCun. The MNIST Database of Handwritten Digits, 1998. http://yann.lecun.com/exdb/mnist/.

[52] S. Lee, S. Purushwalkam, M. Cogswell, D. Crandall, and D. Batra. Why M Heads are Better than One: Training a Diverse Ensemble of Deep Networks, 2015. Technical Report. https://arxiv.org/abs/1511.06314.

[53] S. Lee, S. Purushwalkam Shiva Prakash, M. Cogswell, V. Ranjan, D. Crandall, and D. Batra. Stochastic Multiple Choice Learning for Training Diverse Deep Ensembles. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

[54] H. Liu, M. Long, J. Wang, and M. Jordan. Transferable Adversarial Training: A General Approach to Adapting Deep Classifiers. In *Proc. 36th Int. Conf. on Machine Learning (ICML)*, pages 4013–4022, 2019.

[55] A. Lomuscio and L. Maganti. An Approach to Reachability Analysis for Feed-Forward ReLU Neural Networks, 2017. Technical Report. http://arxiv.org/abs/1706.07351.

[56] A. Lukina, C. Schilling, and T. Henzinger. Into the Unknown: Active Monitoring of Neural Networks. In *Proc. 21st Int. Conf. on Runtime Verification (RV)*, pages 42–61, 2021.

[57] Z. Lyu, N. Gutierrez, A. Rajguru, and W. Beksi. Probabilistic Object Detection via Deep Ensembles. In *Proc. European Conf. on Computer Vision (ECCV)*, pages 67–75, 2020.

[58] Z. Lyu, C. Ko, Z. Kong, N. Wong, D. Lin, and L. Daniel. Fastened Crown: Tightened Neural Network Robustness Certificates. In *Proc. 34th AAAI Conf. on Artificial Intelligence (AAAI)*, pages 5037–5044, 2020.

[59] J. Ma, S. Ding, and Q. Mei. Towards More Practical Adversarial Attacks on Graph Neural Networks. In *Proc. 34th Conf. on Neural Information Processing Systems (NeurIPS)*, 2020.

[60] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. Towards Deep Learning Models Resistant to Adversarial Attacks, 2017. Technical Report. http://arxiv.org/abs/1706.06083.

[61] M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard. DeepFool: A Simple and Accurate Method to Fool Deep Neural Networks. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[62] M. Moshkovitz, Y. Yang, and K. Chaudhuri. Connecting Interpretability and Robustness in Decision Trees through Separation. In *Proc. 38th Int. Conf. on Machine Learning (ICML)*, pages 7839–7849, 2021.

[63] G. Nam, J. Yoon, Y. Lee, and J. Lee. Diversity Matters When Learning From Ensembles. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[64] N. Narodytska, S. Kasiviswanathan, L. Ryzhyk, M. Sagiv, and T. Walsh. Verifying Properties of Binarized Deep Neural Networks, 2017. Technical Report. http://arxiv.org/abs/1709.06662.

[65] M. Ostrovsky, C. Barrett, and G. Katz. An Abstraction-Refinement Approach to Verifying Convolutional Neural Networks. In *Proc. 20th. Int. Symposium on Automated Technology for Verification and Analysis (ATVA)*, 2022.

[66] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. Celik, and A. Swami. Practical Black-Box Attacks against Machine Learning. In *Proc. ACM on Asia Conf. on Computer and Communications Security (CCS*, pages 506–519, 2017.

[67] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. Celik, and A. Swami. The Limitations of Deep Learning in Adversarial Settings. In *IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 372–387, 2016.

[68] P. Prabhakar and Z. Afzal. Abstraction Based Output Range Analysis for Neural Networks, 2020. Technical Report. https://arxiv.org/abs/2007.09527.

[69] C. Qin, J. Martens, S. Gowal, D. Krishnan, K. Dvijotham, A. Fawzi, S. De, R. Stanforth, and P. Kohli. Adversarial Robustness through Local Linearization, 2019. Technical Report. http://arxiv.org/abs/1907.02610.

[70] I. Refaeli and G. Katz. Minimal Multi-Layer Modifications of Deep Neural Networks. In *Proc. 5th Workshop on Formal Methods for ML-Enabled Autonomous Systems (FoMLAS)*, 2022.

[71] W. Ruan, X. Huang, and M. Kwiatkowska. Reachability Analysis of Deep Neural Networks with Provable Guarantees. In *Proc. 27th Int. Joint Conf. on Artificial Intelligence (IJCAI)*, 2018.

[72] A. Shafahi, M. Najibi, A. Ghiasi, Z. Xu, J. Dickerson, C. Studer, L. Davis, G. Taylor, and T. Goldstein. Adversarial Training for Free!, 2019. Technical Report. http://arxiv.org/abs/1904.12843.

[73] A. Shafahi, P. Saadatpanah, C. Zhu, A. Ghiasi, C. Studer, D. Jacobs, and T. Goldstein. Adversarially Robust Transfer Learning, 2019. Technical Report. http://arxiv.org/abs/1905.08232.

[74] C. Shui, A. Mozafari, J. Marek, I. Hedhli, and C. Gagné. Diversity Regularization in Deep Ensembles, 2018. Technical Report. http://arxiv.org/abs/1802.07881.

[75] D. Silver, A. Huang, C. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, and S. Dieleman. Mastering the Game of Go with Deep Neural Networks and Tree Search. *Nature*, 529(7587):484–489, 2016.

[76] K. Simonyan and A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition, 2014. Technical Report. http://arxiv.org/abs/1409.1556.

[77] G. Singh, T. Gehr, M. Puschel, and M. Vechev. An Abstract Domain for Certifying Neural Networks. In *Proc. 46th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL)*, 2019.

[78] M. Sotoudeh and A. Thakur. Correcting Deep Neural Networks with Small, Generalizing Patches. In *Workshop on Safety and Robustness in Decision Making*, 2019.

[79] X. Sun, K. H., and Y. Shoukry. Formal Verification of Neural Network Controlled Autonomous Systems. In *Proc. 22nd ACM Int. Conf. on Hybrid Systems: Computation and Control (HSCC)*, 2019.

[80] M. Svensén and C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer Berlin/Heidelberg, Germany, 2007.

[81] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing Properties of Neural Networks, 2013. Technical Report. http://arxiv.org/abs/1312.6199.

[82] S. Tao. Deep Neural Network Ensembles. In *Int. Conf. on Machine Learning, Optimization, and Data Science*, pages 1–12, 2019.

