BreakDat: (Almost) Breaking the Robust Manifold Defense

Pradyot Prakash
Computer Sciences
UW-Madison
pradyot@cs.wisc.edu

Michael Fernandes
ECE
UW-Madison
mfernandes2@wisc.edu

Gia Hong Geoffrey Lau
ECE
UW-Madison
glau2@wisc.edu

Abstract

While Deep Neural Networks are powerful learning models that have widespread usage in pattern recognition applications, they are susceptible to adversarial examples—small, often imperceptible perturbations to images—that can cause a modern-day classifier to misclassify with a high probability. In this project, we investigate various methods to perform adversarial attacks on the most robust defense algorithm as per our knowledge: the Robust Manifold Defense. Our attack methods include approaches such as the Fast Gradient Sign Method (FGSM), Iterative FGSM (I-FGSM), Single-Pixel Attacks, Max-Min based approach and an algorithm that exploits the geometry of a Variational Autoencoder. Although we were not able to break the defense completely, we were able to force the algorithm to misclassify on MNIST digits 36% of the time using I-FGSM with $\epsilon = 0.2$. Furthermore, our FGSM attack achieved an improvement of at least 8 to 10% in misclassification as compared to the approach taken in the Robust Manifold Defense paper.

1 Introduction

Deep Neural Networks (DNNs) have demonstrated excellent performance and proven to be powerful learning models that achieve state-of-the-art pattern recognition performance in areas such as Computer Vision, Speech Recognition and other domains such as drug discovery and genomics. However, due to imperfections during the training phase, DNNs are highly vulnerable to adversarial examples. Recent research [6] has revealed that the outputs of a DNN can easily be altered by adding some small, often imperceptible perturbations of the input data and lead to misclassifications in modern-day classifiers with high probability. Existence of these perturbations are especially problematic when these classifiers are deployed in the real world, such as in self-driving cars. One reason why defending against these adversarial attacks can be hard is because the natural image is perturbed into some space which is far from the manifold of natural images. Since classifiers aren't trained on images far from the natural images, it does not learn the representation of adversarial examples.

There have been several attempts to make DNNs robust to these adversarial attacks. However, defending against a wide range of attacks remains a challenge. After surveying the literature, the common techniques used to defend against adversarial attacks are through the use of gradient masking techniques. Athayle et al. [1] proposed techniques to overcome these obfuscated gradients and successfully attacked these defenses.

This motivated Ilyas et al. [3] to implement the Robust Manifold Defense, which is the most robust defense algorithm against gradient based attacks as per our knowledge. Their claims of being completely robust to gradient-based attacks motivated us to find several means to undo their defenses. These include other gradient-based attacks not assessed in their work such as the Iterative Fast

ECE 901 Large-Scale Machine Learning and Optimization (May 2018), University of Wisconsin-Madison.

Gradient Sign Method (I-FGSM), a game-theoretic approach that formulates a max-min optimization problem, and brute force single-pixel and N-pixel attacks.

2 Background and Literature Review

2.1 Fast gradient sign methods

The Fast Gradient Sign Method is a simple and quick gradient-based method to generate adversarial images, as described in the work by Goodfellow et al [2]. The method linearizes the cost function and solves for the perturbation that maximizes this cost subject to an l_∞ constraint. The equation to generate adversarial examples is:

$$x_{adv} = x + \epsilon \cdot \text{sign}\left(\nabla_x \ell(x, y, \theta)\right) \tag{1}$$

where ϵ is a hyper-parameter that determines the maximum perturbation to be applied to the original image. ℓ stands for the loss function defined in terms of the input x, the true label y and the parameters of the classifier, θ. In all our experiments, we have used the cross-entropy loss which is a common loss function used for classification.

