Detecting Adversarial Samples Using Influence Functions and Nearest Neighbors

Gilad Cohen
Tel Aviv University
Tel Aviv, 69978
giladco1@mail.tau.ac.il

Guillermo Sapiro
Duke University
North Carolina, 27708
guillermo.sapiro@duke.edu

Raja Giryes
Tel Aviv University
Tel Aviv, 69978
raja@tauex.tau.ac.il

Abstract

Deep neural networks (DNNs) are notorious for their vulnerability to adversarial attacks, which are small perturbations added to their input images to mislead their prediction. Detection of adversarial examples is, therefore, a fundamental requirement for robust classification frameworks. In this work, we present a method for detecting such adversarial attacks, which is suitable for any pre-trained neural network classifier. We use influence functions to measure the impact of every training sample on the validation set data. From the influence scores, we find the most supportive training samples for any given validation example. A k-nearest neighbor (k-NN) model fitted on the DNN's activation layers is employed to search for the ranking of these supporting training samples. We observe that these samples are highly correlated with the nearest neighbors of the normal inputs, while this correlation is much weaker for adversarial inputs. We train an adversarial detector using the k-NN ranks and distances and show that it successfully distinguishes adversarial examples, getting state-of-the-art results on six attack methods with three datasets. Code is available at `https://github.com/giladcohen/NNIF_adv_defense`.

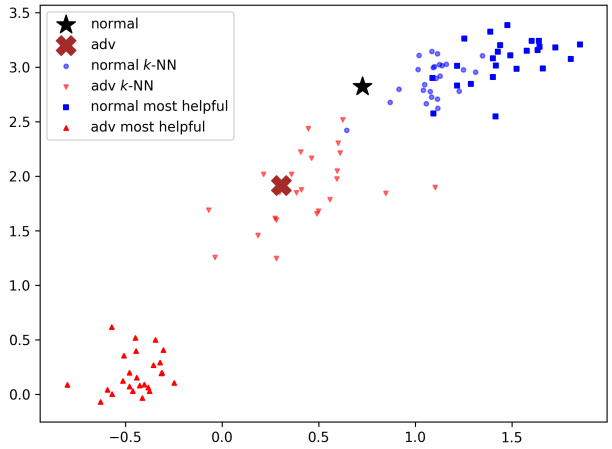

Figure 1. The correspondence between the helpful examples based on influence functions and the k-nearest neighbours (k-NN) in the embedding space of a DNN can help to distinguish adversarial examples from normal ones. We present (using PCA) the embedding space of a DNN for a normal example (black star) with its adversarial version (brown **X**) along with their k-NN (k=25) and 25 most helpful samples. Note that for the normal example, the helpful samples highly correlate with the k-NN in the embedding space. Yet, in the adversarial case, these samples are far from each other. This observation leads us to a technique for detecting adversarial attacks.

1. Introduction

Deep Neural Networks (DNNs) are vastly employed in both the academy and industry, achieving state-of-the-art (SOTA) results in many domains such as computer vision [21, 41, 49], natural language processing [1, 18], and speech recognition [15, 51]. However, studies have shown that DNNs are vulnerable to adversarial examples [12, 46], which are specially crafted perturbations on their input. Adversarial attacks generate such examples that fool machine learning models, inducing them to predict erroneously with high confidence, while being imperceptible to humans. Adversarial subspaces of different DNN classifiers tend to overlap, which makes some adversarial examples generated for a surrogate model fool also other different unseen DNNs [47]. This makes adversarial attacks a real threat to any machine learning model and thus should be kept in mind while deploying a DNN.

The vulnerability of neural networks puts into question their usage in sensitive applications, where an opponent may provide modified inputs to cause misidentifications. For this reason, many methods have been developed to face this challenge. They can be mainly divided into two groups: 1) *proactive* defense methods, which aim at improving the robustness of DNNs to adversarial examples, and 2) *reactive* detection techniques that do not change the DNN but rather try to find whether an attack is associated with a certain input or not.

Contribution. In this work, we focus on the reactive detection problem. We propose a novel strategy for detecting

adversarial attacks that can be applied to any pre-trained neural network. The core idea of the algorithm is that there should be a correspondence between the training data and the classification of the network. If this relationship breaks then it is very likely that we are in the case of an adversarial input.

To this end, we use two "metrics" to check the impact of the training data on the network decision. The first is influence functions [19], which determines how data points in the training set influence the decision of the network for a given test sample. This metric measures how much a small upweighting of a specific training point in the model's loss function affects the loss of a testing point. Thus, it provides us with a measure of how much a test sample classification is affected by each training sample.

Second, we apply a k-nearest neighbor (k-NN) classifier at the embedding space of the network. Various recent works [36, 8, 17, 7] demonstrate a high correlation between the network softmax output and the decision of a k-NN applied at the embedding space of this network (where the neighbors are chosen from the training set). They basically show that the network's decision relies on the nearest neighbors resemblance in the embedding space. Thus, the distance in that space may serve as a measure for the effect of an example on the network output.

Given the influence function and k-NN based measures, we turn to combine them together to generate a novel strategy to detect adversarial examples. The rationale behind our approach is that for a normal input, its k-NN training samples (nearest neighbors in the embedding space) and the most helpful training samples (found using the influence function) should correlate. Yet, for adversarial examples this correlation should break and thus, it will serve as an indication that an attack is happening.

Figure 1 illustrates this relationship between the k-NN and the most helpful training samples. The black star and brown **X** denote a normal and its corresponding adversarial image from CIFAR-10 validation set; the plot is of the embedding space projected using PCA fitted on the training set. For each sample (normal/adv), we find its 25 nearest neighbors (blue circles/red downward triangles) in the DNN embedding space; in addition, we find its 25 most helpful training examples from the training set (marked as blue squares and red upward triangles, respectively). Note that the nearest neighbors and the top most helpful training samples of the normal image are very close in the PCA embedding space, whereas the adversarial image does not exhibit the same correspondence between the training samples.

To check the correlation between the two, we pursue the following strategy: For an unseen input sample, we take the most influential examples from the training set chosen by the influence functions. Then, we check their distance ranking in the embedding space of the network (i.e., what value of k will cause k-NN to take them into account) and their L_2 distance from the input sample's embedding vector. Finally, we use these k-NN features to train a simple Logistic Regression (LR) for detecting whether the input is adversarial or not.