[83] F. Tramer, A. Kurakin, N. Papernot, I. Goodfellow, D. Boneh, and P. McDaniel. Ensemble Adversarial Training: Attacks and Defenses, 2017. Technical Report. http://arxiv.org/abs/1705.07204.

[84] D. Tsipras, S. Santurkar, L. Engstrom, A. Turner, and A. Madry. Robustness May be at Odds with Accuracy, 2018. Technical Report. http://arxiv.org/abs/1805.12152.

[85] C. Urban, M. Christakis, V. Wüstholz, and F. Zhang. Perfectly Parallel Fairness Certification of Neural Networks. In *Proc. ACM Int. Conf. on Object Oriented Programming Systems Languages and Applications (OOPSLA)*, pages 1–30, 2020.

[86] M. Usman, D. Gopinath, Y. Sun, Y. Noller, and C. Păsăreanu. NNrepair: Constraint-based Repair of Neural Network Classifiers, 2021. Technical Report. http://arxiv.org/abs/2103.12535.

[87] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana. Formal Security Analysis of Neural Networks using Symbolic Intervals, 2018. Technical Report. http://arxiv.org/abs/1804.10829.

[88] Y. Wang, S. Jha, and K. Chaudhuri. Analyzing the Robustness of Nearest Neighbors to Adversarial Examples. In *Proc. 35th Int. Conf. on Machine Learning (ICML)*, pages 5120–5129, 2018.

[89] E. Wong, L. Rice, and Z. Kolter. Fast is Better than Free: Revisiting Adversarial Training, 2020. Technical Report. http://arxiv.org/abs/2001.03994.

[90] H. Wu, A. Ozdemir, A. Zeljić, A. Irfan, K. Julian, D. Gopinath, S. Fouladi, G. Katz, C. Păsăreanu, and C. Barrett. Parallelization Techniques for Verifying Neural Networks. In *Proc. 20th Int. Conf. on Formal Methods in Computer-Aided Design (FMCAD)*, pages 128–137, 2020.

[91] H. Wu, A. Zeljić, G. Katz, and C. Barrett. Efficient Neural Network Analysis with Sum-of-Infeasibilities. In *Proc. 27th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 143–163, 2022.

[92] W. Xiang, H. Tran, and T. Johnson. Output Reachable Set Estimation and Verification for Multi-Layer Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems (TNNLS)*, 2018.

[93] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-MNist: a Novel Image Dataset for Benchmarking Machine Learning Algorithms, 2017. Technical Report. http://arxiv.org/abs/1708.07747.

[94] H. Xuan, R. Souvenir, and R. Pless. Deep Randomized Ensembles for Metric Learning. In *Proc. European Conf. on Computer Vision (ECCV)*, 2018.

[95] X. Yang, T. Yamaguchi, H.-D. Tran, B. Hoxha, T. Johnson, and D. Prokhorov. Neural Network Repair with Reachability Analysis, 2021. Technical Report. https://arxiv.org/abs/2108.04214.

[96] X. Yu, B. Han, J. Yao, G. Niu, I. Tsang, and M. Sugiyama. How does Disagreement Help Generalization against Label Corruption? In *Proc. 36th Int. Conf. on Machine Learning (ICML)*, pages 7164–7173, 2019.

[97] T. Zelazny, H. Wu, C. Barrett, and G. Katz. On Reducing Over-Approximation Errors for Neural Network Verification. In *Proc. 22nd Int. Conf. on Formal Methods in Computer-Aided Design (FMCAD)*, 2022.

[98] H. Zhang, M. Shinn, A. Gupta, A. Gurfinkel, N. Le, and N. Narodytska. Verification of Recurrent Neural Networks for Cognitive Tasks via Reachability Analysis. In *Proc. 24th Conf. of European Conference on Artificial Intelligence (ECAI)*, 2020.

[99] H. Zhang, Y. Yu, J. Jiao, E. Xing, L. El Ghaoui, and M. Jordan. Theoretically Principled Trade-off between Robustness and Accuracy. In *Proc. 36th Int. Conf. on Machine Learning (ICML)*, pages 7472–7482, 2019.

[100] Z. Zhou, J. Wu, and W. Tang. Ensembling Neural Networks: Many Could Be Better Than All. *Artificial Intelligence*, 137(1-2):239–263, 2002.

[101] D. Zügner, A. Akbarnejad, and S. Günnemann. Adversarial Attacks on Neural Networks for Graph Data. In *Proc. 24th ACM SIGKDD Int. Conf. on Knowledge Discovery & Data Mining (KDD)*, pages 2847–2856, 2018.

Appendix A
Verifying an Ensemble

Many modern DNN verification tools receive verification queries as $\langle P, N, Q\rangle$ triples, where P is a precondition, N is the network to be verified, and Q is a postcondition. In order to verify a property of an ensemble $\mathcal{E} = \{N_1, \ldots, N_k\}$, we must first transform the k networks into a single composite network $N_{\mathcal{E}}$, which can then be passed to the verifier. This construction is performed as follows:

- By definition, all ensemble members have the same input space, and so their input layers all have the same dimensions. The composite network $N_{\mathcal{E}}$ will also have an input layer of the same dimension.
- Each network N_i is then placed within $N_{\mathcal{E}}$, with the composite network's input layer serving as N_i's input layer. The N_i networks do not affect each other's computation. In particular, the output layer of each N_i network becomes an internal, hidden layer of $N_{\mathcal{E}}$.
- The output layer of $N_{\mathcal{E}}$ is constructed to reflect the ensemble's aggregation mechanism. For simplicity, we focus here on the case where $N_{\mathcal{E}}$ outputs the average of its constituent networks. In this case, if the networks' output domain is of dimension t, network $N_{\mathcal{E}}$'s output layer is a t-dimensional weighted sum layer; and its j'th neuron, n_j, is computed as the weighted sum:

$$n_j = \sum_{i=1}^{k} \frac{1}{k} \cdot n_j^i,$$

where n_j^i is the j'th output neuron of network N_i, currently encoded within $N_{\mathcal{E}}$.

An illustration of this process appears in Fig. 6. We note that similar variants of this construction have been applied in other contexts of DNN verification [49], [64].

