VectorDefense: Vectorization as a Defense to Adversarial Examples

Vishaal Munusamy Kabilan
mkvishaal@gmail.com
Brandon Morris
blm0026@auburn.edu
Anh Nguyen
anhnguyen@auburn.edu
Auburn University

Abstract

Training deep neural networks on images represented as grids of pixels has brought to light an interesting phenomenon known as adversarial examples. Inspired by how humans reconstruct abstract concepts, we attempt to codify the input bitmap image into a set of compact, interpretable elements to avoid being fooled by the adversarial structures. We take the first step in this direction by experimenting with image vectorization as an input transformation step to map the adversarial examples back into the natural manifold of MNIST handwritten digits. We compare our method vs. state-of-the-art input transformations and further discuss the trade-offs between a hand-designed and a learned transformation defense.

1. Introduction

Deep neural networks (DNNs) have been shown to be extremely vulnerable to adversarial examples (AXs)—the inputs specifically optimized to fool them [38, 27]. An imperceptible perturbation can be crafted such that when added to the input of a deep image classifier would change its prediction entirely [38]. These adversarial perturbations can be as minute as one single pixel [37]; computed inexpensively with a single backward pass through the DNN [11] or via black-box optimization algorithms [7, 14, 29].

The inability to resist AXs is a general, task- and dataset-agnostic weakness across most machine learning models [1]. Importantly, AXs generated to fool one target model transfers to fool other unknown models [29]. This poses serious implications on the security and reliability of real-world applications, especially the safety-critical domains like autonomous driving.

Recent research suggests that the adversarial problem is an artifact of classifiers being imperfectly trained i.e. having a non-zero error on test data [9, 35]. However, perfect generalization is impractical to obtain in real-world settings where the input space is high-dimensional, and our models are trained to minimize the empirical risk over a limited training set. Despite a large body of recent research, the adversarial problem remains largely unsolved [1, 43]. A state-of-the-art, effective defense mechanism is adversarial training [23, 11]—directly training a model on both clean¹ and adversarial examples—is not a general solution. Furthermore, we wish the ideal defense layer to be (1) attack-agnostic, and (2) model-agnostic (e.g. not having to alter or impose additional constraints on an existing image classifier).

In this paper, we explore harnessing vectorization (i.e. image tracing) [33, 3]—a simple image transformation method in computer graphics—for defending against AXs. We refer to our method as VectorDefense. Specifically, we

¹We use clean and real images interchangeably to refer to real dataset examples (without any perturbations).
transmit each input bitmap image into a vector graphic image (SVG format)—which is composed of simple geometric primitives (e.g. oval and stroke)—via Potrace [33], and then rasterize it back into the bitmap form before feeding it to the classifier (Fig. 2).

VectorDefense employs vectorization to smooth out the adversarial perturbations, which are often minute, and local at the pixel level (Fig. 1). Here, we attempt to decompose (i.e. vectorize) an input image (e.g. a handwritten digit “9”) into layers of compact, and interpretable elements (e.g. a stroke and an oval in Fig. 2c) that are resolution-independent. We therefore test vectorizing MNIST data, which compose of simple stroke structures, to purify AXs without having to re-train the classifiers.

We make the following contributions:

1. **We show that VectorDefense is a viable input transformation defense to the problem of AXs.** We validate our hypothesis on (1) classifiers trained on MNIST [20]; (2) across 6 state-of-the-art attack methods; and (3) under both white- and gray-box threat models (Sec. 4.2).

2. **We compare and show that VectorDefense performs competitively with state-of-the-art hand-designed input transformation methods including bit-depth reduction [42] and image quilting [12] (Sec. 4.2).**

3. **We evaluate the effectiveness of input transformation techniques over the number of pixels allowed to be perturbed (i.e. a budget).** That is, we propose budget-aware variants of JSMA [30] and C&W \( L_0 \) [5] that craft an AX with a given budget. (Sec. 4.3).

