# Probing for Artifacts: Detecting Imagenet Model Evasions

Jeremiah Rounds
jeremiah.rounds@pnnl.gov

Addie Kingsland
addie.kingsland@pnnl.gov

Michael J. Henry
michael.j.henry@pnnl.gov

Kayla R. Duskin
Pacific Northwest National Laboratory
902 Battelle Blvd, Richland, WA 99354
kayla.duskin@pnnl.gov

## Abstract

*While deep learning models have made incredible progress across a variety of machine learning tasks, they remain vulnerable to adversarial examples crafted to fool otherwise trustworthy models. Previous work has proposed examining the internal activation of Imagenet models to detect adversarial examples. Our work expands the scale and scope of previous research by simultaneously probing every activation within an Imagenet model using a novel probe block. This probe block model is trained against multiple adversarial algorithms to create a more robust detector. Parameterization of the probe block and adversarial classification networks that utilize probe block output are examined in an ablation experiment with probes of Resnet-50, Inception-v3 and Xception. Considered adversarial classification networks include examples built with Mobilenet-v2 which is shown to be better than a VGG alternative for detecting adversarial artifacts. Results are compared to logistic regression feature squeezing results, which we suggest is an improvement to feature squeezing.*

## 1. Introductions

Deep Neural Networks (DNNs) have proven to be effective models for a number of challenging tasks, such as computer vision [16, 41], machine translation [5], and speech recognition [3]. However, DNNs are also known to be vulnerable to adversarial perturbations — inputs to the model that have been subtly perturbed from their original state in order to change the output of the model. While often demonstrated in image classification paradigms [48], these adversarial examples have been shown to exist across a variety of machine learning applications and data modalities [8, 30, 52] and in high-impact topics, such as malware detection [28, 45].

Adversarial examples take advantage of the fact that relatively small changes to input can drastically change DNN outputs. Perturbations too small to be detected by the human eye can successfully alter the output of an otherwise highly accurate DNN model. The imperceptibility of adversarial changes to model inputs is cause for concern for institutions that rely on DNN models to make decisions.

Image classification DNN models trained on Imagenet [41], in particular, now have extensive literature documenting this vulnerability[17]. These include Inception, Resnet, Inception-Resnet-v2, Xception, as evaluated in this work[11, 16, 47, 51], but additional Imagenet-based models such as Faster RCNN variants[39], etc. have all been shown to be vulnerable to adversarial attacks.

To date, there have been three primary strategies for mitigating the threat of adversarial examples. The first is to increase model robustness so that a model will still perform reliably in the presence of adversarial input. The second is to inhibit the creation of successful adversarial examples. The third is to train an adversarial detector to be ran in parallel to the model during inference.

Adversarial examples are hypothesized to reflect a lack of robustness in model features[21]. One method to decrease model sensitivity to small changes in pixel values is to utilize lower RGB input dimensionality; however, that has been proven insufficient to remove the threat of adversarial perturbations[9]. Robustness may be addressed directly with feature manipulations or feature matching[37, 53, 56]. Various forms of feature transformations continue to be explored as a possible defense, for example,[38, 46]. Robustness may be built into a model through more elaborate training strategies[24, 50] or efforts to create more generalized models[25]. While these research areas are promising, it remains that there is not yet a general solution or plan that can create adversarially robust DNN models that does not sacrifice performance and is immune to adversarial perturbations[23, 44].

Another method of safeguarding against adversaries is to prevent their creation in the first place. Since many techniques for constructing adversarial examples use the gradient of the model they are attacking, it was hypothesized that concealing or masking the gradient of the model would impede the creation of successful adversaries]. However, due to the transferability of adversarial examples [48], gradient masking has been shown to be defeated by determined adversaries [36].

In addition to the difficulty of building completely robust DNN models, adversarial research is now increasing the ease and efficiency of attacking a model. Recent research has greatly increased throughput[55], increased transferability[22], lowered information used in oracle attacks[2, 10, 20, 49], and restricted image regions of the input to achieve the adversarial goal[31].

The issues of robustness and defensibility have proven extremely challenging for the research community to solve, while more and more successful adversarial input schemes have been demonstrated. This presents a worrisome outlook for operators of machine learning models and justifies continued research into the third approach: detection. Essentially, this third approach is to have super-human perception, so that if a small change in an RGB pixel value is not seen by a human, it is still noticed by an algorithm. Imperceptible perturbations cannot be discovered by the human eye, so they must be discovered by computer examination.

This paper considers a DNN model-based strategy for detecting artifacts of adversarial perturbations to Imagenet-trained models. We create a training corpus for the investigation of adversarial models and suggest a novel detection scheme based on probing a DNN model's activations at multiple stages of the model. We show our method achieves higher detection rates than several alternatives. We also suggest a practical improvement to feature squeezing called Logistic Regression Feature Squeezing.

## 2. Related Work

Related work in detection may be divided into two broad categories: those methods that look for characteristic artifacts of adversarial perturbation in an input as a classification problem[26, 32, 33] and those that examine model response to further perturbations of the input[18, 19, 29, 54]. The two catagories are commonly called *artifact detection* and *feature squeezing* based on early work in the subject.

Ma *et al*. [32] and Katzir [26] use the distribution of activations within a model to perform artifact detection. Principal component analysis (PCA) may be used on model activations to uncorrelate features [26]. Metzen originally suggested using activation within a DNN as input into another DNN for artifact detection[33] . In [33], VGG16 model is probed for unusual activation values by routing one of its hidden layers to a Convolution Neural Network classifier. A limitation of [33] is that it only considered probing the model at a single depth. The depth at which the model activation was passed to the classifier was chosen via grid search optimizing accuracy of detection. In our work, we expand upon this artifact detection technique by probing the model at multiple depths and treating the weights assigned to the probed activations as learnable parameters in the probe model. In contrast to [33], this new model attaches at each hidden layer of the model and uses a dimension-reducing fully connected layer with ReLU activation to make computation feasible.