A variant of the FGSM involves reducing the perturbation to smaller steps α and applying it over a specified number of iterations, otherwise known as Iterative-FGSM [4]. In performing the iterations, we make sure to clip pixel intensity values if they exceed some interval $[a, b]$. This is to ensure that the perturbation does not go beyond the ϵ-neighbourhood of the original image. In our experiments, we used images whose intensities were in $[0, 1]$. Iterative-FGSM can be understood by the following equation where $x_{adv,t}$ represents the adversarial value at an intermediate time instant t:

$$x_{adv,0} = x, \qquad x_{adv,N+1} = \text{clip}_{x,\epsilon}\left(x_{adv,N} + \alpha \cdot \ \text{sign}(\nabla_x \ell(x_{adv,N}, y, \theta))\right) \tag{2}$$

where generally $\alpha < \epsilon$ and the clip operation is performed as follows, in the case where pixel values lie in the interval $[0, 1]$:

$$\text{clip}_{x,\epsilon}(x') = \min\{1, x + \epsilon, \max\{0,\ x - \epsilon,\ x'\}\}. \tag{3}$$

2.2 Pixel attacks

In pixel attacks, only a handful of the pixels in an image are modified by adding relatively small perturbations to the targeted pixels. Su, Vargas and Sakurai demonstrated in [5] that it is possible to alter the output of a DNN in an extremely limited scenario where only one pixel is modified in an image from the CIFAR-10 and ImageNet datasets. The targeted pixel is determined by Differential Evolution - a population-based optimization algorithm. An advantage of this algorithm is that gradient information is not required for optimization, hence it works even for objective functions that are non-differentiable or even unknown.

3 Motivation and Approach

The space of "natural" images is assumed to lie in a low dimensional manifold of the entire real space. The intuition behind the existence of adversarial examples is that they lie close to the space of the "natural" images but their intrinsic dimensionality is high. Since, the adversarial images are close to the original images, if we can somehow project the noisy image to a natural looking image and feed the denoised image to the classifier, the classifier should do a good job at classification. The Robust Manifold Defense paper exploits this idea and proposes a projection operator to make a classifier robust against adversarial examples.

Finding a projection operator onto a space S requires one to know the geometry of S. That is too strict a demand to make for the space of real looking images which one can expect to be highly non-convex. But some of the recent work in generative modelling tries to solve exactly this problem by using Variational Autoencoders (VAE) and Generative Adversarial Networks (GAN). VAEs and GANs have parts to them – Encoder (E) and Decoder (D). E takes an input vector v and outputs a low-dimensional representation of v, say $z \in Z$. D uses z to construct a vector which is close to v.

Training VAE and GANs require different approaches but they end up identifying the latent space Z quite well. The generator, though not mimicking a projection operator (because the input and output space of G are not of the same dimension), can be used to construct a projection operator.

Given an adversarial example, x_{adv}, we would like to identify a $z_{adv} \in Z$ such that $G(z_{adv}) \approx x_{adv}$. This can be modelled as solving $(\arg\min_{z_{adv}} \|G(z_{adv}) - x_{adv}\|)$. Once we have such a z_{adv}, $G(z_{adv})$ gives us a "natural" looking image such that it is similar to x_{adv}. With this, we define our projection operator, P, as follows:

$$P(x) = G\left(\arg\min_{z \in Z} \|G(z) - x\|_2^2\right). \tag{4}$$

This operator operates in two steps: (1) finds the closest latent space representation of x and (2) uses that to construct a real looking image close to x. For any adversarial example, this is essentially trying to approximate it with a smoothened version of the image which we hope looks like a "natural" image because the decoder G has been trained to generate such images.

3.1 Setting up Robust Manifold Defense Model

We reproduce the Robust Manifold Defense Model described by Ilyas et al. [3]. The paper protects a classifier C by projecting its inputs onto the range of a given generator G (decoder of a Variational Autoencoder). The paper claims that this step provides the classifier with robustness against first-order and combined adversarial attacks. This step can be considered as a prior $P(x)$ defined on the space of natural looking images. The authors call this formulation the **Invert and Classify Algorithm** (*INC*). The *INC* algorithm can be described as follows,

- z^* is obtained performing gradient descent in the z space to minimize $\|G(z) - x\|$. Ideally, $z^* = \arg\min \|G(z) - x\|$.
- The classifier is is applied on the projected input $G(z^*)$ which outputs the class of the x

The INC algorithm is described in Algorithm 1.

