Adversarial Transformation Networks: Learning to Generate Adversarial Examples

Shumeet Baluja and Ian Fischer
Google Research
Mountain View, CA.

Abstract

Multiple different approaches of generating adversarial examples have been proposed to attack deep neural networks. These approaches involve either directly computing gradients with respect to the image pixels, or directly solving an optimization on the image pixels. In this work, we present a fundamentally new method for generating adversarial examples that is fast to execute and provides exceptional diversity of output. We efficiently train feed-forward neural networks in a self-supervised manner to generate adversarial examples against a target network or set of networks. We call such a network an Adversarial Transformation Network (ATN). ATNs are trained to generate adversarial examples that minimally modify the classifier's outputs given the original input, while constraining the new classification to match an adversarial target class. We present methods to train ATNs and analyze their effectiveness targeting a variety of MNIST classifiers as well as the latest state-of-the-art ImageNet classifier Inception ResNet v2.

1. Introduction and Background

With the resurgence of deep neural networks for many real-world classification tasks, there is an increased interest in methods to generate training data, as well as to find weaknesses in trained models. An effective strategy to achieve both goals is to create *adversarial examples* that trained models will misclassify. Adversarial examples are small perturbations of the inputs that are carefully crafted to fool the network into producing incorrect outputs. These small perturbations can be used both offensively, to fool models into giving the "wrong" answer, and defensively, by providing training data at weak points in the model. Seminal work by Szegedy et al. (2013) and Goodfellow et al. (2014b), as well as much recent work, has shown that adversarial examples are abundant, and that there are many ways to discover them.

Given a classifier $f(\mathbf{x}) : \mathbf{x} \in \mathcal{X} \rightarrow y \in \mathcal{Y}$ and original inputs $\mathbf{x} \in \mathcal{X}$, the problem of generating *untargeted* adversarial examples can be expressed as the optimization: $\text{argmin}_{\mathbf{x}^*} L(\mathbf{x}, \mathbf{x}^*)\ s.t.\ f(\mathbf{x}^*) \neq f(\mathbf{x})$, where $L(\cdot)$ is a distance metric between examples from the input space (e.g., the L_2 norm). Similarly, generating a *targeted* adversarial attack on a classifier can be expressed as $\text{argmin}_{\mathbf{x}^*} L(\mathbf{x}, \mathbf{x}^*)\ s.t.\ f(\mathbf{x}^*) = y_t$, where $y_t \in \mathcal{Y}$ is some target label chosen by the attacker.[1]

Until now, these optimization problems have been solved using three broad approaches: (1) By directly using optimizers like L-BFGS or Adam (Kingma & Ba, 2015), as proposed in Szegedy et al. (2013) and Carlini & Wagner (2016). Such optimizer-based approaches tend to be much slower and more powerful than the other approaches. (2) By approximation with single-step gradient-based techniques like fast gradient sign (Goodfellow et al., 2014b) or fast least likely class (Kurakin et al., 2016a). These approaches are fast, requiring only a single forward and backward pass through the target classifier to compute the perturbation. (3) By approximation with iterative variants of gradient-based techniques (Kurakin et al., 2016a; Moosavi-Dezfooli et al., 2016a;b). These approaches use multiple forward and backward passes through the target network to more carefully move an input towards an adversarial classification.

[1] Another axis to compare when considering adversarial attacks is whether the adversary has access to the internals of the target model. Attacks without internal access are possible by transferring successful attacks on one model to another model, as in Szegedy et al. (2013); Papernot et al. (2016a), and others. A more challenging class of blackbox attacks involves having no access to any relevant model, and only getting online access to the target model's output, as explored in Papernot et al. (2016b); Baluja et al. (2015); Tramèr et al. (2016). See Papernot et al. (2015) for a detailed discussion of threat models.

2. Adversarial Transformation Networks

In this work, we propose Adversarial Transformation Networks (*ATN*s). An ATN is a neural network that transforms an input into an adversarial example against a target network or set of networks. ATNs may be untargeted or targeted, and trained in a black-box[2] or white-box manner. In this work, we will focus on targeted, white-box ATNs.

Formally, an ATN can be defined as a neural network:

$$g_{f,\boldsymbol{\theta}}(\mathbf{x}) : \mathbf{x} \in \mathcal{X} \rightarrow \mathbf{x}' \tag{1}$$

where $\boldsymbol{\theta}$ is the parameter vector of g, f is the target network which outputs a probability distribution across class labels, and $\mathbf{x}' \sim \mathbf{x}$, but $\operatorname{argmax} f(\mathbf{x}) \neq \operatorname{argmax} f(\mathbf{x}')$.

Training. To find $g_{f,\boldsymbol{\theta}}$, we solve the following optimization:

$$\underset{\boldsymbol{\theta}}{\operatorname{argmin}} \sum_{\mathbf{x}_i \in \mathcal{X}} \beta L_{\mathcal{X}}(g_{f,\boldsymbol{\theta}}(\mathbf{x}_i), \mathbf{x}_i) + L_{\mathcal{Y}}(f(g_{f,\boldsymbol{\theta}}(\mathbf{x}_i)), f(\mathbf{x}_i)) \tag{2}$$

where $L_{\mathcal{X}}$ is a loss function in the input space (e.g., L_2 loss or a perceptual similarity loss like Johnson et al. (2016)), $L_{\mathcal{Y}}$ is a specially-formed loss on the output space of f (described below) to avoid learning the identity function, and β is a weight to balance the two loss functions. We will omit $\boldsymbol{\theta}$ from g_f when there is no ambiguity.

Inference. At inference time, g_f can be run on any input $\mathbf{x}$ without requiring further access to f or more gradient computations. This means that after being trained, g_f can generate adversarial examples against the target network f even faster than the single-step gradient-based approaches, such as fast gradient sign, so long as $||g_f|| \lessapprox ||f||$.

Loss Functions. The input-space loss function, $L_{\mathcal{X}}$, would ideally correspond closely to human perception. However, for simplicity, L_2 is sufficient. $L_{\mathcal{Y}}$ determines whether or not the ATN is targeted; the *target* refers to the class for which the adversary will cause the classifier to output the maximum value. In this work, we focus on the more challenging case of creating targeted ATNs, which can be defined similarly to Equation 1:

$$g_{f,t}(\mathbf{x}) : \mathbf{x} \in \mathcal{X} \rightarrow \mathbf{x}' \tag{3}$$

where t is the target class, so that $\operatorname{argmax} f(\mathbf{x}') = t$. This allows us to target the exact class the classifier should mistakenly believe the input is.

In this work, we define $L_{\mathcal{Y},t}(\mathbf{y}', \mathbf{y}) = L_2(\mathbf{y}', r(\mathbf{y}, t))$, where $\mathbf{y} = f(\mathbf{x})$, $\mathbf{y}' = f(g_f(\mathbf{x}))$, and $r(\cdot)$ is a reranking function that modifies $\mathbf{y}$ such that $y_k < y_t, \forall\, k \neq t$.

[2]E.g., using Williams (1992) to generate training gradients for the ATN based on a reward signal computed on the result of sending the generated adversarial examples to the target network.

Note that training labels for the target network are not required at any point in this process. All that is required is the target network's outputs $\mathbf{y}$ and $\mathbf{y}'$. It is therefore possible to train ATNs in a self-supervised manner, where they use unlabeled data as the input and make $\operatorname{argmax}\ f(g_{f,t}(\mathbf{x})) = t$.

Reranking function. There are a variety of options for the reranking function. The simplest is to set $r(\mathbf{y}, t) = \operatorname{onehot}(t)$, but other formulations can make better use of the signal already present in $\mathbf{y}$ to encourage better reconstructions. In this work, we look at reranking functions that attempt to keep $r(\mathbf{y}, t) \sim \mathbf{y}$. In particular, we use $r(\cdot)$ that maintains the rank order of all but the targeted class in order to minimize distortions when computing $\mathbf{x}' = g_{f,t}(\mathbf{x})$.

