PixelDefend: Leveraging Generative Models to Understand and Defend against Adversarial Examples

Yang Song [1] **Taesup Kim** [2] **Sebastian Nowozin** [3] **Stefano Ermon** [1] **Nate Kushman** [3]

Abstract

Adversarial perturbations of normal images are usually imperceptible to humans, but they can seriously confuse state-of-the-art machine learning models. What makes them so special in the eyes of image classifiers? In this paper, we show empirically that adversarial examples mainly lie in the low probability regions of the training distribution, regardless of attack types and targeted models. Based on this discovery, we devised *PixelDefend*, a new approach that *purifies* a maliciously perturbed image by moving it back towards the distribution seen in the training data. The purified image is then run through an unmodified classifier, making our method agnostic to both the classifier and the attacking method. As a result, PixelDefend can be used to protect already deployed models and be combined with other model-specific defenses. Experiments show that our method greatly improves resilience across a wide variety of state-of-the-art attacking methods, increasing accuracy on the strongest attack from 32% to 70% for CIFAR-10.

1. Introduction

Recent work has shown that small, carefully chosen modifications to the inputs of a neural network classifier can cause the model to give incorrect labels (Szegedy et al., 2013; Goodfellow et al., 2014). This weakness of neural network models is particularly surprising because the modifications required are often imperceptible, or barely perceptible, to humans. What makes those small modifications so special to deep neural networks?

In this paper, we propose and empirically evaluate the following hypothesis: Even though they have very small deviations from clean images, adversarial examples largely lie in the low probability regions of the distribution that generated the data used to train the model. Therefore, they fool classifiers mainly due to covariate shift. This is analogous to training models on MNIST (LeCun et al., 1998) but testing them on Street View House Numbers (Netzer et al., 2011).

To study this hypothesis, we first need to estimate the probability density of the underlying training distribution. To this end, we leverage recent developments in generative models. Specifically, we choose a PixelCNN (van den Oord et al., 2016b) model for its state-of-the-art performance in modeling image distributions (van den Oord et al., 2016a; Salimans et al., 2017) and tractability of evaluating the data likelihood. In the first part of the paper, we show that a well-trained PixelCNN generative model is very sensitive to adversarial inputs, typically giving them several orders of magnitude lower likelihoods compared to those of training and test images.

Since adversarial examples are generated from clean images by adding imperceptible perturbations, it is possible to decontaminate them by searching for more probable images within a small distance of the original ones. By limiting the L^∞ distance[1], this image *purification* procedure generates only imperceptible modifications to the original input, so that the true labels of the purified images remain the same. The resulting purified images have higher probability under the training distribution, so we can expect that a classifier trained on the clean images will have more reliable predictions on the purified images. Moreover, for inputs which are not corrupted by adversarial perturbations the purified results remain in a high density region.

We use this intuition to build *PixelDefend*, an image purification procedure which requires no knowledge of the attack nor the targeted classifier. PixelDefend approximates the training distribution using a PixelCNN model. The constrained optimization problem of finding the highest probability image within an ϵ-ball of the original is computationally intractable, however, so we approximate it using a greedy decoding procedure. Since PixelDefend does not change the classification model, it can be combined with

[1]Stanford University, Stanford, CA, USA [2]Université de Montréal, Montréal, Quebec, Canada [3]Microsoft Research, Cambridge, United Kingdom. Correspondence to: Yang Song <yangsong@cs.stanford.edu>, Nate Kushman <nkushman@microsoft.com>.

[1]We note that there are many other ways of defining distance of images. In this paper we use L^∞ norm.

other adversarial defense techniques, including adversarial training (Goodfellow et al., 2014), to provide synergistic improvements. We show experimentally that PixelDefend performs exceptionally well in practice, leading to state-of-the art results against a large number of attacks, especially when combined with adversarial training.

2. Background

Attacking methods Given a test image $\mathbf{X}$, an attacking method tries to find a small perturbation $\mathbf{\Delta}$ with $\|\mathbf{\Delta}\|_\infty \leq \epsilon_{\text{attack}}$, such that a classifier f gives different predictions on $\mathbf{X}^{adv} \triangleq \mathbf{X} + \mathbf{\Delta}$ and $\mathbf{X}$. Here colors in the image are represented by integers from 0 to 255. In the experiments, we test and compare Random perturbation (RAND), Fast Gradient Sign Method (FGSM) (Goodfellow et al., 2014), Basic Iterative Method (BIM) (Kurakin et al., 2016), DeepFool (Moosavi-Dezfooli et al., 2016), and Carlini-Wagner method (CW) (Carlini & Wagner, 2017b). More detailed descriptions are provided in Appendix A.