Alternatives to direct classification of artifacts include looking at the response of a model to various transformations of the input. Feature squeezing detects adversarial examples by using transformations that reduce the number of free bits utilized by the input image[54]. Logit or feature differences between paired or noised images can also be useful[19, 29]. It has been hypothesized that there is a sub-manifold explanation for examining feature response to perturbation[18]. Another effort in this family varies the model, instead of the input, and uses rules based on feature response to different model weights[40].

Finally, no review of related work would be complete without stating that a determined researcher, with knowledge of detection techniques, may craft adversarial inputs that are evasive to both a defended model and the detection algorithm[6].

## 3. Adversarial Algorithms

In this section, we provide a summary of the adversarial attack algorithms referenced in our experiments. All algorithms herein create perturbations on floating-point representations of RGB images (96 bits per pixel). Many novel adversarial algorithms exist and new methods are proposed frequently; thus, while we selected a representative subsample of important algorithms, it is by no means exhaustive.

Input images are scaled down (or occasionally up) in height and width to target model resolution with nearest neighbor pixel values in 24-bit RGB and then model preprocessing is applied to the input. An adversarial image is only considered a successful adversary if it can saved as a 24-bit RGB image, with all model preprocessing inverted, preprocessed for a target model again, and remain evasive in a final check. The average size of successful evasive perturbations from an algorithm is measured with the average $L_1$-norm of a perturbation on the 24-bit RGB images:

$$L_1\text{-norm} = \frac{1}{S}\frac{1}{C}\sum_{i}\sum_{(h,w,d)} |x'_{i,(h,w,d)} - x_{i,(h,w,d)}| \quad (1)$$

where $x'_i$ and $x_i$ are adversarial and non-adversarial pairs of images, $i$ indexes $S$ successful adversarial images and $(h, w, d)$ indexes $C$ color channel values within those images.

Many algorithms are parameterized with $\epsilon$ as a parameter that controls the scale of an added perturbation. For

| Algorithm | | Iv3 | R50 | Xcep | IRv2 | VGG16 | 10-Iv3 | 10-VGG16 |
|---|---|---|---|---|---|---|---|---|
| FGSM | (Recall in Table 5) | 1.960 | 1.961 | 1.959 | 1.959 | 1.963 | 1.963 | 1.969 |
| CWL2 | (Recall in Table 6) | 1.655 | 1.602 | 1.594 | 1.840 | 1.559 | 2.690 | 2.459 |
| DeepFool | (Recall in Table 7) | 0.089 | 0.024 | 0.100 | 0.183 | 0.054 | 0.226 | 0.349 |

Table 1. Mean $L_1$-norm of a perturbation on a 24-bit RGB image for different adversarial algorithms (rows) crafted against classification models (columns). Adversarial images are cast to 1 byte per color channel (24-bit), re-transformed for target model, filtered for successful evasion, and a mean $L_1$-norm over successful distortions over the 1-byte color channel difference is reported. FGSM is parameterized $k = 2.0$. CWL2 has learning rate $= 0.001$ and confidence $= 0.10$. DeepFool is modified and parameterized with confidence value $= 0.10$. From top to bottom adversarial artifacts are progressively more difficult to detect. Abbrevations of columns are Inception v3(Iv3), Resnet50 (R50), Xception(Xcep), Inception-Resnet-V2 (IRv2), VGG16, 10-class Inception v3 (10-Iv3), 10-class VGG16 (10-VGG16). Resnet-50 DeepFool produces particularly small perturbations.

consistency, we express $\epsilon$ as $k$ of 255 possible RGB values in a unsigned byte 24-bit image, where $k$ is an integer, but $\epsilon$ is scaled to the dynamic range of the target model input in application. For example with Inception and Inception Resnet models, input is scaled between $[-1, 1]$, so

$$\epsilon = \frac{2}{255}k, \tag{2}$$

but we report $k$. For Resnet50, $\epsilon$ is simply the $k$. This adjustment explains why the FGSM $L_1$-norms reported in Table 1 are nearly constant across models.

**Fast Gradient Sign Method (FGSM)**: FGSM is a non-iterative, non-optimized, fast adversary [14]. An image is perturbed toward a classification boundary based on gradient.

$$\boldsymbol{x} = \boldsymbol{x} + \epsilon\, \textbf{sign}(\nabla J(L, \boldsymbol{x})) \tag{3}$$

where $\boldsymbol{x}$ is model input and $\nabla J(L, \boldsymbol{x}))$ is the gradient of the loss with respect to the input. In a high-dimensional space, $\textbf{sign}(\nabla J(L, \boldsymbol{x}))$ is an approximate direction of a decision boundary. This method, while fast, results in a characteristically splotchy distortion on most successful attacks.

**Carlini and Wagner L2 (CWL2)**: CWL2 crafts adversarial perturbations by optimizing a loss function that favors target model evasion [4, 6, 7, 27]. With CWL2, if a researcher can express a differentiable loss function that achieves adversarial goals against a defense, an optimizer (such as Adam) can find perturbations that defeat defense[4]. For our experiments, we use a reference implementation from the cleverhans project to create CWL2 adversaries[35]. CWL2 has a confidence hyper-parameter that specifies how much greater the logits should be in the adversarial label than the original label. Confidence is $0.1$ for our experiments. In its untargeted form, CWL2 produces adversarial images that are usually invisible to the human eye, having perturbations averaging less than $2/255$ RGB change per color pixel in 24-bit images.

**Modified DeepFool**: DeepFool is an iterative algorithm that minimizes the $L_2$ norm of a perturbation necessary to reach a decision boundary[34]. We modify the DeepFool direction decision and stopping criteria by inserting a confidence parameter.

As previously stated, adversarial algorithms perform perturbations on floating point RGB images, which assign 96 bits per pixel. However, in practice all final perturbed images are saved as unsigned integer RGBs, which assign 24-bits per pixel as is necessary for saving the image to disk with common file formats. This is important because not all adversarial algorithms persist to the unsigned integer representation of the images.