The specific $r(\cdot)$ used in our experiments has the following form:

$$r_\alpha(\mathbf{y}, t) = norm\left(\left\{\begin{array}{rl} \alpha * \max \mathbf{y} & \text{if } k = t \\ y_k & \text{otherwise} \end{array}\right\}_{k \in \mathbf{y}}\right) \tag{4}$$

$\alpha > 1$ is an additional parameter specifying how much larger y_t should be than the current max classification. $norm(\cdot)$ is a normalization function that rescales its input to be a valid probability distribution.

2.1. Adversarial Example Generation

There are two approaches to generating adversarial examples with an ATN. The ATN can be trained to generate just the perturbation to $\mathbf{x}$, or it can be trained to generate an *adversarial autoencoding* of $\mathbf{x}$.

- **Perturbation ATN (*P-ATN*):** To just generate a perturbation, it is sufficient to structure the ATN as a variation on the residual block (He et al., 2015): $g_f(\mathbf{x}) = \tanh(\mathbf{x} + \mathcal{G}(\mathbf{x}))$, where $\mathcal{G}(\cdot)$ represents the core function of g_f. With small initial weight vectors, this structure makes it easy for the network to learn to generate small, but effective, perturbations.

- **Adversarial Autoencoding (*AAE*):** AAE ATNs are similar to standard autoencoders, in that they attempt to accurately reconstruct the original input, subject to regularization, such as weight decay or an added noise signal. For AAE ATNs, the regularizer is $L_{\mathcal{Y}}$. This imposes an additional requirement on the AAE to add some perturbation $\mathbf{p}$ to $\mathbf{x}$ such that $r(f(\mathbf{x}')) = \mathbf{y}'$.

For both ATN approaches, in order to enforce that $\mathbf{x}'$ is a plausible member of $\mathcal{X}$, the ATN should only generate values in the valid input range of f. For images, it suffices to set the activation function of the last layer to be the $tanh$ function; this constrains each output channel to $[-1, 1]$.

Table 1. Baseline Accuracy of Five MNIST Classifiers

Architecture	Acc.
Classifier-Primary (Classifier$_p$) (5x5 Conv)→ (5x5 Conv)→ FC→ FC	98.6%
Classifier-Alternate-0 (Classifier$_{a0}$) (5x5 Conv)→ (5x5 Conv)→ FC → FC	98.5%
Classifier-Alternate-1 (Classifier$_{a1}$) (4x4 Conv)→ (4x4 Conv)→ (4x4 Conv)→ FC → FC	98.9%
Classifier-Alternate-2 (Classifier$_{a2}$) (3x3 Conv)→ (3x3 Conv)→ (3x3 Conv) → FC → FC	99.1%
Classifier-Alternate-3 (Classifier$_{a3}$) (3x3 Conv) → FC → FC → FC	98.5%

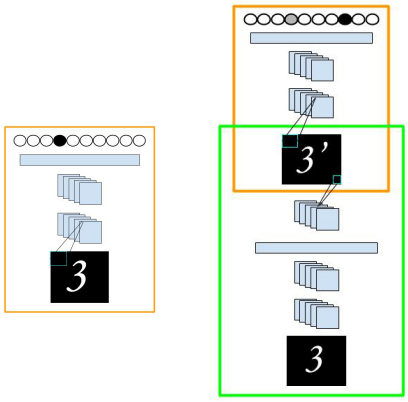

Figure 1. (Left) A simple classification network which takes input image $\mathbf{x}$. (Right) With the same input, $\mathbf{x}$, the ATN emits $\mathbf{x}'$, which is fed into the classification network. In the example shown, the input digit is classified correctly as a 3 (on the left), ATN$_7$ takes $\mathbf{x}$ as input and generates a modified image ($3'$) such that the classifier outputs a 7 as the highest activation and the previous highest classification, 3, as the second highest activation (on the right).

2.2. Related Network Architectures

This training objective resembles standard Generative Adversarial Network training (Goodfellow et al., 2014a) in that the goal is to find weaknesses in the classifier. It is interesting to note the similarity to work outside the adversarial training paradigm — the recent use of feed-forward neural networks for artistic style transfer in images (Gatys et al., 2015)(Ulyanov et al., 2016). Gatys et al. (2015) originally proposed a gradient descent procedure based on "back-driving networks" (Linden & Kindermann, 1989) to modify the inputs of a fully-trained network to find a set of inputs that maximize a desired set of outputs and hidden unit activations. Unlike standard network training in which the gradients are used to modify the weights of the network, here, the network weights are frozen and the input itself is changed. In subsequent work, Ulyanov et al. (2016) created a method to approximate the results of the gradient descent procedure through the use of an off-line trained neural network. Ulyanov et al. (2016) removed the need for a gradient descent procedure to operate on every source image to which a new artistic style was to be applied, and replaced it with a single forward pass through a separate network. Analagously, we do the same for generating adverarial examples: a separately trained network approximates the usual gradient descent procedure done on the target network to find adversarial examples.

3. MNIST Experiments

To begin our empirical exploration, we train five networks on the standard MNIST digit classification task (LeCun et al., 1998). The networks are trained and tested on the same data; they vary only in the weight initialization and architecture, as shown in Table 1. Each network has a mix of convolution (Conv) and Fully Connected (FC) layers. The input to the networks is a 28x28 grayscale image and the output is 10 logit units. Classifier$_p$ and Classifier$_{a0}$ use the same architecture, and only differ in the initialization of the weights. We will primarily use Classifier$_p$ for the experiments in this section. The other networks will be used later to analyze the generalization capabilities of the adversaries. Table 1 shows that all of the networks perform well on the digit recognition task.[3]

We attempt to create an Adversarial Autoencoding ATN that can target a specific class given any input image. The ATN is trained against a particular classifier as illustrated in Figure 1. The ATN takes the original input image, $\mathbf{x}$, as input, and outputs a new image, $\mathbf{x}'$, that the target classifier should erroneously classify as t. We also add the constraint that the ATN should maintain the ordering of all the other classes as initially output by the classifier. We train ten ATNs against Classifier$_p$ – one for each target digit, t.

An example is provided to make this concrete. If a classifier is given an image, $\mathbf{x}_3$, of the digit 3, a successful ordering of the outputs (from largest to smallest) may be as follows: Classifier$_p(\mathbf{x}_3) \rightarrow [3, 8, 5, 0, 4, 1, 9, 7, 6, 2]$. If ATN$_7$ is applied to $\mathbf{x}_3$, when the resulting image, $\mathbf{x'}_3$, is fed into the same classifier, the following ordering of outputs is desired (note that the 7 has moved to the highest output): Classifier$_p$(ATN$_7(\mathbf{x}_3)) \rightarrow [7, 3, 8, 5, 0, 4, 1, 9, 6, 2]$.

Training for a single ATN$_t$ proceeds as follows. The weights of Classifier$_p$ are frozen and never change during ATN training. Every training image, $\mathbf{x}$, is passed through Classifier$_p$ to obtain output $\mathbf{y}$. As described in Equation 4, we then compute $r_\alpha(\mathbf{y}, t)$ by copying $\mathbf{y}$ to a new value, $\mathbf{y}'$,

[3]It is easy to get better performance than this on MNIST, but for these experiments, it was more important to have a variety of architectures that achieved similar accuracy, than to have state-of-the-art performance.