Defense methods Current defense methods generally fall into two classes. They either (1) change the network architecture or training procedure to make it more robust, or (2) modify adversarial examples to reduce their harm. In this paper, we compare Adversarial training with both FGSM and BIM adversarial examples (Goodfellow et al., 2014; Madry et al., 2017), Label Smoothing (Warde-Farley & Goodfellow, 2016)—both belong to the first category, and Feature Squeezing (Xu et al., 2017a;b), which belongs to the second category. We provide more detailed explanations in Appendix B.

Notations and experimental settings We denote the probability given by PixelCNN as $p_{\text{CNN}}(\mathbf{X})$. As a convenient representation of $p_{\text{CNN}}(\mathbf{X})$ for images, we also use the concept of *bits per dimension*, which is defined as $\text{BPD}(\mathbf{X}) \triangleq -\log p_{\text{CNN}}(\mathbf{X})/(I \times J \times K \times \log 2)$ for an image of resolution $I \times J$ and K channels.

We used the CIFAR-10 (Krizhevsky et al.) dataset in experiments. Two state-of-the-art deep neural network image classifiers are examined: ResNet (He et al., 2016) and VGG (Simonyan & Zisserman, 2014). We use the PixelCNN++ (Salimans et al., 2017) implementation for PixelCNN, but replace the mixture of logistics output with softmax. For more experimental settings, please refer to Appendix C.

3. Distribution of adversarial examples

To examine the distribution of adversarial examples, we train a PixelCNN model on the CIFAR-10 (Krizhevsky & Hinton, 2009) dataset and use its log-likelihood as an approximation to the true underlying probability density. The adversarial examples are generated with respect to a ResNet (He et al., 2016) using $\epsilon_{\text{attack}} = 8$, which gets 92% accuracy on the test images.

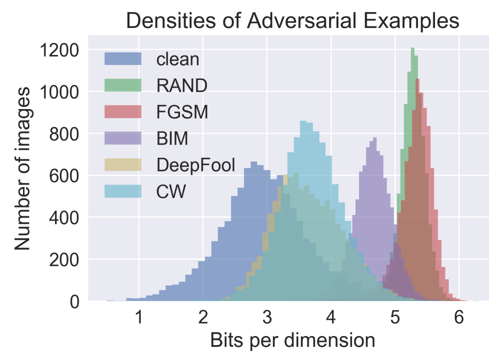

Figure 1. Likelihoods of different perturbed images.

However, the distributions of log-likelihoods show considerable difference between perturbed images and clean images. As summarized in Figure 1, even a 3% perturbation can lead to systematic decrease of log-likelihoods. In Figure 2, we additionally show the results of detecting adversarial examples with PixelCNN log-likelihoods. Note that the PixelCNN model has no information about the attacking methods for producing those adversarial examples, and no information about the ResNet model either.

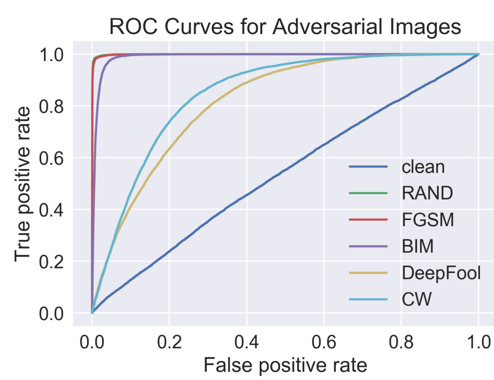

Figure 2. ROC curves of detecting perturbed inputs using $p_{\text{CNN}}(\mathbf{X})$. The clean training images are assigned negative labels while the adversarial examples (or clean test images) have positive labels.

We can see from Figure 1 and Figure 2 that random perturbations also push the images outside of the training distribution, even though they do not have the same adverse effect on accuracy. We believe this is due to an inductive bias that is shared by many neural network models but not inherent to all models, as discussed further in Appendix D.

4. Purifying images with PixelDefend

The basic idea behind PixelDefend is to *purify* input images, by making small changes to them in order to move them back towards the training distribution, *i.e.*, move the images towards a high-probability region. We then classify the purified image using any existing classifier.

Formally, we have training image distribution $p(\mathbf{X})$, and input image $\mathbf{X}$. We wish to find an image $\mathbf{X}^*$ that maximizes $p(\mathbf{X})$ subject to the constraint that $\mathbf{X}^*$ is within the ϵ_{defend}-ball of $\mathbf{X}$:

$$\max_{\mathbf{X}^*} p(\mathbf{X}^*) \quad \text{s.t.} \quad \|\mathbf{X}^* - \mathbf{X}\|_\infty \leq \epsilon_{\text{defend}}. \qquad (1)$$

Here ϵ_{defend} reflects a trade-off, since large ϵ_{defend} may change the meaning of $\mathbf{X}$ while small ϵ_{defend} may not be sufficient for returning $\mathbf{X}$ to the correct distribution. In practice, we choose ϵ_{defend} to be some value that overestimates ϵ_{attack} but still keeps high accuracies on clean images. As in Section 3, we approximate $p(\mathbf{X})$ with the PixelCNN distribution $p_{\text{CNN}}(\mathbf{X})$, which is trained on the same training set as the classifier.