In its original form, DeepFool will often craft a mathematical adversary at 94-bit precision (32-bits per color channel) that do not result in any differences for 24-bit RGB images ($L_1$-norm = 0) after casting from floating point to unsigned byte. The overshoot parameter recommended by [34] is not usually helpful in avoiding this issue on Imagenet models. To overcome this early stopping, we add an additional hyper-parameter to the model logits of the class that DeepFool is evading. This parameter changes DeepFool's selected iterative direction, step size, and willingness to stop. Perturbations remain very small. We selected confidence$= 0.1$, and describe the use of this parameter in the supplement, the average $L_1$-norm of a perturbation on a 24-bit RGB image is approximately $0.10$ (Table 1).

## 4. Detection by Model Probing

Our model detects adversarial artifacts by probing a *base model* for unusual activations in hidden layers. Experimental base models are Resnet50, Inception-v3 and Xception models, and their weights are never unlocked for any training. We view the entirety of internal activation of base models to be extremely high-dimensional feature vectors, as such all layer outputs besides batch normalizations within the base model are used as feature vectors to be fed to a classifier network.

A base model's layers emit activation tensors $\boldsymbol{v} = \boldsymbol{v_1}, ..., \boldsymbol{v_N}$ where $N$ is the number of layers in the model (minus the batch normalization layers). Due to the presence of topologies like residual networks, all considered target models have multiple activations that

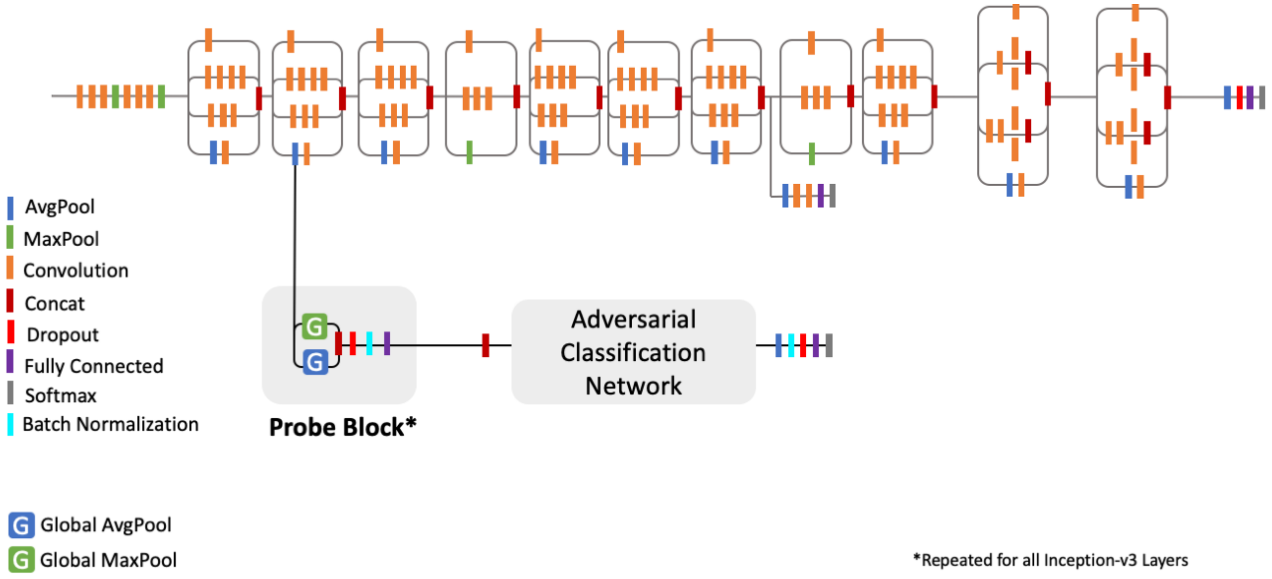

Figure 1. Illustration of probe blocks on Inception-v3. All tensor output in Inception-v3 is routed through a probe block with weights specific to the probed layer. Probe output is concatenated and routed through a final classification unit for artifact detection. The size of probe block output and the form of the classification unit is varied in ablation experiment.

| Probe Layer Index | Input Layer Index | Operation | Output Shape |
|---|---|---|---|
| 1 | Varied Probe $v_i$ | Input | (h,w,d) |
| 2 | 1 | GlobalMaxPool | (d) |
| 3 | 1 | GlobalMeanPool | (d) |
| 4 | 2,3 | Concatenate | (2d) |
| 5 | 4 | Dense | K |
| 6 | 5 | BatchNormalization | K |
| 7 | 6 | ReLU | K |

Table 2. $f_i(\boldsymbol{v_i}|\theta_i, K)$ probe block defined in terms of Keras layer operations. K output dimension varies in experiment while input dimensions (h=height,w=width,d=depth) are determined by the probed layer.

occur at the same depth. In that case, the order in which the tensors are presented in $\boldsymbol{v}$ is ad-hoc, but always the same.

We describe the first layers that operate on elements of $\boldsymbol{v}$ as a *probe block*, which is a functional block of operations that takes as input the activation tensor of a model's $i$th hidden layer, $\boldsymbol{v_i}$, and produces an output of length $K$, where $K$ is chosen by grid search experiment. We define the $i$th probe output, $\boldsymbol{u_i}$, as $\boldsymbol{u_i} = f_i(\boldsymbol{v_i}|\theta_i)$ where $\theta_i$ is unique weights learned for each probe block. The internal computation of a probe block is described in Table 2. One task of the probe block is to resize output from all feature map activations within a base model to one standard dimension to be used downstream in the adversary detection classifier.

| Name | Adversarial Classification Unit |
|---|---|
| Model-Probe-16-FC1 | Dense(512, activation=ReLU)<br>BatchNormalization()<br>DropOut(rate=0.50)<br>Dense(2, activation=ReLU)<br>Softmax() |
| Model -Probe-32-MobileNet-v2 | Reshape(N, K, 1)<br>Conv2D(filters=3, shape=(1,1))<br>MobileNet-v2($\alpha = 1.0$)<br>Dense(512, activation=ReLU)<br>BatchNormalization()<br>DropOut(rate=0.50)<br>Dense(2, activation=ReLU)<br>Softmax() |

Table 3. Subsequent classifier construction on $\boldsymbol{r}$ probed features constructed by concatenating $f_i(\boldsymbol{v_i}|\theta_i, K)$ from Table 2. Model-Probe-16-FC models are the use of probe output into fully connected layers for adversarial classification. Model-Probe-32-MobileNet-v2 is the use of probe output into an unlocked MobileNet-v2 for adversarial classification.