Table 2. Average success of ATN_{0-9} at transforming an image such that it is misclassified by Classifier_p. As β is reduced, the ability to fool Classifier_p increases. How to read the table: Top row of cell: percentage of times Classifier_p labeled $\mathbf{x}'$ as t. Middle row of cell: percentage of times Classifier_p labeled $\mathbf{x}'$ as t and kept the original classification ($\text{argmax}\,\mathbf{y}$) in second place. Bottom row of cell: percentage of all $\mathbf{x}'$ that kept the original classification in second place.

	β: 0.010	0.005	0.001
ATN_a	**69.1%**	**84.1%**	**95.9%**
FC $\rightarrow$ FC$\rightarrow$28x28 Image	91.7%	93.4%	95.3%
	63.5%	78.6%	91.4%
ATN_b	**61.8%**	**77.7%**	**89.2%**
(3x3 Conv)$\rightarrow$ (3x3 Conv) $\rightarrow$	93.8%	95.8%	97.4%
(3x3 Conv) $\rightarrow$ FC $\rightarrow$ 28x28 Image	58.7%	74.5%	86.9%
ATN_c	**66.6%**	**82.5%**	**91.4%**
(3x3 Conv)$\rightarrow$ (3x3 Conv)$\rightarrow$(3x3 Conv)	95.5%	96.6%	97.5%
$\rightarrow$ Deconv: 7x7 $\rightarrow$ Deconv: 14x14 $\rightarrow$ 28x28 Image	64.0%	79.7%	89.1%

setting $y'_t = \alpha * \max(\mathbf{y})$, and then renormalizing $\mathbf{y}'$ to be a valid probability distribution. This sets the target class, t, to have the highest value in $\mathbf{y}'$ while maintaining the relative order of the other original classifications. In the MNIST experiments, we empirically set $\alpha = 1.5$.

Given $\mathbf{y}'$, we can now train ATN_t to generate $\mathbf{x}'$ by minimizing $\beta * L_{\mathcal{X}} = \beta * L_2(\mathbf{x}, \mathbf{x}')$ and $L_{\mathcal{Y}} = L_2(\mathbf{y}, \mathbf{y}')$ using Equation 2. Though the weights of Classifier_p are frozen, error derivatives are still passed through them to train the ATN. We explore several values of β to balance the two loss functions. The results are shown in Table 2.

Experiments. We tried three ATN architectures for the AAE task, and each was trained with three values of β against all ten targets, t. The full 3×3 set of experiments are shown in Table 2. The accuracies shown are the ability of ATN_t to transform an input image $\mathbf{x}$ into $\mathbf{x}'$ such that Classifier_p mistakenly classifies $\mathbf{x}'$ as t.[4] Each measurement in Table 2 is the average of the 10 networks, ATN_{0-9}.

Results. In Figure 2(top), each row represents the transformation that ATN_t makes to digits that were initially correctly classified as 0-9 (columns). For example, in the top row, the digits 1-9 are now all classified as 0. In all cases, their second highest classification is the original correct classification (0-9).

The reconstructions shown in Figure 2(top) have the largest β; smaller β values are shown in the bottom row. The fidelity to the underlying digit diminishes as β is reduced. However, by loosening the constraints to stay similar to the original input, the number of trials in which the trans-

[4]Images that were originally classified as t were not counted in the test as no transformation on them was required.

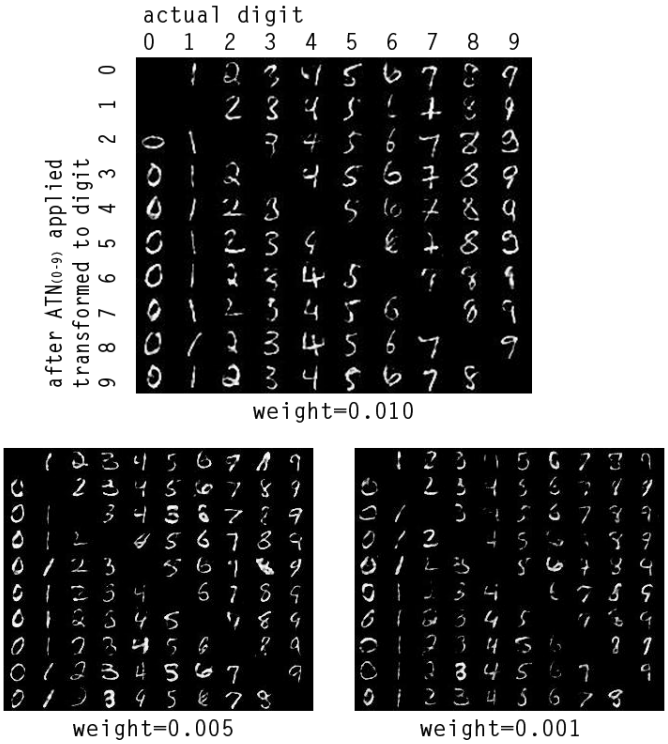

Figure 2. Successful adversarial examples from ATN_t against Classifier_p. Top is with the highest $\beta = 0.010$. Bottom two are with $\beta = 0.005$ & 0.001, respectively. Note that as β is decreased, the fidelity to the underlying digit decreases. The column in each block corresponds to the correct classification of the image. The row corresponds to the adversarial classification, t.

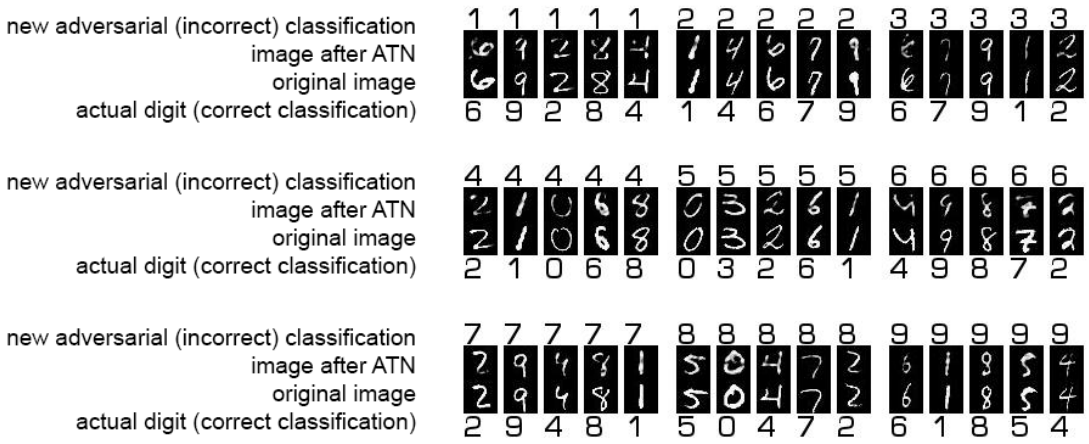

Figure 3. Typical transformations made to MNIST digits against Classifier$_p$. Black digits on the white background are output classifications from Classifier$_p$. The bottom classification is the original (correct) classification. The top classification is the result of classifying the adversarial example. White digits on black backgrounds are the MNIST digits and their transformations to adversarial examples. The bottom MNIST digits are unmodified, and the top are adversarial. In all of these images, the adversarial example is classified as $t = \text{argmax}\,\mathbf{y}'$ while maintaining the second highest output in $\mathbf{y}'$ as the original classification, $\text{argmax}\,\mathbf{y}$.

former network is able to successfully "fool" the classification network increases dramatically, as seen in Table 2. Interestingly, with $\beta = 0.010$, in Figure 2(second row), where there should be a '0' that is transformed into a '1', no digit appears. With this high β, no example was found that could be transformed to successfully fool Classifier$_p$. With the two smaller β values, this anomaly does not occur.

