Generating Adversarial Examples using Adversarial Networks and Functional Attacks

Rajarshi Bhattacharjee
rbhattacharj@umass.edu

Ignacio M. Gavier
igavier@umass.edu

Abstract

Deep learning models have been found to be very vulnerable to *adversarial* inputs, which are visually similar to ordinary input but cause deep networks to misclassify these inputs by a wide margin. The problem of developing new techniques to produce adversarial attacks is an active topic of research. In this project, we propose two novel techniques for generating adversarial examples. Adversarial Transformation Networks (ATN) are feed-forward neural networks trained in a self-supervised manner to generate adversarial examples against a target network. For our first method, we propose training ATNs with the triplet loss as a regularizer instead of the usual output space loss. For our second method we propose a type of functional attack using distortions that locally stretch and squeeze the original sample. We perform extensive experiments using MNIST and CIFAR10 datasets which show the effectiveness of our approach. Though the experiments are limited in scope, they show that ATNs with triplet loss perform pretty well and might be more effective for black box attacks than ordinary ATNs.

1 Introduction

The widespread use of Deep Neural Network (DNN) based machine learning models have brought into focus the question about the vulnerability of these models to different kinds of attacks. *Adversarial examples* [1], [2], [3] are a type of attack in which small perturbations are added to the inputs to the machine learning model which causes the model to misclassify the input example (or more generally produce an incorrect output). An important feature of adversarial examples is that they appear to be almost indistinguishable to the original example to a human observer. Yet they cause machine learning models to misclassify the examples with a high degree of confidence. Research in *adversarial learning* focuses on either techniques to generate new adversarial examples or techniques to defend against adversarial attacks. In our project, we focus on the problem of generating adversarial examples, more specifically adversarial images, to attack deep networks.

In this project, we propose two different types of adversarial attacks. The first technique is based on Adversarial Transformation Networks (ATN) [4]. ATNs are feed-forward networks, trained in a self-supervised manner, to generate adversarial examples to attack a target network. ATNs minimize a loss function consisting of two parts: a loss minimizing the distance between the generated image and original image to maintain perceptual similarity and a loss for the embedding space (*i.e.* output of the penultimate layer of the target network) of the images. We propose training ATNs using the triplet loss for the embedding space. This is motivated by the fact that adversarial attacks are known to work by changing the structure of the embedding space of the examples so that the decision boundaries are perturbed and the network misclassifies them [5]. Thus, using a loss function to directly manipulate the embedding space should be effective.

The second technique is based on modifying the input image using some function. This is also known as *functional attack* and the function is learned to attack the model. We propose the kind of functions that can locally stretch and squeeze the input image. Though we wanted to combine both the attacks at the end to produce a more powerful attack, it was not possible to combine ATNs with functional attacks as the proposed functional attack is highly dependent on the propperties of the specific image which is being attacked. Thus, we cannot use a single neural network for the proposed functional attack on different images.

We perform experiments to show the effectiveness of our proposed approaches using the MNIST and CIFAR10 datasets. For ATNs, we compare the performance of normal ATNs and ATNs with Triplet Loss on both *white* box as well as *black* box targeted attacks. We also compare the performance of the proposed approaches against the state-of-the-art adversarial defense TRADES [6]. Though we could only perform a limited set of experiments due to constraints of computational resources, we find ATNs with triplet loss show comparable performance to normal ATNs for *white* box attack and slightly better performance for *black* box attacks. For the functional attacks, since the attack is specific to each input image and due to limitations in computational capacity, we generated adversarial examples for only a portion of the test sets from the aforementioned datasets. We also compare these results with those arising from attacking the same DNNs trained with the TRADES defense.

The rest of the report is organzied as follows. In section2, we provied the novelty statement for our work. In section 3, we provide an overview of the related work. Inn section 4, we describe our methodlogy in detail. In section 6, we describe the datasets used in detail and in section 7, we discuss the experiments and results. Finally, we conclude by giving an overview in section 7.

2 Novelty Statement

In this work, we propose two new techniques for generating images for adversarial attacks. For the first technique, we propose using the Triplet Loss for training Adversarial Transformation Networks to automatically generate adversarial examples. ATNs and similar architectures have been widely investigated but the methods to train them involve using ad-hoc techniques like the *reranking* function proposed in [4]. Though triplet loss has been proposed previously for adversarial defense, we believe this is the first time they have been proposed in the context of adversarial attacks. For the second method, we propose a new kind of functional attacks, which are techniques where an adversarial example is generated by applying a function to a benign image. Functional attacks have been previously proposed that use a very specific kind of functions, such as FGSM or ReColorAdv, as they modify the benign image pixel-wise; or very general functions, such as stAdv, as they use a spatial distortion in which each pixel can be shifted in any way. In this work we implement parameterized distortion functions to decrease the computational cost in which the benign image is distorted, through local stretching and squeezing.

3 Related Work

In this section, we review the papers on which our attacks are based on and also discuss other closely related papers. Adversarial attacks can be broadly classified into white-box attacks or well black-box attacks. White-box techniques assume that they have full access to the architecture and parameters of the model that is being targeted. Some popular white-box are Projected Gradient Descent (PGD) [7] and Fast Gradient Sign Method (FGSM) [1]. FGSM is an attack where the maximum perturbation to produce adversarial example is bounded by some constant ϵ in the l_∞ norm. Formally, the update equation for FGSM can be written as:

$$\mathbf{x}' = \mathbf{x} + \epsilon.sign(\nabla_x L(\theta, x, y)) \tag{1}$$

where $\mathbf{x}$ is the original input, y is the label, $\mathbf{x}'$ is the adversarial input, $sign(.)$ is the sign function and L is loss function. FGSM essentially increases the value of each pixel in the direction of the steepest ascent of the loss function at that point to increase the loss. PGD is another popular gradient based attack which can be considered as a generalization of the FGSM attack. The update equation for PGD can be written as:

$$\mathbf{x}' = \Pi_S[\mathbf{x} + \alpha(\nabla_x L(\theta, x, y))] \tag{2}$$

Here, $\Pi_S(.)$ is the function to project the input back into the set S which depends on the adversarial budget. For example, the set S is an l_∞ norm ball of appropriate radius. The projection operation ensures that the adversarial budget is not exceeded. There have been other similar white-box attacks which have been proposed like [2], which is another regularization based attack.

Black box attacks generally assume that they do not have access to the model being attacked at all or only have access to the output of the attacked model. In [8], it was shown that it is possible to attack a deep neural network by observing only the labels given by the target DNN to chosen inputs. In [9], it was shown that it is possible to craft adversarial examples by randomly sampling a vector from a predefined orthornormal basis and then adding or subtracting it. More black box techniques have been proposed since then like [10] which also applies a large adversarial perturbation to examples and then iteratively refines it.

