TOWARDS SCALING ROBUSTNESS VERIFICATION OF SEMANTIC FEATURES VIA PROOF VELOCITY

Anonymous authors
Paper under double-blind review

ABSTRACT

Robustness analysis is important for understanding the reliability of neural networks. Despite the significant progress in the verification techniques for both L_p- and semantic features- neighborhoods, existing approaches struggle to scale to deep networks and large datasets. For example, we are unaware of any analyzer that scales to AlexNet trained for ImageNet (consisting of 224x224x3 images). In this work, we take a step towards scaling robustness analysis. We focus on robustness to perturbations of semantic features and introduce the concept of *proof guided by velocity* to scale the analysis. The key idea is to phrase the verification task as a dynamic system and adaptively identify how to split it into subproblems each with maximal proof velocity. We propose a policy to determine the next subproblem based on the past and by leveraging input splitting, input refinement, and bound tightening. We evaluate our approach on CIFAR-10 and ImageNet and show that it can analyze neighborhoods of various features: hue, saturation, lightness, brightness, and PCA.

1 INTRODUCTION

The reliability of deep neural networks (DNNs) has been undermined by adversarial examples: small perturbations to inputs that deceive the network (e.g., Goodfellow et al. (2015)). To understand the robustness of a DNN to adversarial example attacks, most existing works propose to analyze the network's local robustness, i.e., for a given input and a radius ϵ verify that the network is robust at the ϵ-ball around that input (e.g., Müller et al. (2021); Ryou et al. (2021); Xu et al. (2020); Gowal et al. (2019); Singh et al. (2019); Gehr et al. (2018)). Despite the significant progress in their efficiency and precision, verifiers of ϵ-ball neighborhoods struggle to scale to deep networks. That is, at best the maximal verified ϵ-ball neighborhoods consist of indistinguishable images; at worst, no non-trivial ϵ-ball can be verified. Unfortunately, small certified neighborhoods provide little insight on the robustness of the network. To illustrate, consider Figure 1 visualizing an ϵ-ball neighborhood B_ϵ. Such neighborhoods contain all combinations of every possible perturbation within the radius ϵ. This limits the analysis to small neighborhoods confined by the closest decision boundary, and prevents one from understanding other robustness aspects of the network.

In parallel, a new kind of adversarial attacks has been introduced, perturbing semantic features, such as HSV transformations (Hosseini & Poovendran, 2018), PCA transformations (Zhang et al., 2020b; Carlini & Wagner., 2017), and colorization and texture attacks (Bhattad et al., 2020). To understand the robustness of networks to feature attacks, several verifiers have been proposed for analyzing feature neighborhoods (Mohapatra et al., 2020; Balunovic et al., 2019; Singh et al., 2019). Unlike ϵ-ball neighborhoods which consider all possible perturbations, feature neighborhoods impose constraints on the perturbations, restricting them to the feature direction (illustrated by Figure 1). Hence, robust feature neighborhoods are larger than robust ϵ-balls. To verify large feature neighborhoods, several techniques have been proposed. One is encoding of the pixels' dependencies imposed by the features into the analysis to improve precision. Another technique is splitting the input range into smaller parts that can be verified more efficiently. Despite the great effort in improving feature encoding and input partitioning, existing work still struggles to scale to deep networks.

In this work, we present the concept of *proof guided by velocity* to scale robustness verification for feature neighborhoods. The key idea is to phrase the verification problem as a dynamic system driven by a policy that splits the problem into subproblems, each is verified separately, with the

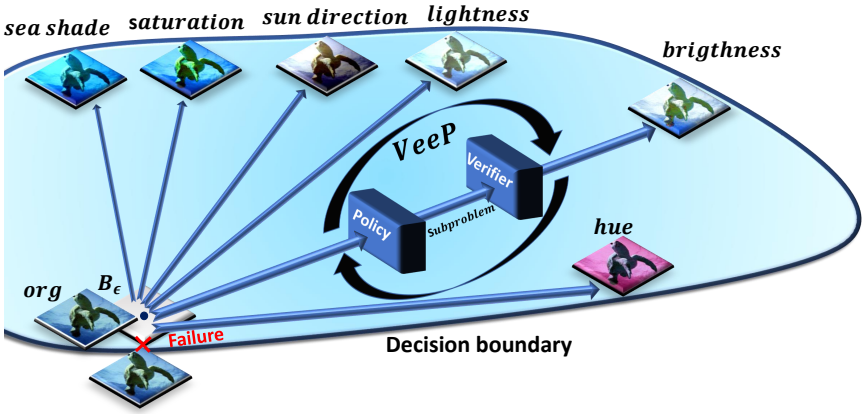

Figure 1: Illustration of VeeP and the difference between an ϵ-ball and feature neighborhoods.

goal of maximizing the verification velocity. The challenge is to compute the optimal split. On the one hand, smaller subproblems are more likely to be verified quickly, and are thus more likely to progress the overall analysis than larger subproblems whose verification is likely to be slow or fail. On the other hand, if we split the problem into too many subproblems, the overall analysis will not terminate in a reasonable time. Splitting into subproblems includes splitting the feature neighborhood into smaller sub-neighborhoods, as well as defining other verification decisions, such as neuron refinement, which affect the proof velocity. Our dynamic system makes progress iteratively. At every step, one subproblem is created and verified (by an existing verifier). Based on the verification results, the policy determines the next subproblem. We design a policy which decides on the next subproblem based on the previous subproblems. For example, if the verifier failed, the size of the next sub-neighborhood is decreased. Our policy also decides which neurons may be refined, based on their predicted effect on the certification result, the refinement gain, and the time overhead.

