Learning to Attack: Adversarial Transformation Networks

Shumeet Baluja and **Ian Fischer**
Google Research
Google, Inc.

Abstract

With the rapidly increasing popularity of deep neural networks for image recognition tasks, a parallel interest in generating adversarial examples to attack the trained models has arisen. To date, these approaches have involved either directly computing gradients with respect to the image pixels or directly solving an optimization on the image pixels. We generalize this pursuit in a novel direction: can a separate network be trained to efficiently attack another fully trained network? We demonstrate that it is possible, and that the generated attacks yield startling insights into the weaknesses of the target network. We call such a network an Adversarial Transformation Network (ATN). ATNs transform any input into an adversarial attack on the target network, while being minimally perturbing to the original inputs and the target network's outputs. Further, we show that ATNs are capable of not only causing the target network to make an error, but can be constructed to explicitly control the type of misclassification made. We demonstrate ATNs on both simple MNIST-digit classifiers and state-of-the-art ImageNet classifiers deployed by Google, Inc.: Inception ResNet-v2.

With the resurgence of deep neural networks for many real-world classification tasks, there is an increased interest in methods to assess the weaknesses in the trained models. *Adversarial examples* are small perturbations of the inputs that are carefully crafted to fool the network into producing incorrect outputs. Seminal work by (Szegedy et al. 2013) and (Goodfellow, Shlens, and Szegedy 2014), 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.

Until now, these optimization problems have been solved using three broad approaches: (1) By directly using optimizers like L-BFGS or Adam (Kingma and Ba 2015), as proposed in (Szegedy et al. 2013) and (Carlini and Wagner 2016). (2) By approximation with single-step gradient-based techniques like fast gradient sign (Goodfellow, Shlens, and Szegedy 2014) or fast least likely class (Kurakin, Goodfellow, and Bengio 2016). (3) By approximation with iterative variants of gradient-based techniques (Kurakin, Goodfellow, and Bengio 2016; Moosavi-Dezfooli et al. 2016; Moosavi-Dezfooli, Fawzi, and Frossard 2016). These approaches use multiple forward and backward passes through the target network to more carefully move an input towards an adversarial classification. Other approaches assume a black-box model and only having access to the target model's output (Papernot et al. 2016; Baluja, Covell, and Sukthankar 2015; Tramèr et al. 2016). See (Papernot et al. 2015) for a discussion of threat models.

Each of the above approaches solved an optimization problem such that *a single set of inputs* was perturbed enough to force the target network to make a mistake. We take a fundamentally different approach: given a well-trained target network, can we create a separate, attack-network that, with high probability, minimally transforms all inputs into ones that will be misclassified? No per-sample optimization problems should be solved. The attack-network should take as input a clean image and output a minimally modified image that will cause a misclassification in the target network. Further, can we do this while imposing strict constraints on the types and amount of perturbations allowed? We introduce a class of networks, called Adversarial Transformation Networks, to efficiently address this task.

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 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 $\text{argmax}\, f(\mathbf{x}) \neq \text{argmax}\, f(\mathbf{x}')$.

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

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

where $L_{\mathcal{X}}$ is a loss function in the input space (e.g., L_2 loss or a perceptual similarity loss like (Johnson, Alahi, and Fei-Fei 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 user-specified 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 $y_k < y_t, \forall\, k \neq t$.

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; this is a novel mechanism through which we can impose strict constraints on the types and amounts of perturbations permitted.

The simplest reranking is to set $r(\mathbf{y}, t) = \operatorname{onehot}(t)$, but other formulations make better use of the implicit signal in $\mathbf{y}$ to encourage better reconstructions. Our reranking functions attempt to keep $r(\mathbf{y}, t) \sim \mathbf{y}$. This is an important, novel, aspect of our work: rather than solely enforcing small perturbations by directly minimizing L_2 changes of the input pixels, we also penalize the ATN if the ranks of the target-network's outputs are changed (except for the targeted class). In particular, we use $r(\cdot)$ that maintains the rank order of all but the targeted class in order to minimize distortions when computing $\mathbf{x}' = g_{f,t}(\mathbf{x})$.

Specifically, $r(\cdot)$ 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.

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 $[-1, 1]$.