4. **We compare and contrast VectorDefense with Defense-GAN [32], a state-of-the-art input transformation method with a learned prior.** Interestingly, we found that these two types of approaches can perform distinctively in many cases (Sec. 4.5).

2. Adversarial Attacks and Defenses

Consider a classifier \( f(\cdot) \) that attempts to predict a true label \( y \) given an input image \( x \). An AX is a slight perturbation of an input, \( x' \), such that \( f(x') \neq y \). The AX can be made visually indistinguishable from the original image. AX crafting algorithms can be also constrained by a norm of the perturbation \( \| \cdot \|_p \) [5].

In this paper, we evaluate our defense under two strong threat models: **white-box** and **gray-box** (characterized by the level of knowledge the adversary has about the victim). White-box: the attacker has access to the network weights, and also any defense method (here, the input transformation layer). Gray-box: the attacker has access to the network parameters, but not the defense method.

2.1. Attacks

Many methods for crafting adversarial examples have been proposed [1]. We briefly describe the most relevant gradient-based attacks, which we use to evaluate VectorDefense (Sec. 4).

A computationally inexpensive procedure is the **fast gradient sign method** (FGSM) [11]. This attack computes the adversarial input through a single backward pass of the network with respect to the input, amounting to a single-step
maximization of the loss function. If the classifier $f(\cdot)$ has a differentiable loss function $J(\cdot, \cdot)$, calculating the adversarial input is written as

$$x' = x + \epsilon \cdot \text{sign}(\nabla_x J(x, y))$$  \hspace{1cm} (1)

where $y$ is the true label and $\epsilon$ determines the strength of the attack (i.e., how much perturbation FGSM is allowed to make). The process can be adapted to an iterative procedure [19] that is more effective with less perturbation:

$$x^{(i+1)} = \text{clip}_x,\epsilon(x^{(i)} + \alpha \cdot \text{sign}(\nabla_{x^{(i)}} J(x^{(i)}, y)))$$  \hspace{1cm} (2)

where $x^{(0)} = x$ is the original image. We refer to this adaptation as iterative fast gradient sign method (I-FGSM). Each iteration, we take a step of size $\alpha$ in the adversarial direction, always staying within $\epsilon$ distance of the original input, measured by the $L_\infty$ norm. That is, a cap is imposed on the maximum perturbation that can be made at any pixel. Optimization continues until the image is misclassified or the maximum perturbation has been reached.

I-FGSM is similar to projected gradient descent (PGD) attack [21] except that PGD starts from a random point within an $\epsilon$-norm ball.

Another gradient-based method is the Jacobian Saliency Map Algorithm (JSMA) [30] which relies on the saliency map computed from the backpropagation. The saliency map shows how sensitive the network’s prediction is to the input pixels. In each step, JSMA selects two pixels to perturb, and repeats the process until a perturbation limit is reached or the DNN prediction becomes incorrect.

The DeepFool algorithm [24] finds an AX by iteratively projecting $x$ onto a linearized approximation of the decision boundary of classifier $f(\cdot)$. In each iteration, DeepFool performs the below update:

$$x^{(i+1)} = x^{(i)} - \epsilon \cdot \frac{f(x^{(i)})}{\|\nabla_{x^{(i)}} f(x^{(i)})\|_2^2} \nabla_{x^{(i)}} f(x^{(i)})$$  \hspace{1cm} (3)

where again $x^{(0)} = x$.

Carlini and Wagner (C&W) attack [6] instead uses Adam optimizer [16] and incorporates a constraint on the adversarial perturbation in three different ways: (1) limiting the amount of changes in $L_2$ distance from the original image; (2) limiting the maximum amount of changes to any pixel via $L_\infty$; (3) limiting the number of pixels that can be
perturbed (i.e. a budget) via $L_0$.