In Figure 3, we provide a closer look at examples of $\mathbf{x}$ and $\mathbf{x}'$ for ATN$_c$ with $\beta = 0.005$. A few points should be noted:

- The transformations maintain the large, empty regions of the image. Unlike many previous studies in attacking classifiers, the addition of salt-and-pepper type noise did not appear (Nguyen et al., 2014; Moosavi-Dezfooli et al., 2016b).
- In the majority of the generated examples, the shape of the digit does not dramatically change. This is the desired behavior: by training the networks to maintain the order beyond the top-output, only minimal changes should be made to the image. The changes that are often introduced are patches where the light strokes have become darker.
- Vertical-linear components of the original images are emphasized in several digits; it is especially noticeable in the digits transformed to 1. With other digits (e.g., 8), it is more difficult to find a consistent pattern of what is being (de)emphasized to cause the classification network to be fooled.

Table 3. Rank Difference in Secondary Outputs, Pre/Post Transformation. Top-5 (Top-9).

	β: 0.010	0.005	0.001
ATN$_a$	0.93 (0.99)	0.98 (1.04)	1.04 (1.13)
ATN$_b$	0.81 (0.87)	0.83 (0.89)	0.86 (0.93)
ATN$_c$	0.79 (0.85)	0.83 (0.90)	0.89 (0.97)

A novel aspect of ATNs is that though they cause the target classifier to output an erroneous top-class, they are also trained to ensure that the transformation preserves the existing output ordering of the target-classifier (other than the top-class). For the examples that were successfully transformed, Table 3 gives the average rank-difference of the outputs with the pre-and-post transformed images (excluding the intentional targeted misclassification).

4. A Deeper Look into ATNs

This section explores three extensions to the basic ATNs: increasing the number of networks the ATNs can attack, using hidden state from the target network, and using ATNs in serial and parallel.

4.1. Adversarial Transfer to Other Networks

So far, we have examined ATNs in the context of attacking a single classifier. Can ATNs create adversarial exam-

Table 4. ATN_b with $\beta = 0.005$ trained to defeat Classifier_p. Tested on 5 classifiers, without further training, to measure transfer. 1st place is the percentage of times t was the top classification. 2nd place measures how many times the original top class ($\text{argmax}\,\mathbf{y}$) was correctly placed into 2nd place, conditioned on the 1st place being correct (Conditional) or unconditioned on 1st place (Unconditional).

	Classifier_p*	Classifier_{a0}	Classifier_{a1}	Classifier_{a2}	Classifier_{a3}
1st Place Correct	82.5%	15.7%	16.1%	7.7%	28.9%
2nd Place Correct (Conditional)	96.6%	84.7%	89.3%	85.0%	81.8%
2nd Place Correct (Unconditional)	79.7%	15.6%	16.1%	8.4%	26.2%

ples that generalize to other classifiers? Much research has studied adversarial transfer for traditional adversaries, including the recent work of Moosavi-Dezfooli et al. (2016a); Liu et al. (2016).

Targeting multiple networks. To test transfer, we take the adversarial examples from the previously trained ATNs and test them against $\text{Classifier}_{a0,a1,a2,a3}$ (described in Table 1).

The results in Table 4 clearly show that the transformations made by the ATN *are not* general; they are tied to the network it is trained to attack. Even Classifier_{a0}, which has the same architecture as Classifier_p, is not more susceptible to the attacks than those with different architectures. Looking at the second place correctness scores (in the same Table 4), it may, at first, seem counter-intuitive that the conditional probability of a correct second-place classification remains high despite a low first-place classification. The reason for this is that in the few cases in which the ATN was able to successfully change the classifier's top choice, the second choice (the real classification) remained a close second (i.e., the image was not transformed in a large manner), thereby maintaining the high performance in the conditional second rank measurement.

Training against multiple networks. Is it possible to create a network that will be able to create a single transform that can attack *multiple* networks? Will such an ATN generalize better to unseen networks? To test this, we created an ATN that receives training signals from multiple networks, as shown in Figure 4. As with the earlier training, the $L_\mathcal{X}$ reconstruction error remains.

The new ATN was trained with classification signals from three networks: Classifier_p, and $\text{Classifier}_{a1,2}$. The training proceeds in exactly the same manner as described earlier, except the ATN attempts to minimize $L_\mathcal{Y}$ for all three target networks at the same time. The results are shown in Table 5. First, examine the columns corresponding to the networks that were used in the training (marked with an *). Note that the success rates of attacking these three classifiers are consistently high, comparable with those when

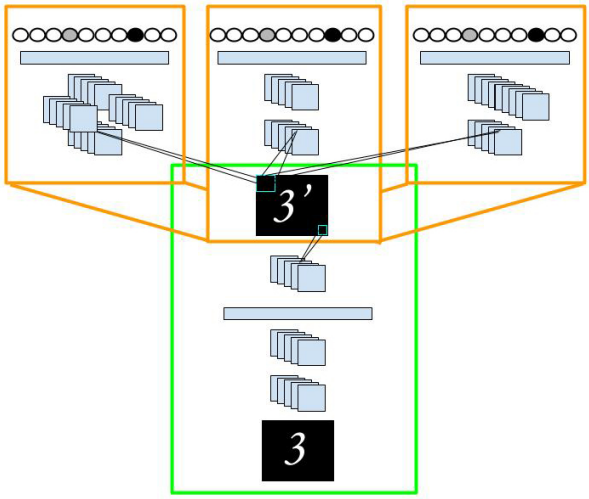

Figure 4. The ATN now has to fool three networks (of various architectures), while also minimizing $L_\mathcal{X}$, the reconstruction error.

the ATN was trained with a single network. Therefore, it is possible to learn a transformation network that modifies images such that perturbation defeats multiple networks.

Next, we turn to the remaining two networks to which the adversary was *not* given access during training. There is a large increase in success rates over those when the ATN was trained with a single target network (Table 4). However, the results do not match those of the networks used in training. It is possible that training against larger numbers of target networks at the same time could further increase the transferability of the adversarial examples.

Finally, we look at the success rates of image transformations. Do the same images consistenly fool the networks, or are the failure cases of the networks different? As shown in Figure 5, for the 3 networks the ATN was trained to defeat, the majority of transformations attacked all three networks successfully. For the unseen networks, the results were mixed; the majority of transformations successfully attacked only a single network.

Table 5. ATN_b retrained with 3 networks (marked with *).

β		Classifier_p*	Classifier_{a0}	Classifier_{a1}*	Classifier_{a2}*	Classifier_{a3}
0.010	1st Place Correct	89.9%	37.9%	83.9%	78.7%	70.2%
	2nd Place Correct (Conditional)	96.1%	88.1%	96.1%	95.2%	79.1%
	2nd Place Correct (Unconditional)	86.4%	34.4%	80.7%	74.9%	55.9%
0.005	1st Place Correct	93.6%	34.7%	88.1%	82.7%	64.1%
	2nd Place Correct (Conditional)	96.8%	88.3%	96.9%	96.4%	73.1%
	2nd Place Correct (Unconditional)	90.7%	31.4%	85.3%	79.8%	47.2%

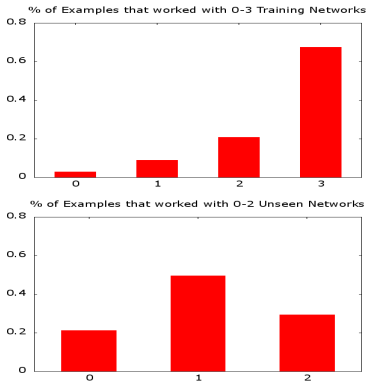

Figure 5. Do the same transformed examples work well on all the networks? (Top) Percentage of examples that worked on exactly 0-3 training networks. (Bottom) Percentage of examples that worked on exactly 0-2 unseen networks. Note: these are all measured on independent test set images.

4.2. "Insider" Information

In the experiments thus far, the classifier, C, was treated as a white box. From this box, two pieces of information were needed to train the ATN. First, the actual outputs of C were used to create the new target vector. Second, the error derivatives from the new target vector were passed through C and propagated into the ATN.