We implement this concept in a system called VeeP (for **v**elocity **p**rover). In addition to the ideas presented above, VeeP also builds on the advancements of both ϵ-ball and feature robustness verification. In particular, it verifies the subproblems with GPUPoly (Müller et al., 2021), a state-of-the-art verifier for ϵ-ball neighborhoods, and it encodes the feature relations by building on the ideas of Semantify-NN (Mohapatra et al., 2020). Combining these ideas with our concept of proof guided by velocity enables VeeP to be the first verifier that can analyze an AlexNet model for ImageNet. VeeP can reason about various feature neighborhoods: brightness, HSL (hue, saturation, lightness), and PCA features. Figure 1 visualizes the meaning of these features. The sea shade and sun direction features are obtained by PCA, which computes semantic features from the dataset. For the turtle image in Figure 1, the diameter of the maximal ϵ-ball which GPUPoly verifies is 0.0001, while the diameter of the maximal brightness neighborhood which VeeP verifies is 0.18 (the diameter is measured in the brightness space) – 1800x bigger than the maximal ϵ.

We evaluate VeeP on ResNet models for CIFAR-10 and AlexNet models for ImageNet over brightness, HSL and PCA neighborhoods. Results show that the verified diameters that VeeP computes for our feature neighborhoods are 99.1x bigger than the verified diameters that GPUPoly computes for ϵ-ball neighborhoods. We further show that these verified diameters are on average at least 97% of the maximal certifiable diameter (which is bounded by the closest adversarial example in the feature direction). We also compare VeeP to Semantify-NN on MNIST and show that VeeP is 135x faster. Finally, we run VeeP over PCA neighborhoods and show that analyzing robustness of networks to dataset-specific features can reveal general robust and non-robust semantic attributes of the network.

To conclude, our main contributions are:

- A *proof guided by velocity framework* for robustness verification of feature neighborhoods.
- A policy deciding on the subproblems based on the analysis of previous subproblems.
- A system called VeeP, implementing the proof guided by velocity framework and leveraging recent advancements of both ϵ-ball and feature robustness verifiers.
- An evaluation on CIFAR-10 and ImageNet networks over brightness, HSL, and PCA.

2 RELATED WORK

In this section, we review the main works on neural network verification, and in particular robustness verification. We refer the reader to a recent survey for a more detailed summary (Li et al., 2020).

Complete verifiers Several works propose complete robustness verifiers, which always determine whether or not a neighborhood is robust. Most works focus on verifying robustness of L_∞ neighborhoods, where each pixel is bounded by an interval. Some complete verifiers encode the robustness problem as an SMT/SAT instance and solve it using existing or enhanced solvers (e.g., Pulina & Tacchella (2012; 2010); Katz et al. (2017); Ehlers (2017)). Others encode the problem as mixed-integer linear problem (e.g., Tjeng et al. (2019); Xiao et al. (2019)). Other works employ branch-and-bound approaches to split the certification problem into smaller subproblems (e.g., Bunel et al. (2018)). Despite the significant effort to scale complete verifiers, for example, using branching heuristics (Elboher et al., 2020) or abstraction-refinement (Wu et al., 2020), exact robustness verification is NP-hard (Katz et al., 2017) and thus cannot scale to large and deep networks.

Incomplete verifiers Another research line propose incomplete verifiers, which rely on overapproximation to scale the analysis to deeper networks than possible by complete verifiers. However, the scalability comes on the expanse of occasionally failing to prove robustness due to the overapproximation error. Existing works propose different ways to find the optimal balance in the efficiency and precision trade-off. Some verifiers employ linear relaxations and compute, for every neuron, lower and upper linear bounds (e.g., Müller et al. (2021); Ryou et al. (2021); Xu et al. (2020); Gowal et al. (2019)). Others rely on semidefinite programming to define convex optimizations which can be solved in polynomial time (Dathathri et al., 2020; Raghunathan et al., 2018). Other works bound the neurons' values by leveraging Lipschitz global and local constants or curvatures (Singla & Feizi (2020); Fazlyab et al. (2019)). These advancements have enabled reasoning about networks with 100k neurons for MNIST and CIFAR-10. However, due to the overapproximation error, they fail to prove robustness of non-trivial neighborhoods for larger networks. In particular, none can prove robustness of non-trivial neighborhoods for ImageNet networks. Recent works propose probabilistic verifiers providing robustness guarantees with confidence intervals (Zhang et al., 2020a; 2021). Although these verifiers scale to large networks and can reason about ImageNet networks, their guarantees are probabilistic.

Robustness to features perturbations Several verifiers analyze local robustness to semantic feature perturbations. Earlier works have focused on rotations, brightness and contrast, and perform certification by translating the feature neighborhoods to L_∞ neighborhoods and then analyzing using existing verifiers (Singh et al., 2019; Wang et al., 2018a). Because of this translation, the certified feature neighborhoods are generally small, especially if the feature is non-linear, in which case the L_∞ bounds tend to be too loose. More recent works encode the feature constraints into the verifier in order to directly certify in the feature domain (Balunovic et al., 2019; Mohapatra et al., 2020). Balunovic et al. (2019) focus on geometric features and rely on Monte Carlo sampling to overapproximate the feature constraints by convex linear bounds. They further refine these bounds by solving an optimization problem and then verify using a standard verifier. Mohapatra et al. (2020) propose adding an input layer that encodes the feature and is added to the original network. It can then run standard verification over this network. Both works focus on analyzing MNIST, CIFAR-10, and GTSRB. In contrast, we show how to scale feature robustness analysis further in order to prove robustness of deeper networks (e.g., ImageNet networks).

Adversarial semantic features attacks We study robustness of networks to feature perturbations, and in particular brightness, PCA, and HSL features. Several works have shown how to craft semantic feature perturbations to obtain adversarial examples. For example, Hosseini & Poovendran (2018) presented an attack relying on HSV color transformations (which is close to HSL). Other works linked adversarial examples to PCA features (Zhang et al., 2020b; Bhagoji et al., 2018; Carlini & Wagner., 2017). Other kinds of feature attacks include facial feature perturbations (Goswami et al., 2018), colorization and texture attacks (Bhattad et al., 2020), features obtained using scale-invariant feature transform (SIFT) (Wicker et al., 2018), and semantic attribute perturbations using multi-attribute transformation models (Joshi et al., 2019). The works of Ilyas et al. (2019) and Goh (2019) show that adversarial examples are linked to non-robust features.

