A Tutorial on Attacking DNNs using Adversarial Examples

Christian Both
ID: 260750691
christian.both@mail.mcgill.ca

Arun Rawlani
ID: 260568533
arun.rawlani@mail.mcgill.ca

Andi Rayhan Chibrandy
ID: 260567400
andi.chibrandy@mail.mcgill.ca

***Abstract*—Deep neural networks (DNN) have achieved state-of-the-art performance on image recognition tasks. However, it has been observed that DNNs are vulnerable to small, targeted perturbations performed on the input image, producing what the literature refers to as adversarial examples. In this paper, we present a tutorial on the topic of adversarial examples in the context of deep neural networks. We first outline two attack methods from the literature and give the readers a demonstration on using these methods to attack various convolutional neural network architectures trained on various image datasets. We also discuss defense mechanisms and implement one of them to demonstrate their effectiveness against adversarial examples. We hope to inspire readers to use this knowledge to design more robust machine learning models.**

I. Introduction

Deep neural networks nowadays have been gaining popularity in various machine learning tasks in part due to advances made in research on neural networks and the availability of more capable machines suited to the computationally intensive task of training them. The ability of deep neural networks to represent higher-level concepts from low-level features and generalize them has, amongst other factors, made them suitable for machine learning tasks involving a highly complex input space, such as image classification.

It was first discussed in [1] that deep neural networks are vulnerable to versions of inputs with slight perturbations added, crafted intentionally to cause misclassification. Many explanations have been offered as to why deep neural networks possess this property, which appears to be in contradiction with their ability to generalize higher level concepts. Further research in [2] suggested and provided some quantitative evidence that this vulnerability to adversarial examples is a result of the linearity of most of the components that build up a neural network.

In this tutorial, we aim to provide a basic overview of research done so far in the topic of adversarial examples in the context of deep neural networks. We begin our tutorial by demonstrating a naive adversary in a toy example, which will help the readers build an intuition as to what adversarial examples are and the failure of naive methods to generate them. We build upon the basic knowledge gained in Section II by giving a richer, more detailed description of the concepts we will be discussing next. We then in section IV discuss the theory behind two of the methods found in literature employed to generate adversarial examples and present two defense mechanisms against adversarial attacks. A demonstration of these methods can be found in section V. The tutorial concludes with a discussion of more recent research on this topic and possible future research directions suggested in the literature.

II. Motivation: A Toy Example

Let us assume that we have a machine learning model M that classifies images. The way M works is that given an input image, denoted X, M will output a list of class scores where each entry M_i denotes the likelihood of the input picture containing the object i that the entry specifies, like the following:

$$Scores(X) = \begin{bmatrix} Chicken = 0.65 \\ Duck = 0.25 \\ . \\ . \\ . \\ Car = 0.07 \end{bmatrix} \tag{1}$$

We also assume that we have an adversary A that seeks to produce a version of X that *looks like* X, denoted X^*, but will lead M into misclassifying the image X^* by changing the likeliest class. We would refer to X^* as an *adversarial example*, which is formally explained in Section III. Unfortunately, A does not know much about the way machine learning classifiers work. He will therefore try to do this the naive way. He then comes up with the following algorithm. He first picks a target class, t, that he wants to output as the new likeliest class. Then, using brute-force search, he finds the combination of pixel changes that will try to maximize M_t and result in M classifying to his chosen target class (we can, for the sake of the example, assume A can query M as many times as he wants). *Will A's algorithm work? What can we learn from this example about actual methods of crafting adversarial images?*

One can see that there are several problems with A's algorithm. First of all, misclassification is only guaranteed if A does not object to making a lot of distortions overall to the image. However, the algorithm may then produce samples that look nothing like the original image X. The second problem is that A's algorithm is very inefficient - the running time is easily exponential just to produce a single image.

However, in the real world, adversaries like A are a lot smarter. As a result they have better ways of generating

these adversarial examples which can negatively affect users utilizing these machine learning algorithms. We would request the reader to keep this toy example in mind as we explain more abstract concepts and attempt to answer these two questions: ***1) What are smarter ways that an adversary can employ to produce adversarial examples? 2) How can we defend our systems against such adversaries?***

III. Background information

In order to explain the two questions above, one must possess a deeper understanding the way our target models work. Below we briefly explain the required concepts to make it easier for the reader to understand the methods discussed in the following sections.

A. Deep Neural Networks

A deep neural network can be described as a very large neural network which is typically organized in multiple layers of individual computing units. Those individual units, called neurons are connected by links with different weights and biases which enables neural networks to model highly non-linear relationships [15]. Being equipped with a complex architecture of several hidden layers, deep neural networks are able to learn a higher-level representation of input data layer by layer and are thus able to solve complex classification problems [4].

Neural Networks in general are trained to minimize a loss function. One can think of this as a function that takes in input data along with the true labels and computes the quality of the network's prediction on that input data. Generally, the lower the value of the loss function, the better the predictive performance of the network.