Attacking MNIST Networks

To begin our empirical exploration, we train five networks on the standard MNIST digit classification task (LeCun, Cortes, and Burges 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 (accuracies

Table 1: Baseline Accuracy of Five MNIST Classifiers

Architecture	Accuracy
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%

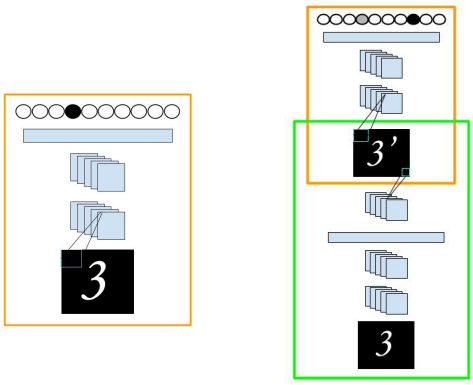

Figure 1: (Left) A simple classification network that takes input image $\mathbf{x}$. (Right) With the same input, $\mathbf{x}$, the ATN emits $\mathbf{x}'$, which is fed into the classification network. In the example shown, the input digit is classified correctly as a 3 (on the left). ATN_7 takes $\mathbf{x}$ as input and generates a modified image ($3'$). When $3'$ is input into the classifier, it outputs a 7 as the highest activation and the previous highest classification, 3, as the second highest (right).

range between 98.5 and 99.1%). Each network has a mix of convolution (Conv) and Fully Connected (FC) layers. The input is a 28x28 gray-scale image and the output is 10 logit units. Classifier_p and Classifier_{a0} use the same architecture, only differing in the weight initialization. To begin, we will use Classifier_p as the target network for the experiments.

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 reranking 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: $\text{Classifier}_p(\mathbf{x}_3) \rightarrow [3, 8, 5, 0, 4, 1, 9, 7, 6, 2]$. 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): $\text{Classifier}_p(\text{ATN}_7(\mathbf{x}_3)) \rightarrow [7, 3, 8, 5, 0, 4, 1, 9, 6, 2]$.

Training ATN_t proceeds as follows. The weights of fully-trained-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, $\mathbf{y}^r$, setting $y^r_t = \alpha * \max(\mathbf{y})$, and then renormalizing $\mathbf{y}^r$ to be a valid probability distribution. This sets the target class, t, to have the highest value in $\mathbf{y}^r$ while maintaining the relative order of the other original classifications. In the MNIST experiments, we empirically set $\alpha = 1.5$.

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. The misclassification column is the percentage of times Classifier_p labeled $\mathbf{x}'$ as t.

	Architecture	Successful Attack
ATN_a	FC → FC → 28x28 Image	69.1% (β=0.010) 84.1% (β=0.005) 95.9% (β=0.001)
ATN_b	(3x3 Conv)→ (3x3 Conv) → (3x3 Conv) → FC → 28x28 Image	61.8% (β=0.010) 77.7% (β=0.005) 89.2% (β=0.001)
ATN_c	(3x3 Conv)→ (3x3 Conv)→ (3x3 Conv) → Deconv:7x7 → Deconv:14x14 → 28x28 Image	66.6% (β=0.010) 82.5% (β=0.005) 91.4% (β=0.001)

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

Experiments. We trained 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. Each measurement in Table 2 is the average of the 10 networks, ATN_{0-9}.

Results. In Figure 2(top), each row is the transformation that ATN_t makes to digits that were initially correctly classified as 0-9 (columns). *E.g.*, 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 transformer 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.

Figure 3 provides 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 numerous previous adversarial studies, salt-and-pepper type noise was not created (Nguyen, Yosinski, and Clune 2014; Moosavi-Dezfooli, Fawzi, and Frossard 2016).