To study the effectiveness of the input transformation defenses over perturbation budget settings, we modify JSMA and C&W $L_0$ to take in an additional input parameter that specifies the number of pixels that can be modified. We refer to these algorithms as (1) Budget-aware JSMA and (2) Budget-aware C&W $L_0$ respectively:

- In every step, budget-aware JSMA selects a pair of pixels to perturb until the budget limit is reached regardless of whether the image is misclassified (i.e. $f(x') \neq y$).
- In contrast, budget-aware C&W $L_0$ maintains a set of all pixels that can be perturbed (initially the entire image), and then iteratively shrinks this set until its size equals a given budget. When it is not possible to find an AX at the given budget, the original image is returned.

2.2. Defenses

A number of defense strategies have been proposed to stymie the effectiveness of adversarial perturbations [1]. Here we describe existing input transformation schemes for later comparing with our VectorDefense. Input transformation defenses seek to “purify” an AX i.e. removing the adversarial perturbations (often small and local) while maintaining the features necessary for correct classification. Input transformations can be categorized into those with hand-designed vs. learned priors.

Under hand-designed priors, many methods have been proposed [1]. An effective technique is bit-depth reduction [42], which reduces the degrees of perturbation freedom within each pixel by quantizing the image. Image quilting [12] instead replaces patches of an image with similar clean patches extracted from a database.

There are input transformation defenses that instead harness a learned prior in the form of Generative Adversarial Networks (GANs) [32, 22, 34, 15], and PixelCNN [36].

Among those, we directly compare VectorDefense with a state-of-the-art method called Defense-GAN [32], which harnesses a GAN [10] to purify AXs. Basically, the idea is similar to conditional iterative image generation techniques [25, 26, 44]. First, we train a generator $G(\cdot)$ to map a latent code $z$ from a simple prior distribution (e.g. Gaussian) to an image $x$ in a target distribution (e.g. handwritten digits in MNIST) which the classifier is also trained on. Then, we search for a latent code $z^*$ such that the resulting image matches a target (adversarial) image $x$ i.e. $G(z^*) \approx x$. The final image found $G(z^*)$ is then fed to the classifier for prediction. Interestingly, this method is not completely robust. Since $G$ is not guaranteed to be perfectly trained and therefore, one can find AXs directly in the generated distribution [2, 15]. Overall, these defenses, and others that do not explicitly use input transformations, are still imperfect at resisting AXs [4, 5, 2, 1].

This work explores a novel, intuitive method that translates an input image into contours and other simple geometric shapes in an attempt to purify AXs. We view VectorDefense as a stepping stone towards decomposing images into compact, interpretable elements to solve the adversarial problem.

3. Methods

VectorDefense is an input transformation defense that aims to remove the adversarial artifacts in an AX, while preserving the class-specific features needed for classification. Specifically, we utilize Potrace vectorization algorithm [33] to translate an input bitmap image into a vector graphic. Then we rasterize the vector graphic back to a bitmap before classification (Fig. 2). We first describe Potrace (Sec. 3.1) and then how the defense method works (Sec. 3.2).

3.1. The Potrace Algorithm

Potrace is an algorithm to convert a black-and-white bitmap image (i.e. a binary image), into an algebraic description of its contours, typically in Bézier curves [33]. Therefore, a vector image is often smooth-edged and does not have pixelation effect (Fig. 2c vs. d). This description is preferred in graphic design since it is scale-invariant. We chose to experiment with Potrace because the algorithm is simple, efficient and has been shown effective for handwritten images [33].

At a high level, the algorithm works by tracing polygons onto the bitmap image based on intensity differences, then optimizing curves to match the contours of those polygons [33]. There are four steps in Potrace:

**Step 1:** An input bitmap image is decomposed into a set of paths that are formed by drawing boundaries that separate black and white regions.

**Step 2:** Potrace approximates each path found by a polygon. The polygons are constructed with the constraints of having the fewest number of edges while matching their respective paths.

**Step 3:** Potrace transforms each polygon into a smooth vector outline (i.e. a set of contours).

**Step 4:** Potrace joins adjacent Bézier curve segments together into a more compact vector file, though this often has imperceptible visual effects on the final output.