Algorithm 1: Invert and Classify

Input : Image x, Generator G, Classifier C
Output : Predicted label

begin
  $z_0 \sim \mathcal{N}(0, 1)$
  for $t \leftarrow 1$ *to* T **do**
    $z_t \leftarrow z_{t-1} - \eta\left(\nabla_z \|x - G(z)\|_2^2\right)$
  end
  $z^* \leftarrow z_t$
  return $C(G(z^*))$

The invert and classify algorithm works as follows: given an input image x, the algorithm solves a minimization problem to find a z^* such that $G(z^*)$ is close to the given image x and is a smoothened version of the input image. $G(z^*)$ is fed to the classifier. The paper solves the minimization problem using gradient descent which makes it a non-differentiable method of projecting on the manifold. This would not be easy to attack since a non-fixed number of gradient steps are required [3].

3.2 Problem setup

Dataset The dataset we use for our project is the MNIST handwritten digit database. The images are grayscale images with pixel intensities lying in the interval $[0, 1]$. The MNIST images used in our experiments were not pre-processed.

Naive Classifier (NC) We looked at two classifier models for our experiments. The first one is a simple tensorflow classifier with the following network configuration:

- Input size: 28×28 images vectorized to 784 dimensional vectors
- Convolutional layer with ReLU activation followed by a max pooling layer: 5×5 kernel, 32 filters, 2×2 strides. Output size: $14 \times 14 \times 32$
- Convolutional layer with ReLU activation followed by a max pooling layer: 5×5 kernel, 64 filters, 2×2 strides. Output size: $7 \times 7 \times 64$
- Fully connected layer followed by a dropout: 1024 width
- Fully connected layer followed by softmax: 10 nodes with each node representing the probability of that class
- Argmax on the output of the previous layer to get the predicted class

The loss function was chosen to be the cross-entropy loss and the Adam optimization algorithm was employed to train the network to over 97% accuracy. We used a batch size of 32, 2000 epochs and a dropout rate of 0.5.

Generative modelling For the generator G, we consider the decoder of a Variational Autoencoder (VAE). The VAE architecture is as follows:

Encoder

- Input size: 784 dimensional vector
- Linear layer followed by a softplus layer: 500 width
- Linear layer followed by a softplus layer: 500 width
- Latent space of dimension 20, i.e., encoder outputs $\mu \in \mathbb{R}^{20}$ and $\Sigma \in \mathbb{R}^{20\times 20}$

Decoder

- Input sampled from μ and Σ to get a vector in $\mathbb{R}^{20}$
- Linear layer followed by a softplus layer: 500 width
- Linear layer followed by a softplus layer: 500 width
- Linear layer followed by sigmoid to get a vector in $[0, 1]^{784}$

Robust classifier (RC) The robust approach is exactly as defined in Algorithm 1.

4 Experiments and Our Methods

We elaborate on the various attack techniques performed on the Robust Manifold Defense model in this section. Our attacks can be broadly divided into two major classes: gradient-based attacks and non-gradient based attacks. All our experiments were conducted using Tensorflow.

4.1 Gradient-Based Attacks

The novelty of Robust Manifold Defense stems from the fact that the gradients are not directly accessible. This makes it difficult to directly perform FGSM on it. The projection step is performed iteratively and the number of steps taken is not fixed. Hence, in order to attack the algorithm, we would have to attack the projection operation (Eq. 4). We compute x_{adv} as follows,

$$x_{adv} = x + \epsilon \cdot \text{sign}(\nabla_x \ell(C_\theta(P(x)), y)) \tag{5}$$

where C_θ is our classifier defined using the parameters θ. One idea would be to rely on numerical methods to get the gradient. We can use first principles and go by the definition of gradient and use that instead of the analytic gradient. To simplify notation, let, $f(x) = \ell(C_\theta(P(x)), y)$ and $x \in \mathbb{R}^d$. Let h be a small scalar which is used in the definition of the gradient. Also, let h_j denote the d dimensional vector with all coordinates but the jth coordinate set to h. With these, the algorithm has been described in Algorithm 2.

4.1.1 Fast Gradient Sign Method (FGSM)

In our experiments, we picked $\epsilon \in \{0.5, 1.0, 2.0\}$ and $h = 0.001$. Table 1 summarizes the inaccuracy in classification of the adversarial examples generated.