We employ a global pooling operation as the first layer of the probe block because that operator is well-defined for any shape input, and the probe block ends with a dense layer with ReLU activation. The probe block's construction and optimization make it a dimension reduction operator.

The outputs of the probe blocks are next reshaped into one tensor to be used as input for the *classification network* that classifies the original input as adversarial or non-adversarial. Depending on classification network design, the $\boldsymbol{u} = \boldsymbol{u_1}, ..., \boldsymbol{u_N}$ probe outputs are either concatenated into an $N * K$ length vector for fully connected operators or reshaped into an $(N, K, 1)$ tensor for convolutional operators. The concatenation of all probe block outputs, called $\boldsymbol{r}$, is then passed to the classification network. Possible architectures for the classification network are described in Table 3 and include options for either a fully connected (FC1) or convolutional network architecture, Mobilenet-v2($\alpha = 1.0$) [42]. For the FC1 architecture $K$ is 16 while for the Mobilenet-v2 architecture $K$ is 32.

During training, the trained weights of the base model are kept frozen and only the weights of the probe blocks and classification network are trained. The loss function is categorical cross-entropy for the binary classification problem where 1 indicates the input is adversarial perturbed and 0 indicated the input is not adversarial. The probe block and classification weights are optimized with Adam optimizer using mini-batch training (batch size = 32)[27] in Keras[12] with a Tensorflow backend[1]. The learning rate is set to 0.001 and decayed by multiplying by 0.75 every time there are four epochs without validation loss improvement.

# 5. Experiments

## 5.1. Data

We create a corpus of training, validation, and test data in which to search for adversarial artifacts. This data corpus contains the ten Imagenet classes selected at random by [33] for experimentation: *palace, joystick, bee, dugong, cardigan, modem, confectionery, valley, Persian cat*, and *stone wall*. The resulting 12849 Imagenet images are divided at random into training, validation (development), and test sets with an 80%, 10%, 10% split.

Each adversarial algorithm we introduced in Section 3 requires knowledge of the target model in order to create adversaries. Therefore, to create each adversarial example necessary for training and testing our model, we must define the following triplet: source data, target model, and adversarial algorithm. Given the above subset of Imagenet as the source data, we select several mainstream models from the Imagenet family of image classifier as target models (Inception-v3, Resnet50, Xception, VGG16 [43] and Inception-Resnet-v2).

Additionally we introduce 10-class versions of Inception-v3 and VGG16 target models. The 10-class models are modified to only emit probabilities for the classes within our training corpus by slicing the logit layer down to only the 10 relevant classes before applying the softmax activation. By creating the 10-class models we have two types of non-targeted Inception adversaries in the training data: a type that builds adversaries on the 1000-class Imagenet models, and a type of adversary that builds perturbations on the 10-class Imagenet models. Adversarial images built against the 10-class variant result in larger perturbations on average for algorithms with stopping criteria (Table 1; DeepFool and CWL2).

| | FPR | FGSM | CWL2 | DeepFool |
|---|---|---|---|---|
| xception-probe-16 FC1 | 0.01 | 0.92 | 0.43 | 0.13 |
| xception-probe-32 mobilenet-v2 | 0.01 | 0.92 | 0.46 | 0.18 |
| xception-probe-16 FC1 | 0.05 | 0.99 | 0.56 | 0.32 |
| xception-probe-32 mobilenet-v2 | 0.05 | 0.98 | 0.58 | 0.40 |
| xception-probe-16 FC1 | 0.10 | 1.00 | 0.65 | 0.45 |
| xception-probe-32 mobilenet-v2 | 0.10 | 0.99 | 0.65 | 0.50 |

Table 4. Recall results for the Xception probe model. The $K = 32$ probe into a Mobilenet is more powerful than the lower capacity models for diffiult-to-detect perturbations such as DeepFool, which were, on average, an order of magnitude smaller than CWL2.

We created adversaries using FGSM, CWL2, and the modified version of DeepFool discussed in Section 3. We experimented with, but ultimately abandoned, probe models specialized to detect one type of adversary at a time. Specializing weights to a single adversarial algorithm has been discussed in [33], and elsewhere. We found that specialized detectors show higher recall at lower false-positive rates than detectors trained against a variety of adversariesr. However, we favor more generalized models which may perform better under circumstances where the type of adversary is unknown. To promote robustness, our models were trained simultaneously against multiple adversaries and so have to learn perturbation artifacts of everything in our training data.

## 5.2. Ablation Study

Several model design decisions were considered in an ablation study with the two most successful probe models highlighted in Table 4. Supplemental results indicate there is an advantage to using more probe units from the base model but no advantage to using a second fully connected layer in the classification network. The higher capacity $K = 32$ probe into Mobilenet-v2 has a consistent advantage over the FC probe in recall in the difficult detections of CWL2 and DeepFool. Based on this ablation experiment, the rest of results in this section focus on probe models that are inputs to Mobilenet.