3.2. Vectorization as a Defense

We describe the components of the VectorDefense pipeline that are hypothesized to be effective in purifying AXs.

**Binarization** Vectorization is an image tracing algorithm based on a color palette [33]. Here, we choose to use only
black and white colors for tracing i.e. we binarize an image first before feeding into Potrace. The intuition that binarizing images effectively reduces the input space dimension where adversarial perturbations could be made has also been confirmed in a concurrent work of bit-depth reduction [42] (here, we reduce images into 1-bit).

**Despeckling** An adversarial example often has many speckles i.e. small color blobs that are not part of the digit but injected to fool the classifier (Fig. 1: top row). Despeckling is an attempt to remove these speckles by dropping all the paths (returned from step 1 in Potrace) that consists of fewer than \( t \) pixels. We empirically find that \( t = 5 \) works the best in removing adversarial speckles (Fig. 4).

![Figure 4: Despeckling removes small color blobs by dropping all the paths that consists of fewer than \( t \) pixels. We show the results of Potrace vectorization on an adversarial example crafted by C&W \( L_0 \) method [5] (a) with different values of \( t = 0, 5, 10, 20 \). t = 0: no despeckling.](image)

**Smoothing** Real handwritten digits often possess a large amount of smoothness (Fig. 3a), which is corrupted by adversarial perturbations (Fig. 3c). The steps 1–3 of Potrace vectorize an image into a set of contours, intuitively smoothing out the noisy perturbations (Fig. 4a vs. b). Therefore, vectorization is also effectively a hand-designed smoothness prior e.g. total variation [12].

### 4. Experiments & Results

We evaluate VectorDefense as an input transformation layer to help a state-of-the-art DNN classifier trained on MNIST correctly classify AXs created by 6 state-of-the-art gradient-based attack methods: I-FGSM, C&W \( L_2 \), PGD, DeepFool, JSMA and C&W \( L_0 \).

We compare VectorDefense against two hand-designed input transformation methods: image quilting [12], and bit-depth reduction [42] because of the following reasons.

Image quilting was shown to be one of the most effective input transformation techniques on natural images, i.e. ImageNet [12]. Here, we implement image quilting for MNIST (details in Sec. S7). Bit-depth reduction obtained state-of-the-art results on MNIST, especially by converting the images into 1-bit (rather than other bit-depth levels) [42]. In addition, comparing VectorDefense (which includes binarization) vs. binarization alone enables us to highlight the effectiveness of the entire vectorization transformation.

In addition, we also compare and contrast with a learned input transformation method: Defense-GAN [32] to shed more light into the pros and cons of hand-designed vs. learned priors.

### 4.1. Experiment setup

**Datasets and Networks** The victim model is a DNN classifier from [39] trained on the MNIST dataset [20]. The network architecture is described in Table S2.

We choose MNIST for two reasons: (1) research is still needed to understand and defend against AXs even at simple, low-dimensional problems [1, 9], which can inform defenses in a high-dimensional space (e.g. ImageNet) [9]; (2) the handwritten digits are made of simple strokes, which can be vectorized well by Potrace into decomposable geometric primitives.

**Implementation** All of the attack algorithms are from the cleverhans library [28], except C&W \( L_0 \) which is from the code released by [6]. Defense methods: Image quilting, Defense-GAN and BPDA are from the code released by [2]. We implemented the rest of the algorithms, and the code to reproduce our results is at https://github.com/VishaalMK/VectorDefense.

**Hyperparameters** Additional hyperparaters for our experiments are reported in Sec. S7.

### 4.2. VectorDefense in a Gray-Box Setting

We evaluate VectorDefense under the gray-box threat model, which has strong assumptions to defend against i.e. the attacker does have knowledge of the victim model, but not that of the defense mechanism being employed.