3 PROOF GUIDED BY VELOCITY: PROBLEM DEFINITION

In this section, we define the problem of *proof guided by velocity* and explain it for the setting of semantic feature verification. In the following, we denote an input by $x \in \mathbb{R}^d$, a neural network classifier by $D : \mathbb{R}^d \to \mathbb{R}^c$, and the classification of x by D by class$(D(x))$.

Neighborhoods, robust neighborhoods, and feature-based neighborhoods Given an input x, a neighborhood $I(x) \subseteq \mathbb{R}^d$ and a classifier D, we say $I(x)$ is robust if $\forall x' \in I(x)$, $class(D(x')) = class(D(x))$. We focus on feature neighborhoods. A feature $f : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$ maps an input x and a diameter δ to the perturbation of x along the feature f by δ. For all features f and inputs x, $f(x, 0) = x$. Given a feature f, an input x, and a diameter δ, the feature neighborhood $I_{f,\delta}(x)$ is the set of all perturbations of x along f with diameter at most δ: $I_{f,\delta}(x) = \{f(x, \delta') \mid 0 \leq \delta' \leq \delta\}$. Feature neighborhoods can be certified with a natural divide-and-conquer approach: given a target diameter δ and a sequence $\delta_0 = 0, \delta_1, \ldots, \delta_t$ such that $\Sigma_i \delta_i = \delta$, we can independently certify each neighborhood $I_{f,\delta_{i+1}}(f(x, \delta_i))$, and if all are robust we can infer that $I_{f,\delta}(x)$ is robust.

Motivation Robustness verifiers struggle to scale to deep networks and large neighborhoods either because the computation cannot terminate in a reasonable time or that the overapproximation error accumulates so that the analysis result is inconclusive. To reduce the overapproximation error, many approaches have been introduced, including splitting the neighborhood into several parts that can be verified separately, refining neurons' bounds, and encoding input relations. The decision of which action to take and with which parameters is key to enable the verifier to complete as fast as possible.

To scale the analysis, we propose to dynamically split the verification problem into subproblems. That is, the analysis executes a series of *verification steps*. Each step invokes a call to the verifier and accordingly picks the next subproblem. The overall computation time is the total execution time of all steps. A key question that arises is: *what is the optimal step series which minimizes the execution time?* To illustrate, assume verifier A can certify a neighborhood with diameter $\delta = 0.1$ in one second and a neighborhood with diameter $\delta = 0.01$ in one millisecond. In this case, if we can partition the larger neighborhood into ten disjoint neighborhoods with diameter $\delta = 0.01$, the analysis completes 10x faster than if let A certify the full neighborhood. Beyond improving execution times, splitting a neighborhood is sometimes the only possibility to successfully certifying a neighborhood (i.e., avoiding inconclusive certification results). To illustrate, assume that we wish to verify a neighborhood with diameter $\delta = 0.1$, and that A can certify any neighborhood in one second. Assume A fails for any neighborhood with diameter $\delta > 0.01$. In this case, if we begin by partitioning the larger neighborhood into ten disjoint neighborhoods with diameter $\delta = 0.01$, the analysis completes with a conclusive result within 10 seconds and avoids the wasted time spent on the failed calls to the analyzer. That is, a good partition into verification steps not only minimizes the execution times of each step but also minimizes the steps with inconclusive results.

We view the verification problem as a dynamic system, where each step is associated with a state s_k. The state consists of the current subproblem's diameter δ_k, the diameter proven so far (given as the sum of the verified diameters $\Sigma_{j<k}\delta_{A,j}$), and other auxiliary data. A verifier A is invoked to solve the verification subproblem and returns the certified diameter $\delta_{A,k}$ (which is δ_k or 0), and possibly other internal verification decisions. Accordingly, a policy function $\mathcal{F}$ picks an action u_k to decide on the next subproblem. The analysis terminates either when the proven diameter is equal to the target diameter or when the policy picks a subproblem whose diameter is below a threshold δ_{min} (indicating that the analysis is close to an adversarial example). Figure 2 illustrates our dynamic system. The challenge is to design a policy minimizing the overall analysis time. We phrase this goal using the concept of *proof velocity*. We next provide the definitions.

Verification step series Given a feature function f, a target diameter δ, an analyzer A, and a set of possible actions $\mathcal{U}$ (e.g., input splitting, refining neurons' bounds), a *verification step series* is a sequence $(s_1, u_1), \ldots, (s_M, u_M)$. Each s_i is a state, consisting of a diameter, the currently certified diameter, and auxiliary data. Each u_i is a an action in $\mathcal{U}$. A pair (s_k, u_k) is called a verification step. The last pair (s_M, u_M) either satisfies that the certified diameter in s_M is equal to the target diameter δ or that u_M proposes a verification step which is below some precision threshold, indicating that practically the verification should terminate and return the (maximal) certified diameter.

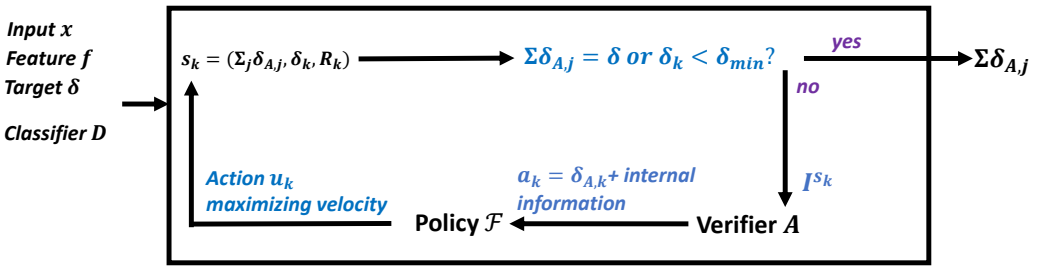

Figure 2: Our dynamic system for analyzing the robustness of D in the neighborhood $I_{f,\delta}(x)$.