3.1 Adversarial Transformation Networks

We now describe Adversarial Transformation Networks [4], on which our first approach is based on, in detail. Let the target network (which we want to attack) be $f : \mathbf{x} \in \mathcal{X} \to y \in \mathcal{Y}$. Here, $\mathcal{X}$ is the input space and $\mathcal{Y}$ is the space of probability distributions across class labels. Then, formally an ATN is defined as a neural network:

$$g_{f,\theta}(\mathbf{x}) : \mathbf{x} \in \mathcal{X} \to \mathbf{x}' \tag{3}$$

Here, θ is the parameter vector of the generator network g which generates adversarial examples to attack the target/discriminator network f. Also, $\mathbf{x}' \sim \mathbf{x}$ but $\arg\max f(\mathbf{x}) \neq \arg\max f(\mathbf{x}')$. In this paper, the focus is on the more challenging case of *targeted* ATNs where the *target* class is the class for which we want the classifier to output the maximum value for the adversarial input. Formally, targeted ATNs can be defined exactly as equation 3 with the following extra constraint: $\arg\max f(\mathbf{x}') = t$ where t is the target class. To find g, we solve the following optimization problem:

$$\arg\min_{\theta} \sum_{\mathbf{x}_i} \beta L_{\mathcal{X}}(g_{f,\theta}(\mathbf{x}_i), \mathbf{x}_i) + L_{\mathcal{Y}}(f(g_{f,\theta}(\mathbf{x}_i)), f(\mathbf{x}_i)) \tag{4}$$

Here, $L_{\mathcal{X}}$ is the loss function in the input space to measure how close the input image is to the adversarial image. $L_{\mathcal{Y}}$ is the loss function for the output space to ensure that $\arg\max f(\mathbf{x}) \neq \arg\max f(g(\mathbf{x}))$ and β is a hyperparameter to balance the two losses. Going forward, we sometimes refer to $g_{f,\theta}$ by only g_f or g when the context is clear. In this paper, the authors specifically choose the L_2 loss function for $L_{\mathcal{X}}$ and $L_{\mathcal{Y}}$ is specified as follows: $L_{\mathcal{Y},t}(\mathbf{y}', \mathbf{y}) = L_2(\mathbf{y}', r(\mathbf{y}, t))$, where $\mathbf{y} = f(\mathbf{x})$ and $\mathbf{y}' = f(g_f(\mathbf{x}'))$. Also, t is the target class and $r(.)$ is a reranking function that modifies $\mathbf{y}$ such that $y_k < y_t, \forall k \neq t$. Specifically, $r(.)$ is defined as:

$$r(\mathbf{y}, t) = norm\left(\begin{Bmatrix} \alpha * \max \mathbf{y} & \text{if } k = t \\ 1 & \text{otherwise} \end{Bmatrix}_{k \in \mathbf{y}} \right) \tag{5}$$

Here, $\alpha > 1$ is a hyperpararmeter which specifies how much larger y_t should be compared to the current maximum. $norm(.)$ is a normalization function to rescales the inputs between 0 and 1 and ensures $r(\mathbf{y}, t)$ is a valid probability distribution. The choice of this reranking function $r(.)$ ensures that $r(\mathbf{y}, t)$ maintains the rank order of all but the targeted class to minimize the distortions when computing $\mathbf{x}'$.

ATN is a semi-black box technique as it requires access to only the output of the target network (for training) but doesn't assume it has access to anything more like the gradients of the target network. Adversarial example are also fast and easy to generate as they can be generated using just a forward pass. There have been more neural networks based approaches for generating adversarial example that have been proposed but ATN is the first paper outlining this approach. More neural network based approaches for generating adversarial examples (for example using Generative Adversarial Networks in [11], [12] and autoencoders in [13]) have been proposed since then.

3.2 Functional Attacks

We now describe the Functional Attack proposed in [14], which is the method that motivated our second approach to generate adversarial examples. Using the notation from the previous section, we call g to the adversarial function that modifies the input image. The authors focused their work on a very specific kind of functions: the pixel-wise functions, *i.e.* $x'_i = g(x_i)$, being i the ith pixel. In order to find the function g for a targeted attack, they solve the following optimization problem

$$\arg\min_{g} L_{\text{adv}}(g; \mathbf{x}) + \lambda L_{\text{smooth}}(g), \tag{6}$$

where $L_{\text{adv}}(g, \mathbf{x}) = \max\left(\max_{k \neq t} \left(g(\mathbf{x}')_i - g(\mathbf{x}')_t\right), 0\right)$ is the adversarial loss and $L_{\text{smooth}}(g)$ is the loss that contributes to find a smooth function g. The authors proposed to solve the optimization using the normal backward propagation method.

Functional attacks that don't use pixel-wise function were also presented in [15]; in this paper the authors use functions that distort the input image using a bidimensional distortion field $(\Delta u, \Delta v)$. The adversarial example is then $\mathbf{x}'(u, v) = \mathcal{B}\left(\mathbf{x}(u + \Delta u, v + \Delta v)\right)$, where u, v are the bidimensional coordinates and $\mathcal{B}(.)$ is the bilinear interpolation of the distorted image to preserve the original size. They optimize an objective function similar to the one presented in equation 6, but using the L-BFGS-B method [16], since backward propagation technique is intractable.

Unlike the attacks using ATNs, which are models that are trained with a batch of images, functional attacks are techniques that generally have to be trained every time we want to attack the network, since the adversarial function g is highly dependent on the original image $\mathbf{x}$. ATN models are deep neural networks that can be trained to generate adversarial examples from a dataset; on the other hand, adversarial functions are just flat models with much less parameters than an ATN, so in general, a function g generating an functional adversarial example from an input sample $\mathbf{x}_1$ doesn't work to generate another adversarial example from a different input sample $\mathbf{x}_2$.

The pixel-wise functional attack is a white box attack since it uses the gradients from the model under attack. Spatially transformed functional attack, on the other hand was proposed as a semi-black box attack since it uses only the output scores from the model under attack. In this work, for our adversarial attack method, we use the white box approach since using backward propagation is much faster than the L-BFGS-B method.

3.3 Adversarial Defenses

There have been a number of adversarial defenses proposed like adversarially robust training [7] which essentially proposes generating adversarial examples for the classifier (using some technique like PGD) and then training the network with the original as well as the adversarial examples. The best known adversarial defense currently is the TRADES defense proposed in [6]. We discuss this defense here as we will be using this defense as a benchmark to see how well our proposed attacks perform. TRADES or Tradeoff-inspired Adversarial Defense via Surrogate-loss minimization essentially proposes minimizing a new objective function for the classifier which needs to be defended. The objective function consists of a regularized surrogate loss function. Formally, for a classifier f, the objective to minimize can be written as:

$$\min_{f} \mathrm{E}\{L(f(\mathbf{x}), y) + \beta \max_{\mathbf{x}' \in \mathrm{B}(\mathbf{x}, \epsilon)} L(f(\mathbf{x}), f(\mathbf{x}'))\} \tag{7}$$

Here, $L(.)$ is a classification-calibrated loss function [17] like cross-entropy loss. $\mathrm{B}(\mathbf{x}, \epsilon)$ is a ball (of a norm which depends on the norm in which the adversarial budget is specified) of radius ϵ centered at $\mathbf{x}$. β is a hyper-parameter which balances the two losses. The first term in the objective is to ensure that the classification is correct *i.e.* $f(\mathbf{x})$ is close to y, and the second term ensures that the decision boundary of classifier is pushed away from the samples by minimizing the prediction error of natural example $f(\mathbf{x})$ and that of adversarial example $f(\mathbf{x}')$. We will be explaining the rest of the details in the experiments section.