**Experiment** For each of the 6 attack methods, we compute 1000 AXs on the target DNN from 1000 random test set images. We feed the AXs through four defense methods: (1) Bit-depth reduction; (2) Image quilting; (3) VectorDefense; and (4) Defense-GAN; to produce purified images, which are then fed to the victim DNN for classification.

All hyperparameters for the 6 attacks are default from cleverhans except C&W \( L_0 \), which is from the code released by [6]. Defense methods: Image quilting, Defense-GAN and BPDA are from the code released by [2]. We implemented the rest of the algorithms, and the code to reproduce our results is at https://github.com/VishaalMK/VectorDefense.

**Results** We found that VectorDefense performed similarly to all three defense methods across four attack methods (I-FGSM; C&W \( L_2 \), DeepFool, and PGD). Table 1 reports the accuracy scores for all 4 defenses vs. 6 attack methods.

Specifically, for C&W \( L_0 \) and JSMA attacks, VectorDefense substantially outperformed existing hand-designed transformation methods (Table 1h). Qualitative comparisons also confirmed our quantitative result (Fig. 5c).

Compared to other attack methods, C&W \( L_0 \) poses a distinct challenge for all input transformation methods by explicitly deleting input features (here, setting many pixels to black in the image; see Fig. S10c & Fig. 1 top row). Under
such a strong attack, Defense-GAN substantially outperformed all hand-designed methods (Table 1g). This result highlighted a big performance difference between a hand-designed vs. a strong learned input transformation method and informed our experiment in the next section.

See Sec. S8 for more qualitative results of VectorDefense defending against each attack method considered.

### 4.3. Gray-Box: Budget-Aware Attack Algorithms

To study the performance of input transformations against state-of-the-art $L_0$ attacks, we make use of (1) Budget-aware JSMA and (2) Budget-aware C&W $L_0$ (described in Sec. 2.1). These attacks explicitly take in as input a budget—the number of pixels the algorithm is allowed to modify.

**Experiment** We sweep across 6 budget settings, from 8 to 48 pixels (in increments of 8 pixels). The existence of an AX is not guaranteed for every (original image, target label) pair, across every budget setting. Across all the budget settings, we therefore evaluate the defenses on a subset of 80 successful, targeted AXs, chosen from a set of one thousand AXs generated via each budget-aware algorithm. A successful AX is one that is (1) misclassified as a target class; and (2) crafted using the exact given budget.

**JSMA Results** Under budget-aware JSMA, VectorDefense outperformed bit-depth reduction and image quilting by a large margin ($\geq 17\%$; Fig. 7a). JSMA poses a challenge to input transformation methods by setting the perturbed pixels to the extreme positive (here, solid white; Fig. 5 heatmaps). The despeckling process is hypothesized to help VectorDefense remove adversarial perturbations more explicitly and effectively than bit-depth reduction and quilting (Fig. 5). However, Defense-GAN is still the most effective in recovering from JSMA attacks.

**C&W $L_0$ Results** Under budget-aware C&W $L_0$ attack, VectorDefense performs similarly to the existing hand-designed input transformations across increasing budget settings (Fig. 7b). Note that as the number of pixels perturbed increases, the average perturbation per pixel decreases (Fig. S14 heatmaps), farther away from the extreme values. This makes it easier for all hand-designed methods to recover from, leading to (1) the increasing accuracy scores as the budget increases (Fig. 7b); and (2) similar performances across VectorDefense, bit-depth reduction and image quilting (Fig. 7b). Overall, Defense-GAN performed superior compared to hand-designed defenses under general $L_0$ attacks with extreme positive and negative perturbations (Fig. S14b–d & Table 1g).

### 4.4. White-Box: Bypassing Input Transformations

Recent research showed that all state-of-the-art defenses are not completely robust to white-box attacks where the attacker has access to the defense mechanism [2]. While VectorDefense is not an exception under the white-box Backward Pass Differential Approximation (BPDA) attack [2], we observed that a large amount of distortion is required to fool the target DNN with VectorDefense.