## 5.3. Sensitivity to Adversarial Model Architecture

We examine the relative importance of the probed base model to adversarial detection against three adversarial algo-

| Model | Iv3 | R50 | Xcep | IRv2 | VGG16 | 10-Iv3 | 10-VGG16 |
|---|---|---|---|---|---|---|---|
| inception-v3-probe-32-mobilenet-v2 | **0.98** | 0.65 | 0.68 | 0.71 | 0.64 | **0.96** | 0.67 |
| resnet50-probe-32-mobilenet-v2 | 0.62 | **0.83** | 0.62 | 0.72 | 0.71 | 0.61 | 0.73 |
| xception-probe-32-mobilenet-v2 | 0.83 | 0.79 | **0.98** | **0.86** | **0.74** | 0.85 | **0.77** |
| Metzen (Our Retrain) | 0.41 | 0.52 | 0.39 | 0.45 | 0.69 | 0.38 | 0.69 |
| LRFS Inception | 0.23 | - | - | - | - | - | - |
| LRFS Xception | - | - | 0.21 | - | - | - | - |

Table 5. FGSM ($k = 2.0$) recall for three types of probe detectors at FPR=$0.05$. Column indicates model adversary used in generating perturbation. Abbrevations of columns are Inception v3(Iv3), Resnet50 (R50), Xception(Xcep), Inception-Resnet-V2 (IRv2), VGG16, 10-class Inception v3 (10-Iv3), 10-class VGG16 (10-VGG16). Reported values are recall. Bold indicates best detection on an adversarial model. Xception generalizes to the best overall detector, but performance can be targeted as needed. Metzen is a VGG16 probe described in related literature and considered further in a comparison section. Logistic Regression feature squeezing (LRFS) was not presented with black-box scenarios in the interest of shortening run-time. LRFS Inception was trained on and applied to Inception-v3 features. LRFS Xception was trained on and applied to Xception features.

| Model | Iv3 | R50 | Xcep | IRv2 | VGG16 | 10-Iv3 | 10-VGG16 |
|---|---|---|---|---|---|---|---|
| inception-v3-probe-32-mobilenet-v2 | 0.53 | 0.25 | 0.25 | 0.35 | 0.24 | **0.91** | 0.58 |
| resnet50-probe-32-mobilenet-v2 | 0.26 | **0.32** | 0.24 | 0.36 | 0.27 | 0.63 | 0.60 |
| xception-probe-32-mobilenet-v2 | 0.38 | **0.32** | **0.58** | **0.51** | **0.32** | 0.79 | **0.67** |
| Metzen (Our Retrain) | 0.18 | 0.18 | 0.16 | 0.24 | 0.26 | 0.49 | 0.54 |
| LRFS Inception | **0.59** | - | - | - | - | - | - |
| LRFS Xception | - | - | 0.51 | - | - | - | - |

Table 6. CWL2 (confidence=0.1) recall for three types of probe detectors at FPR=$0.05$. Columns are as in Table 5.

| Model | Iv3 | R50 | Xcep | IRv2 | VGG16 | 10-Iv3 | 10-VGG16 |
|---|---|---|---|---|---|---|---|
| inception_v3-probe-32-mobilenet-v2 | 0.49 | 0.06 | 0.08 | 0.13 | 0.07 | **0.82** | 0.19 |
| resnet50-probe-32-mobilene-v2 | 0.07 | 0.06 | 0.06 | 0.09 | 0.06 | 0.14 | 0.17 |
| xception-probe-32-mobilenet-v2 | 0.21 | 0.06 | 0.38 | **0.21** | 0.07 | 0.44 | **0.28** |
| Metzen (Our Retrain) | 0.07 | 0.05 | 0.05 | 0.08 | 0.06 | 0.12 | 0.22 |
| LRFS Inception | **0.68** | - | - | - | - | - | - |
| LRFS Xception | - | - | **0.68** | - | - | - | - |

Table 7. Modified DeepFool (confidence=0.1) for three types of probe detectors at FPR=$0.05$. Columns are as in Table 5. Resnet50 had no detection rates significantly above the controlled FPR.

rithms — FGSM (Table 5), CWL2 (Table 6), and DeepFool (Table 7). Recall results are reported with thresholds controlled at a false postive rate (FPR)= $0.05$. Results are reported for each adversaries generated by each adversarial model in our corpus (columns). The detector base model varies as one of: Xception, Inception-v3, Resnet50 (rows).

As shown in Table 6, we find that each detector model displays higher recall when the target model used to create the adversaries is the same as the architecture used in the detector's base model. All detectors are trained using the same dataset and training regime, making it clear that the matching architecture produces the increased performance. This is an intuitive, but not easily explained result because each model has the same training set and approximately the same capacity, yet persists in holding a performance advantage in their own baseline model.

The conclusion from this experiment is that if it is possible to anticipate the architecture used to generate adversaries, better performance can be expected by using the same or similar architecture as the base model in the detector.

## 5.4. Comparison to Other Detectors

We compare probe detection to original work in artifact detection and feature squeezing.

### 5.4.1 Artifact Detection

The most directly related artifact detection model in literature is from Metzen [33]. In Metzen, a VGG16 model is probed with a convolutional model at one specific activation in VGG16. For comparison to our work, we retrain a Metzen *et al*. VGG16 model on this experimental corpus, but according to published instructions from the author. Therefore, reported recalls here are different from original author. The task in this work is harder: these artifact detectors have to simultaneously detect artifacts across a variety of model sources and algorithms. In every case, our proposed alternative shows greater recall than original VGG artifact detection model. Based on these results, the higher capacity "xception-probe 32-mobilenet-v2" introduced in this work is the superior artifact detector.

### 5.4.2 Feature Squeezing

In feature squeezing[54](FS), Imagenet model inputs are manipulated via smoother-like functions that reduce, in some way, the pixel degrees of freedom. For example, one feature squeezer may reduce the number of bits available to a pixel by forcing low-order bytes to be zero, or another feature squeezer may run a local median smoother across an image. The intent is to reduce the complexity of the input in a known manner. In contrast to artifact detection, which detects signatures for adversaries directly from model input, feature squeezing uses the resulting change to model output after applying a squeezer to the input as the signature of an adversary.

Feature squeezers are used as sets or as ensembles. Each of the squeezer operations induces a change in the model probability output, $f(\boldsymbol{x}) = \boldsymbol{p}$, where $f(\boldsymbol{x})$ is a CNN. We calculate the $L_1$ distance between the original model output and new model output as

$$D(f(\boldsymbol{x}), f(s(\boldsymbol{x})) = \sum_k |f_k(\boldsymbol{x}) - f_k(s(\boldsymbol{x})|, \qquad (4)$$

where $\boldsymbol{x}$ is an image input, $s(x)$ is a squeezer on image input, and $k$ indexes Imagenet class probabilities. We denote the particular difference of probability output created by a squeezer function, $s(\boldsymbol{x'})$, as $D_s(\boldsymbol{x'})$.

