Adversarial Examples Detection in Deep Networks with Convolutional Filter Statistics

Xin Li, Fuxin Li
School of Electrical Engineering and Computer Science
Oregon State University
urumican@gmail.com, lif@eecs.oregonstate.edu

Abstract
Deep learning has greatly improved visual recognition in recent years. However, recent research has shown that there exist many adversarial examples that can negatively impact the performance of such an architecture. This paper focuses on detecting those adversarial examples by analyzing whether they come from the same distribution as the normal examples. Instead of directly training a deep neural network to detect adversarials, a much simpler approach was proposed based on statistics on outputs from convolutional layers. A cascade classifier was designed to efficiently detect adversarials. Furthermore, trained from one particular adversarial generating mechanism, the resulting classifier can successfully detect adversarials from a completely different mechanism as well. The resulting classifier is non-subdifferentiable, hence creates a difficulty for adversaries to attack by using the gradient of the classifier. After detecting adversarial examples, we show that many of them can be recovered by simply performing a small average filter on the image. Those findings should lead to more insights about the classification mechanisms in deep convolutional neural networks.

1. Introduction
Recent advances in deep learning have greatly improved the capability to recognize visual objects [13, 26, 7]. State-of-the-art neural networks perform better than human on difficult, large-scale image classification tasks. However, an interesting discovery has been that those networks, albeit resistant to overfitting, would have completely failed if some of the pixels in the image were perturbed via an adversarial optimization algorithm [28, 4]. An image indistinguishable from the original for a human observer could lead to significantly different results from a deep network (Fig. 1).

Those adversarial examples are dangerous if a deep network is utilized in any crucial real application, be it autonomous driving, robotics, or any automatic identification (face, iris, speech, etc.). If the result of the network can be hacked at the will of a hacker, wrong authentications and other devastating effects would be unavoidable. Therefore, there are ample reasons to believe that it is important to identify whether an example comes from a normal or an adversarial distribution. A reliable procedure can prevent robots from behaving in undesirable manners because of the false perceptions it made about the environment.

The understanding of whether an example belongs to the training distribution has deep roots in statistical machine learning. The \textit{i.i.d.} assumption was commonly used in learning theory, so that the testing examples were assumed to be drawn independently from the same distribution of the training examples. This is because machine learning is only good at performing \textit{interpolation}, where some training examples surround a testing example. \textit{Extrapolation} is known to be difficult, since it is extremely difficult to estimate data labels or statistics if the data is extremely different from any known or learned observations. Many current approaches deal with adversarial examples by adding them back to the training set and re-train. However in their experiments, new adversarials can almost always be found from the re-trained classifier. This is because that the space of extrapolation is significantly larger than the area a machine learning algo-
As a simple extension, the results from visualizations of the model provide insights for further research on self-aware learning. The safety of applying these deep networks, and hopefully generalize to adversarially produced from another generalization approach [20] without any special adaptation or additional training. Those confidence estimates may improve the prediction of a deep network. From those intuitions, we extract simple statistics from convolutional filter outputs of various layers in the CNN. A cascade classifier is proposed that utilizes features from many layers to discriminate between normal and adversarial examples.

Experiments show that our features from convolutional filter output statistics can separate between normal and adversarial examples very well. Trained with one particular adversarial generation method, it is robust enough to generalize to adversarials produced from another generation approach [20] without any special adaptation or additional training. Those confidence estimates may improve the safety of applying these deep networks, and hopefully provide insights for further research on self-aware learning.

As a simple extension, the results from visualizations of the features prompted us to perform an average filter on corrupted images, and found out that many correct predictions can be recovered from this simple filtering.