In this section, we examine the possibility of "opening" the classifier, and accessing more of its internal state. From C, the actual *hidden unit activations* for each example are used as additional inputs to the ATN. Intuitively, because the goal is to maintain as much similarity as possible to the original image and to maintain the same order of the non-top-most classifications as the original image, access to these activations may convey usable signals.

Because of the very large number of hidden units that accompany convolution layers, in practice, we only use the penultimate fully-connected layer from C. The results of training the ATNs with this extra information are shown in Table 6. Interestingly, the most salient difference does not come from the ability of the ATN to attack the networks in the first-position. Rather, when looking at the conditional-successes of the second-position, the numbers are improved (compare to Table 2). We speculate that this is because the extra hints provided by the classifier's internal activations (with the unmodified image) could be used to also ensure that the second-place classification, after input modification, was also correctly maintained.

Table 6. Using the internal states of the classifier *as inputs* for the Adversary Networks. Larger font is the percentage of times the adversarial class was classified in the top-space. Smaller font is how many times the original top class was correctly placed into 2nd place, conditioned on the 1st place being correct or not.

	β:		
	0.010	0.005	0.001
ATN_a	68.0% (94.5%/64.5%)	81.4% (96.0%/78.1%)	95.4% (98.1%/93.6%)
ATN_b	68.1% (96.9%/66.5%)	78.9% (98.1%/77.4%)	92.4% (98.9%/91.4%)
ATN_c	67.9% (97.6%/66.4%)	81.0% (98.2%/79.5%)	93.1% (99.0%/92.1%)

4.3. Serial and Parallel ATNs

Separate ATNs are created for each digit (0-9). In this section, we examine whether the ATNs can be used in parallel (can the same original image be transformed by each of the ATNs successfully?) and in serial (can the same image be transformed by one ATN then that resulting image be transformed by another, successfully?).

In the first test, we started with 1000 images of digits from the test set. Each was passed through all 10 ATNs (ATN_c, $\beta = 0.005$); the resulting images were then classified with Classifier_p. For each image, we measured how many ATNs were able to successfully transform the image (success is defined for ATN_t as causing the classifier to output t as the top-class). Out of the 1000 trials, 283 were successfully

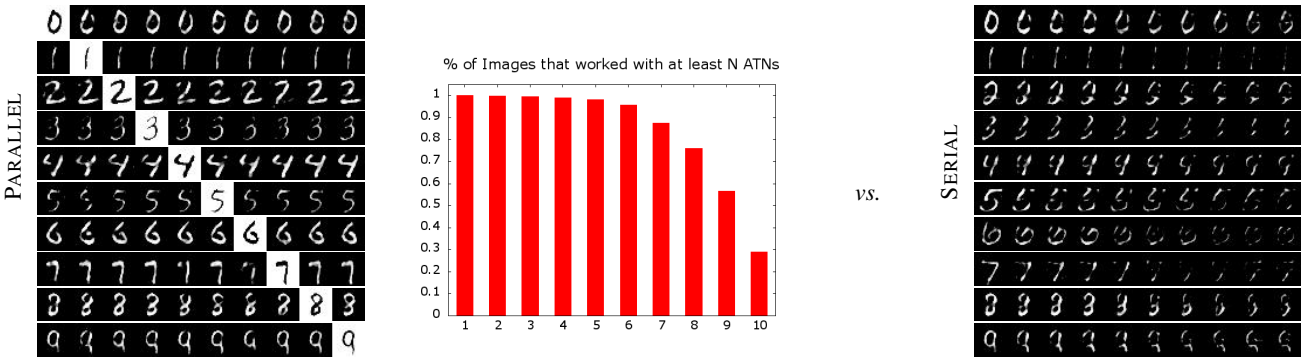

Figure 6. Parallel and Serial Application of 10 ATNs. *Left:* Examples of the same original image (shown in white background) transformed correctly by all ATNs. For example, in the row of 7s, in the first column, the 7 was transformed such that the classifier output a 0 as top class, in the second column, the classifier output a 1, etc. *Middle:* Histogram showing the number of images that were transformed successfully with at least N ATNs (1-10) when used in parallel. *Right:* Serial Adversarial Transformation Networks. In the first column, ATN_0 is applied to the input image. In the second column, ATN_1 is applied to the output of ATN_0, etc. In each of these examples, all 10 of the ATNs successfully transformed the previous image to fool the classifier. Note the severe image degradation as the transformation networks are applied in sequence.

transformed by all 10 of the ATNs. Samples results and a histogram of the results are shown in Figure 6.

A second experiment is constructed in which the 10 ATNs are applied *serially, one-after-the-other*. In this scenario, first ATN_0 is applied to image $\mathbf{x}$, yielding $\mathbf{x}'$. Then ATN_1 is applied to $\mathbf{x}'$ yielding $\mathbf{x}''$ ... to ATN_9. The goal is to see whether the transformations work on previously transformed images. The results of chaining the ATNs together in this manner are shown in Figure 6(right). The more transformations that are applied, the larger the image degradation. As expected, by the ninth transformation (rightmost column in Figure 6) the majority of images are severely degraded and usually not recognizable. Though we expected the degradation in images, there were two additional, surprising, findings. First, in the parallel application of ATNs (the first experiment described above), out of 1000 images, 283 of them were successfully transformed by 10 of the ATNs. In this experiment, 741 images were successfully transformed by 10 ATNs. The improvement in the number of all-10 successes over applying the ATNs in parallel occurs because each transformation effectively diminishes the underlying original image (to remove the real classification from the top-spot). Meanwhile, only a few new pixels are added by the ATN to cause the misclassification as it is also trained to minimize the reconstruction error. The overarching effect is a fading of the image through chaining ATNs together.

Second, it is interesting to examine what happens to the second-highest classifications that the networks were also trained to preserve. Order preservation *did not* occur in this test. Had the test worked perfectly, then for an input-image, $\mathbf{x}$ (e.g., of the digit 8), after ATN_0 was applied, the first and second top classifications of $\mathbf{x}'$ should be 0,8, respectively. Subsequently, after ATN_1 is then applied to $\mathbf{x}'$, the classifications of $\mathbf{x}''$ should be 1,0,8, etc. The reason this does not hold in practice is that though the networks were trained to maintain the high classification (8) of the original digit, $\mathbf{x}$, they were *not trained to maintain the potentially small perturbations* that ATN_0 made to $\mathbf{x}$ to achieve a top-classification of 0. Therefore, when ATN_1 is applied, the changes that ATN_0 made may not survive the transformation. Nonetheless, if chaining adversaries becomes important, then training the ATNs with images that have been previously modified by other ATNs may be a sufficient method to address the difference in training and testing distributions. This is left for future work.

5. ImageNet Experiments

We explore the effectiveness of ATNs on the ImageNet dataset (Deng et al., 2009), which consists of 1.2 million natural images categorized into 1 of 1000 classes. The target classifier, f, used in these experiments is a pre-trained state-of-the-art classifier, Inception ResNet v2 (IR2), that has a top-1 single-crop error rate of 19.9% on the 50,000 image validation set, and a top-5 error rate of 4.9%. It is described fully in Szegedy et al. (2016).

5.1. Experiment Setup

We trained AAE ATNs and P-ATNs as described in Section 2 to attack IR2. Training an ATN against IR2 follows the process described in Section 3.

IR2 takes as input images scaled to 299×299 pixels of 3 channels each. To autoencode images of this size for the

AAE task, we use three different fully convolutional architectures (Table 7):

- *IR2-Base-Deconv*, a small architecture that uses the first few layers of IR2 and loads the pre-trained parameter values at the start of training the ATN, followed by deconvolutional layers;
- *IR2-Resize-Conv*, a small architecture that avoids checkerboard artifacts common in deconvolutional layers by using bilinear resize layers to downsample and upsample between stride 1 convolutions; and
- *IR2-Conv-Deconv*, a medium architecture that is a tower of convolutions followed by deconvolutions.