A critical conjecture for feature squeezing is that there are squeezer functions such that the distributions of $L_1$ distances tend to be higher for adversarial modifications than for images that have not undergone adversarial perturbation. However, appropriate $s(\boldsymbol{x})$ may vary by subject matter and may not be robust to small changes of images or adversaries; that is, with one image set, a particular squeezer is good, but in another, it may not be useful at all. In practice, classification of an image as adversarial or non-adversarial is based on a threshold rule for squeezer distance, $D_s(\boldsymbol{x}) > t$, where $t$ is a chosen threshold. [54] suggests the use of a maximum statistic

$$M(\boldsymbol{x}) = \max(D_1(\boldsymbol{x}), ..., D_n(\boldsymbol{x})). \qquad (5)$$

This is the maximum of all $L_1$ distances induced by an ensemble of squeezers and defines a decision rule for adversarial input, as $M(\boldsymbol{x}) > t_{0.05}$ where $t_{0.05}$ is a threshold that controls false-positive rate at 0.05 on non-adversarial input.

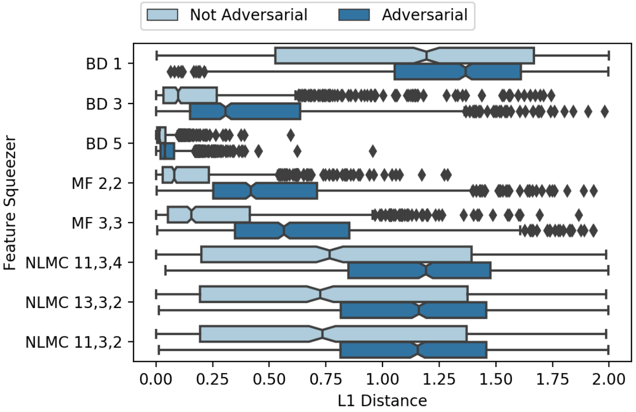

Figure 2. Distribution of all feature squeezers used in [54] on a 1000 image random sample of Xception adversaries. BD 1, 3, 5 are *bit-depth 1, 3, and 5* feature squeezers, respectively. MF 2,2 and 3,3 are *Median Filter* on a 2 by 2 and 3 by 3 moving window. NLMC are parameterizations to a non-local Guassian smoothing algorithm. [54] recommends the maximum of BD 5, MF 2,2 and NLMC 13,3,2 for detecting adversarial modifications. From these boxplots, BD 3, MF 2,2 and MF 3,3 have the least overlap between adversarial and non-adversarial, thus they appear to be more useful feature values than NLMC values. However, if we were to use [54]'s approach, they would rarely factor into the decision boundary because NLMC is so often larger. MF 2,2 itself appears to be a startlingly efficient feature. A simple rescale of the features does not make the maximum much better.

[54] recommends the maximum of bit-depth 5 (BD 5), median filtering 2,2 (MF 2,2), and a non-local means smoother (NLMC 11, 3, 4), which we applied to our own adversarial images corpus. Overall, the distributions of the squeezer $L_1$ distances appears to be useful for detecting adversaries (Figure 2). However, we found that using the simple maximum over those three squeezers did not fairly represent the method as it performed so poorly we could not report the results. Reviewing the probability $L_1$ distances, $D(\boldsymbol{x})$, showed the maximum of squeezer features unnecessarily favored a poor-performing NLMC feature. To avoid the adverse effect of using the maximum operator, we introduce a modification we call *Logistic Regression Feature Squeezing* (LRFS).

The original FS decision rule can be written as:

$$I(\boldsymbol{x}) = (\alpha_{0.05} + \max(D_1(\boldsymbol{x}), ..., D_n(\boldsymbol{x})) > 0 \qquad (6)$$

where $\alpha$ is a false positive rate controlling offset, $\alpha = -t_{0.05}$ from above. The use of a maximum operator over the $D_k(\boldsymbol{x})$ values overemphasizes the conjecture that a squeezer results in extreme values *only* on adversarial images. In contrast, in LRFS we train a logistic regression model to assign weights to each $D_k(\boldsymbol{x})$ and then base the classification decision off of the weighted sum of the $D_k(\boldsymbol{x})$. We express this updated decision rule as

$$I'(\boldsymbol{x}) = (\alpha'_{0.05} + \hat{\beta}_1 D_1(\boldsymbol{x}) + ...\hat{\beta}_n D_n(\boldsymbol{x})) > 0. \qquad (7)$$

The false positive rate controlling the shift of the classification threshold is estimated using the training data.

The log-odds of an image being adversarial are modeled as

$$\mathrm{LO}(\boldsymbol{x}) = \log(p(\boldsymbol{x};\boldsymbol{\beta})/(1-p(\boldsymbol{x};\boldsymbol{\beta}))) = \beta_0 + \sum_j \beta_j D_j(\boldsymbol{x}), \tag{8}$$

where $LO(\boldsymbol{x})$ is log-odds and $p(\boldsymbol{x};\boldsymbol{\beta})$ is the probability of being adversarial in a balanced training set [15]. $\hat{\beta}_1, ..., \hat{\beta}_n$ are estimates that minimize the objective function;

$$\begin{aligned} l(\boldsymbol{x};\boldsymbol{\beta}) = \\ & -\sum_i [y_i \log(p(\boldsymbol{x_i};\boldsymbol{\beta})) + (1-y_i)\log(1-p(\boldsymbol{x_i};\boldsymbol{\beta})] \\ & + c\sum_j \beta_j^2, \end{aligned} \tag{9}$$

where $p(\boldsymbol{x_i};\boldsymbol{\beta}) = 1 - \exp(-\beta_0 - \sum_j \beta_j D_j(\boldsymbol{x_i}))$, and $c$ is a penalty constant.

Unlike with the maximum operator, this log-linear optimization may find $\beta_j$ near 0 when a feature is large but unimportant, or give features relative importance. The maximum operator cannot ignore a feature that is unimportant, which can be quite bad if an included squeezer is not particularly well-suited to the data at hand.