Verifier We assume a robustness verifier A that takes as input I^{s_k} consisting of a neighborhood $I_{f,\delta_k}(x)$ as well as guidance on how to execute internal heuristics. The verifier A returns a_k consisting of the diameter $\delta_{A,k}$ it verified (δ_k if the analysis succeeded and 0 otherwise) and additional information on its internal computations which can be useful for the following subproblems. We assume that A is incomplete and that if A fails, it is due to overapproximation error. However, our definitions extend to verifiers that can determine that a neighborhood has an adversarial example.

Policy function A policy function $\mathcal{F}$ maps a state s_k and an analysis result a_k to an action $u_k \in \mathcal{U}$. This action decides on the next subproblem to verify, which is part of the next state s_{k+1}.

Proof velocity Given a robustness verifier A, we denote the time of verifying a neighborhood $I_{f,\delta}(x)$ by $t_A(I_{f,\delta}(x))$. The proof velocity is the ratio of the certified neighborhood diameter δ_A and the analysis time: $v_A(I) = \frac{\delta_A}{t(I_{f,\delta}(x))}$. The velocity is either a positive number, if A certified the neighborhood, or 0 if it failed. A zero velocity means that the analysis has to refine this neighborhood in order to verify it successfully (e.g., by partitioning it) and that we have not gained from this analysis. We note that this definition also accommodates verifiers that can return failure to indicate on the existence of adversarial examples. In this case, the velocity is zero because no diameter was proven robust. When the verifier is clear from the context, we omit the subscript notation.

Problem definition In the problem of proof guided by velocity the goal is to define states, actions, and a policy function $\mathcal{F}$ such that for every input x, feature f, and maximal diameter δ, the policy returns an action u_k which maximizes the proof velocity of the next verification subproblem:

$$\forall k. \mathcal{F}(s_k, a_k) \in \arg\max_{u_k \in \mathcal{U}} \{v(I^{s_{k+1}}(x))\} \tag{1}$$

where $I^{s_{k+1}}$ is the subproblem sent to the verifier at step $k+1$.

4 VEEP

In this section, we present VeeP, our approach for solving proof guided by velocity (Equation (1)). We start by identifying the main factors affecting the proof velocity of an incomplete verifier. Accordingly, we define the states, actions, and policy function.

Velocity factors of incomplete verifiers A key factor to expediting the proof velocity is splitting the verification problem into subproblems for which the verifier does not fail, because any failed call to the verifier is a waste of time (unless it can return an adversarial example). On the other hand, if we pick too small diameters, the velocity would be too small and the overall analysis will not complete in a reasonable time. Thus, the diameters should be adapted based on the verifier, the neighborhood, and the diameter remained to verify. A second factor to expediting the proof velocity is identifying the expensive computations of the verifier, which are typically tuned based on some heuristic. For example, many verifiers employ a quick overapproximation step, followed by a refinement step which is typically a heavier computation. Examples for refinement steps include backsubstitution (Singh et al., 2019) and MILP encoding (Tjeng et al., 2019). Since the refinement

step is often more expensive, it is not executed for every neuron. The question of which neurons should be refined is crucial for balancing precision and execution time.

State and actions To consider both factors, our verification state is a pair (δ_k, R_k) where δ_k is the next diameter to verify and R_k is the set of neurons which are allowed to be refined (that is, a neuron is refined if the analysis requires to refine it and it is in R_k). Our action set consists of action pairs $\mathcal{U} = \{-, 0, +\} \times \mathcal{P}(D)$. An action $(u_1, u_2) \in \mathcal{U}$ consists of u_1 indicating whether the next diameter should decrease $(-)$, remain unchanged (0), or increase $(+)$, while u_2 is a subset of neurons which may be refined during the analysis.

VeeP policy The goal of the policy $\mathcal{F}$ is to return an action which will lead to a proof of maximal velocity among the current possibilities. To predict the best action, our key assumption is that there is a relation between close verification steps. This assumption is often valid when we progress with small verification steps along the target diameter, and it is what enables us to certify deep networks trained for high-dimensional datasets, for which verifiers often fail even for very small diameters because of overapproximation errors. To decide on the next action, our policy relies on the *quality* of the previous neighborhoods. We next define this metric.

We assume a verifier that for a neighborhood I computes for every neuron an interval bounding its values. Formally, we denote by $x_{m,j}$ the j^{th} neuron at layer m and by $[l_{m,j}, u_{m,j}]$ the interval bounding the possible values of $x_{m,j}$ given any $y \in I$. We denote by $m = o$ the output layer. Using these intervals, the verifier determines whether I is robust with respect to label j by checking if the lower bound of the j^{th} output neuron is greater than the upper bounds of the other output neurons: $l_{o,j} > \max\{u_{o,j'} \mid j' \neq j\}$. We use these intervals to define the quality of I as the difference between that lower bound and the maximal upper bound: $Q(I) = l_j - \max\{u_{j'} \mid j' \neq j\}$. Intuitively, the higher the quality the farther I can expand without risking a certification failure.

Our policy function determines whether to decrease, preserve, or increase the next diameter δ_k based on the recent neighborhoods' quality and recent failures. If there were no recent failures and the last neighborhoods' quality is high, then VeeP adapts the diameter based on the recent observed velocity. Usually, the diameter increases in this case. However, if VeeP observes that the verifier has higher velocity for smaller diameters, it does not increase the diameter. If there is a failure or the quality is too low, the diameter decreases. Otherwise, VeeP does not change the diameter.

Determining the set of neurons which can be refined, R_k, is a more subtle task, as it requires to carefully predict the right balance between precision and execution time. In the extreme case where we let R_k to contain all neurons, we permit the verifier to be as precise as it can, but it may take a very long time, thus reducing the velocity significantly. On the other hand, if R_k is the empty set, we let the verifier to be as fast as it can, but it may be very imprecise, and thus fail, thereby reducing the velocity to zero. Our policy relies on three metrics to determine whether to include a node $x_{m,j}$ in R_k: (i) *the neuron's effect* on the neighborhood's quality, (ii) *the refinement gain*, and (iii) the execution time. Based on these, our policy picks for R_k all neurons except those with the lowest effect on the neighborhood's quality, lowest refinement gain, and highest execution time. To compute these metrics, we assume the verifier returns for every analyzed neighborhood the intervals bounding each neuron, before and after refinement (if applicable), and the execution time of each refinement. We next define the effect of a neuron on the neighborhood's quality and refinement gain.