Algorithm 1 PixelDefend

Input: Image $\mathbf{X}$, Defense parameter ϵ_{defend}, Pre-trained PixelCNN model p_{CNN}
Output: Purified Image $\mathbf{X}^*$

$\mathbf{X}^* \leftarrow \mathbf{X}$
for each row i **do**
  for each column j **do**
    for each channel k **do**
      $x \leftarrow \mathbf{X}[i, j, k]$
      Set feasible range $R \leftarrow [\max(x - \epsilon_{\text{defend}}, 0), \min(x + \epsilon_{\text{defend}}, 255)]$
      Compute the 256-way softmax $p_{\text{CNN}}(\mathbf{X}^*)$.
      Update the purified image $\mathbf{X}^*[i, j, k] \leftarrow \arg\max_{z \in R} p_{\text{CNN}}[i, j, k, z]$
    end for
  end for
end for

Surprisingly, it is hard to solve (1) with gradient-based methods. In fact, one of the advanced gradient-based optimization method—L-BFGS-B (Byrd et al., 1995)—usually produces purified images with even lower log-likelihoods than those of adversarial images (see Appendix E). For efficient optimization, we instead use a greedy technique described in Algorithm 1, which is similar to the greedy decoding process typically used in sequence-to-sequence models (Sutskever et al., 2014). Based on Ramachandran et al. (2017), PixelDefend on average processes 3.6 images per second on one NVIDIA TITAN Xp GPU for CIFAR-10 images. Visually, purified images indeed look much cleaner than adversarially perturbed ones. In Appendix I, we provide sampled purified images for CIFAR-10.

4.1. Adaptive PixelDefend

One improvement is to tune ϵ_{defend} adaptively based on the probability of the input image under PixelCNN. In this way, images that already have high probability under the training distribution would have a very low ϵ_{defend} preventing significant modification, while low probability images would have a high ϵ_{defend} thus allowing significant modifications. We implemented a very simple thresholding version of this, which sets ϵ_{defend} to zero if the input image probability is below a threshold value, and otherwise leaves it fixed at a manually chosen setting. In practice, we set this threshold based on knowledge of the set of possible attacks, so strictly speaking, the adaptive version of our technique is no longer attack-agnostic.

4.2. Experimental results

We carried out a comprehensive set of experiments to test various defenses versus attacks. Detailed information on experimental settings is provided in Appendix C. All experimental results are summarized in Tab. 1. In the upper part of the tables, we show how the various baseline defenses fare against each of the attacks, while in the lower part of the tables we show how our PixelDefend technique works. The cells are formated as $x/y/z$, where x denotes the accuracy (%) on images attacked with $\epsilon_{\text{attack}} = 2$, while y denotes the accuracy with $\epsilon_{\text{attack}} = 8$, and z is the accuracy with $\epsilon_{\text{attack}} = 16$. We use the same ϵ_{defend} for different ϵ_{attack}'s to show that PixelDefend is insensitive to ϵ_{attack}.

From the table we observe that adversarial training successfully defends against the basic FGSM attack, but cannot defend against the more advanced ones. Consistent with Madry et al. (2017), adversarial training with BIM examples is more successful at preventing a wider spectrum of attacks. For example, it improves the accuracy on strongest attack from 2% to 32% on CIFAR-10 when $\epsilon_{\text{attack}} = 8$. But the numbers are still not ideal even with respect to BIM attack itself. As in Tab. 1, it only gets 6% on BIM and 8% on CW when $\epsilon_{\text{attack}} = 16$. We also observe that label smoothing is only effective against simple FGSM attack. Model-agnostic methods, such as feature squeezing, can be combined with other defenses for strengthened performance. We observe that combining it with adversarial training indeed makes it more robust. Actually, Tab. 1 shows that feature squeezing combined with adversarial training dominates using feature squeezing along in all settings. It also gets good performance on DeepFool and CW attacks. However, for iterative attacks with larger perturbations, *i.e.*, BIM, feature squeezing performs poorly. On CIFAR-10, it only gets 2% and 0% accuracy on BIM with $\epsilon_{\text{attack}} = 8$ and 16 respectively.