For the perturbation approach, we use *IR2-Base-Deconv* and *IR2-Conv-FC*, which has many more parameters than the other architectures due to two large fully-connected layers. The use of fully-connected layers cause the network to learn too slowly for the autoencoding approach (AAE ATN), but can be used to learn perturbations quickly (P-ATN).

Hyperparameter search. All five architectures across both tasks are trained with the same hyperparameters. For each architecture and task, we trained four networks, one for each target class: binoculars, soccer ball, volcano, and zebra. In total, we trained 20 different ATNs to attack IR2.

To find a good set of hyperparameters for these networks, we did a series of grid searches through reasonable parameter values for learning rate, α, and β, using only Volcano as the target class. Those training runs were terminated after 0.025 epochs, which is only 1600 training steps with a batch size of 20. Based on the parameter search, for the results reported here, we set the learning rate to 0.0001, $\alpha = 1.5$, and $\beta = 0.01$. All runs were trained for 0.1 epochs (6400 steps) on shuffled training set images, using the Adam optimizer and the TensorFlow default settings.

In order to avoid cherrypicking the best results after the networks were trained, we selected four images from the unperturbed validation set to use for the figures in this paper prior to training. Once training finished, we evaluated the ATNs by passing 1000 images from the validation set through the ATN and measuring IR2's accuracy on those adversarial examples.

5.2. Results Overview

Table 8 shows the top-1 adversarial accuracy for each of the 20 model/target combinations. The AAE approach is superior to the perturbation approach, both in terms of top-1 adversarial accuracy, and in terms of training success. Nonetheless, the results in Figures 9 and 7 show that using an architecture like *IR2-Conv-FC* can provide a qualitatively different type of adversary from the AAE approach.The examples generated using the perturbation approach preserve more pixels in the original image, at the expense of a small region of large perturbations.

In contrast to the perturbation approaches, the AAE architectures distribute the differences across wider regions of the image. However, *IR2-Base-Deconv* and *IR2-Conv-Deconv* tend to exhibit checkerboard patterns, which is a common problem in image generation with deconvolutions (Odena et al. (2016)). The checkerboarding led us to try *IR2-Resize-Conv*, which avoids the checkerboard pattern, but gives smooth outputs (Figure 9). Interestingly, in all three AAE networks, many of the original high-frequency patterns are replaced with high frequencies that encode the adversarial signal.

The results from *IR2-Base-Deconv* show that the same network architectures perform substantially differently when trained as P-ATNs and AAE ATNs. Since P-ATNs are only learning to perturb the input, these networks are much better at preserving the original image, but the perturbations end up being focused along the edges or in the corners of the image. The form of the perturbations often manifests itself as "DeepDream"-like images, as in Figure 8. Approximately the same perturbation, in the same place, is used across all input examples. Placing the perturbations in that manner is less likely to disrupt the other top classifications, thereby keeping $L_{\mathcal{Y}}$ lower. This is in stark contrast to the AAE ATNs, which creatively modify the input, as seen in Figures 9 and 7.

5.3. Detailed Discussion

Adversarial diversity. Figure 7 shows that ATNs are capable of generating a wide variety of adversarial perturbations targeting a single network. Previous approaches to generating adversarial examples often produced qualitatively uniform results – they add various amounts of "noise" to the image, generally concentrating the noise at pixels with large gradient magnitude for the particular adversarial loss function. Indeed, Hendrik Metzen et al. (2017) recently showed that it may be possible to train a detector for previous adversarial attacks. From the perspective of an attacker, then, adversarial examples produced by ATNs may provide a new way past defenses in the cat-and-mouse game of security, since this somewhat unpredictable diversity will likely challenge such approaches to defense. Perhaps a much more interesting consequence of this diversity is its potential application for more comprehensive adversarial training, as described below.

Adversarial Training with ATNs. In Kurakin et al. (2016b), the authors show the current state-of-the-art in