We evaluate our detection strategy on various attack methods and datasets showing its advantage over other leading detection techniques. The results confirm the hypothesis claimed in previous works on the resemblance between k-NN applied on the embedding space and the DNN decision, and show how it can be used for detecting adversarial examples.

2. Related work

In this section, we briefly review existing papers on adversarial attacks and defenses, and related theory.

Theory: Madry et al. used the framework of robust optimization and showed results of adversarial training [26]. They found that projected gradient descent (PGD) is an optimal first order adversary, and employing it in the DNN training leads to optimal robustness against any first order attack. Simon-Gabriel et al. demonstrated that DNNs' vulnerability to adversarial attacks is increased with the gradient of the training loss as a function of the inputs [43]. They also found that this vulnerability does not depend on the DNN model.

Fawzi et al. studied the geometry and complexity of the functions learned by DNNs and provided empirical analysis of the curvatures of their decision boundaries [10]. They showed that a DNN classifier is most vulnerable where its decision boundary is positively curved and that natural images are usually located in the vicinity of flat decision boundaries. These findings are also supported by Moosavi-Dezfooli et al. [31], who found that positively curved decision boundaries increase the likelihood that a small universal perturbation would fool a DNN classifier.

Some works provided guarantees to certify robustness of the network. Hein and Andriushchenko formalized a formal upper bound for the noise required to flip a network prediction [14], while Sinha et al. provided an efficient and fast guarantee of robustness for the worst-case population performance, with high probability [44].

Adversarial attacks: One of the simplest and fastest attack methods is the fast gradient sign method (FGSM) [12]; in this method the attacker linearly fits the cross entropy loss around the attacked sample and lightly perturbs the image pixels in the direction of the gradient loss. This is a fast one-step attack, which is very easy to deploy on raw input images.

The Jacobian-based saliency map attack (JSMA) [37] takes a different approach. Instead of mildly changing all image pixels, this attack is crafted on the L_0 norm, finding one or two pixels which induce the largest change in the loss and modify only them. This is a strong attack, achieving 97%

success rate by modifying only 4.02% of the input features on average. Yet, it is iterative and costly.

Deepfool [32] proposed by Moosavi-Dezfooli et al. is a non-targeted attack[1] that creates an adversarial example by moving the attacked input sample to its closest decision boundary, assuming an affine classifier. In reality most DNNs are very non linear, however, the authors used an iterative method, linearizing the classifier around the test sample at every iteration. Compared to FGSM and JSMA, Deepfool performs less perturbations to the input. It was also employed in the Universal Perturbations attack [30], which is an iterative attack that aims at fooling a group of images using the same minimal, universal perturbation applied on all of them.

Carlini and Wagner [4] proposed a targeted attack[2] (denoted as CW) to impact the defensive distillation method [35]. The CW attack is resilient against most adversarial detection methods. In another work Carlini and Wagner provided an optimization framework [3], which includes a defense-specific loss as a regularization term. This optimization-based attack is argued to be the most effective to date for a white-box threat model where the adversary knows everything related to the trained DNN: training data, architecture, hyper-parameters, weights, etc. Chen et al. [5] included a L_1 regularization to the CW attack, forming the Elastic-net Attack to DNNs (EAD).

Adversarial defenses: A wide range of proactive defense approaches have been proposed, including adversarial (re)training [12, 22, 48, 42, 29], distillation networks [35], gradient masking [48], feature squeezing [50], network input regularization [38, 16], output regularization [14], adjusting weights of correctly predicted labels [40], Parseval networks [6], and k-NN search [9, 45].

However, those defenses can be evaded by the optimization-based attack [3], either wholly or partially. Since there are no known intrinsic properties that differentiate adversarial samples from regular images, proactive adversarial defense is extremely challenging. Instead, recent works have focused on reactive adversarial detection methods, which aim at distinguishing adversarial images from natural images, based on features extracted from DNN layers [28, 24, 39] or from a learned encoder [27]. Feinman et al. [11] proposed a LR detector based on Kernel density (KD) and Bayesian uncertainty features.

Ma et al. [25] characterized the dimensional properties of the adversarial subspaces regions and proposed to use a property called Local Intrinsic Dimentionaloty (LID) . LID describes the rate of expansion in the number of data objects as the distance from the reference sample increases. The authors estimated the LID score at every DNN layer using extreme value theory, where the smallest NN distances are considered as extreme events associated with the lower tail of the data samples' underlying distance distribution. Given a pretrained network and a dataset of normal examples, the authors applied on every sample: 1) Adversarial attack. 2) Addition of Gaussian Noise. The natural and noisy images were considered as negative (non-adversarial) class and the adversarial images were considered as positive class. For each image (natural/noisy/adversarial) they calculated a LID score at every DNN layer. Lastly, a LR model was fitted on the LID features for the adversarial detection task.

Papernot and McDaniel [36] proposed the Deep k-Nearest Neighbors (DkNN) algorithm to estimate better the prediction, confidence, and credibility for a given test sample. Using a pretrained network, they fitted a k-NN model at every layer. Next, they used a left-out calibration set to estimate the nonconformity of every test sample for label j, counting the number of nearest neighbors along the DNN layer which differs from j. They showed that when an adversarial attack is made on a test sample, the real label displays less correspondence with the k-NN labels from the DNN activations along the layers.

Lee et al. [23] trained generative classifiers using the DNN activations of the training set on every layer to detect adversarial examples by applying a Mahalanobis distance-based confidence score. First, for every class and every layer, they computed the empirical mean and covariance of the activations induced by the training samples. Next, using the above class-conditional Gaussian distributions, they calculated the Mahalanobis distance between a test sample and its nearest class-conditional Gaussian. These distances are used as features to train a LR classifier for the adversarial detection task. The authors claimed that using the Mahalanobis distance is significantly more effective than the Euclidean distance employed in [25] and showed improved detection results.

3. Method

We hypothesize that the DNN predictions are influenced by the k-NN of the training data in their hidden layers, especially in the embedding layer. If so, in order to fool the network, an adversarial attack must move the test sample towards a "bad" subspace in the embedding space, where harmful training data can cause the network to misclassify the correct label. To inspect our hypothesis, we fitted a k-NN model on the DNN's activation layers, and also employed the influence functions as used in [19].