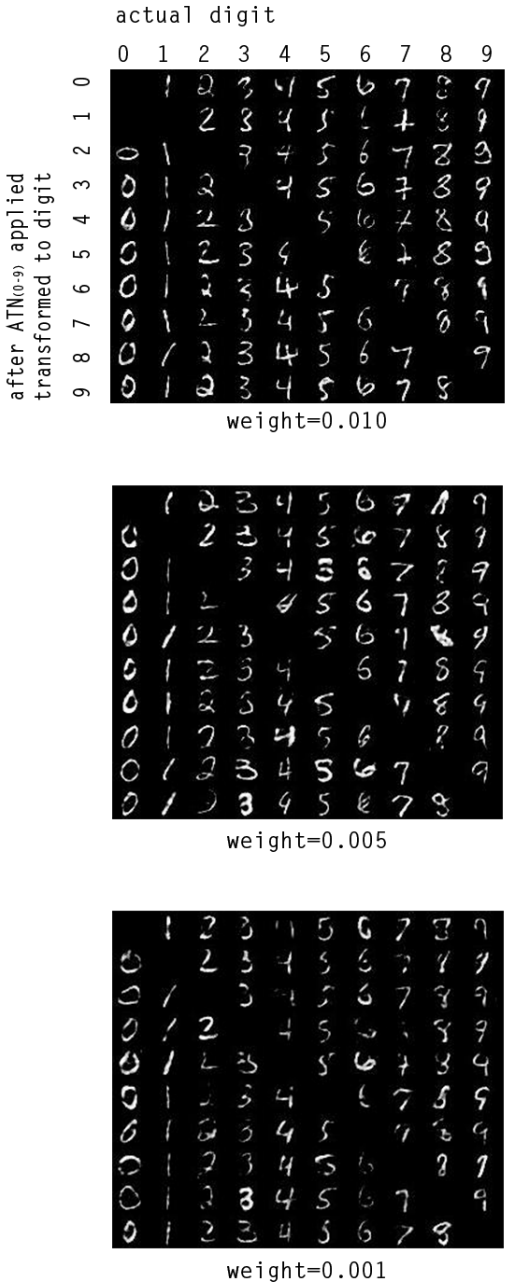

Figure 2: Successful adversarial examples from ATN_t against Classifier_p. Top is with the highest $\beta = 0.010$. Bottom two are with $\beta = 0.005$ & 0.001, respectively. Note that as β is decreased, the fidelity to the underlying digit decreases. The column in each block corresponds to the correct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

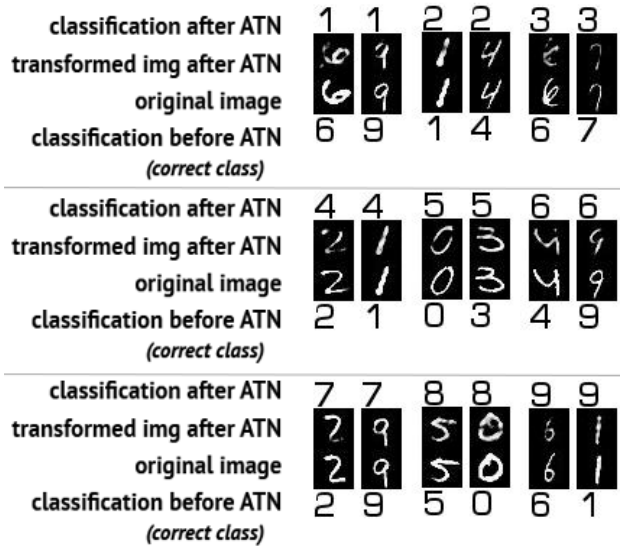

Figure 3: Typical transformations made to MNIST digits against Classifier_p. Black digits on the white background are output classifications from Classifier_p. The bottom classification is the original (correct) classification. The top classification is the result of classifying the adversarial example. 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, $\text{argmax}\,\mathbf{y}$.

- In the majority of cases, the shape of the digit does not change. This is the desired behavior: by training the networks to maintain ranks (other than the top), only minimal changes should be made.
- Vertical-linear components of the original images are emphasized in several digits; it is especially noticeable in the digits transformed to 1.

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 measures how much the classifier's outputs change when presented with the adversarial image. The first number in each cell shows how much the top-5 classifications of the original image moved, on average, when presented with the adversarial image. The second numbers show the same for all the positions (all measurements exclude the intentional target misclassification).

Adversarial Transfer to Unseen Networks

So far, we have examined ATNs in the context of attacking a single classifier. Do ATNs create adversarial examples that generalize to other classifiers (Moosavi-Dezfooli et al. 2016; Liu et al. 2016)? 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).

Table 3: How much do the ranks of the top-5 outputs change when presented with the adversarial image? In parentheses: same statistic measured on all the outputs.

	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)

The results in Table 4 clearly show that the transformations made by the ATN do not transfer; 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.

We can, however, explicitly make the ATN attack multiple networks. For 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 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.