PixelDefend, our model-agnostic and attack-agnostic method, performs well on different classifiers (ResNet and VGG) and different attacks *without modification*. In addi-

Table 1. **CIFAR-10** ($\epsilon_{\text{attack}} = 2/8/16$, $\epsilon_{\text{defend}} = 16$)

NETWORK	TRAINING TECHNIQUE	CLEAN	RAND	FGSM	BIM	DEEP FOOL	CW	STRONGEST ATTACK
ResNet	Normal	**92/92/92**	**92**/87/76	33/15/11	10/00/00	12/06/06	07/00/00	07/00/00
VGG	Normal	89/89/89	89/88/80	60/46/30	44/02/00	57/25/11	37/00/00	37/00/00
ResNet	Adversarial FGSM	91/91/91	90/**88**/84	**88/91/91**	24/07/00	45/00/00	20/00/07	20/00/00
	Adversarial BIM	87/87/87	87/87/86	80/52/34	74/32/06	79/48/25	76/42/08	74/32/06
	Label Smoothing	**92/92/92**	91/**88**/77	73/54/28	59/08/01	56/20/10	30/02/02	30/02/01
	Feature Squeezing	84/84/84	83/82/76	31/20/18	13/00/00	75/75/75	78/78/78	13/00/00
	Adversarial FGSM + Feature Squeezing	86/86/86	85/84/81	73/67/55	55/02/00	**85/85/85**	83/83/83	55/02/00
ResNet	Normal + ***PixelDefend***	85/85/88	82/83/84	73/46/24	71/46/25	80/80/80	78/78/78	71/46/24
VGG	Normal + ***PixelDefend***	82/82/82	82/82/84	80/62/52	80/61/48	81/76/76	81/79/79	80/61/48
ResNet	Adversarial FGSM + ***PixelDefend***	88/88/86	86/86/**87**	81/68/67	**81**/69/**56**	**85/85/85**	**84/84/84**	**81**/69/**56**
	Adversarial FGSM + ***Adaptive PixelDefend***	90/90/90	86/87/**87**	81/70/67	**81/70/56**	82/81/82	81/80/81	**81/70/56**

tion, we can see that augmenting basic adversarial training with PixelDefend can sometimes double the accuracies. We hypothesize that the purified images from PixelDefend are still not perfect, and adversarially trained networks have more toleration for perturbations. This also corroborates the plausibility and benefit of combining PixelDefend with other defenses.

Furthermore, PixelDefend can simultaneously obtain accuracy above 70% for all other attacking techniques, while ensuring that performance on clean images only declines slightly. Models with PixelDefend consistently outperform other methods with respect to the strongest attack. On CIFAR-10, it increases the accuracy from 74% to 81%, 32% to 70% and 6% to 56%, for $\epsilon_{\text{attack}} = 2$, 8, and 16 respectively.

4.3. End-to-End attack of PixelDefend

A natural question that arises is whether we can generate a new class of adversarial examples targeted specifically at the combined PixelDefend architecture of first purifying the image and then using an existing classifier to predict the label of the purified image. We have three pieces of empirical evidence to believe that such adversarial examples are hard to find in general. First, we attempted to apply the iterative BIM attack to an end-to-end differentiable version of PixelDefend generated by unrolling the PixelCNN purification process. However we found the resulting network was too deep and led to problems with vanishing gradients (Bengio et al., 1994), resulting in adversarial images that were identical to the original images. Moreover, attacking the whole system is very time consuming. Empirically, it took about 10 hours to generate 100 attacking images with one TITAN Xp GPU which failed to fool PixelDefend. Secondly, we found the optimization problem in Eq. (1) was not amenable to gradient descent (see more discussions in Appendix E). This makes gradient-based attacks especially difficult. Last but not least, the generative model and classifier are trained separately and have independent parameters. Therefore, the perturbation direction that leads to higher probability images has a smaller correlation with the perturbation direction that results in misclassification. Accordingly, it is harder to find adversarial examples that can fool both of them together. However, we will open source our codes and look forward to any possible attack from the community.

5. Conclusion

In this work, we discovered that state-of-the-art neural density models, *e.g.*, PixelCNN, can detect small perturbations with high sensitivity. This sensitivity broadly exists for a large number of perturbations generated with different methods. An interesting fact is that PixelCNN is only sensitive in one direction—it is relatively easy to detect perturbations that lead to lower probabilities rather than higher probabilities.

Based on the sensitivity, we explore the idea of purifying adversarial examples. We propose the PixelDefend algorithm, and experimentally show that returning adversarial examples to high probability regions of the training distribution can significantly decrease their damage to classifiers. Different from many other defensive techniques, PixelDefend is model-agnostic and attack-agnostic, which means it can be combined with other defenses to improve robustness without modifying the classification model. As a result, PixelDefend is a practical and effective defense against adversarial inputs.

There are also some related ideas which we didn't mention here due to space constraints. We discuss them in Appendix F.

References

Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

Richard H Byrd, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing*, 16 (5):1190–1208, 1995.

Nicholas Carlini and David Wagner. Adversarial examples are not easily detected: Bypassing ten detection methods. *arXiv preprint arXiv:1705.07263*, 2017a.

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

Moustapha Cisse, Piotr Bojanowski, Edouard Grave, Yann Dauphin, and Nicolas Usunier. Parseval networks: Improving robustness to adversarial examples. In *International Conference on Machine Learning*, pp. 854–863, 2017.