Given a neuron $x_{m,j}$, its *effect* on the neighborhood's quality is the absolute value of the gradient of $Q(I)$ by $x_{m,j}$: $e(x_{m,j}) \triangleq |\frac{\partial Q(I)}{\partial x_n}|$. The higher $e(x_{m,j})$ the higher we expect the neighborhood quality to be if we refine $x_{m,j}$. Computing the gradient of a neuron as part of robustness analysis has been introduced before to improve verifiers' precision for L_∞ neighborhoods (Wang et al., 2018a;b). In our context, we use the gradients to identify how to maximize the proof velocity. A neuron's *refinement gain* is the difference between the interval size before and after refinement. That is, given a neuron $x_{m,j}$ with interval $[l_{m,j}, u_{m,j}]$, which was refined to $[l^r_{m,j}, u^r_{m,j}]$, the refinement gain is $g(x_{m,j}) = [(u_{m,j} - l_{m,j}) - (u^r_{m,j} - l^r_{m,j})]$. The higher $g(x_{m,j})$ the higher potential the refinement step has for $x_{m,j}$.

We note that while our policy focuses only on two factors to maximize the proof velocity (the diameter step δ_k and the refinement set R_k), VeeP can be extended to consider other factors.

Table 1: Models used in our experiments.

Dataset	model	$\text{params}_{\#}$	Defense
CIFAR-10	ResNetTiny	311K	PGD (Madry et al., 2018)
CIFAR-10	ResNet18	558K	PGD
CIFAR-10	ResNet34	967K	DiffAI (Mirman et al., 2018)
ImageNet	AlexNetTiny	444K	PGD
ImageNet	AlexNet	600K	PGD

5 EVALUATION

In this section, we evaluate VeeP. We start with implementation aspects and optimizations. We then present our experiments. We begin with a comparison to Semantify-NN over MNIST models. We continue with evaluating VeeP over brightness and HSL features for CIFAR-10 and ImageNet models. Finally, we evaluate VeeP for PCA features and show that some are more robust than others.

Implementation We implemented VeeP in Python using PyTorch[1]. For the verifier, VeeP relies on GPUPoly (Müller et al., 2021). GPUPoly has a heuristic for deciding on the neurons to refine by backsubstitution, and we extended it to consider R_k (i.e., the set of neurons that the policy allows to refine). As an optimization, we build on the idea of Semantify-NN (Mohapatra et al., 2020) that encodes features as input layers with the goal of encoding pixel relations to reduce overapproximation errors. Semantify-NN encodes features using standard fully-connected and convolutional layers. For some features, this approach is practically infeasible for high-dimensional datasets because of the high memory overhead. To illustrate, assume the input dimension is $h \times w \times 3$. The HSL input layers, as defined in Semantify-NN, map an (R,G,B) triple into a single value in the feature domain, resulting in a perturbed output of $h \times w$. This output is then translated back to the input domain. Namely, a fully-connected layer requires $(h \times w) \times (h \times w \times 3)$ weights. For ImageNet, where $h = w = 224$, this layer becomes too big to fit into a standard memory (over 60GB). Instead, we observe that, for some features, the feature layer's weights are mostly zeros and thus this layer can be implemented using sparse layers (Richter & Wattenhofer, 2018; Ardakani et al., 2017).

Evaluation setup We trained models and ran the experiments on a dual AMD EPYC 7742 server with 1TB RAM and eight NVIDIA A100 GPUs. We evaluated VeeP on CIFAR-10 (Krizhevsky, 2009), with images of size 32x32x3, and ImageNet (Deng et al., 2009), with images of size 224x224x3. We consider ResNet (He et al., 2016) and AlexNet (Krizhevsky et al., 2012) models. Since GPUPoly does not support MaxPool layers at the moment, we replaced the MaxPool layers in AlexNet with convolutional ones. This is justified by Springenberg et al. (2015). The CIFAR-10 models were taken from ERAN's repository[2], and we trained the ImageNet models. We summarize the models used in Table 1. To compare VeeP with Semantify-NN, we used two fully-connected models (2 or 4 hidden layers, 1024 neurons at each) from their repository[3] trained for MNIST (Lecun et al., 1998).

Comparison with Semantify-NN We begin by comparing VeeP with Semantify-NN, a state-of-the-art verifier for analyzing feature neighborhoods. To scale, Semantify-NN splits the neighborhoods into equal-size sub-neighborhoods, each is verified separately. This can be viewed as using a constant policy in our framework. For the comparison, we focus on the two MNIST models described above and the brightness feature. Unlike VeeP, which verifies general robustness, Semantify-NN focuses on targeted attacks. That is, it computes the maximal diameter of a neighborhood which does not contain an input of a target class t. To compare both verifiers on the same task, given an image x, we first look for the closest diameter δ_{adv} reaching an adversarial example in the feature domain (using a grid search) and check its label to provide to Semantify-NN. We then let both verifiers certify the maximal diameter up to δ_{adv}. We run this experiment for 100 inputs. Table 2 reports the maximal diameter δ_f and the execution time in seconds t_f. The results indicate that both

[1] We are working towards making our implementation, models, and tests available for reproducibility.

[2] https://github.com/eth-sri/eran

[3] https://github.com/JeetMo/Semantify-NN

Table 2: VeeP vs. Semantify-NN on 100 brightness neighborhoods and two models.