Table 4: ATN_b with $\beta = 0.005$ trained to defeat Classifier_p. Tested on 5 classifiers, without further training, to measure transfer. "Rank-1" is the percentage of times t was the top classification. "Rank-2 (cond)" measures how many times the original top class ($\text{argmax}\,\mathbf{y}$) was correctly placed into 2nd place, when the 1st place was correct (conditional); "Rank-2 (uncond)": not conditional on 1st place being correct.

	Classifier				
	C_p*	C_{a0}	C_{a1}	C_{a2}	C_{a3}
rank 1 correct	83%	16%	16%	8%	29%
rank 2 correct (cond)	97%	85%	89%	85%	82%
rank 2 correct (uncond)	80%	16%	16%	8%	26%

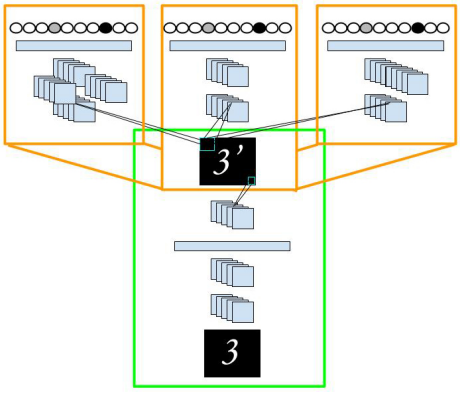

Figure 4: The ATN now has to fool three networks (of various architectures), while also minimizing $L_{\mathcal{X}}$ and maintaining ranks of the outputs.

Next, we turn to the remaining two networks to which the adversary was *not* given access during training. Although the results do not match the networks used in training, there is a large increase in success rates over those when the ATN was trained with a single target network (Table 4). It is likely that simultaneously training against larger numbers of target networks at will further increase the transferability of the adversarial examples; this is left for future exploration.

Table 5: ATN_b retrained with 3 networks (marked with *). (The same notation as Table 4 is used here).

	Classifier				
	C_p*	C_{a0}	C_{a1}*	C_{a2}*	C_{a3}
rank 1 correct	94%	35%	88%	83%	64%
rank 2 cor. (cond)	97%	88%	97%	96%	73%
rank 2 cor. (uncond)	91%	31%	85%	80%	47%

Attacking ImageNet Networks

We explore the effectiveness of ATNs on ImageNet (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 released by Google, Inc.: 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)(Alemi 2016).

We trained AAE ATNs and P-ATNs to attack IR2; training followed the same procedures as were described with the MNIST experiments. 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 6):

- *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;

Table 6: 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)

- *IR2-Resize-Conv*, a small architecture that avoids checkerboard artifacts common in deconvolutional layers by using bilinear resize layers to down-sample and up-sample 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*. *IR2-Conv-FC* has many more parameters than the other architectures due to the large fully-connected layers. These cause the network to learn too slowly for the autoencoding approach (AAE ATN), but can be used to learn perturbations quickly (P-ATN).

Training Details: All networks are trained with the same hyper-parameters. For each architecture and task, we trained four networks, one for each target class: binoculars, soccer ball, volcano, and zebra. In total, 20 different ATNs were trained to attack IR2. To find a good set of hyper-parameters for these networks, a series of grid searches were conducted through reasonable values for learning rate, α, and β, using only a single target (not all 4). Those training runs were terminated after 0.025 epochs – an extremely short training regime (only 1600 training steps with a batch size of 20). Based on the parameter search, we set the learning rate to 0.0001, $\alpha = 1.5$, and $\beta = 0.01$ for all of the networks trained. All runs were trained for 0.1 epochs (6400 steps) on shuffled training set images, using the Adam optimizer and TensorFlow defaults. To avoid hand-picking the best results after the networks were trained, we pre-selected four images from the unperturbed validation set to use for the figures in this paper *prior* to training. Once training completed, we evaluated the ATNs by passing 1000 images from the validation set through the ATN and measuring IR2's accuracy on those adversarial examples.

Table 7 shows the top-1 adversarial accuracy of the 20 model/target combinations. The AAE approach is superior to the perturbation approach. Between 83-92% of the image inputs are successfully transformed into adversarial examples in a single forward pass through the ATN. [1]

Table 7: IR2 ATN Performance

	P-ATN Top-1 Accuracy			
	Binoc.	Soccr.	Volc.	Zeb.
IR2-Base-Deconv	66.0%	56.5%	0.2%	43.2%
IR2-Conv-FC	**79.9%**	**78.8%**	0.0%	**85.6%**
	AAE-ATN Top-1 Accuracy			
	Binoc.	Soccr.	Volc.	Zeb.
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%

The two types of ATNs generate very different attacks. 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, the AAE architectures distribute the differences across wider regions of the image. As seen, with all three AAE networks, many of the *original* high-frequency patterns are replaced with the high frequencies that encode the *adversarial* signal.