2. Deep Convolutional Neural Networks

A deep convolutional neural network consists of many convolutional layers which are connected to spatially/temporally adjacent nodes in the next layer:

$$Z_{m+1} = [T(W_1 * Z_m), T(W_2 * Z_m), \ldots, T(W_k * Z_m)]$$  \(1\)

where \(Z_m\) is the input features at layer \(m\), \(W_1, \ldots, W_K\) are filters that could be much smaller than the size of \(Z_m\) (e.g. \(3 \times 3, 5 \times 5, 7 \times 7\)), \(*\) is the convolution operator, and \(T\) is a nonlinear transformation function such as the rectified linear unit (ReLU) \(T(x) = \max(0, x)\). Other commonly used layers in a CNN include max-pooling layers, or other normalization layers [13] such as batch normalization layers [10]. Most deep networks adopt similar principles while adding more structural complexity in the system such as more layers and smaller filters in each layer [26], multi-layered network within each layer [27], residual network [7], etc. A convolutional neural network makes sense in structured data because it naturally exploits the locality structure in data. In an image, pixels that are located close to each other are naturally more correlated than pixels that are far away [17]. The same holds for temporal data (video, speech) where objects (frames, utterances) that are temporally close can be assumed to be more correlated.

3. Understanding the Trained Deep Classifier

3.1. Adversarial Optimization

The famous result that deep networks can be broken easily [28] is an important motivation of this work. The idea is to start from an existing example (image) and optimize to obtain an example that will be classified to another category while being close to the original example. Namely, the following optimization problem is solved:

$$\min_r \|r\|_1 + L(f_{\theta}(x_0 + r, y))$$

s.t. \(x_0 + r \in [0, 1]^d\)  \(2\)

where \(x_0\) is a known example and \(y\) is an arbitrary category label, \(d\) is the input dimensionality. \(c\) is a parameter that can be tuned for trading off between proximity to the original example \(x_0\) and the classification loss on the other category \(y\). It has been shown, to the astonishment of many, that one can choose an \(r\) with very small norm while completely change the output of the algorithm (e.g. Fig. 1), this can even be done universally for almost all networks, datasets and categories [28, 4]. Besides, adversarials trained
from one network may even fool a related one trained from the same dataset [18]. This has led many people to question whether deep networks are really learning the “proper” rules for classifying those images.

3.2. Adversarial Behavior

In order to gain a deeper understanding of the behavior of a deep network and illustrate the difference between adversarial and normal example distributions, we utilize spectral analysis. As a starting point, we perform principal component analysis (PCA) [11] at the 14-th layer of a VGG network trained on the ImageNet dataset (the first fully-connected layer). The rationale behind using PCA is that each deep learning layer is a nonlinear activation function on a linear transformation, hence a large part of the learning process lies within the linear transformation, for which PCA is a standard tool to analyze.

A linear PCA is performed on the entire collection of 50,000 images from the ImageNet validation set, as well as 4,000 adversarial examples collected using the approach in (2), starting from random images in the collection. The result shows very interesting findings (Fig. 2) and sheds more light on the internal mechanics of those adversarial examples. In Fig. 2(a), we show the PCA projection onto the first two eigenvectors. This cannot separate normal and adversarial examples, as one could possibly imagine. The adversarial examples seem to exactly belong to the same distribution as normal ones. However, it does seem that the adversarial examples reside mostly in the center while the normal examples occupy a bigger chunk of space.

Interestingly, as we move to the tail of the PCA projection space, the picture starts to change significantly. In Fig. 2(b), we can see that there are a significant amount of adversarial examples that has extremely large values w.r.t. to the normal examples in the tail of the distribution. We chose to print the projection on the 3,547-th and 3,844-th eigenvector, but similar distributions can be found all over the tail. As one can see, at such a far end on the tail, the projections of normal examples are very similar to random samples under a Gaussian distribution. An explanation for that could be that under these “uninformative” directions, most of the weighted features are nearly independent w.r.t. each other, hence the distribution of their sum is similar to Gaussian, according to the central limit theorem. However, although normal examples behave similarly to a Gaussian, some adversarial examples are having projections with a deviation as large as 5 or 10 times the standard deviation, which are extremely unlikely to occur under a Gaussian distribution.

\[ p_i(x) = \frac{\exp(f_i(x))}{\sum_i \exp(f_i(x))} \]  

that converts raw predictions \( f_i(x) \) into probabilities. The result shows that normal examples have on average one category with a raw prediction value more than 20, however adversarial examples have only 0.01 category with raw predictions more than 20. The reason that those adversarial examples appear more confident after softmax is because that the predictions on all the other categories are regularized even more. Hence the normalization component of softmax has decided that the single prediction, although much less strong, should be assigned a probability of more than 90%.