4 Methodology

We now describe the details of the attacks proposed by us.

4.1 ATN with Triplet Loss

Triplet Loss The triplet loss is widely used in metric learning. It is trained on triplets of input examples $\{\mathbf{x}_a^i, \mathbf{x}_p^i, \mathbf{x}_n^i\}$ where $\mathbf{x}_a^i$, $\mathbf{x}_p^i$ and $\mathbf{x}_n^i$ are referred to as the anchor example, the positive example and the negative example respectively. The positive and anchor $(\mathbf{x}_a^i, \mathbf{x}_p^i)$ are examples from the same class while $\mathbf{x}_n^i$ is from a different class. Then, our objective is to learn an embedding of the inputs $f(.)$ such that similar examples are mapped close together while dissimilar examples are mapped far apart so that it is easy to classify them. The triplet loss is then defined as:

$$L_{trip}(\mathbf{x}_a^i, \mathbf{x}_p^i, \mathbf{x}_n^i) = max(d(f(\mathbf{x}_a^i), f(\mathbf{x}_a^i)) - d(f(\mathbf{x}_a^i), f(\mathbf{x}_n^i)) + \alpha, 0) \tag{8}$$

Here, $d(.)$ is the distance metric used to measure the similarity between the embeddings and α is the margin of separation between the positive and negative pairs. The objective function is the total loss over all such pairs $\sum_i L_{trip}(\mathbf{x}_a^i, \mathbf{x}_p^i, \mathbf{x}_n^i)$. In our case, we use Euclidean distance metric for $d(.)$ and choose the margin α as 1 for all our experiments. We note that a few adversarial defenses have been proposed recently which uses the triplet loss. In [5], the triplet loss is used as a regularizer. It is used on the outputs of the penultimate layer (along with the perturbed outputs) of the deep network to improve the quality of the representations the neural network is able to learn. This makes the network robust to adversarial perturbations as the embeddings (of the input space) become more well separated. A similar approach of using the triplet loss as a regularizer was also proposed in [18].

Motivation Our first method proposes using the triplet loss with ATN. In [5], it was shown experimentally that adversarial attacks like PGD changes the embedding space of the penultimate layer of deep networks causing the adversarial examples from different classes to move towards the target class while adversarial images from the same class are pushes further apart in such a way that the network classifies them to different classes. Thus, the triplet loss is a natural candidate to use with the ATN as we want to manipulate the embeddings produced by our generator of adversarial examples in a similar way such that, adversarial embeddings from different classes move closer to the target class while those from the same class move further apart.

We also note that the loss $L_{\mathcal{Y}}$, which is the loss for the embedding layer, in the objective function (Equation 4) of the original implementation of the ATN in [4] uses the reranking function $r(.)$ in Equation 5 to ensure that for the adversarial examples, the rank ordering among classes (except the target class) is maintained. However, there is no strong motivation for doing so as the geometry of the adversarial embedding space can be very different from original embedding space as observed in [5], [19]. Thus, our hypothesis is that directly manipulating the embeddings by the triplet loss will yield better adversarial examples.

Objective Function In our formulation, the loss $L_{\mathcal{X}}$ for the input space remains same while the loss for the embedding layer $L_{\mathcal{Y}}$ changes is replaced by the triplet loss. A major factor in how effective the triplet loss is how the triplets are chosen. Generally, we can expect better embeddings and faster learning if the triplets used for training are more discriminative. We note that the definition of the positive and negative examples in the triplet are inverted here as we want to find a generator $g_{f,\theta}$ such that examples from different classes map to the target class. Specifically, for an input image $\mathbf{x}$ not belonging to the target class, we want $f(\mathbf{x})$ and $f(g_{f,\theta}(\mathbf{x}))$ to be be far apart while $f(\mathbf{x})$ must be mapped close to the embeddings of the target class. Thus, if $f(\mathbf{x})$ is the anchor, then $f(\mathbf{x})$ can be the corresponding negative example. There can be multiple different choices for the positive example from the target class. For our experiments, we choose the positive example for every image as just the average of the image embeddings from the target class. Formally, if the images from the target class are $\{\mathbf{x}_t^i\}_{i=1}^n$, the triplet corresponding to an input image $\mathbf{x}$ is $\{f(g(\mathbf{x})), \frac{1}{N}\sum_{i=1}^n f(\mathbf{x}_t^i), f(\mathbf{x})\}$. Thus, we can write the objective function to minimize as the following:

$$\arg\min_{\theta} \sum_{\mathbf{x}_i} \beta L_2(g_{f,\theta}(\mathbf{x}_i), \mathbf{x}_i) + L_{trip}(f(g_{f,\theta}(\mathbf{x}_i)), \frac{1}{N}\sum_{j=1}^{n} f(\mathbf{x}_t^j), f(\mathbf{x}_i)) \tag{9}$$

We also note that the positive (as well as negative) examples chosen corresponding to each image $\mathbf{x}_i$ can be more precise. For example, corresponding to $\mathbf{x}_i$, the positive example can be the one that minimizes $L_2(f(g(\mathbf{x}_i)), f(\mathbf{x}_t))$ *i.e.* the closest embedding from the target class. This will probably give us better results but the tradeoff is that this is computationally expensive to compute for every example at every iteration. However, we find experimentally our choice of positive and negative examples produces reasonable results along with being extremely fast to compute.

Implementation Details We note that this formulation of ATN doesn't give us a scope to specify any **adversarial budget** *i.e.* the maximum perturbation allowed for the adversarial images. While most of our experiments won't specify any adversarial budget, comparison with TRADES requires us to specify an adversarial budget in l_∞ norm. This means if ϵ is the budget and the value of original pixel is p, the corresponding pixel in the adversarial image must be within $[p-\epsilon, p+\epsilon]$. To ensure our ATN output is within the specified budget, if the value of the corresponding pixel in the adversarial image is p', then, if $p' > p+\epsilon$, we set $p' = p+\epsilon$. Similarly, if $p' < p-\epsilon$, we set $p' = p-\epsilon$.