Bradley Efron and Robert J Tibshirani. *An introduction to the bootstrap*. CRC press, 1994.

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

Zhitao Gong, Wenlu Wang, and Wei-Shinn Ku. Adversarial and clean data are not twins. *arXiv preprint arXiv:1704.04960*, 2017.

Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

Kathrin Grosse, Praveen Manoharan, Nicolas Papernot, Michael Backes, and Patrick McDaniel. On the (statistical) detection of adversarial examples. *arXiv preprint arXiv:1702.06280*, 2017.

Shixiang Gu and Luca Rigazio. Towards deep neural network architectures robust to adversarial examples. *arXiv preprint arXiv:1412.5068*, 2014.

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*, pp. 770–778, 2016.

Matthias Hein and Markus Maier. Manifold denoising. In *Advances in neural information processing systems*, pp. 561–568, 2007.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research). URL `http://www.cs.toronto.edu/~kriz/cifar.html`.

Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale. *arXiv preprint arXiv:1611.01236*, 2016.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Yanpei Liu, Xinyun Chen, Chang Liu, and Dawn Song. Delving into transferable adversarial examples and black-box attacks. *arXiv preprint arXiv:1611.02770*, 2016.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

Jan Hendrik Metzen, Tim Genewein, Volker Fischer, and Bastian Bischoff. On detecting adversarial perturbations. *arXiv preprint arXiv:1702.04267*, 2017.

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*, pp. 2574–2582, 2016.

Aran Nayebi and Surya Ganguli. Biologically inspired protection of deep networks from adversarial attacks. *arXiv preprint arXiv:1703.09202*, 2017.

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, volume 2011, pp. 5, 2011.

Nicolas Papernot, Patrick McDaniel, and Ian Goodfellow. Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. *arXiv preprint arXiv:1605.07277*, 2016a.

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

Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In

Security and Privacy (SP), 2016 IEEE Symposium on, pp. 582–597. IEEE, 2016c.

Prajit Ramachandran, Tom Le Paine, Pooya Khorrami, Mohammad Babaeizadeh, Shiyu Chang, Yang Zhang, Mark A Hasegawa-Johnson, Roy H Campbell, and Thomas S Huang. Fast generation for convolutional autoregressive models. *arXiv preprint arXiv:1704.06001*, 2017.

Tim Salimans, Andrej Karpathy, Xi Chen, and Diederik P Kingma. Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications. *arXiv preprint arXiv:1701.05517*, 2017.

Hiroaki Sasaki, Aapo Hyvärinen, and Masashi Sugiyama. Clustering via mode seeking by direct estimation of the gradient of a log-density. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 19–34. Springer, 2014.

K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pp. 3104–3112, 2014.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

Florian Tramèr, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. The space of transferable adversarial examples. *arXiv preprint arXiv:1704.03453*, 2017.

Aaron van den Oord, Nal Kalchbrenner, Lasse Espeholt, Oriol Vinyals, Alex Graves, et al. Conditional image generation with pixelcnn decoders. In *Advances in Neural Information Processing Systems*, pp. 4790–4798, 2016a.

Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. *arXiv preprint arXiv:1601.06759*, 2016b.

Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pp. 1096–1103. ACM, 2008.

David Warde-Farley and Ian Goodfellow. 11 adversarial perturbations of deep neural networks. *Perturbations, Optimization, and Statistics*, pp. 311, 2016.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

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

Weilin Xu, David Evans, and Yanjun Qi. Feature squeezing mitigates and detects carlini/wagner adversarial examples. *arXiv preprint arXiv:1705.10686*, 2017b.

Ciyou Zhu, Richard H Byrd, Peihuang Lu, and Jorge Nocedal. Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization. *ACM Transactions on Mathematical Software (TOMS)*, 23(4): 550–560, 1997.

A. Attacking methods

Random perturbation (RAND) Random perturbation is arguably the weakest attacking method, and we include it as the simplest baseline. Formally, the randomly perturbed image is given by

$$\mathbf{X}^{adv} = \mathbf{X} + \mathcal{U}(-\lfloor\epsilon_{\text{attack}}\rfloor, \lfloor\epsilon_{\text{attack}}\rfloor),$$

where $\mathcal{U}(a, b)$ denotes an element-wise uniform distribution of integers from $[a, b]$.