We note that this issue was also pointed out by [2] in a different manner and they proposed a solution in the OpenMax classifier, which we compare against in the experiments.

But besides that, it seems that such extremal and standard deviation statistics are evident features that could help discriminating normal and adversarial examples. Unfortunately, they only occur as a statistic from a large sample, as any single point in Fig. 2(a) looks similar to a single point in the normal distribution. We have tried to utilize the tail distributions (Fig. 2(b)) to create a classifier which easily achieved 99% accuracy separating adversarials from normals, however we subsequently found out that since the tail almost do not contribute to the classification, knowing this defense, the adversarial example can easily optimize to remove their footprints on the tail distributions.

This leads us to think about an approach that would turn a single image into a distribution, so that we can use statistics as detectors for adversarial examples. An image is a distribution of detectors. Especially, the output of each filter from each convolutional layer is an image which could be

\[ \text{Note this is without a ReLU transformation. ReLU would destroy the negative part of the data distribution so that it no longer looks like a Gaussian. However, some tail effects can be observed even in the distribution after ReLU.} \]
treated as a distribution where the samples are the pixels. Therefore, in the following section we aim to build a classifier based on collecting statistics from such distributions.

4. Identifying Adversarial Examples

4.1. Feature Collection

Suppose the output at a convolutional layer \( m \) is an \( W \times H \times K \) tensor, where \( W \) and \( H \) represent the width and height of the image at that stage (smaller than original after max-pooling), and \( K \) represents the number of convolutional filters. Such a tensor can be considered as a \( K \)-channel image where each pixel has a \( K \)-dimensional feature. We consider the feature on every pixel to be a \( K \)-channel image, and each pixel has a \( K \)-dimensional output. A random vector drawn from the distribution \( D_m \) of convolutional filter outputs, a \( K \)-dimensional distribution.

The list of statistics we collect is:

- Normalized PCA coefficients
- Minimal and Maximal values
- 25-th, 50-th and 75-th percentile values

on each of the \( K \)-dimensional features. Normalized PCA coefficients are collected via Algorithm 1. Extremal and percentile statistics are straightforward to understand.

The features we collect are non-subdifferentiable, hence essentially preventing adversaries to use gradient-based attacks to counter the classifier. Although we are interested in a generative adversarial network-type adversary which would learn to avoid our detector, such adversaries would have to resort to derivative-free optimization methods, which currently do not scale to the size of a realistic image. The best derivative-free approach we have tried scales up to several hundreds of variables. The genetic algorithm in [20] scales better, but as we will soon show, their low-level feature statistics are so different from natural images, making them very easy to be detected, even without training on any data from their adversarial generation algorithm.

**Algorithm 1** PCA Statistics Extraction

1: **INPUT:** Image \( I \), layer \( m \).
2: For all normal images in a training set, compute their CNN filter output of layer \( m \) to form an example matrix \( Z_m \).
3: Compute the mean \( e \) and PCA projection matrix \( W \) of \( Z_m \).
4: Compute the standard deviation \( s \) on each dimension in the PCA projection \( W^T (Z_m - e 1^T) \).
5: For each image \( I \), project its CNN filter output of layer \( m \) \( Z_{ml} \) using PCA: \( z_{ml} = W^T (Z_{ml} - e 1^T) \), and normalize them by dividing the standard deviation \( s \) on each respective dimension.
6: Collect the statistic for each image as \( x_I = \frac{1}{n} \| z_{ml} \|_1 \), where \( L_1 \) norm is the vector \( L_1 \) norm. The resulting statistic is \( K \)-dimensional.