Influence functions can interpret a DNN by pointing out which of the training samples helped the DNN to make its prediction, and which training samples were harmful, i.e., inhibited the network from its prediction. Koh and Liang [19] suggested to measure the influence a train image z has

[1]Non-targeted attacks are adversarial attacks which aim to make the prediction incorrect regardless of the spricifc erroneous class.

[2]Targeted attacks are adversarial attacks which aim to make the prediction classified to a particular erroneous class.

on the loss of a specific test image z_{test}, by the term:

$$I_{up,loss}(z, z_{test}) = -\nabla_\theta L(z_{test}, \theta)^T H_\theta^{-1} \nabla_\theta L(z, \theta), \quad (1)$$

where H is the Hessian of the machine learning model, L is its loss, and θ are the model parameters. In the definition of Eq. (1) z and z_{test} are images.

For each test example z_{test}, we calculate Eq. (1) per each training example z in the training set. Then, we sort all $I_{up,loss}(z, z_{test})$ scores, determining the top M helpful and harmful training examples for a specific z_{test}. Next, for each of the 2xM selected training points we find its rank and distance from the testing example by fitting a k-NN model on the embedding space using all the training examples' embedding vectors. We feed the embedding vector of each test sample z_{test} to the k-NN model to extract all the nearest neighbors' ranks (denoted as $\mathcal{R}$) and distances (denoted $\mathcal{D}$) of the examples in the training set. The $\mathcal{R}$ and $\mathcal{D}$ features can also be extracted from any other hidden activation layer within the DNN, and not solely from the embedding vector. $\mathcal{R}^{M\uparrow}$, $\mathcal{D}^{M\uparrow}$ and $\mathcal{R}^{M\downarrow}$, $\mathcal{D}^{M\downarrow}$ are all the ranks and distances of the helpful and harmful training examples, respectively.

We apply an adversarial attack on z_{test} and repeat the aforementioned process on the new, crafted image. Both the normal and adversarial features ($\mathcal{R}^{M\uparrow}$, $\mathcal{D}^{M\uparrow}$, $\mathcal{R}^{M\downarrow}$, $\mathcal{D}^{M\downarrow}$) are used to train a LR classifier for the adversarial detection task. The detector training scheme is described in Alg. 1.

We name our adversarial detection method Nearest Neighbor Influence Functions (NNIF). We assume that the training, validation, and testing sets are not contaminated with adversarial examples, as in [3]. We start by generating an adversarial validation set from the normal validation set (step 4). The M most helpful and harmful training examples associated with the validation image prediction (either normal or adversarial) are found using the influence function in step 22 (see supp. material for the INFLUENCEFUNCTION procedure). The NNIF features are then evaluated by the k-NN model, extracting the ranks and distances (from $\mathcal{R}$ and $\mathcal{D}$) of the most influential training points found above. This is done for both the normal validation images (step 8) and for the adversarial images (step 12). This scheme can be carried out on the embedding layer alone, or employed for all L activation layers within the DNN.

Finally, a LR classifier is trained using the NNIF features. Images from the test set are classified to either adversarial (positive) or normal (negative) based on the NNIF features extracted from the M most helpful/harmful training examples, ($\mathcal{R}^{M\uparrow}$, $\mathcal{D}^{M\uparrow}$, $\mathcal{R}^{M\downarrow}$, $\mathcal{D}^{M\downarrow}$).

Training our NNIF detector is very time consuming, requiring us to calculate Eq. (1) on the entire training set for every validation image, having a time complexity of $\mathcal{O}(N_{train} \cdot N_{val})$, where N_{train} and N_{val} are the size of the training and validation sets, respectively. For an adversarial detection the complexity time is $\mathcal{O}(N_{train})$, since we need to find the top M helpful/harmful training examples for every new incoming test image.

Papernot and McDaniel [36] focused on improving credibility and robustness in DNN. They used the nearest neighbors in the activation layers for interpretability. As an additional competing strategy, we convert their original DkNN algorithm [36] to an adversarial detection method. This is done by collecting the empirical p-values calculated in the DkNN strategy and formulating a reactive adversarial detector by training a LR model on these features. While NNIF also uses nearest neighbors, instead of inspecting the labels of the nearest neighbors, we examine the correlation between them and the image's most helpful/harmful training examples using the influence functions.

4. Results

This section shows the power of our NNIF adversarial detector against six adversarial attack strategies (norms in parentheses): FGSM (L_∞), JSMA (L_0), DeepFool (L_2), CW (L_2), PGD (L_∞), and EAD (L_1), as introduced in Section 2. The PGD attack was used with input perturbation as implemented by [26]. We selected these attacks for our experiments due to their effectiveness, diversity, and popularity. For versatility, we used Deepfool and EAD as non-targeted attacks. PGD was We applied these attacks on three datasets: CIFAR-10, CIFAR-100 [20], and SVHN [33]. NNIF performance is compared to the SOTA LID and Mahalanobis detectors (Section 2) and also to the DkNN adversarial detector (Section 3). Lastly, we analyzed the robustness of NNIF against a white-box setting. Before presenting our results, we first describe the experimental setup used in our analysis.

4.1. Experimental setup

Training and Testing: Each of the three image datasets was divided into three subsets: *training* set, *validation* set, and *testing* set, containing 49k, 1k, and 10k images respectively. Since our NNIF method is time consuming (especially the procedure INFLUENCEFUNCTION in Alg. 1), we randomly selected 49k and 1k *training* and *validation* samples, respectively, from the official SVHN training set and 10k testing samples from the official SVHN testing set. Any *validation* or *testing* image not correctly classified by the DNN was discarded. For every image in the *validation* and *testing* sets, we generated adversarial examples using all six attack methods, as describe in Step 4 in Alg. 1. Then, an equal number of normal and adversarial *validation* images were used to train a LR classifier, which was later applied on the remaining *testing* images for calculating the detectors metrics. We used the *cleverhans* library [34] to carry out all the adversarial attacks.