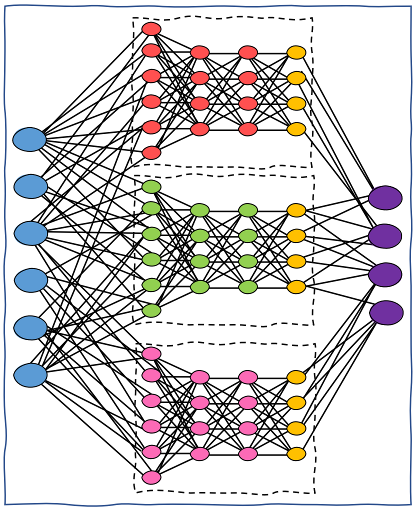

Fig. 6: An ensemble comprised of three different DNNs, each depicted in a different color. The input to the ensemble is passed, as is, to each of the individual DNN members, and the output of the ensemble is the average of the outputs of the individual members.

Once $N_{\mathcal{E}}$ is constructed, it can be verified for different properties, e.g., adversarial robustness around a given input point x_0, using a standard encoding of that verification query.

Appendix B
Checking for Mutual Errors of Ensemble Members

Let N_1 and N_2 be two ensemble members, and suppose we wish to check whether these networks have a mutual error within a given ϵ-ball B around point x_0, whose ground-truth label is l. We can achieve this as follows:

- We begin by constructing a composite network N_c that effectively evaluates N_1 and N_2, side-by-side. This is similar to the process described in Section A, but with two differences. First, this time we only compose two networks, and so the blowup in size is not as significant. Second, we do not construct an output layer that aggregates the outputs of N_1 and N_2; instead, the outputs of both N_1 and N_2 are concatenated into a single output layer of N_c (which is consequently twice as large as the output layers of N_1 and N_2).
- We use the precondition P to restrict the inputs of N_c to the ϵ-ball B. Specifically, let $x_0 = \langle x_0^1, \ldots, x_0^n \rangle$; then:

$$P = \bigwedge_{i=0}^{n} |x^i - x_0^i| \leq \epsilon,$$

  where $x = \langle x^1, \ldots, x^n \rangle$ is the input vector for N_c.
- We use the postcondition Q to ensure that both networks N_1 and N_2 misclassify input x; that is, neither selects label l as its output. This is achieved by requiring that, among the outputs of N_1, the neuron that represents l is not assigned the maximal value; and likewise for N_2. More specifically, let $y_1^1, \ldots, y_r^1$ denote the outputs of N_1, and let $y_1^2, \ldots, y_r^2$ denote the outputs of N_2, so that the outputs of N_c are $y_1^1, \ldots, y_r^1, y_1^2, \ldots, y_r^2$. The postcondition Q in this case is

$$\bigvee_{i \neq l} (y_i^1 \geq y_l^1) \wedge \bigvee_{i \neq l} (y_i^2 \geq y_l^2)$$

  Here, y_l^1 and y_2^l represent the correct labels, and the postcondition requires that at least one other label be assigned a greater score, both in N_1 and in N_2.

. It is straightforward to show that the query $\langle P, N_c, Q \rangle$ is `SAT` if N_1 and N_2 have a mutual error in B, and is `UNSAT` otherwise. This query can be dispatched using any number of existing DNN verification tools.

An illustration of this process appears in Fig. 3. In this example, the precondition restricts the inputs to an ϵ-ball around a correctly classified digit "9"; the input is processed by the two networks independently; and the postcondition requires both networks to misclassify the input.

We note that in our experiments, we used the common practice of simplifying the query's postcondition, by only considering the disjunct $y_i \geq y_l$ for the label i that achieved the second-highest score on x_0 (the "runner up"). This simplification is solely to expedite the experiments; and the full postcondition could be encoded, as well.

Appendix C
Additional Evaluation Details

A. MNIST

Our first set of experiments focused on the MNIST digit recognition dataset. Examples of inputs and perturbed inputs from this dataset appear in Figs. 7 and 8.

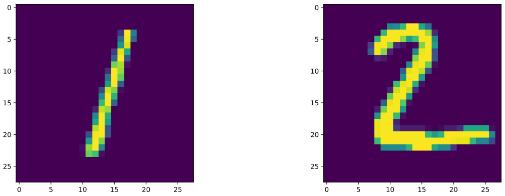

Fig. 7: Examples of two images from the MNIST dataset. The left image is labeled “1”, while the right image is labeled “2”. All images are 28×28 grayscale images.

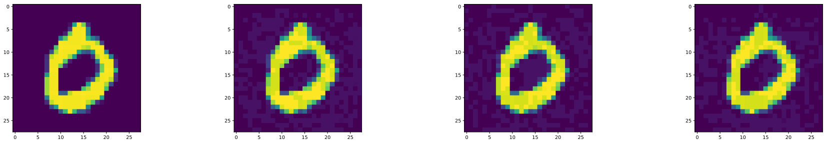

Fig. 8: Adversarial perturbations returned by the SAT verification queries in our experiments. From left to right are: a non-perturbed “0” image from the test set; and three $\epsilon = 0.05$-perturbations of the original image, causing misclassification by members N_1 and N_2 (second from the left), by members N_1 and N_3 (third from the left), and by the whole ensemble $\mathcal{E}_1 = \{N_1, N_2, N_3, N_4, N_5\}$ (on the right).

1) Improving the Robust Accuracy of $\mathcal{E}_1$: In order to improve the robust accuracy of $\mathcal{E}_1$, we computed the mutual error scores for each pair of ensemble members. This computation was performed by dispatching 1200 verification queries for every pair, using 6 different ϵ values. The results appear in Table IV, grouped by network; e.g., the N_1 column shows the aggregated results of all the pairwise queries where N_1 appeared. Recall that a higher number of SAT results (or, equivalently, a lower number of UNSAT results) indicates that a network is more prone to simultaneous errors with its counterparts. These results were then used to compute the uniqueness score for each network, as presented in Table V. Specifically, for the 200 inputs points, per pair, per ϵ value, for each member $N_t \in \mathcal{E}$, we calculated the uniqueness score as: $1 - (\#\texttt{SAT}) \cdot \frac{1}{200\cdot|\mathcal{E}\setminus N_t|}$.[9]

As the uniqueness scores show, network N_2 obtains the lowest score (see Table V) for each value of ϵ, save for $\epsilon = 0.01$, where the margins are very small — presumably because the small perturbation size prevents any of the networks from erring, almost at all. This clearly indicates that N_2 is a prime candidate for replacement, although its original accuracy rate is actually the highest among all networks comprising ensemble $\mathcal{E}_1$ (see Table I). Networks N_5 and N_3 are not far behind, often obtaining the second-lowest scores for the various epsilon values; whereas networks N_1 and N_4 are clearly the stronger of the lot. In our experiments we thus chose to replace N_2 and N_5. After choosing the members to replace, we set out to search for the best replacement candidate from $\mathcal{E}_2$, relative to the remaining ensemble members. As an example, we supply the analysis at Table VI, indicating N_9 is one of the two leading candidates to be added to $\mathcal{E}_1 \setminus \{N_2\}$ and replace N_2, in order to improve the overall robust accuracy (see the left plot in Fig. 4). A similar analysis to the one presented in Table VI can be extracted (based on our experiments summarized in the supplied artifact [6]), and demonstrates that N_9 is also one of the leading candidates to replace N_5 in $\mathcal{E}_1 \setminus \{N_5\}$.