Algorithm 2: Numerical gradient

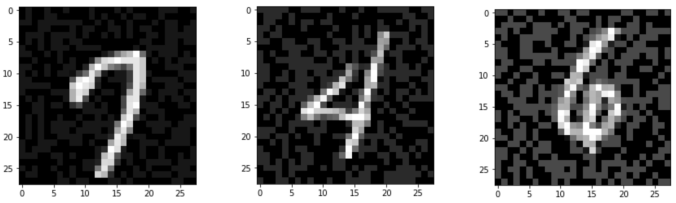
Input : function f, $h \in \mathbb{R}$, $x \in \mathbb{R}^d$
Output : $\nabla_x f(x)$

for $j \leftarrow 1$ *to* d **do**
  $(\nabla f(x))_j \leftarrow \frac{f(x+h_j)-f(x-h_j)}{2h}$
end
return $\nabla_x f(x)$

ϵ	# of adversarial examples generated	Misclassification rate (%)
0.05	150	19.3
0.1	150	22
0.2	150	24.7

Table 1: Misclassification rate using FGSM on the robust classifier

The success rate for misclassifying adversarial examples on the MNIST dataset is better than that reported by Ilyas et al. [3] for their FGSM attack. In their experiments, misclassification rates were at most 10% for $\epsilon \leq 0.2$. It should also be observed that the larger ϵ is, the noisier the generated image is and the more perceptible the perturbation is to the human eye. Some examples of MNIST digit images generated by FGSM are shown in Figure 1.



(a) $\epsilon = 0.05$, original label 7, classified as 9 (b) $\epsilon = 0.1$, original label 4, classified as 7 (c) $\epsilon = 0.2$, original label 6, classified as 2

Figure 1: Adversarial examples generated by FGSM on the robust classifier

4.1.2 Iterative Fast Gradient Sign Method (I-FGSM)

In order to try and improve the misclassification rate, a separate code was run to determine an average number of iterations of I-FGSM to be performed to generate adversarial examples with high confidence of misclassification. This average number of iterations turns out to be 6, but for the purpose of our experiments with I-FGSM, 5 iterations were performed. Table 2 summarizes the results obtained using I-FGSM to generate adversarial examples with $h = 0.001$.

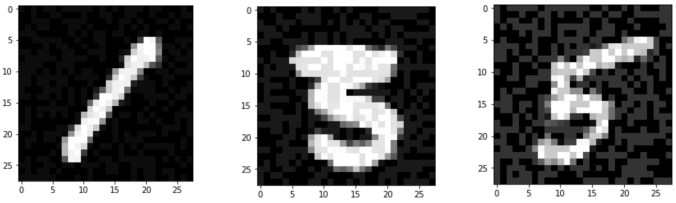
ϵ	# of adversarial examples generated	Misclassification rate (%)
0.05	25	8
0.1	25	20
0.2	25	36

Table 2: Misclassification rate using I-FGSM on the robust classifier

Compared to the vanilla FGSM, I-FGSM produces adversarial images that achieves larger misclassification rate for larger ϵ but is unable to match the performance when ϵ is less than 0.1. Sample

adversarial images generated by I-FGSM are presented in Figure 2. Adversarial examples generated by I-FGSM are perceivably less noisy as compared to their counterparts created through FGSM (as in Figure 1), and this difference is more obvious at larger values of ϵ.

I-FGSM seems to be a better algorithm to create adversarial examples. However, there is a trade-off as we need to perform more computation to create a single example. This is further made worse because the numerical gradient computation is inherently slow because we need to iterate over each pixel of the image.



(a) $\epsilon = 0.05$, original label 1, classified as 3

(b) $\epsilon = 0.1$, original label 3, classified as 8

(c) $\epsilon = 0.2$, original label 5, classified as 8

Figure 2: Adversarial Examples Generated by I-FGSM on the robust classifier

4.2 Max-Min attack

The above approaches used the gradient information to construct adversarial examples using the FGSM algorithm. However, the projection step (Eq. 4) is robust enough and the misclassification rate is low. If we dig deeper into the potential reason for this, it might be because the projection operator is able to remove the adversarial perturbations for most of the adversarially constructed images. So to defeat the Robust Classifier (RC), if we can find ways to misdirect the generator G to a "bad" z, then one can hope that the classifier following that would predict an incorrect label.