Dataset	Model		VeeP		Semantify-NN	
		δ_{adv}	δ_f	$t_f[s]$	δ_f	t_f[s]
MNIST	fully-connected 2x1024	0.7000	0.6801	0.06	0.6884	13.03
MNIST	fully-connected 4x1024	0.6822	0.6747	1.25	0.6770	68.72

Table 3: Maximal brightness and HSL neighborhoods vs. maximal ϵ-ball neighborhoods.

Dataset	Model	Feature	d_ϵ	d_f	δ_f	δ_{adv}
CIFAR-10	ResNetTiny	Brightness	0.0047	0.4191	0.4191	0.4195
CIFAR-10	ResNetTiny	Hue	0.0047	0.1834	3.7249	3.7390
CIFAR-10	ResNetTiny	Saturation	0.0047	0.1590	0.8630	0.8796
CIFAR-10	ResNet18	Brightness	0.0046	0.4096	0.4096	0.4146
CIFAR-10	ResNet18	Saturation	0.0046	0.1387	0.8227	0.8493
ImageNet	AlexNetTiny	Brightness	0.0025	0.2210	0.2210	0.2226
ImageNet	AlexNetTiny	Hue	0.0025	0.1104	1.6320	1.6360
ImageNet	AlexNetTiny	Saturation	0.0025	0.0787	0.4445	0.4557
ImageNet	AlexNetTiny	Lightness	0.0025	0.1631	0.1764	0.1771
ImageNet	AlexNet	Brightness	0.0009	0.3042	0.3042	0.3112
ImageNet	AlexNet	Hue	0.0009	0.0871	0.4849	0.5337
ImageNet	AlexNet	Saturation	0.0009	0.0817	0.4852	0.5052
ImageNet	AlexNet	Lightness	0.0009	0.2286	0.2398	0.2592

approaches prove more than 97% of the maximal certifiable diameter, however VeeP is 55x–217x faster than Semantify-NN. This factor is crucial for scaling to high-dimensional datasets and deep networks. This factor also shows the advantage of our dynamic policy over a constant policy.

Brightness and HSL neighborhoods Next, we evaluate VeeP for four semantic features: brightness, representing a linear feature, and HSL, representing non-linear features. HSL (hue, saturation, and lightness) is a color space transformation, where hue defines the position in the color wheel, saturation controls the image's colorfulness and lightness the perceived brightness. For every model and feature, we run VeeP on 50 inputs to compute the maximal certifiable diameter δ_f. We compare δ_f to the diameter of the closest adversarial example in the feature domain δ_{adv} (obtained using a grid search). For every δ_f, we compute d_f, the average pixel perturbation caused by the feature perturbation. To compute d_f for an image x, we look for the maximally perturbed x' in $I_{f,\delta_f}(x)$ and average the differences between x and x'. We compare d_f to d_ϵ, which is the maximal diameter of the ϵ-ball neighborhood that GPUPoly verifies. Table 3 reports these results. Our results indicate that VeeP proves on average at least 97% of the potentially certifiable diameters. The table also shows that focusing on feature neighborhoods enables us to certify neighborhoods whose diameters are larger by 99.1x compared to the diameters of the maximally certified ϵ-ball neighborhoods. Figure 3 illustrates the differences between the certifiable feature neighborhoods and the ϵ-ball ones. Each triple shows an ImageNet image, the maximally perturbed image in the maximally certified ϵ-ball, and the maximally perturbed image in the maximally certified feature neighborhood. These triples exemplify that the certified feature neighborhoods contain images that are visually different, thereby showing the gain of analyzing robustness of networks to feature perturbations.

PCA features Next, we evaluate VeeP for features derived from the dataset using PCA. Since PCA features are computed from the dataset (unlike the previous features), they are more likely to expose robust and non-robust aspects of the model, which is also computed from the dataset. In this experiment, we focus on the ResNet34 model for CIFAR-10. We further focus on a single class – the airplane class– in order to allow PCA extract features unique to that class (e.g., the sky color feature is more relevant to the airplane class than the car class). We run PCA on the training inputs to compute the first five PCA dimensions. For each dimension, we consider two features f_i^+ and f_i^-, one for each direction (i.e., $\pm\delta$). For each feature and for 100 airplane images, we run VeeP to compute the maximal diameter. Figure 4 shows the average pixel perturbation d_f of the maximally

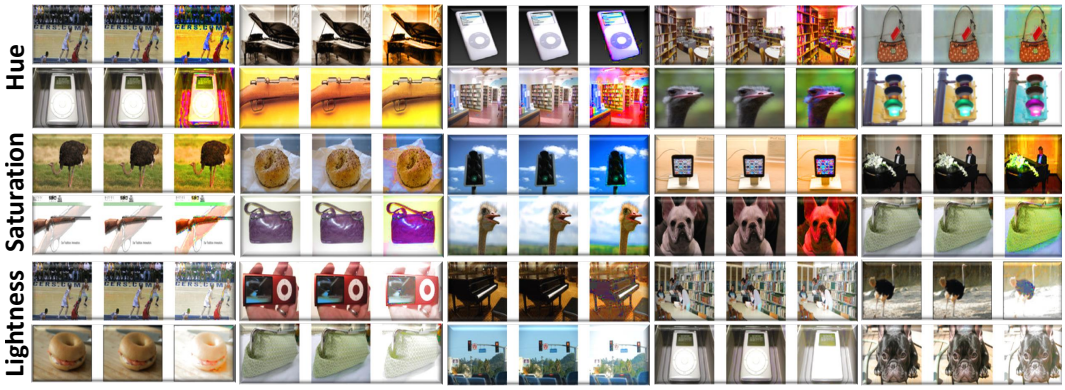

Figure 3: Illustration of images in certified ϵ-ball and feature neighborhoods. Each triple shows an ImageNet image, the maximally perturbed image in the maximal ϵ-ball neighborhood, and the maximally perturbed image in the maximal feature neighborhood (for the HSL features).