We perform two types of experiments. We try to attack a classifier f after training with the outputs of the classifier f. This is the straightforward semi-black box mode of attack. We also try out the fully black-box mode of attack by attacking a completely different classifier f' using our generator network. This shows how transferable our attacks actually are. More details about the implementation of the experiments are given in the Experiments and Results section.

4.2 Functional Attacks

We now present the mathematical formulations for our functional attack approach. We propose a function that generates a distortion field as in [15] (see the previous section), but instead of using a generic distortion field $(\Delta u, \Delta v)$ where distortions are independent for each pixel, we parameterize it in order to reduce the number of variables to be learned during the attack and make the backward propagation tractable.

Parameterized distortion. There are infinity number of possibilities to define a parameterized distortion field, but we will focus only on the functions that locally stretch and squeeze the input image $\mathbf{x}$. For this kind of functions, we choose the following distortion field:

$$(\Delta u, \Delta v) = k(r)\left(\alpha_u \sin\theta, \alpha_v \cos\theta\right), \tag{10}$$

where α_u and α_v are the distortion amplitudes, which indicate how much distorted is the image, $k(r)$ is a radial function described in the next paragraph, and r and θ are the polar coordinates with origin (c_u, c_v), which is the distortion center: $r = \sqrt{(u-c_u)^2 + (v-c_v)^2}$ and $\theta = \arctan\frac{u-c_u}{v-c_v}$.

We choose $k(r)$ to be a Rayleigh-shaped function since it is smooth and includes only one parameter:

$$k(r) = -(r/\sigma^2)\exp\left(-r^2/2\sigma^2\right). \tag{11}$$

where σ is the radius of influence of the distortion.

Note that by parameterizing the distortion field, we have reduced the number of parameters to be learned to five $(c_u, c_v, \sigma, \alpha_u, \alpha_v)$. Also, note that positive or negative values for αs indicate that $\mathbf{x}'$ is squeezed or stretched with respect to $\mathbf{x}$, while zero values indicates no distortion.

We can generalize this distortion field for multiple superposed distortions to make the attack more effective:

$$(\Delta u, \Delta v) = \sum_l k^l(r^l)\left(\alpha_u^l \sin\theta^l, \alpha_v^l \cos\theta^l\right), \tag{12}$$

where $l = \{1, \ldots, L\}$ indicates the index among L distortions. Therefore, the resultant distortion field has $5L$ parameters to be learned for generating the attack. Having reduced the number of parameters, we now can implement the backward propagation method.

Forward. Let $\mathbf{x}$ be the original image and $\mathbf{x}'$ the adversarial image. The modified image at (u, v) can be expressed as $\mathbf{x}'(u,v) = \mathcal{B}(\mathbf{x}(u+\Delta u, v+\Delta v))$. To simplify the following expressions, let $s = u + \Delta u$ and $t = v + \Delta v$. Therefore, the distorted image at coordinates (u, v), can be rewritten as:

$$\mathbf{x}'(u,v) = a_0 + a_1 t + a_2 s + a_3 ts, \tag{13}$$

where a_i are the interpolation coefficients given by:

$$\begin{cases} a_0 = & \mathbf{x}(s_1,t_1)s_2t_2 & -\mathbf{x}(s_1,t_2)s_2t_1 & -\mathbf{x}(s_2,t_1)s_1t_2 & +\mathbf{x}(s_2,t_2)s_1t_1 \\ a_1 = & -\mathbf{x}(s_1,t_1)s_2 & +\mathbf{x}(s_1,t_2)s_2 & +\mathbf{x}(s_2,t_1)s_1 & -\mathbf{x}(s_2,t_2)s_1 \\ a_2 = & -\mathbf{x}(s_1,t_1)t_2 & +\mathbf{x}(s_1,t_2)t_1 & +\mathbf{x}(s_2,t_1)t_2 & -\mathbf{x}(s_2,t_2)t_1 \\ a_3 = & \mathbf{x}(s_1,t_1) & -\mathbf{x}(s_1,t_2) & -\mathbf{x}(s_2,t_1) & +\mathbf{x}(s_2,t_2) \end{cases} \tag{14}$$

and $s_1 = \lfloor s \rfloor$, $s_2 = \lceil s \rceil$, $t_1 = \lfloor t \rfloor$, $t_2 = \lceil t \rceil$.

Backward. In order to learn the parameters of each distortion, we can compute the gradients of the objective with respect to the distortion parameters using the following gradients:

$$\begin{cases} \nabla_{c_u}\mathbf{x}'(u,v) & = (u-c_u)\left[\frac{\Delta v(a_1+a_3s)+\Delta u(a_2+a_3t)}{\sigma^2} - \frac{\Delta u(a_2+a_3t)}{(u-c_u)^2}\right] \\ \nabla_{c_v}\mathbf{x}'(u,v) & = (v-c_v)\left[\frac{\Delta v(a_1+a_3s)+\Delta u(a_2+a_3t)}{\sigma^2} - \frac{\Delta v(a_1+a_3s)}{(v-c_v)^2}\right] \\ \nabla_{\sigma}\mathbf{x}'(u,v) & = \frac{1}{\sigma}\left(\frac{r^2}{\sigma^2} - 2\right)\left[\Delta v(a_1+a_3s) + \Delta u(a_2+a_3t)\right] \\ \nabla_{\alpha_u}\mathbf{x}'(u,v) & = \frac{1}{\alpha_u}\left[\Delta u(a_2+a_3t)\right] \\ \nabla_{\alpha_v}\mathbf{x}'(u,v) & = \frac{1}{\alpha_v}\left[\Delta v(a_1+a_3s)\right] \end{cases} \tag{15}$$

Note that we have obviated the l index to make cleaner expressions. Also, note that in order to compute the gradient of the objective with respect to one parameter, we should sum over all the coordinates (u, v).

Objective function. With this formulation, we can also define a simpler version for equation 6 where we replace the regularization term for smoothness of g, L_{smooth}, with the squared norms of each α_u^l and α_v^l. Note that, the more distorted the image, the more penalization for the loss function, which is consistent with the loss defined in [14] and [15]. Therefore, the optimization problem that we solve for our functional attack formulation is:

$$\underset{\mathbf{c}_u,\mathbf{c}_v,\boldsymbol{\sigma},\boldsymbol{\alpha}_u,\boldsymbol{\alpha}_v}{\arg\min} \; L_{\text{adv}}(\mathbf{c}_u, \mathbf{c}_v, \boldsymbol{\sigma}, \boldsymbol{\alpha}_u, \boldsymbol{\alpha}_v; \mathbf{x}) + \lambda \left(||\boldsymbol{\alpha}_u||_2^2 + ||\boldsymbol{\alpha}_v||_2^2 \right), \tag{16}$$

where each bold parameter corresponds to the vector of parameters for each distortion l. Here, we don't consider the centers of distortion to collaborate with the penalization since they are not relevant for the smoothness of g. In the same way, the radius of influence is also irrelevant for the smoothness.