**Experiment** Here we employ BPDA [2] to bypass VectorDefense. Basically, BPDA has access to a black-box input transformation method (here, VectorDefense) and uses it to approximate the gradient to compute AXs. We compare results of running BDPA to attack the target DNN with no defense vs. with VectorDefense under the same experimental setup. Optimization runs for 50 steps.

$L_2$ dissimilarity often describes the average distortion necessary to fool a classifier [2]. Here, we report the average $L_2$ dissimilarity for the successfully crafted AXs.

**Results** Though the adversary ends up being successful, VectorDefense makes the BPDA distort the images much more vs. when there is no defense ($L_2$ distortion of 5.30 vs. 2.97). Fig. S13 shows a qualitative comparison.

### 4.5. Hand-designed vs. Learned Prior

Throughout the previous results, we have qualitatively seen the superiority of Defense-GAN—an input transformation method with a learned prior—compared to hand-designed methods including VectorDefense. We further compare and contrast the process of purifying AXs by these two types of input transformation approaches.

Under strong attacks with 40-pixel perturbations,
Number of pixels perturbed by JSMA attack

(a) 8  (b) 16  (c) 24  (d) 32  (e) 40  (g) 48

Figure 5: Qualitative comparison of input transformations on budget-aware JSMA. Clean images (clean) are added to adversarial perturbations (row 2) to produce adversarial images (adv). Each column (a–g) corresponds to a specific budget setting denoted on top. Rows 4–7 show the results of VectorDefense (vec), bit-depth reduction [42] (bit), image quilting [12] (quilt) and Defense-GAN [32] (dgan) respectively. VectorDefense effectively maps adversarial images back to the natural manifold.

Defense-GAN substantially outperformed all hand-designed transformations (Fig. 7; yellow lines at 40).

To shed more light into this result, we qualitatively examine cases when Defense-GAN succeeded but others failed and showed interesting observations in Fig. 6. Here, the images are strongly perturbed such that the AXs actually qualitatively change into the target class (Fig. 6 row 3; digit “6” after perturbation does look like a “0”). However, Defense-GAN prior is so strong that it pulls the image back into the correct class “6” (Fig. 6d). In contrast, VectorDefense removes the speckles around the digit and turns the image into a “0”—the label that may make sense to humans but is not the original label. Similar opposite behaviors of Defense-GAN vs. VectorDefense can be observed in other cases (Fig. 6).

5. Discussion and Future Work

We present VectorDefense—a novel defense to AXs that transforms a bitmap input image into the space of vector
Figure 7: Classification accuracy of MNIST classifier (Table S2) on purified adversarial examples, crafted using (a) Budget-aware JSMA algorithm and (b) Budget-aware C&W $L_0$ algorithm and purified by four input transformations (1) VectorDefense; (2) Image quilting [12]; (3) Bit-depth reduction [42]; and (4) Defense-GAN [32]. Black dotted lines show the classifier accuracy on clean images, which serves as an upper bound to the effectiveness of a defense. A budget-setting of zero corresponds to the classification accuracy on clean images. Higher is better.

graphics and back, prior to classification. We showed how this transformation smooths out the visual artifacts created by state-of-the-art attack algorithms.

Under strong perturbations, VectorDefense and other hand-designed input transformation methods underperformed Defense-GAN, a method with a strong, learned prior. While this result may not be too surprising, we view VectorDefense as a stepping stone towards decomposing images into compact, interpretable elements to solve the AX problem. Simple use of vectorization in this work alone reduces effectiveness of AXs, and opens up promising extensions towards more robust machine learning:

- The vector representation (SVG) could be further compacted via Ramer-Peucker-Douglas algorithm [8, 31] or constrained to compose of elements from a strict set of geometric primitives (e.g. only straight strokes).

- Since the vector images are resolution-independent, one could rasterize them back into much smaller-sized images. In lower-dimensional space, it can be easier to defend against AXs [9].