Though not shown in detail for space limitations, note that the results obtained from *IR2-Base-Deconv* revealed 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 excel at preserving the majority of 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, or small ghost/edges images placed in high frequency areas (Mordvintsev, Olah, and Tyka 2015) (see Figures 5-E & 6-bottom

[1] With the selected hyper-parameters, the P-ATNs for Volcano diverged during training, and the perturbations were far from the image manifold. Nonetheless, we kept the hyper-parameters fixed in these experiments to accurately represent the challenge of training P-ATNs. We were able to more reliably train P-ATNs for Volcano with different hyper-parameters found during our grid search.

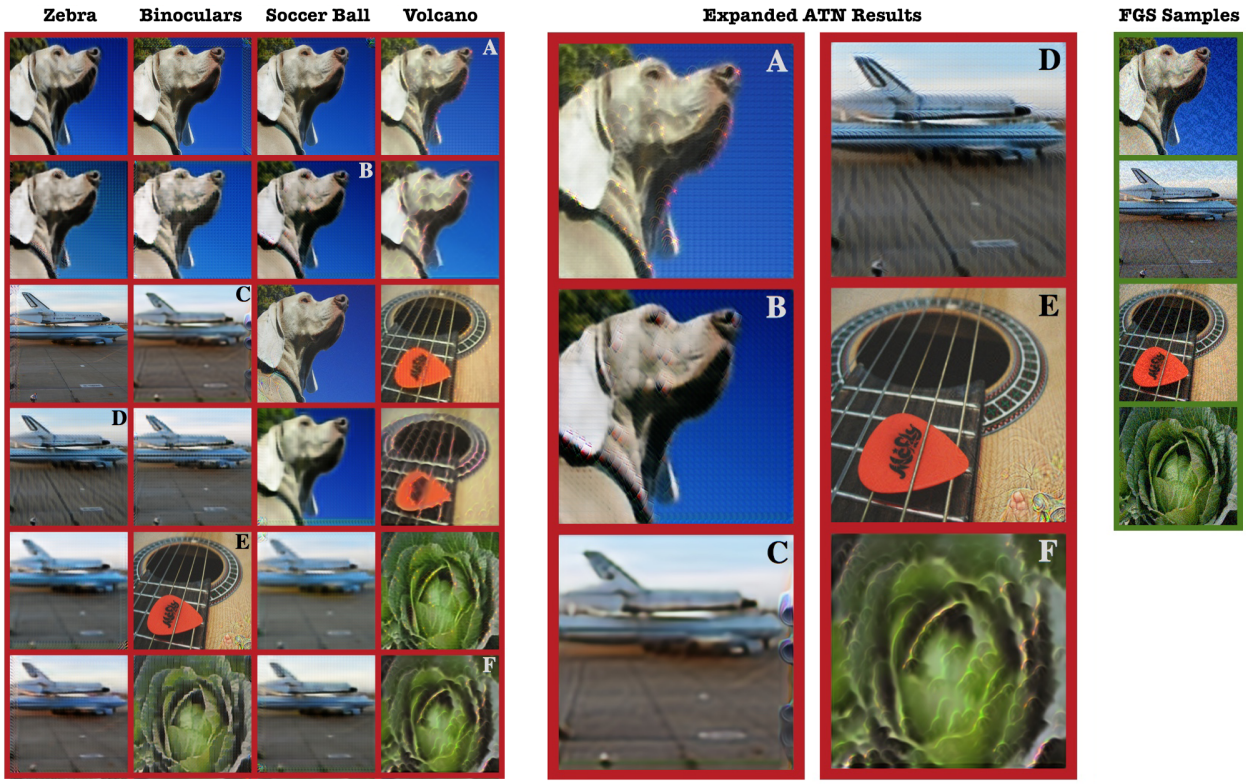

Figure 5: **Adversarial diversity.** *Left 4 columns:* successful adversarial examples for different target classes from a variety of ATNs. From the left: Zebra, Binoculars, Soccer Ball, Volcano. *Middle 2 columns:* zoomed examples. Unlike existing adversarial techniques where adversarial examples look similar, these adversarial examples exhibit a great deal of diversity. Consider (D), in which the space shuttle is mistaken as a zebra, the ATN made the lines on the tarmac darker and more organic — evocative of a zebra's stripes. Yet clearly no human would mistake this for an image of a zebra. Similarly, in (A), the dog's face has been speckled with a few orange dots, and these are sufficient to convince IR2 that it is a volcano. 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*. *Right column:* Adversaries generated by gradient based methods (FGS) shown for comparison. Note the qualitative properties: these largely consist of additional noise, and no semantic information about the classifier is conveyed.