Fast gradient sign method (FGSM) Goodfellow et al. (2014) proposed the generation of malicious perturbations in the direction of the loss gradient $\nabla_{\mathbf{X}} L(\mathbf{X}, y)$, where $L(\mathbf{X}, y)$ is the loss function used to train the model. The adversarial examples are computed by

$$\mathbf{X}^{adv} = \mathbf{X} + \epsilon_{\text{attack}} \operatorname{sign}(\nabla_{\mathbf{X}} L(\mathbf{X}, y)).$$

Basic iterative method (BIM) Kurakin et al. (2016) tested a simple variant of the fast gradient sign method by applying it multiple times with a smaller step size. Formally, the adversarial examples are computed as

$$\mathbf{X}_0^{adv} = \mathbf{X}, \quad \mathbf{X}_{n+1}^{adv} = \operatorname{Clip}_{\mathbf{X}}^{\epsilon_{\text{attack}}} \left\{\mathbf{X}_n^{adv} + \alpha \operatorname{sign}(\nabla_{\mathbf{X}} L(\mathbf{X}_n^{adv}, y))\right\},$$

where $\operatorname{Clip}_{\mathbf{X}}^{\epsilon_{\text{attack}}}$ means we clip the resulting image to be within the ϵ_{attack}-ball of $\mathbf{X}$. Following Kurakin et al. (2016), we set $\alpha = 1$ and the number of iterations to be $\lfloor\min(\epsilon_{\text{attack}} + 4, 1.25\epsilon_{\text{attack}})\rfloor$. This method is also called Projected Gradient Descent (PGD) in (Madry et al., 2017).

DeepFool DeepFool (Moosavi-Dezfooli et al., 2016) was the first method trying to minimize the perturbation needed to have a misclassification. It works by iteratively linearizing the decision boundary. However, compared to FGSM and BIM, this method is much slower in practice. We clip the resulting image so that its perturbation is no larger than ϵ_{attack}.

Carlini-Wagner (CW) Carlini & Wagner (2017b) proposed an efficient optimization objective for iteratively finding the adversarial examples with the smallest perturbations. As with DeepFool, we clip the output image to make sure the perturbations are limited by ϵ_{attack}.

B. Defense methods

Adversarial training This defense works by generating adversarial examples on-the-fly during training and including them into the training set. FGSM adversarial examples are the most commonly used ones for adversarial training, since they are fast to generate and easy to train. Although training with higher-order adversarial examples (*e.g.*, BIM) has witnessed some success in small datasets (Madry et al., 2017), other work has reported failure in larger ones (Kurakin et al., 2016). We consider both variants in our work.

Label smoothing In contrast to adversarial training, label smoothing (Warde-Farley & Goodfellow, 2016) is agnostic to the attack method. It converts one-hot labels to soft targets, where the correct class has value $1 - \epsilon$ while the other (wrong) classes have value $\epsilon/(N - 1)$. Here ϵ is a small constant and N is the number of classes. When the classifier is re-trained on these soft targets rather than the one-hot labels it is significantly more robust to adversarial examples. This method was originally devised to achieve a similar effect as *defensive distillation* (Papernot et al., 2016c), and their performance is comparable. We didn't compare to defensive distillation since it is more computationally expensive.

Feature squeezing Feature squeezing (Xu et al., 2017a) is both attack-agnostic and model-agnostic. Given any input image, it first reduces the color range from $[0, 255]$ to a smaller value, and then smooths the image with a median filter. The resulting image is then passed to a classifier for predictions. Since this technique does not depend on attacking methods and classifiers, it can be combined with other defensive methods such as adversarial training, similar to PixelDefend.

C. Experimental settings

Dataset CIFAR-10 is a dataset that is broadly used for image classification tasks. It consists of $60,000$ examples, where $50,000$ are used for training and $10,000$ for testing, and each sample is a 32×32 color image associated with 1 of 10 classes.

Adversarial Training We have tested adversarial training with both FGSM and BIM examples. During training, we take special care of the label leaking problem as noted in Kurakin et al. (2016)—we use the predicted labels of the model to generate adversarial examples, instead of using the true labels. This prevents the adversarially trained network to perform better on adversarial examples than clean images by simply retrieving ground-truth labels. Following Kurakin et al. (2016), we also sample ϵ_{attack} from a truncated Gaussian distribution for generating FGSM or BIM adversarial examples, so that the adversarially trained network won't overfit to any specific ϵ_{attack}. This is different from Madry et al. (2017), where the authors train and test with the same ϵ_{attack}. In addition, we randomly sample ϵ_{attack} from $\mathcal{N}(0, \delta)$, take the absolute value and truncate it to $[0, 2\delta]$, where $\delta = 8$.

Feature Squeezing For implementing the feature squeezing defense, we reduce the number of colors to 32 for CIFAR-10. The numbers are chosen to make sure color reduction will not lead to significant deterioration of image quality. After color depth reduction, we apply a 2×2 median filter with reflective paddings, since it is reported in Xu et al. (2017b) to be most effective for preventing CW attacks.

Models We use ResNet (62-layer) and VGG (16-layer) as classifiers. In our experiments, normally trained networks have the same architectures as adversarially trained networks. In practive, VGG is more robust than ResNet due to using of dropout layers. The network architecture details are described in Appendix G. For the PixelCNN generative model, we adopted the implementation of PixelCNN++ (Salimans et al., 2017), but modified the output from mixture of logistic distributions to softmax.