Note that, unlike the attack with ATNs, this optimization problem has to be solved for every attack, *i.e.* for every image. This is necessary since the optimal parameters are highly dependent on the image we are modifying. Therefore, two different images, even if they have the same label, will learn completely different parameters.

Parameters constraints. We have seen that using a parameterized distortion to generate the adversarial attack reduces the computational learning cost with respect to a non-parameterized distortion, which takes into account the shift for every pixel. However, in some cases, the introduced parameters should be restricted to certain bounds. In our previous formulation, the centers of distortion must be restricted to be located within the edges of the image, or even further in. Also, the radii of influence must be restricted to be less than or equal to the minimum distance from their corresponding center of distortion to the edges of the image. The amplitudes, on the other hand, should be restricted to not produce such a distortion that the order of the pixels is reversed, *i.e.* a pixel that is located to the right of another in the original image cannot end up being placed to the left of it in the adversarial image.

We can define our constrained parameters to be learned based on unconstrained parameters as following:

$$\begin{cases} c_u &= u_0 + \frac{4}{5}\frac{u_{\max}-u_{\min}}{2}\left(2S(c'_u)-1\right) \\ c_v &= v_0 + \frac{4}{5}\frac{v_{\max}-v_{\min}}{2}\left(2S(c'_v)-1\right) \\ \sigma &= \sigma_{\max} S(\sigma') \\ \alpha_u &= \frac{\sigma^2\delta_u \exp(1.5)}{4}\left(2S(\alpha'_u)-1\right) \\ \alpha_v &= \frac{\sigma^2\delta_v \exp(1.5)}{4}\left(2S(\alpha'_v)-1\right) \end{cases} \tag{17}$$

where $S(.)$ is the logistic function, $(u_0, v_0), (u_{\min}, v_{\min}), (u_{\max}, v_{\max})$ are the coordinates of the center, the top left corner and the bottom right corner of the image. In the first two equations, we have restricted the centers of distortion to not lay close to the edge. Also, δ_u and δ_v are the vertical and horizontal distances between two adjacent pixels; and $\sigma_{\max} = \min(u_{\max} - u_{\min}, v_{\max} - v_{\min})/10$, which ensures no distortion in the edges.

5 Data Sets

We perform experiments using the MNIST [20] and CIFAR10 [21] datasets. The MNIST and CIFAR10 datasets from downloaded from *torchvision* are PILImages which are automatically normalized between 0 and 1. The NIST dataset consists of grayscale images of handwritten digits. Each image is of size 28×28. There are 60,000 training images and 10,000 test images in the dataset.Each image has an associated label specifying which class of digits it belongs to. There are a total of 10 classes, one for each of the digits from 0 to 9. The CIFAR10 dataset contains 60,000 32×32 color (RGB) images in 10 different classes. The 10 different classes and their corresponding labels are as follows: airplanes:'0', automobile: '1', birds: '2', cats: '3', deer: '4', dogs: '5', frogs: '6', horses: '7', ships: '8', and trucks: '9'. In the experiments we will refer to each class by their corresponding code. There are 6,000 images of each class. For our experiments, we use a train-test split of 50,000-10,000. Both the datasets are available with the *torchvision* package of PyTorch [22] machine learning library.

We use these versions for our experiments. The versions provided are PILImages (Python Imaging Library format) where each pixel has been normalized between 0 and 1 automatically.

We choose these datasets for our experiments as they have been widely used in papers related to adversarial learning and thus benchmark results area already available for them. Specifically, MNIST has been used in the original paper for ATN [4] and CIFAR10 has been used in the paper of functional attacks [14]. Moreover, these datasets are comparatively smaller than other datasets like Imagenet and thus, experiments using them require less computational resources and time.

6 Experiments and Results

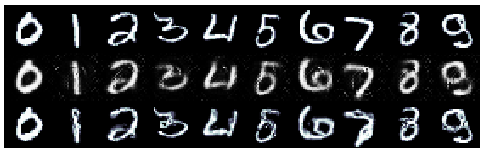
All experiments are performed using Pytorch using its Automatic Differentiation library. For the experiments with ATN with triplet loss, the first challenge was to try to reproduce the results from the main paper [4]. The paper only provides high-level details of the experiments. Moreover, no code repository was provided with the paper. So, the first challenge was to reproduce the original ATN results to establish a baseline by minimizing the objective in Equation 4. Also, though the experiments are limited in scope due to time and computational constraints, they provide us some basic idea about how the models perform in general.

We note that we will be reporting the results for ATN with triplet loss and functional attacks separately. This is because of different training and testing protocols used for the attacks. Generating an adversarial image using the proposed functional attack takes more than a minute per image on average. Hence, it was not possible to test the functional attacks on the full test sets for MNIST and CIFAR10 which contains 10,000 images. Thus, we report the test accuracy for each experiment using the proposed functional attack by randomly sampling a set of 1,000 images from the test set.

The CNN architectures for the discriminator networks used for ATNs and functional attacks is given in Table 1. These architectures are pretty standard and similar to the ones used in [4]. In this table, Conv$_{(n,a\times b)}$ refers to convolutional layers with n filters and a $a \times b$ kernel. MaxPool$_{(k)}$ refers to a max polling layer of kernel size $k \times k$. FC$_{a\times b}$ refers to a fully convolutional layer of size $a \times b$. Also, N is the number of pixels in the image: 784 for MNIST and 1024 for CIFAR10.

The architectures for the generator networks for the ATNs for MNIST and CIFAR10 are given in Table 2. The generator for MNIST is a simple autoencoder with a single hidden layer. The generator for CIFAR10 is based on the architecture provided in Kaggle [1] for CIFAR10. Here, DeConv refers to a deconvolutional layer. We chose these architectures as they are simple and can be trained fast. We also note that the generator networks contain a $tanh$ layer at the end as we want the generated images to be in the range -1 and 1 as the images in the experiments with ATNs are normalized between 1 and -1 unless otherwise stated. Throughout the experiments we refer to the results obtained using the original formulation of ATN as simply ATN and the results obtained using ATN with the triplet loss as ATNTrip. As stated previously, for both MNIST and CIFAR10 the train-test split is already available to us. For experiments with both MNIST and CIFAR10, we used a validation set whose size is 5% of the training set size. The validation set is sampled randomly from the training set.

Figure 1 shows ten images from each dataset (first row), along with the adversarial images generated using targeted adversarial attack by the ATN with Triplet loss (second row) and the functional attack (third row). In both the attacks, the target was class 0 (digit 0 for MNIST and airplane for CIFAR10)


(a) MNIST

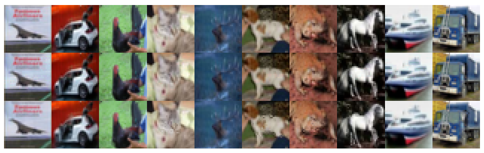

(b) CIFAR10

Figure 1: Images from MNIST and CIFAR10 test sets (first row), being misclassified as 0 (MNIST) and airplane (CIFAR10) after the ATN attack (second row) and functional attack (third row).