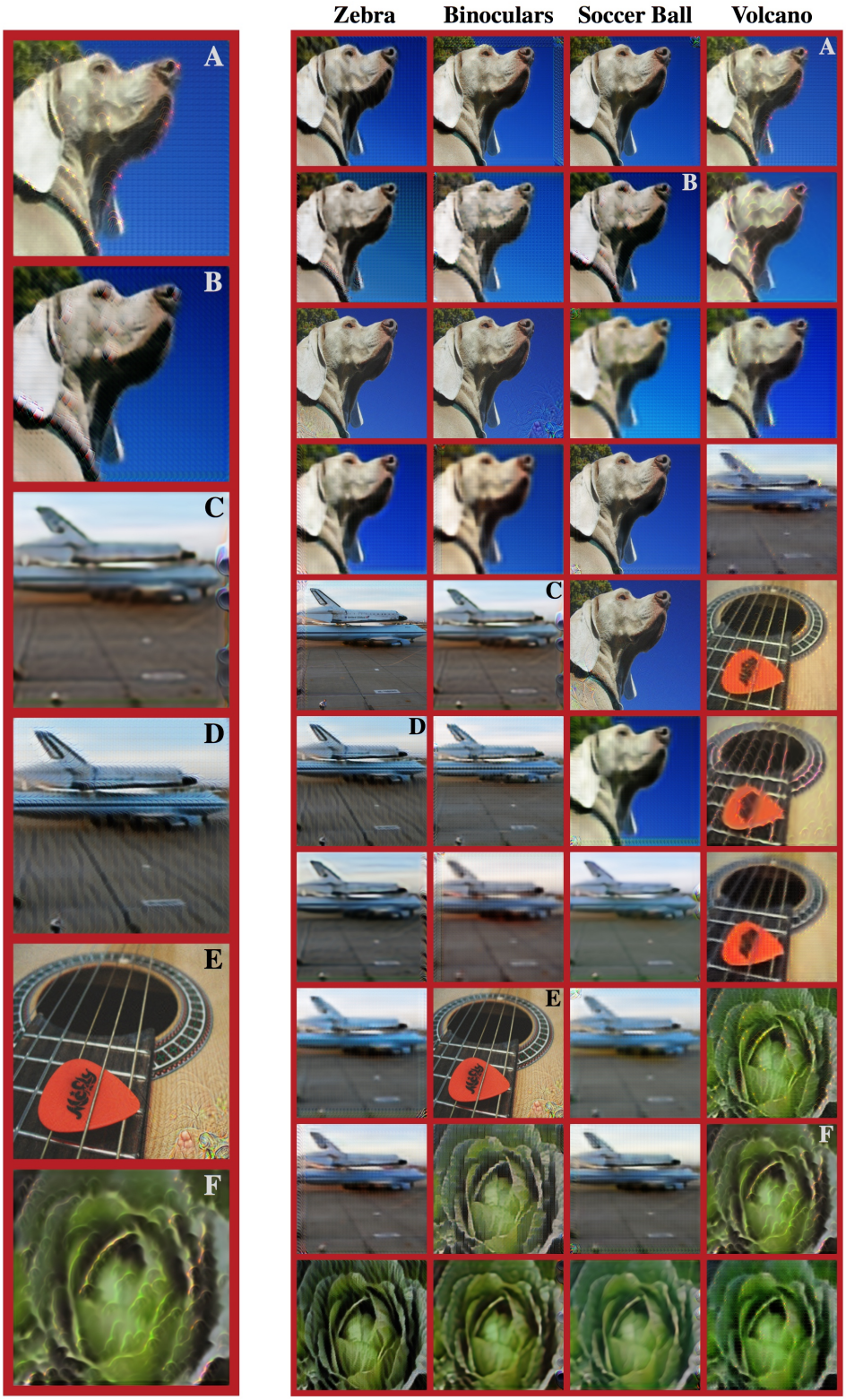

Figure 7. **Adversarial diversity.** Left column: selected zoomed samples. Right 4 columns: successful adversarial examples for different target classes from a variety of ATNs. From the left: Zebra, Binoculars, Soccer Ball, Volcano. These images were selected at random from the set of successful adversaries against each target class. Unlike existing adversarial techniques, where adversarial examples tend to look alike, these adversarial examples exhibit a great deal of diversity, some of which is quite surprising. For example, consider the second image of the space shuttle in the "Zebra" column (D). In this case, the ATN made the lines on the tarmac darker and more organic, which is somewhat evocative of a zebra's stripes. Yet clearly no human would mistake this for an image of a zebra. Similarly, the dog's face in (A) has been speckled with a few orange dots (but not the background!), and these are sufficient to convince IR2 that it is a volcano. This diversity may be a key to improving the effectiveness of adversarial training, as a more diverse pool of adversarial examples may lead to better network generalization. Images A, B, and D are from AAE ATN *IR2-Conv-Deconv*. Images C and F are from AAE ATN *IR2-Resize-Conv*. Image E is from P-ATN *IR2-Conv-FC*.

暗

停机坪

using adveraries for improving training. With single step and iterative gradient methods, they find that it is possible to increase a network's robustness to adversarial examples, while suffering a small loss of accuracy on clean inputs. However, it works only for the adversary the network was trained against. It appears that ATNs could be used in their adversarial training architecture, and could provide substantially more diversity to the trained model than current adversaries. This adversarial diversity might improve model test-set generalization and adversarial robustness.

Because ATNs are quick to train relative to the target network (in the case of IR2, hours instead of weeks), reliably produce diverse adversarial examples, and can be automatically checked for quality (by checking their success rate against the target network and the $L_{\mathcal{X}}$ magnitude of the adversarial examples), they could be used as follows: Train a set of ATNs targeting a random subset of the output classes on a checkpoint of the target network. Once the ATNs are trained, replace a fraction of each training batch with corresponding adversarial examples, subject to two constraints: the current classifier incorrectly classifies the adversarial example as the target class, and the $L_{\mathcal{X}}$ loss of the adversarial example is below a threshold that indicates it is similar to the original image. If a given ATN stops producing successful adversarial examples, replace it with a newly trained ATN targeting another randomly selected class. In this manner, throughout training, the target network would be exposed to a shifting set of diverse adversaries from ATNs that can be trained in a fully-automated manner.[5,6]

DeepDream perturbations. *IR2-Conv-FC* exhibits interesting behavior not seen in any of the other architectures. The network builds a perturbation that generally contains spatially coherent, recognizable regions of the target class. For example, in Figure 8, a consistent soccer-ball "ghost" image appears in all of the transformed images. While the methods and goals of these perturbations are quite different from those generated by DeepDream (Mordvintsev et al., 2015), the qualitative results appear similar. *IR2-Conv-FC* seems to learn to distill the target network's representation of the target class in a manner that can be drawn across a large fraction of the image.[7] This result hints at a direct relationship between DeepDream-style techniques and adversarial examples that may improve our ability to find and correct weaknesses in our models.

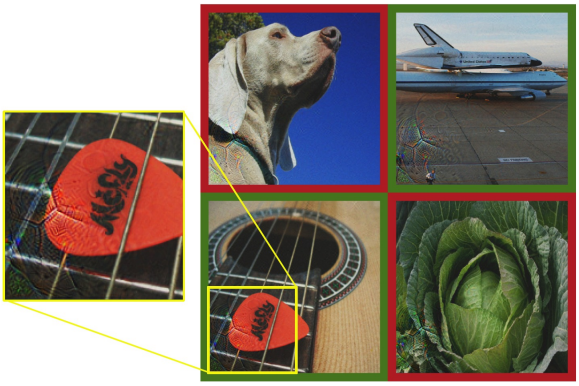

Figure 8. **DeepDream-style perturbations.** Four different images perturbed by *IR2-Conv-FC*, targeting soccer ball. The images outlined in red were successful adversarial examples against IR2. The images outlined in green did not change IR2's top-1 classification. The network has learned to add approximately the same perturbation to all images. The perturbation resembles part of a soccer ball (lower-left corner). The results are akin to those found in DeepDream-like processes (Mordvintsev et al., 2015).

High frequency data. The AAE ATNs all remove high frequency data from the images when building their reconstructions. This is likely to be due to limitations of the underlying architectures. In particular, all three convolutional architectures have difficulty exactly recreating edges from the input image, due to spatial data loss introduced when downsampling and padding. Consequently, the $L_{\mathcal{X}}$ loss penalizes high confidence predictions of edge locations, leading the networks to learn to smooth out boundaries in the reconstruction. This strategy minimizes the overall loss, but it also places a lower bound on the error imposed by pixels in regions with high frequency information.

This lower bound on the loss in some regions provides the network with an interesting strategy when generating an AAE output: it can focus the adversarial perturbations in regions of the input image that have high-frequency noise. This strategy is visible in many of the more interesting images in Figure 7. For example, many of the networks make minimal modification to the sky in the dog image, but add substantial changes around the edges of the dog's face, exactly where the $L_{\mathcal{X}}$ error would be high in a non-adversarial reconstruction.

6. Conclusions and Future Work

Current methods for generating adversarial samples involve a gradient descent procedure on individual input ex-

[5]This procedure conceptually resembles GAN training (Goodfellow et al., 2014a) in many ways, but the goal is different: for GANs, the focus is on using an easy-to-train discriminator to learn a hard-to-train generator; for this adversarial training system, the focus is on using easy-to-train generators to learn a hard-to-train multi-class classifier.

[6]Note also that we can run the adversarial example generation in this algorithm on unlabeled data, as described in Section 2. Miyato et al. (2016) also describe a method for using unlabeled data in a manner conceptually similar to adversarial training.

[7]This is likely due to the final fully-connected layer, which has one weight for each pixel and channel, allowing the network to specify a particular output at each pixel.

Figure 9. **Architecture comparisons.** Left 3: Adversarial autoencoding ATNs. Right 2: Perturbation ATNs. All five networks in this figure are trained with the same hyperparameters, apart from the target class, which varies among zebra, soccer ball, and volcano. The images in the bottom row show the absolute difference between the original image (not shown) and the adversarial examples. There are substantial differences in how the different architectures generate adversarial examples. *IR2-Base-Deconv* and *IR2-Conv-Deconv* tend to exhibit checkerboard patterns in their reconstructions, which is a common problem in image generation with deconvolutions in general. *IR2-Resize-Conv* avoids the checkerboard pattern, but tends to give very smooth outputs. *IR2-Base-Deconv* performs quite differently when trained as a P-ATN rather than an AAE ATN. The P-ATNs focus their perturbations along the edges of the image or in the corners, where they are presumably less likely to disrupt the other top classifications. In both P-ATNs, it turns out that the networks learn to mostly ignore the input and simply generate a single perturbation that can be applied to any input image without much change, although the perturbations between networks on the same target image vary substantially. This is in stark contrast to the AAE ATNs, which creatively modify each input individually, as seen here and in Figure 7.

Table 7. ImageNet ATN Architectures.