PixelCNN The PixelCNN (van den Oord et al., 2016b; Salimans et al., 2017) is a generative model with tractable likelihood especially designed for images. The model defines the joint distribution over all pixels by factorizing it into a product of conditional distributions.

$$p_{\text{CNN}}(\mathbf{X}) = \prod_i p_{\text{CNN}}(x_i | x_{1:(i-1)}).$$

The pixel dependencies are in raster scan order (row by row and column by column within each row). We train the PixelCNN model for each dataset using only clean (not perturbed) image samples. In Appendix H, we provide clean sample images as well as generated image samples from PixelCNN (see Figure 4).

Adaptive Threshold We chose the adaptive threshold discussed in Section 4.1 using validation data. We set the threshold at the lowest value which did not decrease the performance of the strongest adversary. In Tab. 1, the threshold was chosen to be 3.2. As a reference, the mean value of bits per dimension for clean CIFAR-10 test images is 3.0. However, we admit that using a validation set to choose the best threshold makes the adaptive version of PixelDefend not strictly attack-agnostic.

D. On random perturbations

One may observe that random perturbations also live outside of the high density area. Although many classifiers are robust to random noise, it is not a property granted by the dataset. The fact is that robustness to random noise could be from model inductive bias, and there exist classifiers which have high generalization performance on clean images, but can be attacked by small random perturbations.

It is easy to construct a concrete classifier that are susceptible to random perturbations. Our ResNet on CIFAR-10 gets 92.0% accuracy on the test set and 87.3% on randomly perturbed test images with $\epsilon_{\text{attack}} = 8$. According to our PixelCNN, 175 of 10000 test images have a bits per dimension (BPD) larger than 4.5, while the number for random images is 9874. Therefore, we can define a new classifier

$$\text{ResNet'}(\mathbf{X}) \triangleq \begin{cases} \text{ResNet}(\mathbf{X}), & \text{BPD}(\mathbf{X}) < 4.5 \\ \text{random label}, & \text{BPD}(\mathbf{X}) \geq 4.5 \end{cases},$$

which will get roughly $92\% \times 9825/10000 + 10\% \times 175/10000 \approx 90.6\%$ accuracy on the test set, while only $87.3\% \times 126/10000 + 10\% \times 9874/10000 \approx 11.0\%$ accuracy on the randomly perturbed images. This classifier has comparable generalization performance to the original ResNet, but will give incorrect labels to most randomly perturbed images.

Figure 3. The bits-per-dimension distributions of purified images from FGSM adversarial examples. We tested two purification methods, L-BFGS-B and greedy decoding, the latter of which is used in PixelDefend. A good purification method should give images that have lower bits per dimension compared to FGSM images and ideally similar bits per dimension compared to clean ones.

E. Gradient-based Optimization

Surprisingly, even gradient-based optimization faces great difficulty on (1). We found that one advanced methods in gradient-based constrained optimization, L-BFGS-B (Byrd et al., 1995) (we use the `scipy` implementation based on Zhu et al. (1997)), actually decreases $p_{\text{CNN}}(\mathbf{X})$ for most random initializations within the ϵ_{defend}-ball. To show the effectiveness of Algorithm 1 compared to L-BFGS-B, we take the first 10 images from CIFAR-10 test set, attack them by FGSM with $\epsilon_{\text{attack}} = 8$, and purify them with L-BFGS-B and PixelDefend respectively. We used random start points for L-BFGS-B and repeated 100 times for each image. As depicted in Figure 3, most L-BFGS-B attempts failed at minimizing the bits per dimension of FGSM adversarial examples. Because of the rugged gradient landscape of PixelCNN, L-BFGS-B even results in images that have lower probabilities. In contrast, PixelDefend works much better in increasing the probabilities of purified images, although their probabilities are still lower compared to clean ones.

F. Related work

Most recent work on detecting adversarial examples focuses on adding an outlier class detection module to the classifier, such as Grosse et al. (2017), Gong et al. (2017) and Metzen et al. (2017). Those methods require the classification model to be changed, and are thus not model-agnostic. Feinman et al. (2017) also presents a detection method based on kernel density estimation and Bayesian neural network uncertainty. However, Carlini & Wagner (2017a) shows that all those methods can be bypassed.

Grosse et al. (2017) also studied the distribution of adversarial examples from a statistical testing perspective. They reported the same discovery that adversarial examples are outside of the training distribution. However, our work is different from theirs in several important aspects. First, the kernel-based two-sample test used in their paper needs a large number of suspicious inputs, while our method only requires one data point. Second, they mainly tested on first-order methods such as FGSM and JSMA (Papernot et al., 2016b). We show the efficacy of PixelCNN on a wider range of attacking methods, including both first-order and iterative methods. Third, we further demonstrate that random perturbed inputs are also outside of the training distribution.