2) Worsening the Robust Accuracy of $\mathcal{E}_2$: Next, we conducted a similar analysis of the networks comprising $\mathcal{E}_2$; the results appear in Tables VII and VIII. This time, we set out to identify the strongest members of the ensemble. As the maximal entries (in bold) of Table VIII indicate, networks N_9 and N_{10} obtain higher uniqueness scores than their counterparts. We selected N_4 as the replacement for N_{10}, as our analysis (presented in Table IX) indicated that this member has a lower uniqueness score, relative to $\mathcal{E}_2 \setminus \{N_{10}\}$. We note that any of the $\mathcal{E}_1$ members can worsen the robust accuracy, as our metrics indicate all members of $\mathcal{E}_1$ achieve a lower uniqueness score than N_{10}, when inserted into $\mathcal{E}_2 \setminus \{N_{10}\}$. A similar analysis to the one presented in Table IX can be extracted (based on our experiments summarized in the supplied artifact [6]), and demonstrates that N_4 is also one of the leading candidates to replace N_9 in $\mathcal{E}_2 \setminus \{N_9\}$. As expected, the total robust accuracy worsened when conducting the switch (see the right plot in Fig. 4).

[9] In out experiments: $|\mathcal{E} \setminus N_t| = 4$, and the amount of all SAT queries for each member N_t is stated, per ϵ, in Table IV.

TABLE IV: The results of the verification queries used to compute the mutual error scores of $\mathcal{E}_1$'s constituent networks on the MNIST dataset. For each network N_i, the $\texttt{TIMEOUT}$ values are: $800 - (\#\texttt{SAT}) - (\#\texttt{UNSAT})$.

ϵ	N_1		N_2		N_3		N_4		N_5	
	# SAT	UNSAT	# SAT	UNSAT	# SAT	UNSAT	# SAT	UNSAT	# SAT	UNSAT
0.01	6	794	6	794	7	793	6	794	3	797
0.02	74	726	93	707	75	724	63	736	91	709
0.03	270	517	283	503	270	515	223	562	258	523
0.04	474	297	507	266	485	272	449	302	483	287
0.05	621	142	646	127	625	130	601	148	623	139
0.06	694	82	716	60	698	68	692	79	686	71

TABLE V: The uniqueness scores for the constituent networks of $\mathcal{E}_1$ on the MNIST dataset. The minimal scores are in bold.

ϵ	N_1	N_2	N_3	N_4	N_5
0.01	99.25	99.25	**99.13**	99.25	99.63
0.02	90.75	**88.38**	90.63	92.13	88.63
0.03	66.25	**64.63**	66.25	72.13	67.75
0.04	40.75	**36.63**	39.38	43.88	39.63
0.05	22.38	**19.25**	21.88	24.88	22.13
0.06	13.25	**10.5**	12.75	13.5	14.25

We observe that for all four novel ensembles we constructed ($\mathcal{E}_1^{2\to9}$, $\mathcal{E}_1^{5\to9}$, $\mathcal{E}_2^{9\to4}$, $\mathcal{E}_2^{10\to4}$), and presented in Fig. 4, the accuracy rates range between 97.7% and 98.7% — i.e., were higher than the accuracy rates of each of the individual DNNs comprising them. We also observe that there is a slight negative correlation between an ensemble's robust accuracy and its accuracy: by improving the robust accuracy of an ensemble, we risk slightly decreasing its accuracy, and vice versa. This finding is in accordance with previous research [84].

B. *Fashion-MNIST*

As mentioned in Section IV, we repeated the process on additional networks, trained on the Fashion-MNIST dataset (examples of inputs and perturbed inputs from this dataset appear in Figs. 9 and 10). In this experiment, we noticed a low variance among the relative uniqueness scores of the members comprising $\mathcal{E}_3$, compared to a larger variance among the relative uniqueness scores of the members comprising $\mathcal{E}_4$. This led us to focus on $\mathcal{E}_4$, in order to check whether our method allows improving (or worsening) the ensemble's robust accuracy. The high variance among the uniqueness scores of $\mathcal{E}_4$'s members can be seen in Table XIII.

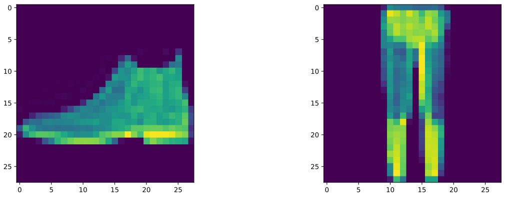

Fig. 9: Examples of two images from the Fashion-MNIST dataset of clothing items. The left image is labeled “Sneaker”, while the right image is labeled “Trouser”. All images are 28×28 grayscale images.

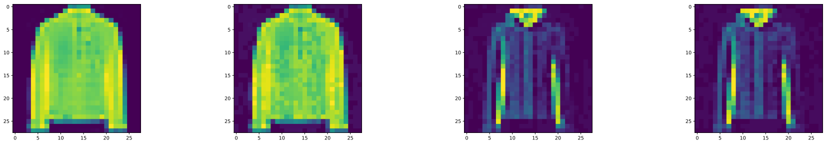

Fig. 10: An image from the Fashion-MNIST dataset labeled as a “Coat” (first on the left), and an $\epsilon = 0.04$-perturbation of that image misclassified by $\mathcal{E}_3$ (second on the left). The two images on the right are two adversarial examples of another “Coat”-labeled image, which cause a joint error for the pairs (N_{13}, N_{14}) and (N_{14}, N_{15}).

The mutual error scores of $\mathcal{E}_3$'s members appear in Table X, and these give rise to the uniqueness scores displayed in Table XI. For $\mathcal{E}_4$, the mutual error scores appear in Table XII, and the uniqueness scores appear in Table XIII. As can be seen, member N_{18} is the most unique member of $\mathcal{E}_4$, and replacing it with N_{13}, a member from $\mathcal{E}_3$ with a low uniqueness

TABLE VI: The uniqueness scores for each replacing candidate from $\mathcal{E}_2$, relative to $\mathcal{E}_1 \setminus \{N_2\}$. The maximal scores are in bold.