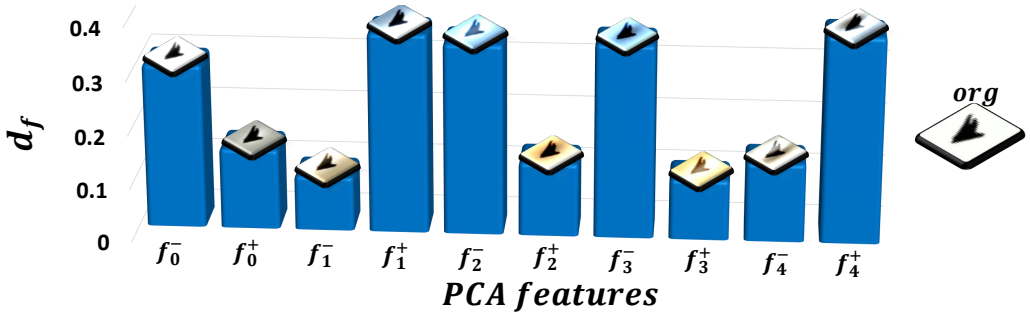

Figure 4: Average pixel perturbation d_f of the maximally certified PCA neighborhoods.

certified PCA neighborhoods. The bars illustrate the semantic meaning of perturbing their feature. The figure shows there is a large variance between the features. For example, features that add the sky color blue shades are more robust than ones that add yellow shades. Verifying a large diameter for a certain feature on many inputs may suggest that the model is robust for perturbations along this feature. To test this hypothesis, we consider an adversarial attack perturbing a given feature up to a limit of d_f over 300 airplane images (different from the previous 100 images). We run this attack, for the most robust feature f_4^+ and the least robust feature f_1^-, with a limit of $d_f = 0.2$. The attack's success rate is 93.3%, for f_1^-, and only 10.6%, for f_4^+. This may suggest that PCA features are linked to global robustness properties; we leave this for future work.

Time analysis Finally, we discuss the execution time of VeeP. On the smaller models, ResNetTiny and AlexNetTiny, and the DiffAI model, VeeP completes within 14.7 minutes on average. For the bigger models, VeeP completes within 2.1 hours on average. These execution times can be linearly reduced by increasing the number of GPUs.

6 Conclusion

We presented the concept of proof guided by velocity to scale feature robustness to deep networks and high-dimensional datasets. The idea is to phrase the certification problem as a dynamic system, splitting the task into smaller subproblems, each maximizes the proof velocity. We propose an instantiation of the dynamic system and a policy that builds on past certification results to dynamically decide on the next subproblem. We implement this concept in VeeP, and we evaluate it for various kinds of feature neighborhoods. We show that the average diameter of the neighborhoods that VeeP verifies is at least 97% of the maximal certifiable diameter. VeeP is also the first verifier to prove robustness of feature neighborhoods for an AlexNet model for ImageNet within a couple of hours.

ETHICS STATEMENT

Our work proposes a scalable verifier to prove the robustness of networks to feature perturbations. This potentially has a positive impact because it can help gain a better understanding of the network behavior and possibly increase user trust. However, our work also enables one to identify non-robust features of the network, which may be exploited by attacks and thus have a negative impact.

REFERENCES

Arash Ardakani, Carlo Condo, and Warren J. Gross. Sparsely-connected neural networks: Towards efficient VLSI implementation of deep neural networks. *In 5th International Conference on Learning Representations, ICLR*, 2017.

Mislav Balunovic, Maximilian Baader, Gagandeep Singh, Timon Gehr, and Martin T. Vechev. Certifying geometric robustness of neural networks. *In Annual Conference on Neural Information Processing Systems*, 2019.

Arjun Nitin Bhagoji, Daniel Cullina, Chawin Sitawarin, and Prateek Mittal. Enhancing robustness of machine learning systems via data transformations. *In 52nd Conference on Information Sciences and Systems, CISS*, 2018.

Anand Bhattad, Min Jin Chong, Kaizhao Liang, Bo Li, and David A. Forsyt. Unrestricted adversarial examples via semantic manipulation. *In 8th International Conference on Learning Representations, ICLR*, 2020.

Rudy Bunel, Ilker Turkaslan, Philip H. S. Torr, Pushmeet Kohli, and Pawan Kumar Mudigonda. A unified view of piecewise linear neural network verification. *In Advances in Neural Information Processing Systems 31*, 2018.

Nicholas Carlini and David A. Wagner. Adversarial examples are not easily detected: Bypassing ten detection methods. *In Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security, AISec*, 2017.

Sumanth Dathathri, Krishnamurthy Dvijotham, Alexey Kurakin, Aditi Raghunathan, Jonathan Uesato, Rudy Bunel, Shreya Shankar, Jacob Steinhardt, Ian Goodfellow, Percy Liang, and Pushmeet Kohli. Enabling certification of verification-agnostic networks via memory-efficient semidefinite programming. *In Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. *In IEEE conference on computer vision and pattern recognition*, 2009.

Rüdiger Ehlers. Formal verification of piece-wise linear feed-forward neural networks. In Deepak D'Souza and K. Narayan Kumar (eds.), *In Automated Technology for Verification and Analysis*, 2017.

Yizhak Yisrael Elboher, Justin Gottschlich, and Guy Katz. An abstraction-based framework for neural network verification. In Shuvendu K. Lahiri and Chao Wang (eds.), *In Computer Aided Verification - 32nd International Conference, CAV*, 2020.

Mahyar Fazlyab, Alexander Robey, Hamed Hassani, Manfred Morari, and George J. Pappas. Efficient and accurate estimation of lipschitz constants for deep neural networks. *In Advances in Neural Information Processing Systems 32*, 2019.

Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin T. Vechev. AI2: safety and robustness certification of neural networks with abstract interpretation. *In IEEE Symposium on Security and Privacy*, 2018.

Gabriel Goh. A discussion of 'adversarial examples are not bugs, they are features': Two examples of useful, non-robust features. *Distill*, 2019. doi: 10.23915/distill.00019.3.

Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *In 3rd ICLR*, 2015.

Gaurav Goswami, Nalini K. Ratha, Akshay Agarwal, Richa Singh, and Mayank Vatsa. Unravelling robustness of deep learning based face recognition against adversarial attacks. *In Proceedings of 34th AAAI*, 2018.