4.2. Classifier Cascade

[29] proposed a famous strategy for face detection by using a cascaded boosting classifier composed by a sequence
of base classifiers. A cascade classifier is ideal when it is easy to identify many of the examples from a category but some important cases can be difficult. In Fig. 4, $SC_N$ represents the classifier at each stage. $X$ is the input of the cascade classifier. The negatives in a cascade classifier from each stage will be outputted directly, while the positives will go to the next stage.

In our case, the normal category is much easier to detect than the adversarial category (see e.g. Fig. 6). In our initial experiments with VGGNet, we found that more than 80% of normal examples can be determined from the first convolutional layer with 100% precision. Therefore, we constructed a cascade classifier based on convolutional layers: the first stage works with features collected from the outputs of the first convolutional layer, the second with the second layer, etc. (Fig. 4). The base classifiers will not solely consider statistics from their own stage, instead, after one stage of training, the remaining positive examples will be concatenated to the corresponding features on the next stage.

The operations that are represented by Fig. 4 can also be summarized as Algorithm 2.

**Algorithm 2 Training Process of a cascade of Classifier**

1. $N_{pool} \leftarrow$ Normal example pool, $P_{train} \leftarrow$ Training set of $N_p$ perturbed examples, $L \leftarrow$ Number Of convolutional layers
2. while current layer $\leq L$ Or $N_{pool} \neq \emptyset$ do
3. Draw $N_p$ sized subset $P_{normal}$ from $N_{pool}$
4. $T \leftarrow P_{normal} \cup P_{train}$
5. Train SVM on $T$
6. Predict SVM on $N_{pool}$, eliminate those predicted as normal above a threshold (described in text)
7. end while

The overall false positive rate of a $K$ stage cascade classifier can be represented as: $F = \prod_{i=1}^{K} f_i$, where $f_i$ is the false positive rate at each layer. And similarly the true positive rate can be represented in the same form: $T = \prod_{i=1}^{K} t_i$, where $t_i$ is the true positive rate at each stage. In order to maximize recall, we maintain a high true positive rate and select a classification threshold which corresponds to a high true positive rate (97% in AlexNet and 98% in VGG).

5. Related Work

Szegedy et al. [28] proposes the adversarial optimization formulation in eq. (2). [4] proposes an explanation of the adversarial mechanism, and proposed a simpler adversarial optimization mechanism that only corrupts based on the signs of gradient of the network. The fact that such examples can be generated so easily with the gradient sign method shows that adversarial examples come from attacking the magnifying effect coming from the linearity in the network. [20] proposes another mechanism to generate adversarial examples by deep networks with high confidence (Fig. 5). [19] proposes another efficient approach. [23] proposes an approach to generate adversarial examples that match the convolutional filter outputs as well as perturbing the data. [25, 8] propose approaches to sample adversaries or minimax optimization for making learning more robust. While most of the work are done on standard benchmarks such as MNIST, CIFAR and ImageNet, [14] is an interesting work on projecting the adversaries in physical world.

Recently, there have been a lot of focus on training adversarial generation networks to create Generative Adversarial Networks (GANs) [3, 22, 32, 24]. These networks play a two-player game where a generator network aims to generate adversarial examples that will not be correctly classified by another discriminator network, and the goal is to generate images more and more similar to natural images. It has been shown that these networks generate images that resemble natural images. However, this generative approach is different from our goal, where we aim to create discriminative networks that discriminate from images that are already indistinguishable from natural images (e.g. Fig.1).

Mechanisms for detecting and countering adversarial examples have also been proposed [5, 21]. [18] proposes to use the foveation mechanism to alleviate adversarial examples when it is already known to be adversarial, but did not attempt to detect adversarial. The open-set deep networks proposed by [2] seek to alleviate concerns from a soft-max classification by creating an abstain option. The universal classifier [31] is similar but with more theoretical arguments.