- Another extension of VectorDefense would be to learn a generative model on this vector space.

- In high-dimensional color image space, simple image tracing may not yield compact and interpretable elements (Sec. S6); therefore, we did not explore further vectorizing natural images. Instead it might be interesting to de-render an image into a scene graph [41], and train a prior over the graphs [18].

Acknowledgements

We thank Zhitao Gong, Chengfei Wang for feedback on the drafts; and Nicholas Carlini and Nicolas Papernot for helpful discussions.
[32] P. Samangouei, M. Kabkab, and R. Chellappa. Defense-gan: Protecting classifiers against adversarial attacks using generative models. 2018. 2, 4, 5, 6, 7, 8

[33] P. Selinger. Potrace: a polygon-based tracing algorithm. Potrace (online), http://potrace.sourceforge.net/potrace.pdf (2009-07-01), 2003. 1, 2, 4, 11

[34] S. Shen, G. Jin, K. Gao, and Y. Zhang. Ape-gan: Adversarial perturbation elimination with gan. ICLR Submission, available on OpenReview, 2017. 4

[35] L. Smith and Y. Gal. Understanding measures of uncertainty for adversarial example detection. arXiv preprint arXiv:1803.08533, 2018. 1

[36] Y. Song, T. Kim, S. Nowozin, S. Ermon, and N. Kushman. Pixeldefend: Leveraging generative models to understand and defend against adversarial examples. arXiv preprint arXiv:1710.10766, 2017. 4

[37] J. Su, D. V. Vargas, and S. Kouichi. One pixel attack for fooling deep neural networks. arXiv preprint arXiv:1710.08864, 2017. 1

[38] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. arXiv preprint arXiv:1312.6199, 2013. 1

[39] F. Tramèr, A. Kurakin, N. Papernot, D. Boneh, and P. McDaniel. Ensemble adversarial training: Attacks and defenses. arXiv preprint arXiv:1705.07204, 2017. 5

[40] W. Wang, A. Wang, A. Tamar, X. Chen, and P. Abbeel. Safer classification by synthesis. arXiv preprint arXiv:1711.08534, 2017.

[41] J. Wu, J. B. Tenenbaum, and P. Kohli. Neural scene de-rendering. In Proc. CVPR, volume 2, 2017. 8

[42] W. Xu, D. Evans, and Y. Qi. Feature squeezing: Detecting adversarial examples in deep neural networks. arXiv preprint arXiv:1704.01155, 2017. 2, 4, 5, 6, 7, 8

[43] X. Yuan, P. He, Q. Zhu, R. R. Bhat, and X. Li. Adversarial examples: Attacks and defenses for deep learning. arXiv preprint arXiv:1712.07107, 2017. 1

[44] J.-Y. Zhu, P. Krähenbühl, E. Shechtman, and A. A. Efros. Generative visual manipulation on the natural image manifold. In European Conference on Computer Vision, pages 597–613. Springer, 2016. 4
S6. Tracing color images

Fig. S8 shows samples from SVHN and CIFAR-10 datasets traced via Potrace (each RGB channel being traced independently). While color images can be traced via vectorization (Fig. S8), the output vector are often not composed of simple compact, interpretable geometric primitives. Therefore, we did not explore it further in this work.

![SVHN CIFAR-10](image)

Figure S8: Tracing color images using Potrace: Top row shows test images in the bitmap space. Bottom row shows the corresponding vector graphic outputs.

S7. Experimental Setup

**VectorDefense** To vectorize the input image, we used the open-source implementation of the Potrace algorithm[^1]. In our experiments, we make two changes to the default potrace algorithm.

- We set the *turn policy*[^2] to black, which prefers to connect the black components in Step 1 (Sec. 3.1).
- We increased the default maximum size of removed speckles, the *despeckling* value, to 5 (Sec. 3.2).

With the image converted to a scalable vector graphics (SVG) format, we render it with the open-source Inkscape[^3] program, having the same width and height as the original image.