Since the DkNN method requires a calibration set, we randomly selected 33% of the *validation* set examples (after discarding the misclassifications) for calibrating it. Note

Algorithm 1 Adversarial detection using Nearest Neighbors Influence Functions (NNIF)

Input: Training set (X_{train}, Y_{train}) and validation set (X_{val}, Y_{val})
Input: Pre-trained DNN with L activation layers and parameters θ
Input: M: Number of top influence samples to collect
Output: Detector($\mathcal{R}^{M\uparrow}$, $\mathcal{D}^{M\uparrow}$, $\mathcal{R}^{M\downarrow}$, $\mathcal{D}^{M\downarrow}$) ▷ An adversarial example detector

$N_{train} = |X_{train}|$, $N_{val} = |X_{val}|$ ▷ Number of examples in train- and validation-set
Initialize: R^{+}_{norm}=[], D^{+}_{norm}=[], R^{-}_{norm}=[], D^{-}_{norm}=[] ▷ Normal image features
Initialize: R^{+}_{adv}=[], D^{+}_{adv}=[], R^{-}_{adv}=[], D^{-}_{adv}=[] ▷ Adversarial image features
$(X^{adv}_{val}, Y^{adv}_{val})$:= adversarial attack on (X_{val}, Y_{val}) ▷ Generate a new adversarial dataset by attacking the validation set
for *l* in [1,L] **do**
    Fit k-NN[l] model on layer *l*. $k = N_{train}$
    for (x_i, y_i) in (X_{val}, Y_{val}) **do**
        $\mathcal{R}^{M\uparrow}$, $\mathcal{D}^{M\uparrow}$, $\mathcal{R}^{M\downarrow}$, $\mathcal{D}^{M\downarrow}$:= NNFEATURES(x_i, k-NN[l]) ▷ Get NNIF helpful/harmful features for normal images
        R^{+}_{norm}.append($\mathcal{R}^{M\uparrow}$), D^{+}_{norm}.append($\mathcal{D}^{M\uparrow}$), R^{-}_{norm}.append($\mathcal{R}^{M\downarrow}$), D^{-}_{norm}.append($\mathcal{D}^{M\downarrow}$)
    end for
    for (x_i, y_i) in $(X^{adv}_{val}, Y^{adv}_{val})$ **do**
        $\mathcal{R}^{M\uparrow}$, $\mathcal{D}^{M\uparrow}$, $\mathcal{R}^{M\downarrow}$, $\mathcal{D}^{M\downarrow}$:= NNFEATURES(x_i, k-NN[l]) ▷ Get NNIF helpful/harmful features for adv images
        R^{+}_{adv}.append($\mathcal{R}^{M\uparrow}$), D^{+}_{adv}.append($\mathcal{D}^{M\uparrow}$), R^{-}_{adv}.append($\mathcal{R}^{M\downarrow}$), D^{-}_{adv}.append($\mathcal{D}^{M\downarrow}$)
    end for
end for
$NNIF_{pos}$ = (R^{+}_{adv}, D^{+}_{adv}, R^{-}_{adv}, D^{-}_{adv})
$NNIF_{neg}$ = (R^{+}_{norm}, D^{+}_{norm}, R^{-}_{norm}, D^{-}_{norm})
Detector($\mathcal{R}^{M\uparrow}$, $\mathcal{D}^{M\uparrow}$, $\mathcal{R}^{M\downarrow}$, $\mathcal{D}^{M\downarrow}$) = train a classifier on ($NNIF_{pos}$, $NNIF_{neg}$)

procedure NNFEATURES(x_i, k-NN[l]) ▷ Collecting nearest neighbors features
    Initialize: R^{+}=[], D^{+}=[], R^{-}=[], D^{-}=[] ▷ image's nearest neighbors features
    $\mathcal{R}$, $\mathcal{D}$:= Apply k-NN on activation layer *l*, get training examples' ranks & L_2 distances out of activations of sample x_i
    H^{+}_{inds}, H^{-}_{inds} := INFLUENCEFUNCTION($(x_i, y_i), (X_{train}, Y_{train})$) ▷ get indices of the 2xM most influencing training samples. This procedure is presented in the supp. material.
    for j in H^{+}_{inds} **do** ▷ Collect M helpful ranks and distances
        R^{+}.append($\mathcal{R}[j]$)
        D^{+}.append($\mathcal{D}[j]$)
    end for
    for j in H^{-}_{inds} **do** ▷ Collect M harmful ranks and distances
        R^{-}.append($\mathcal{R}[j]$)
        D^{-}.append($\mathcal{D}[j]$)
    end for
    return R^{+}, D^{+}, R^{-}, D^{-}
end procedure

that although Papernot and McDaniel [36] showed that the nearest neighbors can qualitatively detect adversarial attacks (see Fig. 7 in [36]), they did not formalize an adversarial detector. We employ their empirical p-values as features for the adversarial detection task.

Training DNNs: We trained all DNNs on the *training* set while decaying the learning rate using the *validation* set's accuracy score. All the DNNs used in our experiments are Resnet-34 [13] with global average pooling layer prior to the embedding space. The embedding vector was multiplied by a fully-connected layer for the logits calculation. We trained all three datasets for 200 epochs, with L_2 weight regularization of 0.0004, using a Stochastic Gradient Decent optimizer with momentum 0.9 and Nesterov updates. For evaluation we used the model checkpoint with the best (highest) validation accuracy on the image classification task. We follow the checklist in [2] and report the full DNN validation/test accuracies for the clean models when not under attack and the attacks success rates (see supp. material). These DNNs perform close to the SOTA and thus are sufficient for being used in an adversarial study without fine tuning [11].