Self-aware learning (classification with an abstain option) had been proposed in e.g. [12, 16]. It is relevant to robust learning (e.g. [15]), however robust learning usually seek to directly optimize the minimax loss under adversarial conditions, instead of outputting an abstain option. [1, 30] also focuses on classification with an abstain option.

6. Experiments

Our algorithm is tested on 2 approaches to generate adversaries. The main one is data generated using the L-BFGS
algorithm by [28]. We generated 4,000 adversarials from a random subset of the ILSVRC-2012 validation set (total of 50,000 images). In order to test the out-of-sample generalization capability, we included another dataset, which includes 5,000 EA-adversarial images generated using the algorithm in [20]. These datasets are tested with 2 different network structures, VGG-16 [26] and AlexNet [13]. All input images had been normalized and re-shaped into $224 \times 224$ for VGG-16 model and $227 \times 227$ for AlexNet. The MatConvNet toolbox is used for the deep networks. Features are extracted from the statistics of interest, mentioned in section 4.1, in the lower 9 convolutional layers from the VGG-16 network, and in all 5 convolutional layers from AlexNet.

All the classifiers were trained with a subset of 2,000 L-BFGS adversarials and a randomly selected subset of 45,000 normal images in the ILSVRC-2012 dataset. Testing was done on a random held-out dataset with the other 5,000 normal images and 2,000 L-BFGS adversarials for the L-BFGS experiments, and on a randomly selected 5,000 normal images and 5,000 EA-adversarials for the EA experiments. The training and testing sets do not overlap in any ways: for all the testing L-BFGS adversarials, the corresponding normal images were not seen in the training set. EA experiments are only conducted with AlexNet since we found out that EA-adversarials did not succeed fooling the VGG-16 network. We have also tried some other fast adversarial generation approaches such as [4], but they generally do not generate adversarials that fool classifiers as completely (with more than 90% confidence) or as universally (on 100% of the images) as the L-BFGS algorithm.

Additional experiment results using DeepFool and on ResNet can be found in the supplementary material.

6.1. Single-Layer Results

The first experiments we did were by extracting statistics on each single convolutional layer output, instead of using the cascade classifier. From Tables 1 and 2, one can see that the statistics from a single layer have some discriminative power to distinguish normal examples from adversarials, but are not extremely effective. However, EA-adversarials were much easier to distinguish, even though our classifier was trained only on L-BFGS adversarials instead of EA ones (Table 3). We only need the first three convolutional layers to reach an overall 97.34% classification accuracy. We believe the reason is that our features capture natural image statistics, and because EA-adversarials look so unnatural, their statistics are vastly different than natural images (see Sec. 6.3 for more discussions).

<table>
<thead>
<tr>
<th>Network Layer</th>
<th>2nd</th>
<th>3rd</th>
<th>4th</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Accuracy</strong></td>
<td>57.5 ± 0.7</td>
<td>67.3 ± 0.7</td>
<td>70.9 ± 0.6</td>
</tr>
<tr>
<td><strong>Accuracy</strong></td>
<td>74.9 ± 0.9</td>
<td>78.95 ± 0.6</td>
<td></td>
</tr>
</tbody>
</table>

Table 1. Classification Result with AlexNet for Normal vs. L-BFGS-adversarials

<table>
<thead>
<tr>
<th>Network Layer</th>
<th>2nd</th>
<th>3rd</th>
<th>4th</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Accuracy</strong></td>
<td>72.1 ± 0.7</td>
<td>84.1 ± 0.7</td>
<td>80.3 ± 0.6</td>
</tr>
<tr>
<td><strong>Accuracy</strong></td>
<td>81.4 ± 0.9</td>
<td>74.3 ± 0.6</td>
<td>73.9 ± 0.6</td>
</tr>
<tr>
<td><strong>Accuracy</strong></td>
<td>74.2 ± 0.7</td>
<td>71.2 ± 0.7</td>
<td>74.3 ± 0.8</td>
</tr>
</tbody>
</table>