left). For P-ATNs, approximately the same perturbation, in the same place, is used across all input examples. Placing the perturbations there is less likely to disrupt the other top classifications, thereby keeping L_y lower. This is in stark contrast to the AAE ATNs, which creatively modify a larger portion of the input, in semantically interpretable manners, as seen in Figures 5 & 6.

Discussion

Perhaps most importantly to ascertaining the weaknesses of the targeted network, Figure 5 shows that ATNs are capable of generating a wide variety of small perturbations that all cause the targeted network to make mistakes. In Figure 5, consider the zoomed example, D. In D, the space shuttle image that is normally classified correctly is transformed into one that is mistaken as a zebra. To accomplish this, the ATN made the lines on the tarmac darker and more "organic" — evocative of a zebra's stripes. Given how much of the image is (to humans) still visually about the Space Shuttle, this misclassification provides clear insights into what mistakenly triggers the network's Zebra response. Similarly, in example A (Figure 5), the dog's face has been speckled with only a few orange dots. These few pixels are sufficient to convince the very-well trained, state-of-the-art ImageNet classifiers, that the image is of a volcano, despite the overwhelming evidence of a dog. It is crucial to understand the role of preserving the target-network's output here; the vast majority of the image remains intact, thus preserving the majority of the target network's classification probabilities (the classifier's output vector). Based on empirical studies, had rank-preservation not been employed, image transformations would have been less constrained, and thereby significantly more disruptive.

In comparison, previous adversarial example generation often produced qualitatively uniform results (though achieved in a variety of manners). Many amounted to adding what appears as "salt-and-pepper" type noise to the image, generally concentrating the noise at pixels with large gradient magnitude for the particular adversarial loss function.

Figure 6: **Architecture comparisons.** Top 2: Adversarial autoencoding ATNs. Bottom 2: Perturbation ATNs. All networks here are trained with the same hyper-parameters, apart from the target class, which varies among zebra, soccer ball, and volcano. There are substantial differences in how each transforms the input (see the areas within the magenta circles). The P-ATNs make perturbations along the edges, where they are less likely to disrupt the other top classifications. This contrasts the AAE ATNs, which creatively modify each input individually, as seen here and in Figure 5. In the top 2 images, note the heavy reliance on the content of the image and the placement of perturbations and the differences in the attack methods: large wavy lines on the dog's face, compared with the small speckled orange dots when trained for volcano (right). Finally, for the P-ATNs, note how small some of the perturbations are; yet, they are enough to cause mistakes in a well-trained ImageNet classifier.

To demonstrate, we also tackled the same problem with fast gradient sign (FGS) using a perturbation magnitude of 0.15. Note that FGS adversaries are *untargeted*: the goal is to make the classifier make *any* mistake, not to target a particular class. Even with this easier goal, we found that IR2 was far less susceptible to these attacks. IR2 was able to correctly classify 51.2% of the FGS generated examples (including the 4 examples shown in Figure 5) (right). When the perturbation magnitude was increased to 0.5 (yielding unacceptably noisy images), the ability to fool IR2 remained significantly lower than the rates in with ATNs. Most important, however, is that even when FGS was successful, there is little to be gleaned from the example generated. Though the pixel-noise that FGS finds sometimes fools IR2, the ATNs produce coherent features that are both interpretable and more successful in attacking a well trained network.[2] (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.

One possible explanation for the occurrence of the "misleading" information in the high frequency regions of the image can be seen in Figure 5: some of the AAE-ATNs do not completely reconstruct the original image's high frequency data. The convolutional architectures have difficulty exactly recreating edges from the input image, due to spatial data loss introduced when down-sampling and padding. Consequently, the $L_{\mathcal{X}}$ loss leads the networks to learn smooth boundaries in the reconstruction. Because the high-frequency regions cannot be reconstructed fully, this provides an interesting opportunity for the ATNs: they can make perturbations in regions that have already lost their high-frequency information. This strategy is visible in many examples; for example, in the dog image, many networks make minimal modification to the sky, but add orange pixels around the edges of the dog's face, exactly where the $L_{\mathcal{X}}$ error would be high even in a non-adversarial reconstruction.