[1]https://www.kaggle.com/ljlbarbosa/convolution-autoencoder-pytorch

Symbol	Neural Network Architecture
NN_1	Conv$_{(16,3\times3)}$ $\rightarrow$ ReLU $\rightarrow$ MaxPool$_{(2)}$ $\rightarrow$ Conv$_{(32,3\times3)}$ $\rightarrow$ ReLU $\rightarrow$ MaxPool$_{(2)}$ $\rightarrow$ BatchNorm $\rightarrow$ FC$_{(2N\times1,024)}$ $\rightarrow$ Drop$_{(0.2)}$ $\rightarrow$ ReLU $\rightarrow$ FC$_{(1,024\times1,024)}$ $\rightarrow$ Drop$_{(0.2)}$ $\rightarrow$ ReLU $\rightarrow$ FC$_{(1,024\times10)}$
NN_2	Conv$_{(16,5\times5)}$ $\rightarrow$ ReLU $\rightarrow$ MaxPool$_{(2)}$ $\rightarrow$ Conv$_{(32,3\times3)}$ $\rightarrow$ ReLU $\rightarrow$ MaxPool$_{(2)}$ $\rightarrow$ BatchNorm $\rightarrow$ FC$_{(2N\times1,024)}$ $\rightarrow$ Drop$_{(0.2)}$ $\rightarrow$ ReLU $\rightarrow$ FC$_{(1,024\times512)}$ $\rightarrow$ Drop$_{(0.2)}$ $\rightarrow$ ReLU $\rightarrow$ FC$_{(512\times10)}$
NN_3	Conv$_{(16,3\times3)}$ $\rightarrow$ ReLU $\rightarrow$ MaxPool$_{(2)}$ $\rightarrow$ BatchNorm $\rightarrow$ FC$_{(4N\times1,024)}$ $\rightarrow$ Drop$_{(0.2)}$ $\rightarrow$ ReLU $\rightarrow$ FC$_{(1,024\times512)}$ $\rightarrow$ Drop$_{(0.2)}$ $\rightarrow$ ReLU $\rightarrow$ FC$_{(512\times10)}$

Table 1: Neural Network Architectures for the Discriminator Networks

Dataset	Neural Network Architecture
MNIST	FC$_{(784\times400)}$ $\rightarrow$ ReLU $\rightarrow$ FC$_{(400\times784)}$ $\rightarrow$ tanh
CIFAR10	Conv$_{(16,3\times3)}$ $\rightarrow$ ReLU $\rightarrow$ MaxPool$_{(2)}$ $\rightarrow$ Conv$_{(4,3\times3)}$ $\rightarrow$ ReLU $\rightarrow$ MaxPool$_{(2)}$ $\rightarrow$ DeConv$_{(16,2\times2)}$ $\rightarrow$ ReLU $\rightarrow$ DeConv$_{(3,2\times2)}$ $\rightarrow$ tanh

Table 2: Neural Network Architectures for the Generator Networks

6.1 Performance of ATN with Triplet Loss

We first compare the performance of the original ATN architecture with ATN using triplet loss. We normalize the MNIST and CIFAR10 datasets between -1 and 1 after downloading the pre-normalized datasets (between 0 and 1) using the *torchvision* package. We first train the NN_1 network using MNIST (and CIFAR10) and then perform targeted attacks on it by training the corresponding generator network from Table 2. The results, in terms of accuracies (along with the original accuracy of the classifiers) are reported in Tables 3 and 4 for MNIST and CIFAR10 respectively. We don't set any specific adversarial budget for these experiments so there is substantial drop in accuracy due to the attacks. Though we cannot conclude the performance of ATN with Triplet loss is substantially better from the experiments, we note that the accuracies are close in value. Also, an advantage of ATNTrip over ATN was that it required fewer epochs to generate sharper images (10 on average for ATNTrip vs 15 for ATN).

Hyperparameter Tuning For all experiments with ATN, we keep the value of α (in Equation 5) fixed at 1.5 (following [4]). As stated previously, the quality of images generated by ATN is very sensitive to α and so we kept this value fixed. For all ATN related experiments, we use the Adam optimizer. The hyperparameters to be tuned for all experiments thus include the regularization parameter β (in objective functions 4 and 9 for ATN and ATN with Triplet Loss respectively) and the learning rate for Adam optimizer. For β, the range of values we searched over is 0.001, 0.01, 0.1, 10, 100 and for the learning rate, we searched over the values 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2} to 10^{-1}. Generally, for our experiments, the optimal values for β were found to be between 0.01 to 10 while for learning rate, the optimal values were between 10^{-4} to 10^{-2}. The weight decay for Adam didn't seem to sufficiently alter the results and so it was kept at the default value of 0. We used a batch size of 20 for all experiments. The maximum number of epochs was set at 20. The actual number of epochs for each experiment was decided by observing when the generated adversarial images are sufficiently similar in appearance to the original images while the accuracy of the classifier for the generated adversarial images is sufficiently low.

Target	0	1	2	3	4	5	6	7	8	9
ATN	22.39	21.35	20.44	22.89	21.38	32.02	28.08	19.19	24.88	29.69
ATNTrip	21.82	24.67	16.28	20.47	17.57	36.82	18.35	18.55	25.32	21.85

Table 3: Performance (in terms of accuracy) of targeted attacks using ATN and ATN with Triplet Loss on trained NN_1 classifier for MNIST (original accuracy: 99.12%)

Target	0	1	2	3	4	5	6	7	8	9
ATN	30.14	36.33	28.64	34.86	28.62	37.90	32.44	22.58	24.89	25.00
ATNTrip	26.92	38.30	26.32	36.66	26.30	25.53	39.21	28.72	21.67	23.32

Table 4: Performance (in terms of accuracy) of targeted attacks using ATN and ATN with Triplet Loss on trained NN_1 classifier for CIFAR10 (original accuracy: 67.42%)

6.2 Black box Attacks using ATN with Triplet Loss

Though adversarial attacks by ATNs are not very transferable in general [4] we perform experiments to see if using the triplet loss has an advantage for transferring of attacks. We train our ATN and ATNTrip using the pre-trained NN_1 classifier and then find the accuracy of pre-trained NN_2 and NN_3 classifiers on the adversarial images generated using the ATNs. The results for MNIST and CIFAR10 are reported in Tables 5 and 6. The ATNs are trained with class 0 as the traget class for both datasets. The second column refers to the test accuracy on actual images of the networks. The third and fourth columns refer to the test accuracies of images generated by ATN and ATNtrip respectively. It is seen that the triplet loss is a little more effective for black-box attacks as clearly demonstrated by the larger drop in accuracy for ATNTrip for NN_3 for both MNIST and CIFAR10 and for NN_2 in case of CIFAR10. This shows that generating adversarial images by taking into account the geometry of the embedding space has an advantage for producing better adversarial images which can be used for black box attacks.