Parameter tuning: The number of neighbors (k) for LID

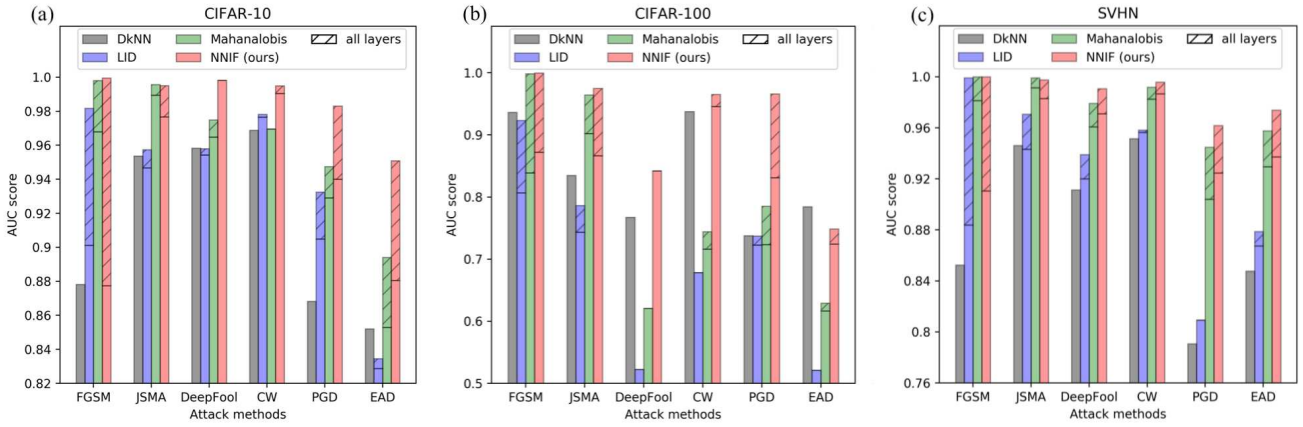

Figure 2. Comparison of AUC scores for detection of FGSM, JSMA, Deepfool, CW, PGD, and EAD attacks on three datasets: (a) CIFAR-10, (b) CIFAR-100, and (c) SVHN. The black, blue, green, and red bars correspond to the DkNN, LID, Mahalanobis, and NNIF defense methods, respectively. The hatched pattern bars correspond to AUC scores increase where taking into consideration all the DNN activation layers instead of just the penultimate activation layer. Each attack cluster of bars is divided to four columns which correspond to the methods (from left to right): DkNN, LID, Mahalanobis, and NNIF. Our NNIF detector surpasses previous SOTA methods by a large margin for most of the attacks.

and DkNN, the noise magnitude (ϵ) for the Mahalanobis method, and the number of top influence samples to collect (M) for NNIF were chosen using nested cross validation within the *validation* set, based on the AUC values of the detection ROC curve. We tuned k for DkNN using an exhaustive grid search between [10, $N/\#classes$], where N is the dataset size and $\#classes$ is the number of classes. For LID the number of nearest neighbors was tuned using a grid search over the range [10, 40) while using a minibatch size of 100 (as in [25]). For the Mahalanobis method we tuned ϵ using an exhaustive grid search in log-space between [$1E^{-5}$, $1E^{-2}$], and M was tuned using a grid search over [10, 500]. The selected parameters are presented in the supp. material.

Running INFLUENCEFUNCTION in Alg. 1 for an entire *training* set is very slow. Thus, for every *testing* set we randomly selected only 10k out of the 49k samples in the *training* set and calculated $I_{up,loss}$ (Eq. (1)) just for them. Although this is a coarse approximation of the real nearest neighbors distribution in the *training* set on the DNN embedding space, this approximation is sufficient for achieving new SOTA adversarial detection. We emphasize that this approximation was done only for the *testing* set, and not for the *validation* set.

Activation layers: The LID, Mahalanobis, and NNIF detectors can be trained using either features from the embedding space alone or using all the activation layers in the network. The DkNN detector portrays very poor results when it is applied on all the DNN's features (data not shown) and therefore, we present all the DkNN results by training features from the embedding space alone.

Threat model: We consider two treat models, black-box and white-box settings. Unless stated otherwise, the default threat model is black-box, where the attacker is unaware that an adversarial detection is employed. In this setting, only the model's parameters are given to the adversary. In Section 4.5 we also consider a white-box setting, where the attacker knows the model parameters, and also the adversarial detection scheme.

4.2. Detection of adversarial attacks

Figure 2 shows the discrimination power (AUC score) of the four inspected adversarial detectors: DkNN (black), LID (blue), Mahalabolis (green) and NNIF (red), on three popular datasets: CIFAR-10, CIFAR-100, and SVHN. We compare between the detection scores calculated for six adversarial attacks: FGSM, JSMA, Deepfool, CW, PGD, and EAD. The solid bars correspond to detections where only the penultimate activation layer was utilized. In some cases, considering all the layers in the DNN activations boosts the LID/Mahalanobis/NNIF scores; this is portrayed as a complementary hatched patterned bar above the solid bar.

Our method surpasses all other detectors for distinguishing Deepfool, CW, and PGD attacks, for all the datasets. On FGSM and JSMA our NNIF detector also demonstrates SOTA results, matching the Mahalanobis detector's performance. Against EAD we show new SOTA for CIFAR-10 and SVHN, but not for CIFAR-100. Table 1 summarizes the AUC scores of the detectors using features from all the DNN's activation layers. The only exception is the DkNN method, which is employed only on the embedding space. In the supp. material, we include results for more attacks and a similar table for the obtained AUC scores using only the DNN's penultimate layer.

Table 1. Comparison of AUC scores (%) for various adversarial detection methods. Results obtained using all the DNN's activation layers for LID/Mahalanobis/NNIF and only the embedding space for DkNN.

Dataset	Detector	FGSM	JSMA	Deepfool	CW	PGD	EAD
CIFAR-10	DkNN	87.81	95.37	95.82	96.88	86.83	85.20
	LID	98.18	95.74	95.80	97.82	93.24	83.46
	Mahalanobis	99.80	**99.56**	97.49	96.48	94.74	89.41
	NNIF (ours)	**99.96**	99.50	**99.32**	**99.5**	**98.31**	**95.09**
CIFAR-100	DkNN	93.65	83.46	76.71	93.77	73.78	**78.42**
	LID	92.33	78.63	51.61	67.83	73.71	51.11
	Mahalanobis	99.87	96.44	62.05	74.43	78.53	62.93
	NNIF (ours)	**99.96**	**97.50**	**77.17**	**96.51**	**96.60**	74.86
SVHN	DkNN	85.24	94.61	91.13	95.15	79.07	84.77
	LID	99.92	97.06	93.90	95.82	80.12	87.86
	Mahalanobis	**100.00**	**99.91**	97.92	99.18	94.47	95.77
	NNIF (ours)	**100.00**	99.76	**99.06**	**99.59**	**96.18**	**97.40**

Table 2. Ablation test for adversarial attack detection: Calculating AUC score and accuracy for selected features. Attacking CIFAR-10 dataset using Deepfool.