Conclusions and Future Work

Current methods for generating adversarial samples involve a gradient descent procedure on individual input examples. We have presented a fundamentally different approach to creating an adversary: a separate neural network can be trained to attack a target network. This network, an ATN, operates by converting any input into an adversarial example with high probability. Even on well trained state-of-the-art ImageNet classifiers, ATNs could, in a single pass, transform between 83-92% of image inputs into adversarial attacks. ATNs are efficient to train, fast to execute (only a single forward pass), and produces diverse, successful, adversarial examples that also reveal weaknesses in trained classifiers.

In the future, it will be interesting to measure the difficulty of detecting ATN attacks compared to attacks generated by previous adversarial techniques. This may provide a quantitative metric into determining which methods produces the most realistic examples. Extending this line of inquiry, exploring the use of a Generative Adversarial Networks as discriminators during training may even further improve the realism of the ATN outputs. Finally, an avenue not explored here, but that may be important when access to the target network is limited, is training ATNs in a black-box manner, similar to recent work in (Tramèr et al. 2016), or using REINFORCE (Williams 1992) to compute gradients for the ATN using the target network simply as a reward signal.

[2] In terms of speed, the timing of FGS to generate an adversarial example was approximately 0.58 seconds / sample generated. For IR2-Conv-Deconv, it was 0.16 seconds, for IR2-Resize-Conv and IR2-Conv-FC the timing was 0.55 and 0.61s.

References

Alemi, A. 2016. Improving inception and image classification in tensorflow. https://research.googleblog.com/2016/08/improving-inception-and-image.html.

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

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

Deng, J.; Dong, W.; Socher, R.; Li, L.-J.; Li, K.; and Fei-Fei, L. 2009. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, 248–255. IEEE.

Goodfellow, I. J.; Shlens, J.; and Szegedy, C. 2014. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2015. Deep residual learning for image recognition. *CoRR* abs/1512.03385.

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

Johnson, J.; Alahi, A.; and Fei-Fei, L. 2016. Perceptual losses for real-time style transfer and super-resolution. In *European Conference on Computer Vision*.

Kingma, D., and Ba, J. 2015. Adam: A method for stochastic optimization. In *iclr*.

Kurakin, A.; Goodfellow, I. J.; and Bengio, S. 2016. Adversarial examples in the physical world. *CoRR* abs/1607.02533.

LeCun, Y.; Cortes, C.; and Burges, C. J. 1998. The mnist database of handwritten digits.

Liu, Y.; Chen, X.; Liu, C.; and Song, D. 2016. Delving into transferable adversarial examples and black-box attacks. *CoRR* abs/1611.02770.

Moosavi-Dezfooli, S.; Fawzi, A.; Fawzi, O.; and Frossard, P. 2016. Universal adversarial perturbations. *CoRR* abs/1610.08401.

Moosavi-Dezfooli, S.; Fawzi, A.; and Frossard, P. 2016. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE CVPR*, 2574–2582.

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

Nguyen, A. M.; Yosinski, J.; and Clune, J. 2014. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. *CoRR* abs/1412.1897.

Papernot, N.; McDaniel, P.; Jha, S.; Fredrikson, M.; Celik, Z. B.; and Swami, A. 2015. The limitations of deep learning in adversarial settings. In *Proceedings of the 1st IEEE European Symposium on Security and Privacy*.

Papernot, N.; McDaniel, P.; Goodfellow, I.; Jha, S.; Celik, Z. B.; and Swami, A. 2016. Practical black-box attacks against deep learning systems using adversarial examples. *arXiv preprint arXiv:1602.02697*.

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

Szegedy, C.; Ioffe, S.; Vanhoucke, V.; and Alemi, A. 2016. Inception-v4, inception-resnet and the impact of residual connections on learning. *arXiv preprint arXiv:1602.07261*.

Tramèr, F.; Zhang, F.; Juels, A.; Reiter, M. K.; and Ristenpart, T. 2016. Stealing machine learning models via prediction apis. In *USENIX Security*.

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