$candidate \backslash \epsilon$	0.01	0.02	0.03	0.04	0.05	0.06
N_6	**99.88**	94.38	85.63	63.88	44.5	26.5
N_7	**99.88**	94.88	82.63	65.75	44.5	28.88
N_8	**99.88**	96.38	84.25	63.88	42	25
N_9	**99.88**	**97.88**	**89.13**	72	47.88	31.75
N_{10}	99.63	97.25	87.63	**74.63**	**55.25**	**39.25**

TABLE VII: The results of the verification queries used to compute the mutual error scores of $\mathcal{E}_2$'s constituent networks on the MNIST dataset.

ϵ	N_6		N_7		N_8		N_9		N_{10}	
	# SAT	UNSAT	# SAT	UNSAT	# SAT	UNSAT	# SAT	UNSAT	# SAT	UNSAT
0.01	1	799	1	799	0	800	0	800	0	800
0.02	21	779	26	774	20	780	9	791	18	782
0.03	77	723	87	713	78	721	58	742	64	735
0.04	199	601	202	598	201	599	162	638	166	634
0.05	337	452	345	445	350	443	327	460	295	494
0.06	484	301	489	293	503	288	464	315	430	345

score (see Table XIV), worsens the robust accuracy. In the opposite direction, for various ϵ-sized perturbations, N_{20} is the least unique member, and replacing it with N_{15}, a member from $\mathcal{E}_3$ with a higher uniqueness score (see Table XV), increases the robust accuracy. As in the MNIST experiments, the uniqueness scores of the replacing members are always compared to the remaining members. The robust accuracy changes are presented in Fig. 5.

Our results for Fashion-MNISNT suggest that an ensemble that attains high robust accuracy on different agreement points with *the same* label ("Coat", in our experiments) also achieves high accuracy on points from the test data for which this is the correct label (even if these are not agreement points). We note, however, that our results also indicate that, unlike in the MNIST case, robust accuracy with respect to data points with one label does not imply robust accuracy with respect to points with *a different* label. This is not surprising; the classification of the Fashion-MNISNT dataset is more challenging, and so there is no reason to expect that two members of an ensemble that rarely err together on the region of the input that corresponds to a certain label, also seldom err simultaneously on other regions. Our results suggest that to attain high robust accuracy with respect to the underlying distribution, the choice of the ensemble should be done on a validation set that consists of agreement points whose labels are distributed similarly to the empirically observed distribution in the training data (which is expected to approximate the underlying distribution).

Note. Throughout the appendices, all the uniqueness scores presented are normalized relative to 100.

TABLE VIII: The uniqueness scores for the constituent networks of $\mathcal{E}_2$ on the MNIST dataset. The maximal scores are in bold.

ϵ	N_6	N_7	N_8	N_9	N_{10}
0.01	99.88	99.88	**100**	**100**	**100**
0.02	97.38	96.75	97.5	**98.88**	97.75
0.03	90.38	89.13	90.25	**92.75**	92
0.04	75.13	74.75	74.88	**79.75**	79.25
0.05	57.88	56.88	56.25	59.13	**63.13**
0.06	39.5	38.88	37.13	42	**46.25**

TABLE IX: The uniqueness scores for each replacing candidate from $\mathcal{E}_1$, relative to $\mathcal{E}_2 \setminus \{N_{10}\}$. The minimal scores are in bold.

$candidate \backslash \epsilon$	0.01	0.02	0.03	0.04	0.05	0.06
N_1	100	96.38	85.25	66.25	45	27.38
N_2	**99.75**	95.25	84.75	**63.5**	**41.25**	**26.13**
N_3	**99.75**	96.38	84.75	65.13	43.13	26.75
N_4	99.88	95.75	87.75	69	46.75	29.5
N_5	99.88	**95**	**83.88**	65.13	44	28.5

TABLE X: The results of the verification queries used to compute the mutual error scores of $\mathcal{E}_3$'s constituent networks on the Fashion-MNIST dataset.

ϵ	N_{11}		N_{12}		N_{13}		N_{14}		N_{15}	
	# SAT	UNSAT	# SAT	UNSAT	# SAT	UNSAT	# SAT	UNSAT	# SAT	UNSAT
0.01	43	757	40	760	48	752	43	757	28	772
0.02	235	565	228	572	242	558	233	567	214	586
0.03	493	307	437	363	484	316	469	331	435	365
0.04	651	148	611	188	655	145	629	171	610	190
0.05	759	38	720	75	758	40	745	53	736	58
0.06	795	3	789	8	793	4	785	8	788	7

TABLE XI: The uniqueness scores for the constituent networks of $\mathcal{E}_3$ on the Fashion-MNIST dataset.

ϵ	N_{11}	N_{12}	N_{13}	N_{14}	N_{15}
0.01	94.63	95	94	94.63	96.5
0.02	70.63	71.5	69.75	70.88	73.25
0.03	38.38	45.38	39.5	41.38	45.63
0.04	18.63	23.63	18.13	21.38	23.75
0.05	5.13	10	5.25	6.88	8
0.06	0.63	1.38	0.88	1.88	1.5

TABLE XII: The results of the verification queries used to compute the mutual error scores of $\mathcal{E}_4$'s constituent networks on the Fashion-MNIST dataset.

ϵ	N_{16}		N_{17}		N_{18}		N_{19}		N_{20}	
	# SAT	UNSAT	# SAT	UNSAT	# SAT	UNSAT	# SAT	UNSAT	# SAT	UNSAT
0.01	75	725	48	752	30	770	45	755	72	728
0.02	261	539	221	579	159	641	229	571	266	534
0.03	529	271	494	306	441	359	491	309	529	271
0.04	692	108	672	128	633	167	664	136	685	115
0.05	774	26	753	47	739	59	756	43	770	29
0.06	793	5	787	10	780	19	790	9	792	7

TABLE XIII: The uniqueness scores for the constituent networks of $\mathcal{E}_4$ on the Fashion-MNIST dataset.

ϵ	N_{16}	N_{17}	N_{18}	N_{19}	N_{20}
0.01	90.63	94	96.25	94.38	91
0.02	67.38	72.38	80.13	71.38	66.75
0.03	33.88	38.25	44.88	38.63	33.88
0.04	13.5	16	20.88	17	14.38
0.05	3.25	5.88	7.63	5.5	3.75
0.06	0.88	1.63	2.5	1.25	1

TABLE XIV: The uniqueness scores for each replacing candidate from $\mathcal{E}_3$, relative to $\mathcal{E}_4 \setminus \{N_{18}\}$. The minimal scores are in bold.