$\mathcal{R}^{M\uparrow}$	$\mathcal{D}^{M\uparrow}$	$\mathcal{R}^{M\downarrow}$	$\mathcal{D}^{M\downarrow}$	AUC(%)	acc(%)
			✓	82.11	77.03
		✓		66.14	61.47
		✓	✓	83.25	78.44
	✓			99.79	97.68
	✓		✓	**99.82**	97.51
	✓	✓		99.79	99.29
	✓	✓	✓	99.81	97.34
✓				98.27	96.69
✓			✓	97.73	97.21
✓		✓		98.28	96.73
✓		✓	✓	97.62	97.12
✓	✓			99.79	97.73
✓	✓		✓	99.81	97.78
✓	✓	✓		99.79	97.71
✓	✓	✓	✓	**99.82**	**97.86**

4.3. Ablation study

To quantify the contribution of each one of the features ($\mathcal{R}^{M\uparrow}$, $\mathcal{D}^{M\uparrow}$, $\mathcal{R}^{M\downarrow}$, $\mathcal{D}^{M\downarrow}$) on the NNIF method performance, we conducted an ablation study on CIFAR-10 dataset. Table 2 shows the AUC and accuracy results for Deepfool attack using features from the DNN's embedding space only. In the supp. material we present an extended ablation study with more attacks: FGSM, JSMA, and CW.

Our analysis shows that the most influential feature is $\mathcal{D}^{M\uparrow}$, which is the L_2 distance from the most helpful training examples on the embedding space. In most cases, our NNIF detector performance using $\mathcal{D}^{M\uparrow}$ is nearly as good as the performance upon utilizing all four features. The least important feature is $\mathcal{R}^{M\downarrow}$, which barely helps the adversarial detection. Intuitively it makes sense because we have noticed that the classes of the most harmful training examples always differ from the normal examples' class and mostly differ from the adversarial examples' class, and thus their rankings ($\mathcal{R}^{M\downarrow}$) are expected to be high for both cases (normal/adversarial). On the other hand, the distances from the most harmful training examples ($\mathcal{D}^{M\downarrow}$) are beneficial for the detection. The most helpful ranks ($\mathcal{R}^{M\uparrow}$) is a beneficial feature when used by itself, alas incorporating it with $\mathcal{D}^{M\uparrow}$ did not improve the results. We therefore deduce that the information added by $\mathcal{R}^{M\uparrow}$ can already be inferred from $\mathcal{D}^{M\uparrow}$ in our detector.

We also show that the features $\mathcal{R}^{M\uparrow}$, $\mathcal{D}^{M\uparrow}$, $\mathcal{D}^{M\downarrow}$ affect every attack differently. We calculated the probability density functions for these three features on CIFAR-10, applying the Deepfool and CW attacks (shown in the supp. material). From these histograms it can be easily observed that $\mathcal{R}^{M\uparrow}$ or $\mathcal{D}^{M\uparrow}$ are more useful for detecting Deepfool adversarial attacks than CW ones. On the other hand, the $\mathcal{D}^{M\downarrow}$ feature discriminates CW attacks better than Deepfool attacks.

A deployment of any learning based detector on systems is risky since an attacker could potentially have access to the LR classifier's parameters. Thus, it is helpful to deploy instead a detector which inspects only one feature and applies a simple thresholding. Our results show that this scheme is possible with NNIF using only the $\mathcal{D}^{M\uparrow}$ feature for all attacks.

4.4. Generalization to other attacks

To evaluate how well our detection method can be transferred to unseen attacks, we trained LR classifiers on the features obtained using the FGSM attack, and then evaluated the classifies on the other (unseen) attacks. The AUC scores are shown in Table 3. It can be observed that our NNIF method shows the best generalization everywhere except to JSMA. Table 3 results were collected using only the penultimate layer in the DNN (the embedding vector); A similar generalization table with additional attacks, using all the DNN layers, is provided in the supp. material. Notice that the generalization is weaker for all methods in this case.

Table 3. Generalization of adversarial detection from FGSM attack to unseen attacks. The LR classifier is trained on the features extracted after applying FGSM attack, and then evaluated on JSMA, Deepfool, CW, PGD, and EAD.

Dataset	Detector	FGSM (seen)	JSMA	Deepfool	CW	PGD	EAD
CIFAR-10	DkNN	87.81	94.89	95.21	96.76	85.10	83.28
	LID	90.12	94.67	95.43	97.66	90.29	82.52
	Mahalanobis	96.80	**98.95**	95.03	89.57	91.39	68.87
	NNIF (ours)	87.75	94.81	**97.98**	**98.98**	**93.94**	**86.95**
CIFAR-100	DkNN	93.65	83.16	62.41	92.22	73.60	62.67
	LID	80.68	74.33	52.25	67.84	72.25	52.10
	Mahalanobis	83.90	**90.20**	59.96	68.72	69.42	59.34
	NNIF (ours)	87.23	80.76	**78.82**	**93.16**	**81.87**	**70.49**
SVHN	DkNN	85.24	93.43	89.84	92.20	75.99	79.81
	LID	88.38	93.93	91.32	94.22	80.26	84.24
	Mahalanobis	98.14	**99.00**	91.46	87.51	86.26	80.62
	NNIF (ours)	91.06	97.91	**95.79**	**98.16**	**89.80**	**91.99**

4.5. Attack against NNIF

Here we consider a white-box threat model. In this setting, the adversary knows more than just the model parameters. We assume that the attacker is familiar with the adversarial defense scheme, but does not have access to the detector's parameters. Since the NNIF algorithm utilizes the entire training set, these data are also accessible to the attacker in our white-box setting. We employ a similar attack strategy as was proposed in [3] to evade the KD-based detector, and define a modified objective for the CW minimization:

$$\text{minimize } \|x - x_{adv}\|_2^2 + c \cdot \left(\ell_{cw}(x_{adv}) + \ell_*(D(x_{adv}))\right), \tag{2}$$

where ℓ_{cw} is the original adversarial loss term used in [4], and $D(x_{adv})$ is the sum over all the distances (in the embedding space) between the adversarial image and the original image's most helpful training samples (D^+_{adv} in Alg. 1). More rigorously, we define:

$$\ell_*\left(D(x_{adv})\right) := \sum D^+_{adv} = \sum_{i=1}^{M} \left\|\text{DNN}(x_{adv}) - \text{DNN}\left(X_{train}(H^+_{inds}[i])\right)\right\|_1, \tag{3}$$

where DNN$(\cdot)$ is the network transformation from the input image to the embedding vector in the penultimate layer, and $H^+_{inds}[i]$ is the index of the i^{th} most helpful training sample. Lastly, c is a constant which balances between the fidelity to the original image and the adversarial strength.