IR2-Base-Deconv (3.4M parameters)	IR2 MaxPool 5a (35x35x192) → Pad (37x37x192) → Deconv (4x4x512, stride=2) → Deconv (3x3x256, stride=2) → Deconv (4x4x128, stride=2) → Pad (299x299x128) → Deconv (4x4x3) → Image (299x299x3)
IR2-Resize-Conv (3.8M parameters)	Conv (5x5x128) → Bilinear Resize (0.5) → Conv (4x4x256) → Bilinear Resize (0.5) → Conv (3x3x512) → Bilinear Resize (0.5) → Conv (1x1x512) → Bilinear Resize (2) → Conv (3x3x256) → Bilinear Resize (2) → Conv (4x4x128) → Pad (299x299x128) → Conv (3x3x3) → Image (299x299x3)
IR2-Conv-Deconv (12.8M parameters)	Conv (3x3x256, stride=2) → Conv (3x3x512, stride=2) → Conv (3x3x768, stride=2) → Deconv (4x4x512, stride=2) → Deconv (3x3x256, stride=2) → Deconv (4x4x128, stride=2) → Pad (299x299x128) → Deconv (4x4x3) → Image (299x299x3)
IR2-Conv-FC (233.7M parameters)	Conv (3x3x512, stride=2) → Conv (3x3x256, stride=2) → Conv (3x3x128, stride=2) → FC (512) → FC (268203) → Image (299x299x3)

Table 8. IL2 ATN Performance

	P-ATN Target Class Top-1 Accuracy			
	Binoculars	Soccer Ball	Volcano	Zebra
IR2-Base-Deconv	66.0%	56.5%	0.2%	43.2%
IR2-Conv-FC	**79.9%**	**78.8%**	0.0%	**85.6%**
	AAE ATN Target Class Top-1 Accuracy			
	Binoculars	Soccer Ball	Volcano	Zebra
IR2-Base-Deconv	**83.0%**	**92.1%**	88.1%	**88.2%**
IR2-Resize-Conv	69.8%	61.4%	**91.1%**	80.2%
IR2-Conv-Deconv	56.6%	75.0%	87.3%	79.1%

amples. We have presented a fundamentally different approach to finding examples by training neural networks to convert inputs into adversarial examples. Our method is efficient to train, fast to execute, and produces remarkably diverse, successful adversarial examples.

Future work should explore the possibility of using ATNs in adversarial training. A successful ATN-based system may pave the way towards models with better generalization and robustness.

Hendrik Metzen et al. (2017) recently showed that it is possible to detect when an input is adversarial, for current types of adversaries. It may be possible to train such detectors on ATN output. If so, using that signal as an additional loss for the ATN may improve the outputs. Similarly, exploring the use of a GAN discriminator during training may improve the realism of the ATN outputs. It would be interesting to explore the impact of ATNs on generative models, rather than just classifiers, similar to work in Kos et al. (2017). Finally, it may also be possible to train ATNs in a black-box manner, similar to recent work in Tramèr et al. (2016); Baluja et al. (2015), or using REINFORCE (Williams, 1992) to compute gradients for the ATN using the target network simply as a reward signal.

References

Baluja, Shumeet, Covell, Michele, and Sukthankar, Rahul. The virtues of peer pressure: A simple method for discovering high-value mistakes. In *Int. Conf. on Computer Analysis of Images and Patterns*, pp. 96–108. Springer, 2015.

Carlini, Nicholas and Wagner, David. Towards evaluating the robustness of neural networks. *arXiv preprint arXiv:1608.04644*, 2016.

Deng, Jia, Dong, Wei, Socher, Richard, Li, Li-Jia, Li, Kai, and Fei-Fei, Li. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pp. 248–255. IEEE, 2009.

Gatys, Leon A., Ecker, Alexander S., and Bethge, Matthias. A neural algorithm of artistic style. *CoRR*, abs/1508.06576, 2015. URL http://arxiv.org/abs/1508.06576.

Goodfellow, Ian, Pouget-Abadie, Jean, Mirza, Mehdi, Xu, Bing, Warde-Farley, David, Ozair, Sherjil, Courville, Aaron, and Bengio, Yoshua. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, pp. 2672–2680, 2014a.

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

He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL http://arxiv.org/abs/1512.03385.

Hendrik Metzen, J., Genewein, T., Fischer, V., and Bischoff, B. On Detecting Adversarial Perturbations. *ArXiv e-prints*, February 2017.

Johnson, Justin, Alahi, Alexandre, and Fei-Fei, Li. Perceptual losses for real-time style transfer and super-resolution. In *European Conference on Computer Vision*, 2016.

Kingma, Diederik and Ba, Jimmy. Adam: A method for stochastic optimization. 2015.

Kos, Jernej, Fischer, Ian, and Song, Dawn. Adversarial examples for generative models. *arXiv preprint arXiv:1702.06832*, 2017.

Kurakin, Alexey, Goodfellow, Ian J., and Bengio, Samy. Adversarial examples in the physical world. *CoRR*, abs/1607.02533, 2016a.

Kurakin, Alexey, Goodfellow, Ian J., and Bengio, Samy. Adversarial machine learning at scale. *CoRR*, abs/1611.01236, 2016b.

LeCun, Yann, Cortes, Corinna, and Burges, Christopher JC. The mnist database of handwritten digits, 1998.

Linden, Alexander and Kindermann, J. Inversion of multilayer nets. In *Neural Networks, 1989. International Joint Conference*, pp. 425–430. IEEE, 1989.

Liu, Yanpei, Chen, Xinyun, Liu, Chang, and Song, Dawn. Delving into transferable adversarial examples and black-box attacks. *CoRR*, abs/1611.02770, 2016. URL http://arxiv.org/abs/1611.02770.

Miyato, Takeru, Maeda, Shin-ichi, Koyama, Masanori, Nakae, Ken, and Ishii, Shin. Distributional smoothing with virtual adversarial training. In *International Conference on Learning Representations*, 2016.

Moosavi-Dezfooli, Seyed-Mohsen, Fawzi, Alhussein, Fawzi, Omar, and Frossard, Pascal. Universal adversarial perturbations. *CoRR*, abs/1610.08401, 2016a.

Moosavi-Dezfooli, Seyed-Mohsen, Fawzi, Alhussein, and Frossard, Pascal. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE CVPR*, pp. 2574–2582, 2016b.

Mordvintsev, A., Olah, C., and Tyka, M. Inceptionism: Going deeper into neural networks. http://googleresearch.blogspot.com/2015/06/inceptionism-going-deeper-into-neural.html, 2015.

Nguyen, Anh Mai, Yosinski, Jason, and Clune, Jeff. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. *CoRR*, abs/1412.1897, 2014. URL http://arxiv.org/abs/1412.1897.

Odena, Augustus, Dumoulin, Vincent, and Olah, Chris. Deconvolution and checkerboard artifacts. *Distill*, 2016. http://distill.pub/2016/deconv-checkerboard.

Papernot, Nicolas, McDaniel, Patrick, Jha, Somesh, Fredrikson, Matt, Celik, Z Berkay, and Swami, Ananthram. The limitations of deep learning in adversarial settings. In *Proceedings of the 1st IEEE European Symposium on Security and Privacy*, 2015.

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

Papernot, Nicolas, McDaniel, Patrick, Goodfellow, Ian, Jha, Somesh, Celik, Z Berkay, and Swami, Ananthram. Practical black-box attacks against deep learning systems using adversarial examples. *arXiv preprint arXiv:1602.02697*, 2016b.

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

Szegedy, Christian, Ioffe, Sergey, Vanhoucke, Vincent, and Alemi, Alex. Inception-v4, inception-resnet and the impact of residual connections on learning. *arXiv preprint arXiv:1602.07261*, 2016.

Tramèr, Florian, Zhang, Fan, Juels, Ari, Reiter, Michael K, and Ristenpart, Thomas. Stealing machine learning models via prediction apis. In *USENIX Security*, 2016.

Ulyanov, Dmitry, Lebedev, Vadim, Vedaldi, Andrea, and Lempitsky, Victor S. Texture networks: Feed-forward synthesis of textures and stylized images. *CoRR*, abs/1603.03417, 2016. URL http://arxiv.org/abs/1603.03417.

Williams, Ronald J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.