Table 2. Classification Result with VGG-16 for Normal vs. L-BFGS-Adversarials

6.2. Experiment for LBFGS-Adversarials Detection

Next we test the cascade classifier on both AlexNet and VGG-16. The parameter $C$ is set to 0.005. On AlexNet, the average accuracy of the cascade classifier reaches 83.4% over 20 random trials, and the AUC (area-under-curve) metric is 90.7%. We compared against the recently published OpenMax method [2]. To learn the Weibull distribution required for OpenMax, the EVT was applied on the same training set as the algorithm. Figure 6(a) shows the results, where we were able to outperform OpenMax by over 9% in area-under-curve (AUC) and 11% in terms of accuracy.

In VGG-16, the results were even better. The accuracy of the classifier was on average 90.665% over 20 random trials. Fig.6(b) shows the ROC curve. We believe the fact that VGG has a lot more layers than AlexNet helps setting more constraints on the layer statistics, and is subsequently helpful for detecting adversarial examples.

Finally, the cascade classifier was tested on EA-adversarials. We obtained more than 96% accuracy with 0 false positive rate, with a final accuracy of 97.3% and AUC of 98.2% (Fig.6(c)). In other words, our algorithm is rarely fooled by EA-adversarials, even without training on them.
6.3. Visualization of Statistics

Our experiment results show that EA-adversarials are easy to detect with our detector. To gain more insight into this result, we made a few comparisons between the statistics of interest extracted from normal images, LBFGS-adversarials and EA-adversarials.

We visualized the average of the statistics that are used for the detection task from the first layer of the AlexNet on all its dimensions. As can be seen in Fig. 7(a), the difference on the PCA projection statistics on extracted from EA-adversarials and that of the normal images is very dramatic. Meanwhile, compared to the EA-adversarials, the statistics from LBFGS-adversarials have much less difference from the normal data and the difference does not change very much across different dimensions.

From Fig. 7(b), one can see that LBFGS-adversarials have smaller extremal values than normal images. This might imply that the LBFGS optimization worked to diminish strong signals from the original image by introducing small pixel perturbations, and that helped our classifiers separating them from normal images. From Fig. 7(c), we see the EA-adversarials evidently differ from normal images. Those results illustrate why EA-adversarials are easier to detect. We suspect it would be easy to reach 100% accuracy, had we actually trained on some EA-adversarials. The accuracy, had we actually trained on some EA-adversarials. The decision to detect.

We assume that the training input is drawn i.i.d. from a distribution $P(x, y)$, where $x$ is the input and $y$ is the output. Assume that the testing input is drawn from a mixture distribution between $P(x, y)$ and $Q(x, y)$:

$$P_m = \Omega P(x, y) + (1 - \Omega)Q(x, y)$$

where $\Omega \in \{0, 1\}$ is an unknown mixture weight, and $Q(x, y)$ is an adversarial distribution. Assume that we have a classifier that includes a function $f(x)$, and a boolean strategy $a_i$ between predict and abstain that can be chosen for each individual $x_i$. Assume that the expected error from our classifier on the adversarial distribution is $e_a$ (which could be assumed, if no other prior is present, as the random guessing error of $C^{-1}$ for a $C$-class classification problem). Further assume that abstaining always incur a fixed cost $e_a$. As long as $e_a < e_q$, abstaining would be better than predicting on the example drawn from the adversarial distribution, however, $e_a$ should be set sufficiently large so that the classifier would still make predictions when confident, instead of abstaining everything.