The objective of the minimization in Eq. (2) is to apply the CW attack while keeping x_{adv} close to the most helpful training samples of the original image. In theory, we should have demanded this proximity for the nearest neighbors of x_{adv}, and not for x_{adv} itself, but differentiating over the nearest neighbor algorithm is not feasible. It should also be noted that this attack was performed only on the penultimate activation layer, with the features that correspond to the most helpful examples: $\mathcal{R}^{M\uparrow}$ and $\mathcal{D}^{M\uparrow}$.

We applied this white-box attack on 4000 random samples of CIFAR-10 test set. We show the performances of DkNN, LID, Mahalanobis, and NNIF detectors on the original CW compared to our CW-Opt attack in Table 4. Results for other datasets are in the supp. material. For every detector we used the same hyper-parameters which yielded the best defense results using only the last layer. Following [3], we present the results in this experiment in term of accuracy, instead of AUC used in previous tests.

From Table 4 we observe that the proposed white-box attack decreases NNIF detection accuracy by only 1%. Therefore, we conclude that our NNIF defense algorithm is robust to a white-box setting. In addition, we note that the new attack impairs all the defense algorithms which rely on L_2 distance of nearest neighbors in the embedding space: DkNN, LID, and NNIF. Yet, for Mahalanobis we observe an adverse effect. This is somewhat expected since Mahalanobis estimates a global Gaussian for each class, and does not consider local features in the embedding space.

Table 4. Defense accuracy (%) for a white-box attack targeting the NNIF detector on CIFAR-10.

Attack	DkNN	LID	Mahalanobis	NNIF
CW	93.45	91.43	90.70	91.95
CW-Opt	90.99	89.74	92.29	90.81

5. Discussion and conclusions

In this paper, we addressed the task of detecting adversarial attacks. We showed that for normal (untempered) images, there exists a strong correlation between their nearest neighbors in the DNN's embedding space and their most helpful training examples, found using influence functions. Our empirical results show that the L_2 distance from a test image embedding vector to its most helpful training inputs ($\mathcal{D}^{M\uparrow}$) is a strong measure for the detection of adversarial examples. The aforementioned distance combined with the nearest neighbors ranking order of the training inputs were used to achieve a SOTA adversarial detection performance for six attacks (FGSM, JSMA, Deepfool, CW, PGD, EAD) on three datasets: CIFAR-10, CIFAR-100, and SVHN. Furthermore, we showed that our detector is robust in a white-box setting.

One possible avenue for future research is to inspect how the nearest neighbors are correlated with the most helpful/harmful training examples using different distance metrics or by employing a transform on the DNN embedding vectors. We emphasize that we mainly used the L_2 distance throughout our analysis, thus, we suspect that using another distance metric such as Mahalanobis [23] could improve our results further.

Another open issue for future research is the long computation time, which is required to calculate the influence functions for the entire training set. It is obvious that in order to deploy our NNIF algorithm, a significant improvement in computation time is needed, especially for real time applications or systems, which mandate fast detection pace. A possible solution to this problem may be a form of hash map from the nearest neighbors to the most influence training examples. Every training example can be encoded with a probability vector for its influence on a specific class; then, instead of employing a simple k-NN search in the embedding space, we can average over the probability of each class.

Acknowledgements. GS is partially supported by ARO, NGA, ONR, NSF, and gifts from Amazon, Google, and Microsoft. RG and GC are supported by ERC-StG grant no. 757497 (SPADE) and gifts from NVIDIA, Amazon, and Google.

References

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv:1409.0473*, 2014.

[2] Nicholas Carlini, Anish Athalye, Nicolas Papernot, Wieland Brendel, Jonas Rauber, Dimitris Tsipras, Ian Goodfellow, Aleksander Madry, and Alexey Kurakin. On evaluating adversarial robustness. *arXiv:1902.06705*, 2019.

[3] Nicholas Carlini and David A Wagner. Adversarial examples are not easily detected: Bypassing ten detection methods. In *AISec@CCS*, 2017.

[4] Nicholas Carlini and David A Wagner. Towards evaluating the robustness of neural networks. *2017 IEEE Symposium on Security and Privacy (SP)*, pages 39–57, 2017.

[5] Pin-Yu Chen, Yash Sharma, Huan Zhang, Jinfeng Yi, and Cho-Jui Hsieh. Ead: Elastic-net attacks to deep neural networks via adversarial examples. In *AAAI*, 2018.

[6] Moustapha Cisse, Piotr Bojanowski, Edouard Grave, Yann Dauphin, and Nicolas Usunier. Parseval networks: Improving robustness to adversarial examples. In *ICML*, 2017.

[7] Gilad Cohen, Guillermo Sapiro, and Raja Giryes. DNN or k-NN: That is the generalize vs. memorize question. *arXiv:1805.06822*, 2018.

[8] Maik Döring, László Györfi, and Harro Walk. Rate of convergence of k-nearest-neighbor classification rule. *J. Mach. Learn. Res.*, 18(1):8485–8500, 1 2017.

[9] Abhimanyu Dubey, Laurens van der Maaten, Zeki Yalniz, Yixuan Li, and Dhruv Kumar Mahajan. Defense against adversarial images using web-scale nearest-neighbor search. *CVPR*, pages 8759–8768, 2019.

[10] Alhussein Fawzi, Seyed-Mohsen Moosavi-Dezfooli, Pascal Frossard, and Stefano Soatto. Classification regions of deep neural networks. *arXiv:1705.09552*, 2017.

[11] Reuben Feinman, Ryan R Curtin, Saurabh Shintre, and Andrew B Gardner. Detecting adversarial samples from artifacts. *arXiv:1703.00410*, 2017.

[12] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *ICLR*, 2015.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CVPR*, pages 770–778, 2016.