This motivates our game theoretic formulation. Let v be the adversarial perturbation that is added to our original image x and hence $x_{adv} = x + v$. The projection step tries to solve the optimization problem,

$$\min_z \|G(z) - x_{adv}\|_2^2 = \min_z \|G(z) - (x + v)\|_2^2. \tag{6}$$

If we can design our v such that whatever the G does, we try to make it worse. This can be better understood through the following optimization problem,

$$\max_{\|v\|_\infty \leq \epsilon} \min_z \|G(z) - (x + v)\|_2^2. \tag{7}$$

Hence, we try to find a perturbation v which tries to make the projection step worse. By imposing the constraint that $\|v\|_\infty \leq \epsilon$, we ensure that the adversarial perturbations are small and the image $x_{adv} = x + v$ still resembles the original image x. Thus, when such a $x + v$ is fed to RC, we might hope that the projection operation doesn't go through as nicely one would have hoped and the classifier would misclassify with a higher probability. Note that this formulation is not optimal because we would want to maximize the misclassification rate and this does not factor that in. But this is the best we can do since any formulation involving the classifier C would be intractable experimentally because the projection operator doesn't have a closed form expression.

Relaxing the l_∞ constraint, the above optimization problem is equivalent to solving two sub-problems (SP_1 and SP_2),

$$SP_1: \quad \min_z \|G(z) - (x + v)\|_2^2 \tag{8}$$

$$SP_2: \quad \min_v -\|G(z) - (x + v)\|_2^2 + \lambda \|v\|_\infty \tag{9}$$

where λ regulates the size of v.

4.2.1 Experiments

The VAE architecture described earlier was used for the experiments. Tensorflow's Gradient Descent Optimizer with a learning rate of 0.01 was used to optimize the two sub-problems. This experiment

was not too stable due to the difficulty involved in solving two sub problems of varying difficulties. Note that solving this problem is similar to the training of GANs which is known to be unstable and highly dependent on the parameter initialization. Furthermore, since SP_1 is generally a harder problem to optimize (because of the presence of a non-convex G), for every optimization step of SP_2, we take multiple gradient descent steps for SP_1. We choose this number to be 5.

To solve both SP_1 and SP_2, we used a $\gamma = 50$. Such a large value of γ was important to constrain the size of v to be small. With such hyper-parameter settings, we iterated for 10000 times until the value of $\|G(z) - (x + v)\|$ had dropped down to about 0.03. Even with tuning γ extensively, we were able to find v which had high $\|v\|_\infty$. Figure 3 shows an image generated by this process. Unfortunately, even with such a high perturbation, the RC did not misclassify.

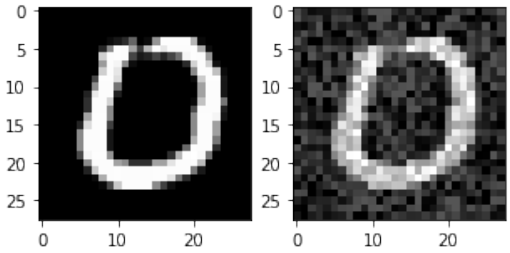


Figure 3: Max-min attack. Left: original image, right: image obtained by solving the optimization problem 7. $\|v\|_\infty = 0.25$, original label: 1, predicted label: 1

4.3 Attack by exploiting the geometry of Variational Autoencoders

Variational Autoencoders identify parts of the latent space which correspond to objects which resemble each other closely. Since we wish to misclassify an image x, if we can add some small noise v to it such that the encoder E of VAE identifies a latent space for $x + v$ which is different from that of x, we can possibly try to fool the decoder G. So if we solve the optimization problem,

$$\min_{\|v\|_\infty \leq \epsilon} \|VAE(x + v) - x'\|_2^2 \tag{10}$$

where $VAE(.)$ represents the output of the Variational Autoencoder and $x' \neq x$ is the target image that we want the VAE to output. By keeping the l_∞ norm of v to be small, we ensure that $x + v$ looks similar to the original image x. Relaxing the l_∞ constraint, we get the problem,

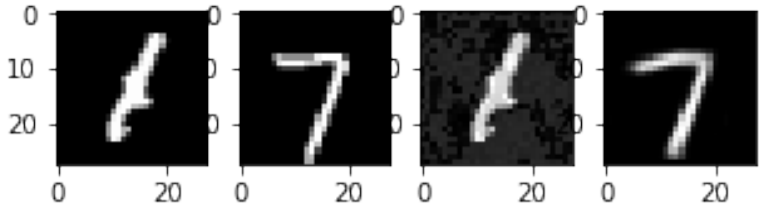
$$\min_v \|VAE(x + v) - x'\|_2^2 + \gamma\|v\|_\infty \tag{11}$$

where γ controls the size of v.