For each testing input, the testing of the self-aware classifier is then trying to optimize $\min_a E_{P_m} L_a(x, y)$ where

$$L_a(x_i, y_i) = \begin{cases} P(y_i \neq f(x_i)), & \text{if } a_i = \text{predict}, \\ e_q & \text{if } a_i = \text{predict}, (x_i, y_i) \sim Q(x, y) \\ e_a & \text{if } a_i = \text{abstain} \end{cases}$$

hence the classifier needs to select between making a prediction using its function $f(x)$ and risk paying $e_q$ versus abstaining. It is easy to derive the optimal strategy:

$$a_i = \text{predict}, \quad \text{if } P(\Omega = 1|x_i)P(y_i \neq f(x_i)) + P(\Omega = 0|x_i)e_q < e_a$$

$$a_i = \text{abstain}, \quad \text{otherwise}$$
Our approach can be seen as estimating $P(\Omega = 1|x_i)$ in this framework. Experiments about the effect of such self-aware learning is shown in the supplementary material. We eagerly hope to apply it in realistic applications in future work.

### 7.2. Image Recovery

Insights from [4] indicate that the adversarial mechanism is very specifically attacking vulnerable gradients starting from the first convolutional layer. Insights from the previous experiments also suggest that LBFGS-adversarials work to diminish filter responses from the first convolutional layer. Therefore a natural idea would be to destroy the adversarial effects in the first convolutional layer to try to recover the original image. We tried a very simple approach: applying a small (e.g. $3 \times 3$) average filter on the adversarial image before using the CNN to classify it. The positive and negative adverse gradients will average out in this approach, and make the masked activations from the normal images more prominent. In Table 4 we illustrate such recovery results: after using a $3 \times 3$ average filter on identified adversarial examples, the classification accuracy improved from almost 0% to 73.0%, showcasing the effectiveness of this simple average filter.

Table 4. Recovery Results. Simply using a $3 \times 3$ average filter we can recover a large proportion of adversarial examples after detecting them using the algorithm described previously. More complex cancellation approaches such as foveation in [18] that utilizes cropping can achieve better results.

<table>
<thead>
<tr>
<th>Approach</th>
<th>Top-5 Accuracy (Recovered Images)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original Image (Non-corrupted)</td>
<td>86.5%</td>
</tr>
<tr>
<td>3 $\times$ 3 Average Filter</td>
<td>73.0%</td>
</tr>
<tr>
<td>5 $\times$ 5 Average Filter</td>
<td>68.0%</td>
</tr>
<tr>
<td>Foveation (Object Crop MP) [18]</td>
<td>82.6%</td>
</tr>
</tbody>
</table>

Those results show that we can both detect and recover from adversarial examples with high accuracy. But the main reason we performed this (overly simplistic) experiment is to show how simple it might be to cancel out some adversarial perturbations. Importantly, this result indicates that current deep convolutional networks are too locally focused: these are corruptions that can be cancelled out by a simple $3 \times 3$ average filter, however they can adversely impact the entire result of the deep network. For human with a large receptive field, they will not even care about what happens within a $3 \times 3$ area. Therefore, we believe that future deep learning approaches should focus on enlarging the receptive field in order to reduce the chance of being fooled by adversarial examples. Another potential direction is to research classification approaches that do not require a softmax-type normalization, in order to avoid regularizing attacks such as the ones used in the adversarial optimization in (2).

### 8. Conclusion

This paper proposes an approach that detects adversarial examples using simple statistics on convolutional layer outputs. A cascade classifier was designed based on simple statistics on filter outputs from each layer. And it was capable of detecting more than 85% of the adversarial examples. Experiments showed that our cascade classifier significantly outperforms state-of-the-art on detecting adversarial examples. Experiment also showed transfer learning capabilities of our classifier, since the classifier we trained with LBFGS adversarials are capable of detecting EA-adversarials as well. Insights drawn from these experiments lead us to perform simple $3 \times 3$ average filter to corrupted images, which successfully recovered most of them. In the future, we would like to explore GAN-type generative adversarial networks from the current results, with multiple rounds of adversarial detection and counter-detection.

### Acknowledgements

This paper was supported by Future of Life grants 2015-143880 and 2016-158701.
References