#### 5.4.3 Logistic Regression Feature Squeezing Results

We trained two LRFS models for comparisons. One was trained on Inception-v3 adversaries and the other was trained on Xception adversaries. Each model estimated coefficients with an $L_2$-penalized objective function. The penalty value $c$ is selected by grid search in a 10-fold cross-validation on the training data.

For Inception-v3 the squeezer coefficients are: $LO(\boldsymbol{x}) = \alpha'_{0.05} + 1.19D_1(\boldsymbol{x}) + 3.69D_2(\boldsymbol{x}) + 0.61D_3(\boldsymbol{x})$, where $LO(\boldsymbol{x})$ is the log-odds of an image being adversarial and $D_1$, $D_2$, and $D_3$ are the $L_1$ distances in model output incurred by bit-depth 5 (BD 5), median filtering 2,2 (MF 2,2), and a non-local means smoother (NLMC 11, 3, 4) squeezer functions, respectively. A final LRFS decision rule that controlled FPR at 0.05 is $I'(\boldsymbol{x}) = -3.25 + 1.19D_1(\boldsymbol{x}) + 3.69D_2(\boldsymbol{x}) + 0.61D_3(\boldsymbol{x}) > 0$. The logistic regression emphasizes the distribution of the median smoother over non-local means which is consistent with the gestalt of Figure 2. There were no measured advantages to scaling or using PCA on this feature set, but this method may theoretically be extended to many more squeezers, and then dimension reduction may become more valuable. We only used the three recommended squeezers from [54] because we wanted the minimum adaption to make FS work as LRFS. For the Xception model the decision rule is similar, but with even more emphasis placed on the MF 2,2 feature: $I'(\boldsymbol{x}) = -3.73 + 0.12D_1(\boldsymbol{x}) + 6.43D_2(\boldsymbol{x}) + 0.55D_3(\boldsymbol{x}) > 0$

The resulting recall performance of the LRFS models are reported in Tables 5, 6, and 7. From this work we corroborate two generalities from [54]. First, feature squeezing is very poor at detecting FGSM adversaries, and second, compared to artifact detection, feature squeezing is very good at detecting DeepFool adversaries. What is interesting about that statement is that FGSM is typically an easier artifact to detect, and modified DeepFool makes very small perturbations that are difficult to detect as artifacts. This may indicate that feature squeezing and artifact detection may be best deployed as complementary systems rather than as competitive systems.

## 6. Conclusions

In this work we found that probing base models at multiple depths to detect adversarial artifacts, along with training adversarial detectors on a variety of adversarial attacks produces effective and reasonably generalizable detection models. We also illustrate the connection between creating and detecting adversarial examples using the same model architecture and show that some presumed knowledge of potential adversaries will always be valuable (Section 5.3).

We suggest a generalized improvement to the feature squeezing detection method: to view the squeezers $L_1$ distance as a feature vector in a classifier, in this case, Logistic Regression. The use of artifact detection and Logistic Regression Feature Squeezing are complementary because each has scenarios in which they are the strongest detector (Tables 5,6,7). When an adversarial perturbation is too small, for example, average perturbation $L_1$-norm $\approx 0.10$, switching to a feature model response, rather than input inspection, makes intuitive sense.

We have observed artifact detection is dependent on training corpus. Future work is to use techniques such as Domain Adversarial Training to reduce that dependence[13]. Also, the approaches demonstrated through this work could be combined by integrating the feature squeezing inputs into the probe model artifact detection via concatenating probe activations from each squeezed input. These new research directions would be an exciting continuation of probing for artifacts.

## References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[2] Moustafa Alzantot, Yash Sharma, Supriyo Chakraborty, Huan Zhang, Cho-Jui Hsieh, and Mani B Srivastava. Genattack: Practical black-box attacks with gradient-free optimization.

In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1111–1119. ACM, 2019.

[3] Dario Amodei, Rishita Anubhai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Jingdong Chen, Mike Chrzanowski, Adam Coates, Greg Diamos, et al. End to end speech recognition in english and mandarin. In *International Conference on Machine Learning (ICML)*, 2016.

[4] Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. *arXiv preprint arXiv:1802.00420*, 2018.

[5] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[6] Nicholas Carlini and David Wagner. Adversarial examples are not easily detected: Bypassing ten detection methods. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pages 3–14. ACM, 2017.

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

[8] Nicholas Carlini and David Wagner. Audio adversarial examples: Targeted attacks on speech-to-text. In *2018 IEEE Security and Privacy Workshops (SPW)*, pages 1–7. IEEE, 2018.

[9] Jiefeng Chen, Xi Wu, Vaibhav Rastogi, Yingyu Liang, and Somesh Jha. Towards understanding limitations of pixel discretization against adversarial attacks. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 480–495. IEEE, 2019.

[10] Minhao Cheng, Thong Le, Pin-Yu Chen, Jinfeng Yi, Huan Zhang, and Cho-Jui Hsieh. Query-efficient hard-label black-box attack: An optimization-based approach. *arXiv preprint arXiv:1807.04457*, 2018.

[11] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1251–1258, 2017.

[12] François Chollet et al. Keras. `https://keras.io`, 2015.

[13] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks. *The Journal of Machine Learning Research*, 17(1):2096–2030, 2016.

[14] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations (ICLR)*, 2015.

[15] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media, 2009.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[17] Dan Hendrycks, Kevin Zhao, Steven Basart, Jacob Steinhardt, and Dawn Song. Natural adversarial examples. *arXiv preprint arXiv:1907.07174*, 2019.

[18] Shengyuan Hu, Tao Yu, Chuan Guo, Wei-Lun Chao, and Kilian Q Weinberger. A new defense against adversarial images: Turning a weakness into a strength. In *Advances in Neural Information Processing Systems*, pages 1633–1644, 2019.

[19] Bo Huang, Yi Wang, and Wei Wang. Model-agnostic adversarial detection by random perturbations. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pages 4689–4696. AAAI Press, 2019.