Some other work has focused on modifying the classifier architecture to increase its robustness, *e.g.*, Gu & Rigazio (2014), Cisse et al. (2017) and Nayebi & Ganguli (2017). Although they have witnessed some success, such modifications of models might limit their representative power and are also not model-agnostic.

Our basic idea of moving points to higher-density regions is also present in other machine learning methods not specifically designed for handling adversarial data; for example, the *manifold denoising method* of Hein & Maier (2007), the *direct density gradient estimation* of Sasaki et al. (2014), and the *denoising autoencoders* of Vincent et al. (2008) all move data points from low to high-density regions. In the future some of these methods could be adapted to amortize the purification process directly, that is, to learn a *purification network*.

G. Image classifier architectures

G.1. ResNet classifier for CIFAR-10 & Fashion MNIST

<table>
<tr><th>NAME</th><th colspan="2">CONFIGURATION</th></tr>
<tr><td>Initial Layer</td><td colspan="2">conv (filter size: 3×3, feature maps: 16 (4), stride size: 1×1)</td></tr>
<tr><td>Residual Block 1</td><td>batch normalization & leaky relu
conv (filter size: 3×3, feature maps: 16, stride size: 1×1)
batch normalization & leaky relu
conv (filter size: 3×3, feature maps: 16, stride size: 1×1)
residual addition</td><td>$\times 10$ times</td></tr>
<tr><td rowspan="2">Residual Block 2</td><td colspan="2">batch normalization & leaky relu
conv (filter size: 3×3, feature maps: 32, stride size: 2×2)
batch normalization & leaky relu
conv (filter size: 3×3, feature maps: 32, stride size: 1×1)
average pooling & padding & residual addition</td></tr>
<tr><td>batch normalization & leaky relu
conv (filter size: 3×3, feature maps: 32, stride size: 1×1)
batch normalization & leaky relu
conv (filter size: 3×3, feature maps: 32, stride size: 1×1)
residual addition</td><td>$\times 9$ times</td></tr>
<tr><td rowspan="2">Residual Block 3</td><td colspan="2">batch normalization & leaky relu
conv (filter size: 3×3, feature maps: 64, stride size: 2×2)
batch normalization & leaky relu
conv (filter size: 3×3, feature maps: 64, stride size: 1×1)
average pooling & padding & residual addition</td></tr>
<tr><td>batch normalization & leaky relu
conv (filter size: 3×3, feature maps: 64, stride size: 1×1)
batch normalization & leaky relu
conv (filter size: 3×3, feature maps: 64, stride size: 1×1)
residual addition</td><td>$\times 9$ times</td></tr>
<tr><td>Pooling Layer</td><td colspan="2">batch normalization & leaky relu & average pooling</td></tr>
<tr><td>Output Layer</td><td colspan="2">fc_10 & softmax</td></tr>
</table>

G.2. VGG classifier for CIFAR-10 & Fashion MNIST

<table>
<tr><th>NAME</th><th colspan="2">CONFIGURATION</th></tr>
<tr><td rowspan="2">Feature Block 1</td><td>conv (filter size: 3×3, feature maps: 16, stride size: 1×1)
batch normalization & relu</td><td>$\times 2$ times</td></tr>
<tr><td colspan="2">max pooling (stride size: 2×2)</td></tr>
<tr><td rowspan="2">Feature Block 2</td><td>conv (filter size: 3×3, feature maps: 128, stride size: 1×1)
batch normalization & relu</td><td>$\times 2$ times</td></tr>
<tr><td colspan="2">max pooling (stride size: 2×2)</td></tr>
<tr><td rowspan="2">Feature Block 3</td><td>conv (filter size: 3×3, feature maps: 512, stride size: 1×1)
batch normalization & relu</td><td>$\times 3$ times</td></tr>
<tr><td colspan="2">max pooling (stride size: 2×2)</td></tr>
<tr><td rowspan="2">Feature Block 4</td><td>conv (filter size: 3×3, feature maps: 512, stride size: 1×1)
batch normalization & relu</td><td>$\times 3$ times</td></tr>
<tr><td colspan="2">max pooling (stride size: 2×2) & flatten</td></tr>
<tr><td>Classifier Block</td><td colspan="2">dropout & fc_512 & relu
dropout & fc_10 & softmax</td></tr>
</table>

H. Sampled images from PixelCNN

H.1. CIFAR-10

Figure 4. True and generated images from CIFAR-10. The upper part shows true images sampled from the dataset while the bottom part shows generated images from PixelCNN.

I. Sampled purified images from PixelDefend

I.1. CIFAR-10

Figure 5. The upper part shows adversarial images generated from FGSM attack while the bottom part shows corresponding purified images by PixelDefend. Here $\epsilon_{\text{attack}} = 8$ and $\epsilon_{\text{defend}} = 16$.