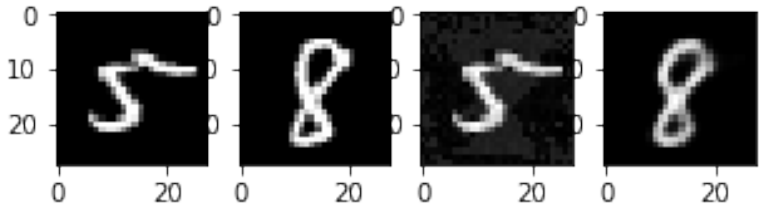
4.3.1 Experiments

The same VAE architecture described in a previous section was used. The experiments were performed with $\gamma = 0.5$ as it led to the best value of $\|v\|_\infty \leq 0.1$. Eq. 11 was solved by using the Gradient Descent Optimizer of Tensorflow by running the optimization for 10000 iterations. The iterations converged quite quickly with a learning rate of 0.1. $x_{adv} = x + v$ was fed as input to RC and the misclassification rate turned out to be around **17%**.

Figure 4 plots images for two digits generated in the experiment. It is evident that the optimization program is able to find small perturbations which can make the VAE predict a different digit for these two cases. This was our observation for all the digits too—there exist small perturbations which can help us get a target image $x_\prime$. However, as we noted earlier that the misclassification rate with this procedure was only 17% suggesting that the robust method is able to remove the adversarial noise for most of the inputs. This can be explained by noting that even though we are able to fool the encoder to represent the input digit differently, the projection step $P(x)$ is quite robust and multiple iterations involved in the projection step are able to remove the adversarial noise quite well so that the classification turns out to be correct for the majority of digits.



(a) $\|v\|_\infty = 0.10$, original label: 1, predicted label: 8



(b) $\|v\|_\infty = 0.08$, original label: 5, predicted label: 5

Figure 4: From left to right: original image (x), target image (x'), adversarial image ($x_{adv} = x + v$), and output of VAE ($VAE(x_{adv})$). (a) shows an image which got misclassified by RC and (b) shows an image which was classified correctly by RC

4.4 Single-Pixel and N-Pixel Attacks

In our pixel attack, we explore two brute-force variations of the pixel attack. For each experiment, a proportion of pixels in the MNIST digit images were randomly chosen and modified in two separate ways:

- Saturation: of the pixels to be attacked in the original image, if their intensities are less than 0.5, they are set to 1. Conversely, if the pixel's intensity is greater than 0.5, they are set to 0.

$$new\_pixel\_value = \begin{cases} 1 & if\ pixel\_value < 0.5 \\ 0 & if\ pixel\_value \geq 0.5 \end{cases} \tag{12}$$

- Midpoint: regardless of pixel intensity in the original image, the pixel intensity is set to 0.5.

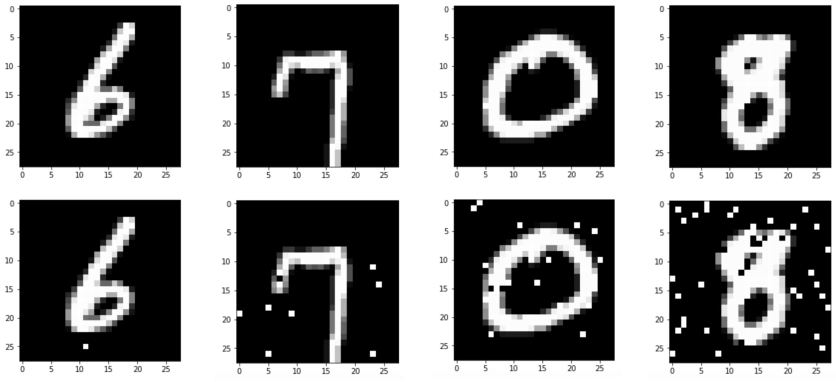
$$new\_pixel\_value = 0.5 \tag{13}$$

We perform pixel attacks on $N \in \{1, 8, 16, 40, 78\}$ pixels of the MNIST digits which comprise 784 pixels each. This corresponds to a proportion of 0.001%, 1%, 2%, 5% and 10% of the pixels in each image respectively. 1000 adversarial examples were generated in each experimental run and the results are reported in Tables 3 and 4. Sample adversarial images which we generated are shown in Figures 5 and 6.