$candidate\backslash\epsilon$	0.01	0.02	0.03	0.04	0.05	0.06
N_{11}	**90.75**	66.75	**31.5**	**14.38**	**3.38**	**0.5**
N_{12}	94.25	69.25	44.63	20.5	9.25	2.13
N_{13}	91.25	**66.5**	34.88	15.13	3.75	0.63
N_{14}	92.75	66.88	36.88	18.13	5.75	1.38
N_{15}	96.25	72.88	43.88	21.5	6.75	0.88

TABLE XV: The uniqueness scores for each replacing candidate from $\mathcal{E}_3$, relative to $\mathcal{E}_4 \setminus \{N_{20}\}$. The maximal scores are in bold.

$candidate\backslash\epsilon$	0.01	0.02	0.03	0.04	0.05	0.06
N_{11}	92.63	71.5	35.88	16.38	4.38	1.13
N_{12}	95.25	73.63	**47.5**	21.63	**10**	**3**
N_{13}	93	70.63	38.75	16.5	4.63	1.13
N_{14}	94.5	71.38	41	19.75	6.38	1.63
N_{15}	**96.75**	**76.25**	46.5	**23.25**	7.88	1.63

Appendix D
Implementation of Gradient Attacks

A. Attacking Multiple Networks Simultaneously

To evaluate the mutual error of two neural networks N_1, N_2 over a set A of agreement points, we must determine, for every point $a \in A$, whether there exists an adversarial input $x_0 \in B_{a,\epsilon}$ such that both N_1 and N_2 misclassify x_0.

In order to compare our verification-driven approach to gradient-based methods, we adjusted common gradient-based approaches to suit this task. This was needed because gradient/optimization attacks are usually geared toward finding adversarial inputs for a *single* neural network in question; and so we had to tailor them to search for adversarial inputs that fool *multiple* neural networks simultaneously. We thus present here a novel framework for modifying existing optimization attacks, originally designed to find inputs that fool a single DNN, and extend them to support simultaneous attacks on multiple DNNs.

A naive approach for designing such an attack is to first construct a larger ensemble network from the individual neural networks, and then apply existing techniques. However, because the output of an ensemble is determined by averaging the outputs of its individual networks, the attack might trick the ensemble by fooling a single neural network by a large margin, instead of fooling all of the individual neural networks simultaneously. Consequently, we propose a different approach.

As a first step, our framework computes a network-specific loss for each DNN. This is done by utilizing the loss function of the network-specific attack that is being modified. Next, rather than taking a step in the direction of the original loss function's gradient, as is common in gradient-based attacks on single networks, our method accumulates all the losses into a single "regulating" loss function. This function prioritizes the various single-network losses, and performs a step in a calculated direction. Intuitively, the regulator function should discount (but not ignore) negative losses corresponding to networks that were already fooled, and "focus" on networks that are not yet fooled by the adversarial input being constructed. An overview of this architecture (instantiated for two networks) appears in Fig. 11.

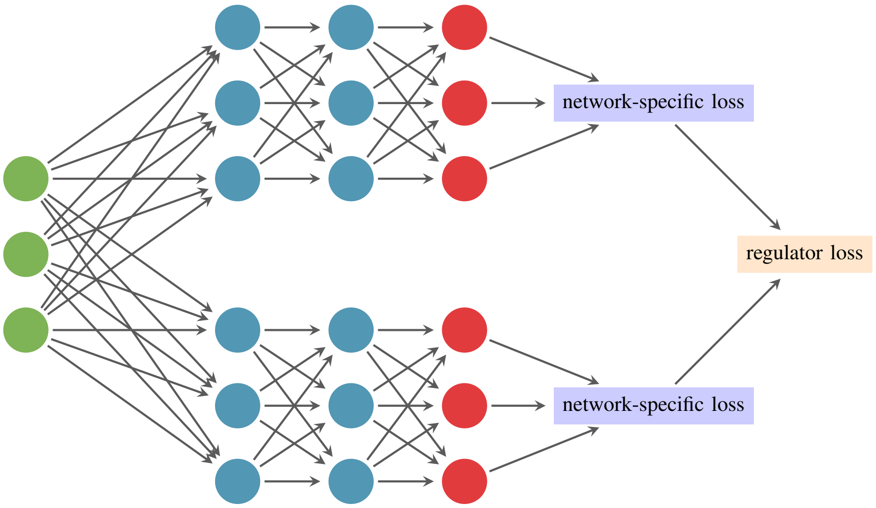

Fig. 11: A scheme of the extended loss function for a gradient attack on multiple networks. In this case, each of the two separate networks has a predefined loss for a network-specific gradient attack. In our framework, both losses are accumulated into a novel regulating loss function which is then optimized in order to find a local optimum, which is hopefully in accordance with an adversarial attack that fools both networks simultaneously.

Given two neural networks (N_1, N_2) and an input x correctly classified by both networks into class t:

$$N_1(x) = N_2(x) = t$$

, we define two search strategies:

- a *targeted* attack aims to cause each network to make a *specific* misclassification. Formally, given two labels: l_1,l_2 such that $l_1 \neq t$ and $l_2 \neq t$, a targeted attack is successful if it can produce a perturbed input x_ϵ such that:

$$N_1(x_\epsilon) = l_1 \wedge N_2(x_\epsilon) = l_2$$

- An *untargeted* attack, on the other hand, aims to cause each network to make *any* misclassification. Formally, an untargeted attack is successful if it can produce a perturbed input x_ϵ such that:

$$N_1(x_\epsilon) \neq t \wedge N_2(x_\epsilon) \neq t$$

We note that both strategies are originally defined for optimizing attacks on a single DNN, and hence we slightly extended their definition to fit the setting in which we simultaneously optimize an attack on multiple DNNs.

In addition, we built our framework upon two popular gradient attacks:

- *FGSM* (Fast-Gradient Sign Method [38]) is a highly scalable and efficient attack in which the perturbation for the constructed adversarial input is calculated based on moving a single step in the direction of the gradient, characterizing a predefined loss on a DNN (with fixed parameters).
- *I-FGSM* (Iterative FGSM [48]) is an iterative algorithm that conducts an FGSM procedure multiple times.