[14] Matthias Hein and Maksym Andriushchenko. Formal guarantees on the robustness of a classifier against adversarial manipulation. In *NIPS*, 2017.

[15] Geoffrey Hinton, Li Deng, Dong Yu, George Dahl, Abdel-Rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara Sainath, and Brian Kingbury. Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

[16] Daniel Jakubovitz and Raja Giryes. Improving dnn robustness to adversarial attacks using jacobian regularization. In *ECCV*, 2018.

[17] Heinrich Jiang, Been Kim, Melody Guan, and Maya Gupta. To trust or not to trust a classifier. In *NIPS*, pages 5546–5557, 2018.

[18] Yoon Kim. Convolutional neural networks for sentence classification. In *EMNLP*, pages 1746–1751, 2014.

[19] Pang Wei Koh and Percy S. Liang. Understanding black-box predictions via influence functions. In *ICML*, 2017.

[20] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[21] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances In Neural Information Processing Systems*, pages 1–9, 2012.

[22] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. Adversarial machine learning at scale. *arXiv:1611.01236*, 2017.

[23] Kimin Lee, Kibok Lee, Honglak Lee, and Jinwoo Shin. A simple unified framework for detecting out-of-distribution samples and adversarial attacks. *NeurIPS*, 2018.

[24] Xin Li and Fuxin Li. Adversarial examples detection in deep networks with convolutional filter statistics. *ICCV*, pages 5775–5783, 2017.

[25] Xingjun Ma, Bo Li, Yisen Wang, Sarah M Erfani, Sudanthi N R Wijewickrema, Michael E Houle, Grant Schoenebeck, Dawn Song, and James Bailey. Characterizing adversarial subspaces using local intrinsic dimensionality. *arXiv:1801.02613*, 2018.

[26] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *ICLR*, 2018.

[27] Dongyu Meng and Hao Chen. Magnet: A two-pronged defense against adversarial examples. In *ACM*, 2017.

[28] Jan Hendrik Metzen, Tim Genewein, Volker Fischer, and Bastian Bischoff. On detecting adversarial perturbations. In *ICLR*, 2017.

[29] Takeru Miyato, Shin-ichi Maeda, Masanori Koyama, Ken Nakae, and Shin Ishii. Distributional smoothing with virtual adversarial training. In *ICLR*, 2015.

[30] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal adversarial perturbations. In *CVPR*, 2017.

[31] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, Pascal Frossard, and Stefano Soatto. Analysis of universal adversarial perturbations. *ArXiv:1705.09554*, 2017.

[32] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: A simple and accurate method to fool deep neural networks. *CVPR*, pages 2574–2582, 2016.

[33] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. 2011.

[34] Nicolas Papernot, Fartash Faghri, Nicholas Carlini, Ian Goodfellow, Reuben Feinman, Alexey Kurakin, Cihang Xie, Yash Sharma, Tom Brown, Aurko Roy, Alexander Matyasko, Vahid Behzadan, Karen Hambardzumyan, Zhishuai Zhang, Yi-Lin Juang, Zhi Li, Ryan Sheatsley, Abhibhav Garg, Jonathan Uesato, Willi Gierke, Yinpeng Dong, David Berthelot, Paul Hendricks, Jonas Rauber, and Rujun Long. Technical report on the cleverhans v2.1.0 adversarial examples library. *arXiv:1610.00768*, 2018.

[35] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami. Distillation as a defense to adversarial perturbations against

deep neural networks. In *IEEE Symposium on Security and Privacy (SP)*, 2016.

[36] Nicolas Papernot and Patrick D McDaniel. Deep k-nearest neighbors: Towards confident, interpretable and robust deep learning. *arXiv:1803.04765*, 2018.

[37] Nicolas Papernot, Patrick D McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. *IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 372–387, 2016.

[38] Andrew Slavin Ross and Finale Doshi-Velez. Improving the adversarial robustness and interpretability of deep neural networks by regularizing their input gradients. In *AAAI*, 2017.

[39] Bita Darvish Rouhani, Mohammad Samragh, Tara Javidi, and Farinaz Koushanfar. Towards safe deep learning: Unsupervised defense against generic adversarial attacks. 2018.

[40] Andras Rozsa, Manuel Gunther, and Terrance E. Boult. Towards robust deep neural networks with bang. In *WACV*, 2018.

[41] Florian Schroff, Dmitry Kalenichenko, and James Philbin. FaceNet: A unified embedding for face recognition and clustering. In *CVPR*, 2015.

[42] Uri Shaham, Yutaro Yamada, and Sahand Negahban. Understanding adversarial training: Increasing local stability of supervised models through robust optimization. *Neurocomputing*, 307:195–204, 2018.

[43] Carl-Johann Simon-Gabriel, Yann Ollivier, Léon Bottou, Bernhard Schölkopf, and David Lopez-Paz. First-order adversarial vulnerability of neural networks and input dimension. In *ICML*, 2019.

[44] Aman Sinha, Hongseok Namkoong, and John Duchi. Certifiable distributional robustness with principled adversarial training. In *ICLR*, 2018.

[45] Chawin Sitawarin and Dávid Wágner. Defending against adversarial examples with k-nearest neighbor. *ArXiv*, abs/1906.09525, 2019.

[46] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *ICLR*, 2014.

[47] Florian Tramer, Nicolas Papernot, Ian Goodfellow, Dan Boneh1, and Patrick McDaniel. The space of transferable adversarial examples. *arXiv:1704.03453*, 2017.

[48] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. Ensemble adversarial training: Attacks and defenses. In *ICLR*, 2018.

[49] Athanasios Voulodimos, Nikolaos Doulamis, Anastasios Doulamis, and Eftychios Protopapadakis. Deep learning for computer vision: A brief review. *Computational Intelligence and Neuroscience*, 2018.

[50] Weilin Xu, David Evans, and Yanjun Qi. Feature squeezing: Detecting adversarial examples in deep neural networks. *arXiv:1704.01155*, 2018.

[51] Ying Zhang, Mohammad Pezeshki, Philemon Brakel, Saizheng Zhang, César Laurent, Yoshua Bengio, and Aaron C Courville. Towards end-to-end speech recognition with deep convolutional neural networks. In *Interspeech*, 2016.