Pixels attacked (%)	# of pixels attacked (N)	# of adversarial examples generated	Misclassification rate (%)
0.001	1	1000	18.5
1	8	1000	19.5
2	16	1000	20.3
5	40	1000	21.8
10	78	1000	28.2

Table 3: Misclassification rate of adversarial examples generated by N-pixel attack with varying N, using Saturation method

From Table 3, misclassification rate increases as we increase the percentage of pixels that undergo perturbations. However, the difference in misclassification rate is only 3.3% for a nearly 5% increase in the number of pixels under attack.



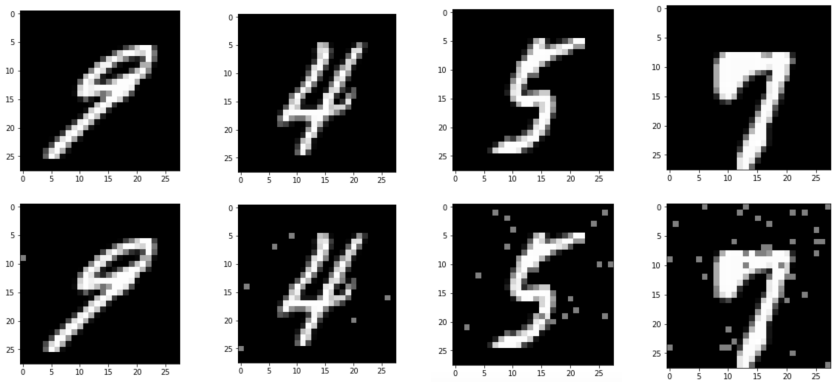
(a) 1-pixel attack (b) 8-pixel attack (c) 16-pixel attack (d) 40-pixel attack

Figure 5: Adversarial examples by N-pixel attack. Top row: original images, bottom row: adversarial examples using the Saturation method

% of pixels attacked	# of pixels attacked (N)	# of adversarial examples	Misclassification Rate (%)
0.001	1	1000	20.5
1	8	1000	19.0
2	16	1000	20.2
5	40	1000	19.2
10	78	1000	22.7

Table 4: Misclassification rate using N-pixel attack using the Midpoint method

In Table 4, we observe that the Midpoint method for N-pixel attack does not result in perceivable impact to the classifier's performance on the adversarial examples. Comparing the adversarial images generated by the two variants of the N-pixel attack method in Figures 5 and 6, it is obvious that the Saturation method creates adversaries that are more perceptible to the human eye.



(a) 1-pixel attack (b) 8-pixel attack (c) 16-pixel attack (d) 40-pixel attack

Figure 6: Adversarial examples by N-pixel attack. Top row: original images, bottom row: adversarial examples using the Midpoint method

5 Conclusion

In this project, we have explored various gradient-based, game-theoretic and pixel attack methods in our attempt to defeat the Robust Manifold Defense by Ilyas et al [3]. Out of all these attacks, our FSGM attack was more effective than the approach explored by Ilyas et al. - increasing misclassification rates by at least 8 to 10%. Furthermore, iterative-FGSM caused the classifier to misclassify on 36% of adversarial images when the maximum allowed perturbation was set to $\epsilon = 0.2$. While this result is better than what was reported in the original paper, we believe that a more thorough investigation of our approaches would lead to a stronger attack on the defense algorithm. We also have some unexplored ideas revolving around the use of GANs to learn to create adversarial examples in an automated way.

References

[1] Anish Athalye, Nicholas Carlini, and David A. Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. *CoRR*, abs/1802.00420, 2018.

[2] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*, 2015.

[3] Andrew Ilyas, Ajil Jalal, Eirini Asteri, Constantinos Daskalakis, and Alexandros G. Dimakis. The robust manifold defense: Adversarial training using generative models. *CoRR*, abs/1712.09196, 2017.

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

[5] Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai. One pixel attack for fooling deep neural networks. *CoRR*, abs/1710.08864, 2017.

[6] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *CoRR*, abs/1312.6199, 2013.