[20] Andrew Ilyas, Logan Engstrom, Anish Athalye, and Jessy Lin. Black-box adversarial attacks with limited queries and information. *arXiv preprint arXiv:1804.08598*, 2018.

[21] Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Logan Engstrom, Brandon Tran, and Aleksander Madry. Adversarial examples are not bugs, they are features. *arXiv preprint arXiv:1905.02175*, 2019.

[22] Nathan Inkawhich, Wei Wen, Hai Helen Li, and Yiran Chen. Feature space perturbations yield more transferable adversarial examples. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7066–7074, 2019.

[23] Saumya Jetley, Nicholas Lord, and Philip Torr. With friends like these, who needs adversaries? In *Advances in Neural Information Processing Systems*, pages 10749–10759, 2018.

[24] Guoqing Jin, Shiwei Shen, Dongming Zhang, Feng Dai, and Yongdong Zhang. Ape-gan: Adversarial perturbation elimination with gan. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3842–3846. IEEE, 2019.

[25] Harini Kannan, Alexey Kurakin, and Ian Goodfellow. Adversarial logit pairing. *arXiv preprint arXiv:1803.06373*, 2018.

[26] Ziv Katzir and Yuval Elovici. Detecting adversarial perturbations through spatial behavior in activation spaces. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–9. IEEE, 2019.

[27] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[28] Bojan Kolosnjaji, Ambra Demontis, Battista Biggio, Davide Maiorca, Giorgio Giacinto, Claudia Eckert, and Fabio Roli. Adversarial malware binaries: Evading deep learning for malware detection in executables. In *2018 26th European Signal Processing Conference (EUSIPCO)*, pages 533–537. IEEE, 2018.

[29] Jinkyu Koo, Michael Roth, and Saurabh Bagchi. Hawkeye: Adversarial example detector for deep neural networks. *arXiv preprint arXiv:1909.09938*, 2019.

[30] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial examples in the physical world. 2016.

[31] Hyun Kwon, Hyunsoo Yoon, and Daeseon Choi. Restricted evasion attack: Generation of restricted-area adversarial example. *IEEE Access*, 7:60908–60919, 2019.

[32] Shiqing Ma, Yingqi Liu, Guanhong Tao, Wen-Chuan Lee, and Xiangyu Zhang. Nic: Detecting adversarial samples with neural network invariant checking. In *NDSS*, 2019.

[33] Jan Hendrik Metzen, Tim Genewein, Volker Fischer, and Bastian Bischoff. On detecting adversarial perturbations.

*arXiv preprint arXiv:1702.04267*, 2017.

[34] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2574–2582, 2016.

[35] Nicolas Papernot, Ian Goodfellow, Ryan Sheatsley, Reuben Feinman, and Patrick McDaniel. cleverhans v2. 0.0: an adversarial machine learning library. *arXiv preprint arXiv:1610.00768*, 10, 2016.

[36] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia conference on computer and communications security*, pages 506–519, 2017.

[37] Chongli Qin, James Martens, Sven Gowal, Dilip Krishnan, Krishnamurthy Dvijotham, Alhussein Fawzi, Soham De, Robert Stanforth, and Pushmeet Kohli. Adversarial robustness through local linearization. In *Advances in Neural Information Processing Systems*, pages 13824–13833, 2019.

[38] Edward Raff, Jared Sylvester, Steven Forsyth, and Mark McLean. Barrage of random transforms for adversarially robust defense. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6528–6537, 2019.

[39] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[40] Yibin Ruan and Jiazhu Dai. Twinnet: A double sub-network framework for detecting universal adversarial perturbations. *Future Internet*, 10(3):26, 2018.

[41] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[42] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018.

[43] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[44] Dong Su, Huan Zhang, Hongge Chen, Jinfeng Yi, Pin-Yu Chen, and Yupeng Gao. Is robustness the cost of accuracy?–a comprehensive study on the robustness of 18 deep image classification models. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 631–648, 2018.

[45] Octavian Suciu, Scott E Coull, and Jeffrey Johns. Exploring adversarial examples in malware detection. In *2019 IEEE Security and Privacy Workshops (SPW)*, pages 8–14. IEEE, 2019.

[46] Bo Sun, Nian-hsuan Tsai, Fangchen Liu, Ronald Yu, and Hao Su. Adversarial defense by stratified convolutional sparse coding. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 11447–11456, 2019.

[47] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

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

[49] Chun-Chen Tu, Paishun Ting, Pin-Yu Chen, Sijia Liu, Huan Zhang, Jinfeng Yi, Cho-Jui Hsieh, and Shin-Ming Cheng. Autozoom: Autoencoder-based zeroth order optimization method for attacking black-box neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 742–749, 2019.

[50] Jianyu Wang and Haichao Zhang. Bilateral adversarial training: Towards fast training of more robust models against adversarial attacks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 6629–6638, 2019.

[51] Yutong Wang, Kunfeng Wang, Zhanxing Zhu, and Fei-Yue Wang. Adversarial attacks on faster r-cnn object detector. *Neurocomputing*, 2019.

[52] Cihang Xie, Jianyu Wang, Zhishuai Zhang, Yuyin Zhou, Lingxi Xie, and Alan Yuille. Adversarial examples for semantic segmentation and object detection. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1369–1378, 2017.

[53] Cihang Xie, Yuxin Wu, Laurens van der Maaten, Alan L Yuille, and Kaiming He. Feature denoising for improving adversarial robustness. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 501–509, 2019.

[54] Weilin Xu, David Evans, and Yanjun Qi. Feature squeezing: Detecting adversarial examples in deep neural networks. *arXiv preprint arXiv:1704.01155*, 2017.

[55] Zhewei Yao, Amir Gholami, Peng Xu, Kurt Keutzer, and Michael W Mahoney. Trust region based adversarial attack on neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 11350–11359, 2019.

[56] Haichao Zhang and Jianyu Wang. Defense against adversarial attacks using feature scattering-based adversarial training. In *Advances in Neural Information Processing Systems*, pages 1829–1839, 2019.