Sven Gowal, Krishnamurthy Dvijotham, Robert Stanforth, Rudy Bunel, Chongli Qin, Jonathan Uesato, Relja Arandjelovic, Timothy Arthur Mann, and Pushmeet Kohli. Scalable verified training for provably robust image classification. *In IEEE/CVF International Conference on Computer Vision, ICCV*, 2019.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *In IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

Hossein Hosseini and Radha Poovendran. Semantic adversarial examples. *In IEEE Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops*, 2018.

Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Logan Engstrom, Brandon Tran, and Aleksander Madry. Adversarial examples are not bugs, they are features. *In Advances in Neural Information Proc. Systems 32*, 2019.

Ameya Joshi, Amitangshu Mukherjee, Soumik Sarkar, and Chinmay Hegde. Semantic adversarial attacks: Parametric transformations that fool deep classifiers. *In International Conference on Computer Vision, ICCV*, 2019.

Guy Katz, Clark W. Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. *In 29th Computer Aided Verification Conference*, 2017.

Alex Krizhevsky. Learning multiple layers of features from tiny images. *CoRR, abs/1708.07747*, 2009.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *In Advances in Neural Information Processing Systems 25*, 2012.

Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *In Proceedings of the IEEE 1998;86(11):2278e324*, 1998.

Linyi Li, Xiangyu Qi, Tao Xie, and Bo Li. Sok: Certified robustness for deep neural networks. *CoRR*, abs/2009.04131, 2020. URL `https://arxiv.org/abs/2009.04131`.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *In 6th International Conference on Learning Representations, ICLR*, 2018.

Matthew Mirman, Timon Gehr, and Martin T. Vechev. Differentiable abstract interpretation for provably robust neural networks. *In Proceedings of the 35th International Conference on Machine Learning ICML*, 2018.

Christoph Müller, François Serre, Gagandeep Singh, Markus Püschel, and Martin Vechev. Scaling polyhedral neural network verification on gpus. *In Proceedings of Machine Learning and Systems 3*, 2021.

Jeet Mohapatra, Tsui-Wei Weng, Pin-Yu Chen, Sijia Liu, and Luca Daniel. Towards verifying robustness of neural networks against A family of semantic perturbations. *In IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR*, 2020.

Luca Pulina and Armando Tacchella. An abstraction-refinement approach to verification of artificial neural networks. *In Computer Aided Verification, 22nd International Conference, CAV*, 2010.

Luca Pulina and Armando Tacchella. Challenging SMT solvers to verify neural networks. *In AI Commun. 10.3233/AIC-2012-0525*, 2012.

Aditi Raghunathan, Jacob Steinhardt, and Percy Lian. Semidefinite relaxations for certifying robustness to adversarial examples. *In Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

Oliver Richter and Roger Wattenhofer. Treeconnect: A sparse alternative to fully connected layers. *In IEEE 30th International Conference on Tools with Artificial Intelligence*, 2018.

Wonryong Ryou, Jiayu Chen, Mislav Balunovic, Gagandeep Singh, Andrei Marian Dan, and Martin T. Vechev. Scalable polyhedral verification of recurrent neural networks. *In Computer Aided Verification - 33rd International Conference, CAV*, 2021.

Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin T. An abstract domain for certifying neural networks. *In Proc. ACM Program. Lang*, 2019.

Sahil Singla and Soheil Feizi. Second-order provable defenses against adversarial attacks. *In Proceedings of the 37th International Conference on Machine Learning, ICML*, 2020.

Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin A. Riedmiller. Striving for simplicity: The all convolutional net. *In 3rd International Conference on Learning Representations, ICLR*, 2015.

Vincent Tjeng, Kai Y. Xiao, and Russ Tedrake. Evaluating robustness of neural networks with mixed integer programming. *In 7th International Conference on Learning Representations, ICLR*, 2019.

Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Efficient formal safety analysis of neural networks. *In 31th Advances in Neural Information Processing Systems*, 2018a.

Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Formal security analysis of neural networks using symbolic intervals. *In 27th USENIX Security Symposium*, 2018b.

Matthew Wicker, Xiaowei Huang, and Marta Kwiatkowska. Feature-guided black-box safety testing of deep neural networks. *In 24th Tools and Algorithms for the Construction and Analysis of Systems TACAS*, 2018.

Haoze Wu, Alex Ozdemir, Aleksandar Zeljic, Kyle Julian, Ahmed Irfan, Divya Gopinath, Sadjad Fouladi, Guy Katz, Corina S. Pasareanu, and Clark W. Barrett. Parallelization techniques for verifying neural networks. In *In Formal Methods in Computer Aided Design, FMCAD*, 2020.

Kai Yuanqing Xiao, Vincent Tjeng, Nur Muhammad (Mahi) Shafiullah, and Aleksander Madry. Training for faster adversarial robustness verification via inducing relu stability. *In 7th International Conference on Learning Representations, ICLR*, 2019.

Kaidi Xu, Zhouxing Shi, Huan Zhang, Yihan Wang, Kai-Wei Chang, Minlie Huang, Bhavya Kailkhura, Xue Lin, and Cho-Jui Hsieh. Automatic perturbation analysis for scalable certified robustness and beyond. *In Advances in Neural Information Processing Systems 33*, 2020.

Dinghuai Zhang, Mao Ye, Chengyue Gong, Zhanxing Zhu, and Qiang Liu. Black-box certification with randomized smoothing: A functional optimization based framework. *In Advances in Neural Information Processing Systems 33*, 2020a.

Dinghuai Zhang, Mao Ye, Chengyue Gong, Zhanxing Zhu, and Qiang Liu. Scalable certified segmentation via randomized smoothing. *In Proceedings of the 38th International Conference on Machine Learning, ICML*, 2021.

Yonggang Zhang, Xinmei Tian, Ya Li, Xinchao Wang, and Da. Tao. Principal component adversarial example. *In IEEE Trans. Image Process*, 2020b.