	Original	ATN	ATNTrip
NN_1	99.12	22.39	21.82
NN_2	99.04	90.72	92.80
NN_3	98.63	68.94	62.53

Table 5: Black box attack on MNIST classifier using ATNs

	Original	ATN	ATNTrip
NN_1	67.42	30.14	26.92
NN_2	68.48	27.14	23.11
NN_3	64.73	21.31	18.04

Table 6: Black box attack on CIFAR10 classifier using ATNs

6.3 Performance of ATNs on Models with TRADES Defense

For testing the performance of adversarial attacks with ATNs with respect to the TRADES defense, we follow the protocol outlined in [6] and in their github page [2]. We donwloaded the pre-trained models SmallCNN and WideResNet-34-10 (WRN-34-10) for MNIST and CIFAR10 respectively. The models have been trained on the given datasets with the TRADES defense. The test sets have also been provided. We train our generators for MNIST and CIFAR10 using these models with class 0 as the target and generate the adversarial images corresponding to images provided in the test set and then check the accuracy of the SmallCNN and WRN-34-10 models on these adversarial images. We note two key differences from previous experiments. The provided pretrained models have been trained with images normalized between 0 and 1. So we changed the final activation function of both our generators in Table 2 to sigmoid from $tanh$. Also, the experiments with TRADES have specified adversarial budgets in l_∞ norms (0.3 for MNIST and 0.031 for CIFAR10). Since we didn't take into account an adversarial budget in our previous experiments, we modify the final results generated by the ATNs by simply restricting the output to be in the specified range as explained under **Implementation details** in section 4.1. The results for ATNs with TRADES are provided in 10. It is seen that the general performance of both ATN and ATNTrip is pretty poor for both datasets. We suspect that it is due to the fact that we are clamping the images due to the small adversarial budgets. However, ATNTrip performs slightly better than ATN for CIFAR10. Note that we only tested against TRADES by training with class 0 as a target for both MNIST and CIFAR10 due to a lack of resources.

[2]https://github.com/yaodongyu/TRADES

6.4 Experiments with Functional Attacks

Now we check the performance of functional attacks against neural networks trained on MNIST and CIFAR10. We performed all our attacks on the NN_1 CNN architecture described in Table 1. Using the mathematical formulations described in section 4.2, we implemented the distortion layer with L distortions. For the MNIST dataset, we set $L = 100$, while for CIFAR10 we set $L = 150$. We used Adam optimization algorithm to find the parameters $\mathbf{c}_u, \mathbf{c}_v, \boldsymbol{\sigma}, \boldsymbol{\alpha}_u, \boldsymbol{\alpha}_v$, with a learning rate equal to 5×10^{-2} and weight decay equal to 10^{-1}, to optimize the objective as indicated in equation 16, with $\lambda = 1$. The hyperparameters for the optimizer were selected using an exponential random search, while the value for λ was selected to be such that: (1) the image was not distorted in a way such that a human eye could realize it was modified, and (2) the attack was often successful. For this hyperparameter, we do not use the same value as the authors of [14] or [15] since this is a different objective function.

	MNIST	CIFAR10
Original accuracy (%)	99.02	63.63
Accuracy with targeted attack (%)	25.83	13.82
Attack success (%)	36.44	68.52

Table 7: Accuracies without attack for the CNNs, accuracies under functional attacks and attack success.

After optimizing the parameters as described in section 4.2, we obtained the accuracies showed in table 7. We can observe from the table that functional attacks on the MNIST classifier are more successful when they are untargeted attacks, while targeted attacks are only half as successful. Compared to the method presented in [15], our functional attack is not able to attack the classifier models as effectively. In their proposals, the authors reported an accuracy of roughly 0% for their attack, which is significantly lower than in our proposed attack. We will explain why this happens in the following paragraph. With respect to attacks on the CIFAR10 classifier, we see that the success of the attack does not depend on whether the attack is made with or without a target, but rather there is a constraint on the effectiveness of the attack itself. However, the attack success is significantly higher than over the MNIST classifier. The authors of [15] and [14] reported 1% and 3% accuracies under attacks on their classifiers.

Target	0	1	2	3	4	5	6	7	8	9
MNIST	32.11	17.89	26.67	23.16	31.11	30.61	37.37	24.74	9.60	28.57
CIFAR10	17.17	11.11	9.60	5.21	15.93	13.27	16.36	21.05	12.26	17.44

Table 8: Performance (in terms of accuracy) of targeted functional attacks

One of the most important results to note from this type of attack is that for the MNIST classifier the effectiveness of the attack is very low compared to the state-of-the-art techniques mentioned above. To analyze this problem further, we show in Table 8 the attack success results for different targets. In particular, looking at the row corresponding to the MNIST classifier, we can notice a strong dependence of the attack success on the target. See number 8, for example, which has a significantly higher success rate than the rest, while number 6 has almost zero success rate. We attribute this difference to the type of parameterized distortion we proposed to perform the functional attack. The type of distortions described in 4.2 defines σ as the *radius of influence of the distortion*, which is constant in any direction, *i.e.* the distortion is radially homogeneous with respect to the distortion center. This causes the distorted image to be prone to have circles or rings (see figure 1a), which marks a tendency for the classification result to be an 8.

With respect to the attack success on the CIFAR10 classifier, we can see from the table 8 that there is no great disparity depending on the target of the attack as there is for the MNIST classifier. However, the results are significantly lower than those reported for the state-of-art techniques, as previously mentioned. The reason for this low efficiency is mainly due to the fact that our technique allows only one type of local image deformation: stretching and squeezing. Although with our technique we decreased the number of parameters to be optimized with respect to existing techniques, our spectrum of distortion types was too limited and did not allow a high attack efficiency.

6.5 Performance of Functional Attacks on Models with TRADES Defense

In this section we describe the functional attacks performed on the MNIST and CIFAR10 classifier models but previously trained with the TRADES technique. Both the architectures of the classifier models and the optimizers and hyperparameters are identical to those of the attacks performed in section 6.4, with the exception of having replaced the loss with that indicated in equation 7. In order to test the effectiveness of the functional attacks against models trained to be robust to adversarial attacks, we took 1,000 samples from the test set and performed the parameter optimization for functional attacks. Table 9 shows the values of accuracy of the models defended with TRADES without being attacked and being attacked using functional attacks. We can observe that, for both MNIST and CIFAR10, the accuracies obtained are higher than those obtained in the attacks on the undefended models, which indicates the robustness that this defense technique gives to the models. However, note that for MNIST the attack success turns out to be very similar to that of the attack on the undefended classifier. This implies that there are samples in the test set that are more vulnerable to spatial distortions, no matter how robust the model is. The same is not true for CIFAR10, where the attack success is much lower in the defended model.