In our case, in order to search for adversarial examples that simultaneously affect two networks around an agreement point, we designed three novel attacks that we refer to as **Gradient Attack (G.A.) 1, 2, and 3**, differing from each other in the method in which they extend FGSM or I-FGSM, and in their adversarial search strategy (targeted or untargeted):

- G.A. 1: is based on a *targeted* FGSM algorithm
- G.A. 2: is based on a *targeted* I-FGSM algorithm
- G.A. 3: is based on an *untargeted* I-FGSM algorithm

B. Tailoring and Optimizing the Loss Functions

The aforementioned attacks optimize the following loss functions we defined:

- For the targeted network-specific loss function we used: $l_s\left(N_i, x\right) \equiv N_i\left(x\right)_t - N_i\left(x\right)_{r_a}$ where t and r_a are the indices of the true label the "runner-up" label (the second largest value of the output vector) of $N_i\left(a\right)$.
- For the untargeted network-specific loss function we used: $l_s\left(N_i, x\right) \equiv N_i\left(x\right)_t - N_i\left(x\right)_{r_x}$ where t is the true label of $N_i\left(a\right)$ and r_x is the runner-up of $N_i\left(x\right)$.
- For the regulator loss function we used the sum of ELU activations on the network-specific loss functions: $l_r\left(N_1, N_2, ..., N_k, x\right) = \sum_{i=1}^{k} \text{ELU}\left(l_s\left(N_i, x\right)\right)$.

We note that our tailored loss functions are specific compositions of the common ELU ("Exponential Linear Unit") activation. Formally, ELU : $\mathbb{R} \rightarrow \mathbb{R}$ is defined as:

$$\text{ELU}(x) = \begin{cases} x & x \geq 0 \\ e^x - 1 & x \leq 0 \end{cases}$$

and is depicted on the left side of Fig. 12. This function is often used as an activation function. However, we note that its properties (such as converging quickly) make it also an effective tool to construct our regulator function.

Specifically, we believe $l_r\left(N_1, N_2, ..., N_k, x\right) = \sum_{i=1}^{k} \text{ELU}\left(l_s\left(N_i, x\right)\right)$ to be a successful regulator function as each $\text{ELU}\left(l_s\left(N_i, x\right)\right)$ term has advantageous properties:

1) as long as its input (the network-specific loss) is positive, i.e., the specific network this loss corresponds with is not yet fooled, it outputs the loss unchanged, keeping its part in the final loss unchanged as well — and hence ensuring that it will be optimized.
2) but, if its input is negative, that is, the network this loss corresponds to is currently fooled, it will discount the final value of the regulated loss function.[10]

These (desirable) characteristics are depicted on the right side of Fig. 12, demonstrating an accumulation of two network-specific losses.

C. Measuring Uniqueness Scores with Gradient Attacks

Below are tables with the main results for the uniqueness scores calculated on the initial ensembles of both datasets (MNIST and Fashion-MNIST). The equivalent (verification-based) uniqueness scores of the initial MNIST ensembles ($\mathcal{E}_1$, $\mathcal{E}_2$) are presented in Tables V and VIII, and the equivalent (verification-based) uniqueness scores of the initial Fashion-MNIST ensembles ($\mathcal{E}_3$, $\mathcal{E}_4$) are presented in Tables XI and XIII of the appendix.

For the complete results, as well as the ensemble members' uniqueness scores after multiple iterations and swaps, see our supplied artifact [6].

TABLE XVI: The uniqueness scores for the constituent networks of $\mathcal{E}_1$ on the MNIST dataset, as calculated by **G.A. 1**. The average uniqueness score is **95.35**.

ϵ	N_1	N_2	N_3	N_4	N_5
0.01	99.88	99.88	100	99.75	100
0.02	99.5	98.75	99.25	99.5	99.25
0.03	97.75	96.25	97.25	98.88	97.63
0.04	95.88	92.5	94.63	96.13	94.63
0.05	92.38	88.13	91.5	94.13	92.63
0.06	88.88	85.13	88.5	92.5	89.5

[10] The discount depends on the negativity of the function. Intuitively, if it is close to zero, it will not discount it by much and if it is very negative it will all but ignore it.

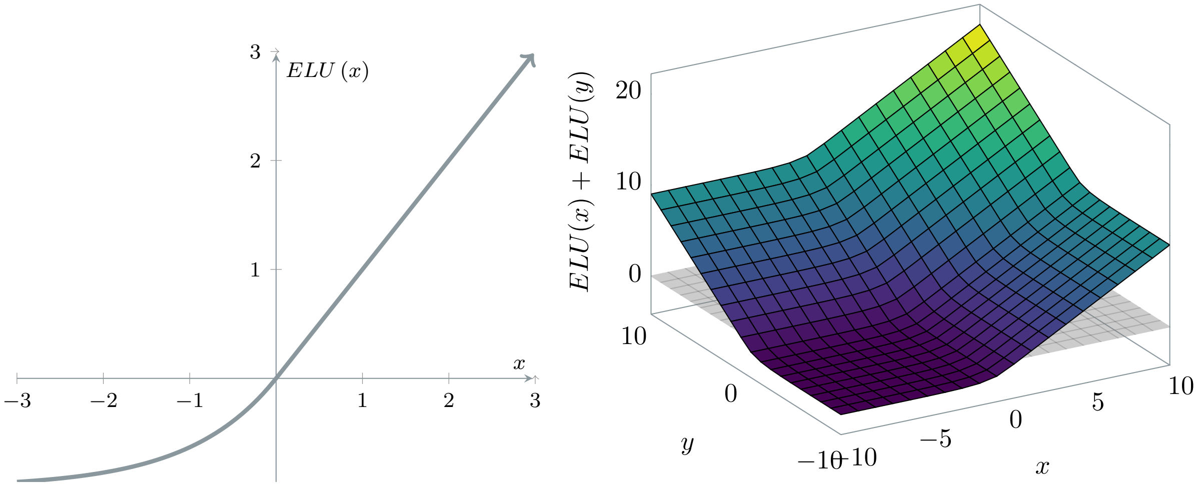

Fig. 12: Left: the ELU activation function. Right: An accumulated sum of two ELU functions. Each axis corresponds to the network-specific loss attributed to one of the two networks simultaneously attacked. Each quadrant ($[+,-]^2$) corresponds with different regulating behavior.

TABLE XVII: The uniqueness scores for the constituent networks of $\mathcal{E}_2$ on the MNIST dataset, as calculated by **G.A. 1**. The average uniqueness score is **97.8**.