**Bit-depth reduction** We reduce the images to 1-bit in our experiments.

**Image-quilting** In our implementation of image quilting on MNIST, we use a quilting patch size of 4x4 and a database of 60,000 images from the MNIST training set. The patch is selected at random from one of $K = 10$ nearest neighbours.

**Defense-GAN** We set the parameters of L and R to 200 and 10 respectively.

S8. Additional results by VectorDefense

We show below five figures demonstrating how VectorDefense purifies AXs created by five different attack algorithms: I-FGSM (Fig. 3), JSMA (Fig. S9), C&W $L_0$ (Fig. S10), C&W $L_2$ (Fig. S11), and DeepFool (Fig. S12).

![Table S2](image)

| CONFIGURATION |
|---------------|
| Conv(64, 5, 5) + Relu |
| Conv(64, 5, 5) + Relu + Dropout(0.25) |
| FC(128) + Relu + Dropout(0.5) |
| FC + Softmax |

[^1]: http://potrace.sourceforge.net/
[^2]: [33]
[^3]: https://inkscape.org/
Figure S9: JSMA: Given starting images (a), adversarial images are generated using JSMA (c). VectorDefense effectively purifies the adversarial images (e). The perturbations introduced by the adversary (b) and the purification process (d) can be obtained by subtracting (a) from (c) and (a) from (e) respectively. All images start out classified correctly with label $l$ and are targeted to have a label $l + 1$. The images were chosen as the first of their class from the MNIST test set.
Figure S10: C&W $L_0$: Given starting images (a), adversarial images are generated using C&W $L_0$ (c). VectorDefense effectively purifies the adversarial images (e). The perturbations introduced by the adversary (b) and the purification process (d) can be obtained by subtracting (a) from (c) and (a) from (e) respectively. All images start out classified correctly with label $l$ and are targeted to have a label $l + 1$. The images were chosen as the first of their class from the MNIST test set.
Figure S11: C&W $L_2$: Given starting images (a), adversarial images are generated using C&W $L_2$ (c). VectorDefense effectively purifies the adversarial images (e). The perturbations introduced by the adversary (b) and the purification process (d) can be obtained by subtracting (a) from (c) and (a) from (e) respectively. All images start out classified correctly with label $l$ and are targeted to have a label $l + 1$. The images were chosen as the first of their class from the MNIST test set.
Figure S12: DeepFool: Given starting images (a), adversarial images are generated using DeepFool (c). VectorDefense effectively purifies the adversarial images (e). The perturbations introduced by the adversary (b) and the purification process (d) can be obtained by subtracting (a) from (c) and (a) from (e) respectively. All images start out classified correctly with label \( l \) and are targeted to have a label \( l + 1 \). The images were chosen as the first of their class from the MNIST test set.
Figure S13: Distortion introduced by BPDA against VectorDefense. Given starting images (a), adversarial images are generated using PGD (c) corresponding to a $L_2$ dissimilarity of 2.97. Once VectorDefense is augmented to the image classification pipeline, the adversarial images are then generated using PGD with BPDA (e) corresponding to a $L_2$ dissimilarity of 5.30. The perturbations introduced by the adversary under no defense (b) and the adversary in the presence of VectorDefense (d) can be obtained by subtracting (a) from (c) and (a) from (e) respectively. All images start out classified correctly with label $l$ and are targeted to have a label $l + 1$. The images were chosen as the first of their class from the MNIST test set.
Number of pixels perturbed in our $L_0$ attack

(a) 8  (b) 16  (c) 24  (d) 32  (e) 40  (g) 48

Figure S14: Qualitative comparison of input transformations on budget-aware C&W $L_0$. This figure follows the same conventions as in Fig. 5. Row 2: the intensities of perturbations gradually decrease as we move from the least (a) to the greatest number of pixels (g). Hand-designed input transformation methods can only remove, but cannot fill in missing pixels as Defense-GAN (bottom row).