Accuracy (%)	MNIST	CIFAR10
Original	98.92	72.62
With targeted attack	44.99	48.64
Attack success	33.10	25.97

Table 9: Performance of functional attacks on models trained with TRADES

Accuracy (%)	MNIST	CIFAR10
Original	99.48	84.92
ATN	97.32	82.40
ATNTrip	97.68	78.10

Table 10: Performance of ATN on models trained with TRADES

7 Conclusion

We have proposed two new adversarial attacks: an attack based on ATN with triplet loss and a functional attack. We performed adversarial attacks on classifiers trained on MNIST and CIFAR10 datasets based on our proposed attacks. A problem we faced while training ATNs is that they are extremely sensitive to hyperparameters like learning rate and number of epochs. The quality of the generated images vary greatly with a small change in the hyperparameter settings as well as the ATN architectures. We could only try simple ATN architectures and limited hyperparameters due to computational constraints. Though our results show some effectiveness for the triplet loss, more extensive experiments are needed to be performed to determine how useful the triplet loss really is in this setting. Morevover, we have performed all our experiments with a very simple choice of selecting the triplets, as described in Equation 9 to perform fast computations. Better but more computationally intensive triplet mining techniques can be used with ATNs which will likely provide faster convergence and better results. Also, since it is difficult to impose an adversarial budget directly on the original formulation of ATNs, we had to resort to an ad-hoc method of clamping the pixels of the generated adversarial image by the ATN so that it does not exceed the budget which resulted in poor performance against TRADES. It would be useful to find ways to modify ATNs so that they can directly generate adversarial images within a given adversarial budget. Overall the results indicate that ATN with triplet loss provides a promising way for generating adversarial attacks. ATNs with triplet were especially shown to be more effective for black box attacks over ordinary ATNs and showed faster convergence over ordinary ATNs. Another interesting line of work could be using the Triplet loss with other ATN style architectures like AdvGAN [11]

For functional attacks, we proposed a new type of distortion functions, in which the image is modified by locally stretching and squeezing it. This type of functions, while offering a lower computational cost to minimize the objective function, provides limitations when attacking the classifier models. We also performed attacks on MNIST and CIFAR10 classifiers, distorting images of the test sets. However, the results obtained did not equal the state-of-the-art techniques. The main cause was the limitation in the type of parameters chosen, which could be extended by using more general parameterized functions than those proposed, such as twisting, shearing, etc. Another problem we faced with this type of attack is due to the low resolution of the dataset images. Being such low resolution images, the spatial transformations do not cause such a significant effect. Using this attack on higher resolution datasets could lead to good results.

References

[1] I. J. Goodfellow, J. Shlens, and C. Szegedy, “Explaining and harnessing adversarial examples,” in *International Conference on Learning Representations (ICLR)*, 2015.

[2] N. Carlini and D. Wagner, “Towards evaluating the robustness of neural networks,” in *2017 IEEE Symposium on Security and Privacy (S&P)*, pp. 39–57, IEEE, 2017.

[3] A. Kurakin, I. Goodfellow, and S. Bengio, “Adversarial examples in the physical world,” *https://arxiv.org/abs/1607.02533*, 2017.

[4] S. Baluja and I. Fischer, “Learning to attack: Adversarial transformation networks,” in *Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18)*, 2018.

[5] C. Mao, Z. Zhong, J. Yang, C. Vondrick, and B. Ray, “Metric learning for adversarial robustness,” in *Advances in Neural Information Processing Systems (NeuRIPS)*, 2019.

[6] H. Zhang, Y. Yu, J. Jiao, E. Xing, L. E. Ghaoui, and M. Jordan, “Theoretically principled trade-off between robustness and accuracy,” in *Proceedings of the 36th International Conference on Machine Learning (ICML)*, pp. 7472–7482, 2019.

[7] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, “Towards deep learning models resistant to adversarial attacks,” in *International Conference on Learning Representations (ICLR)*, 2018.

[8] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, “Practical black-box attacks against machine learning,” in *Proceedings of the 2017 ACM Asia Conference on Computer and Communications Security (ASIA CCS)*, 2017.

[9] C. Guo, J. R. Gardner, Y. You, A. G. Wilson, and K. Q. Weinberger, “Simple black-box adversarial attacks,” in *International Conference on Machine Learning (ICML)*, 2019.

[10] W. Brendel, J. Rauber, and M. Bethge, “Decision-based adversarial attacks: Reliable attacks against black-box machine learning models,” in *International Conference on Learning Representations (ICLR)*, 2018.

[11] C. Xiao, B. Li, J. yan Zhu, W. He, M. Liu, and D. Song, “Generating adversarial examples with adversarial networks,” in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pp. 3905–3911, 2018.

[12] Y. Jang, T. Zhao, S. Hong, and H. Lee, “Adversarial defense via learning to generate diverse attacks,” in *International Conference on Computer Vision (ICCV)*, 2019.

[13] J. Feng, Q.-Z. Cai, and Z.-H. Zhou, “Learning to confuse: Generating training time adversarial data with auto-encoder,” in *Advances in Neural Information Processing Systems (NeuRIPS)*, 2019.

[14] C. Laidlaw and S. Feizi, “Functional adversarial attacks,” in *Advances in Neural Information Processing Systems (NeuRIPS)*, pp. 10408–10418, 2019.

[15] C. Xiao, J.-Y. Zhu, B. Li, W. He, M. Liu, and D. Song, “Spatially transformed adversarial examples,” in *International Conference on Learning Representations*, 2018.

[16] C. Zhu, R. H. Byrd, P. Lu, and J. Nocedal, “Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization,” *ACM Transactions on Mathematical Software (TOMS)*, vol. 23, no. 4, pp. 550–560, 1997.

[17] P. L. Bartlett, M. I. Jordan, and J. D. McAuliffe, “Convexity, classification, and risk bounds,” *Journal of the American Statistical Association*, vol. 101, no. 47, pp. 138–156, 2006.

[18] P. Li, J. Yi, B. Zhou, and L. Zhang, “Improving the robustness of deep neural networks via adversarial training with triplet loss,” in *Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI)*, 2019.

[19] A. Ilyas, S. Santurkar, D. Tsipras, L. Engstrom, B. Tran, and A. Madry, “Adversarial examples are not bugs, they are features,” in *Advances in Neural Information Processing Systems (NeuRIPS)*, pp. 125–136, 2019.

[20] Y. LeCun, C. Cortes, and J. Burges, “The mnist database of handwritten digits.”

[21] A. Krizhevsky, “Learning multiple layers of features from tiny images,” tech. rep., 2009.

[22] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, “Pytorch: An imperative style, high-performance deep learning library,” in *Advances in Neural Information Processing Systems 32* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), pp. 8024–8035, Curran Associates, Inc., 2019.