ϵ	N_6	N_7	N_8	N_9	N_{10}
0.01	100	100	100	100	100
0.02	99.75	99.88	99.75	99.88	99.75
0.03	99.38	98.75	98.75	99.5	98.88
0.04	98.38	97.5	97.88	98.88	97.88
0.05	96.63	95.75	95.75	97.25	96.63
0.06	93.38	92.5	92.75	94	94.63

TABLE XVIII: The uniqueness scores for the constituent networks of $\mathcal{E}_3$ on the Fashion-MNIST dataset, as calculated by **G.A. 1**. The average uniqueness score is **67.51**.

ϵ	N_{11}	N_{12}	N_{13}	N_{14}	N_{15}
0.01	96.88	97.5	96.75	97	98.13
0.02	87.13	87.5	86.63	85.38	88.63
0.03	71.5	73.13	69.25	71.25	72.88
0.04	58.25	61.38	55.5	59.5	61.13
0.05	47.88	51.88	44.38	47.75	49.88
0.06	41.13	46.38	36.88	41.5	42.38

TABLE XIX: The uniqueness scores for the constituent networks of $\mathcal{E}_4$ on the Fashion-MNIST dataset, as calculated by **G.A. 1**. The average uniqueness score is **68.42**.

ϵ	N_{16}	N_{17}	N_{18}	N_{19}	N_{20}
0.01	95	97.5	98.5	97.63	95.13
0.02	82.75	87.63	90	87.5	83.38
0.03	69.88	74.88	78.13	76	71.13
0.04	56.63	59.63	64.13	65	57.38
0.05	46.5	47.13	52.63	56.38	48.63
0.06	38.88	39.75	43	49.63	42.25

TABLE XX: The uniqueness scores for the constituent networks of $\mathcal{E}_1$ on the MNIST dataset, as calculated by **G.A. 2**. The average uniqueness score is **87.53**.

ϵ	N_1	N_2	N_3	N_4	N_5
0.01	99.88	99.88	99.75	99.5	100
0.02	98.38	97.75	98.38	98.38	98.38
0.03	95	92	93.75	95	92.5
0.04	87	85	87.38	89.13	86
0.05	77.75	76	78.13	81.38	77.75
0.06	67.5	65.13	68.25	72.38	68.5

TABLE XXI: The uniqueness scores for the constituent networks of $\mathcal{E}_2$ on the MNIST dataset, as calculated by **G.A. 2**. The average uniqueness score is **94.5**.

ϵ	N_6	N_7	N_8	N_9	N_{10}
0.01	99.88	99.88	100	100	100
0.02	99.5	99.75	99.38	99.75	99.38
0.03	98.38	97.13	98	99	98
0.04	95.38	93.88	94.5	95.88	95.88
0.05	89.63	88.63	89.63	92.13	91.25
0.06	82.38	82.38	84	85.13	86.38

TABLE XXII: The uniqueness scores for the constituent networks of $\mathcal{E}_3$ on the Fashion-MNIST dataset, as calculated by **G.A. 2**. The average uniqueness score is **60.82**.

ϵ	N_{11}	N_{12}	N_{13}	N_{14}	N_{15}
0.01	96.5	97	96.25	96.38	97.63
0.02	83.63	84.38	82.63	81.75	84.88
0.03	64.25	65.75	61.25	65.38	66.38
0.04	46.25	51.75	45	50.25	50.5
0.05	37.5	42.63	35.25	39.75	39.63
0.06	31.25	37.13	27.88	33.38	32.38

TABLE XXIII: The uniqueness scores for the constituent networks of $\mathcal{E}_4$ on the Fashion-MNIST dataset, as calculated by **G.A. 2**. The average uniqueness score is **59.72**.

ϵ	N_{16}	N_{17}	N_{18}	N_{19}	N_{20}
0.01	93.38	96.63	97.75	97	93.75
0.02	78.5	84.25	87.75	84.88	78.63
0.03	60.75	63.13	70.13	68	60.25
0.04	44.63	46.25	52.5	52.5	45.63
0.05	33.75	34.88	39.63	42.38	34.88
0.06	26.75	27.63	31.25	35.13	29

TABLE XXIV: The uniqueness scores for the constituent networks of $\mathcal{E}_1$ on the MNIST dataset, as calculated by **G.A. 3**. The average uniqueness score is **83.06**.

ϵ	N_1	N_2	N_3	N_4	N_5
0.01	99.63	99.63	99.5	99.5	100
0.02	97.13	95.88	97.13	97.13	96.5
0.03	89.75	88.13	90.25	92	89.87
0.04	77.88	77.75	79.38	83.5	79.75
0.05	67.88	66.25	71	74.75	70.12
0.06	61.88	58.38	62.13	65.88	63.25

TABLE XXV: The uniqueness scores for the constituent networks of $\mathcal{E}_2$ on the MNIST dataset, as calculated by **G.A. 3**. The average uniqueness score is **92.52**.

ϵ	N_6	N_7	N_8	N_9	N_{10}
0.01	99.88	99.88	100	100	100
0.02	99.38	99.38	98.75	99.75	99
0.03	97.5	95.88	96.88	97.88	97.63
0.04	93.13	91	92.25	93.63	93.5
0.05	85.38	84.88	86.38	88.63	87.25
0.06	77.5	78.25	79.38	80.63	82

TABLE XXVI: The uniqueness scores for the constituent networks of $\mathcal{E}_3$ on the Fashion-MNIST dataset, as calculated by **G.A. 3**. The average uniqueness score is **58.66**.

ϵ	N_{11}	N_{12}	N_{13}	N_{14}	N_{15}
0.01	95.5	96.25	95.25	95.75	97
0.02	81.75	82.25	81.63	79.75	82.88
0.03	59.75	61	59	61.88	63.13
0.04	42.37	48	42.38	46.5	47.25
0.05	35.62	39.88	33.38	37.75	37.88
0.06	30.13	35.13	27.88	31.88	31

TABLE XXVII: The uniqueness scores for the constituent networks of $\mathcal{E}_4$ on the Fashion-MNIST dataset, as calculated by **G.A. 3**. The average uniqueness score is **58.67**.

ϵ	N_{16}	N_{17}	N_{18}	N_{19}	N_{20}
0.01	92.75	96.38	97.63	96.63	93.38
0.02	76.5	83.13	86	82.25	76.38
0.03	58	60.25	66.25	65	57
0.04	41.63	44.5	51.13	50.75	43.5
0.05	33.5	35.13	38.13	41.88	35.63
0.06	28.25	29.25	32.38	36.38	30.5