B. Adversarial Examples

Adversarial examples can be described as inputs that are specially crafted to cause a machine learning model to produce an incorrect output [14]. More formally, we define X as a legitimate input that is classified by a learner to a class Y.

$$F(X) = Y \tag{2}$$

We construct an adversarial example X* by adding a small perturbation vector δX to X resulting in a misclassification Y*.

$$F(X + \delta_X) = Y^* \tag{3}$$

However, in order to produce a *good adversarial example* we aim to find the smallest perturbation that produces misclassifcation. This leads us to the following optimization problem to solve:

$$\underset{\delta_X}{argmin} ||\delta_X|| \mathbf{s.t.} F(X + \delta_X) = Y^* \tag{4}$$

However, it is non-trivial to solve this optimization problem due to the non-linear and non-convex properties of a DNN [22]. As a result, we employ techniques that can approximate a reduced perturbation to misclassify the input.

It is now understood that adversarial samples generated to attack a specific network architecture will mislead not only models of similar architectures, but also generalize across different representations [14]. For example, adversarial examples generated to for a logistic regression model can also successfully cause a decision tree to misclassify. It is thus possible to successfully create adversarial examples without having access to the underlying model, which is termed "Black-Box" attack [10].

C. Adversarial Machine Learning

It is important to put adversarial examples in the context to the recognized field of adversarial machine learning, which links the crafting of those artificial images to the bigger picture of *threats* in the real world. Discussing adversarial behaviour is especially relevant when talking about defense mechanisms against adversarial examples. Generally, the term adversarial machine learning deals with adversary attacks on a learning system, e.g. to make a classifier produce false classification which benefits the adversary, and discusses defenses and countermeasures of such an attack [12]. As described in [11] those attacks will naturally occur whenever machine learning is used to prevent illegal or unsanctioned activity. It follows that whenever there is an economic incentive, the adversary will attempt to circumvent the protection provided. Examples given in [12] name amongst others spam filtering, network-intrusion detection and virus detection as applications which are sensitive to adversarial machine learning. As presented in [13], adversarial attacks on learners (e.g. classifiers) can be seen as a game between classifier and adversary to defeat the opponent's strategy. Although several potential scenarios have been pointed out as to how adversarial examples might be used to attack machine learning applications [8], [10], they do not necessarily imply an adversarial attack itself. They might even be intentionally created to improve the learner's ability to generalize, as discussed later in our tutorial.

D. Threats

As DNNs are already implemented in several applications and give a promising field for more future applications in the real world, it is of importance to explore the potential harmfulness of adversarial examples. For example, it is pointed out in [10] that self-driving cars equipped with a DNN classifier to scan the environment and detect traffic signs could possibly be causing accidents when signs requiring a certain action (e.g. stop sign) are manipulated and thus misclassified. Other potentially harmful scenarios can be imagined in manipulating face-recognition systems with adversarial inputs, as well as in fraud detection systems.

IV. Attacking and Defending a Neural Network

We take this opportunity to refer back to A's algorithm described in Section II. We recall that one of the biggest challenges that the algorithm encounters was that it was doing a brute force search in order to find the desired combination of pixel-wise perturbation, resulting in its being exponential in running time. It also risks generating adversarial examples that do not resemble what natural samples from the original class

would look like. Intuitively, if a smarter way of choosing how to perturb pixels can be used instead of a brute force search scheme, one can build an algorithm to produce adversarial examples that not only will take shorter to run but also can potentially induce a smaller amount of perturbation, preventing human detectability.

With these targets in mind, we introduce two methods that are often employed to generate adversarial examples, followed by two defense mechanisms which can provide resistance against such attacks.

A. Fast Gradient Sign Method

Recall from section II that we need a smarter way to pick a combination of pixel perturbations that will produce adversarial examples reliably. Recall also from subsection III-A that neural networks are typically trained to minimize a loss function. We can see that if we can produce samples that cause our loss function to increase when these samples are passed in as input, then we might have a way to produce adversarial examples. We can intuitively see the potential of gradient-based approaches here. Indeed, the Fast Gradient Sign Method operates along a similar line of approach.

It was found in [2] that taking the sign of the gradient of the loss function of a deep neural network with respect to its input and applying perturbations based on this to input images reliably produces adversarial examples. This approach is thus termed the "Fast Gradient Sign Method". Now we explain the formulation behind fast gradient sign approach. Let w be the parameters or weights in the network, x the input to the network and y the target classes that we train our network with. Now we denote the cost function of our neural network as $L(w,x,y)$. To generate adversarial sample x_{adv}, the fast gradient sign method uses the following update rule:

$$x_{adv} = x + \varepsilon sign(\nabla_x L(w,x,y)) \tag{5}$$

In equation 5, ε denotes the magnitude of the perturbation we want applied to the image. The bigger the value of ε, the higher the amount of perturbation applied will be. This will result in more certain misclassification. However, too high a value will result in a sample that does not resemble the original image.

This approach was found to reliably produce adversarial examples using various datasets and classification algorithms while still being relatively cheap to compute - the gradient can be efficiently derived using the back propagation algorithm. On top of this, the amount of perturbation applied can be controlled by changing the value of ε. This approach thus addresses in part some of the problems that the brute force algorithm in section II faces.

B. Jacobian Saliency Map Approach

The Fast Gradient Sign Approach is a good start to solve the first question asked in Section II. However, it still does not guarantee a misclassification once the perturbation has been added to the input, meaning that the adversarial examples may not be good enough to trick the system. Fortunately, for the adversary A, there is a smarter approach to find effective perturbations that guarantee misclassification, at the expense of a longer running time.

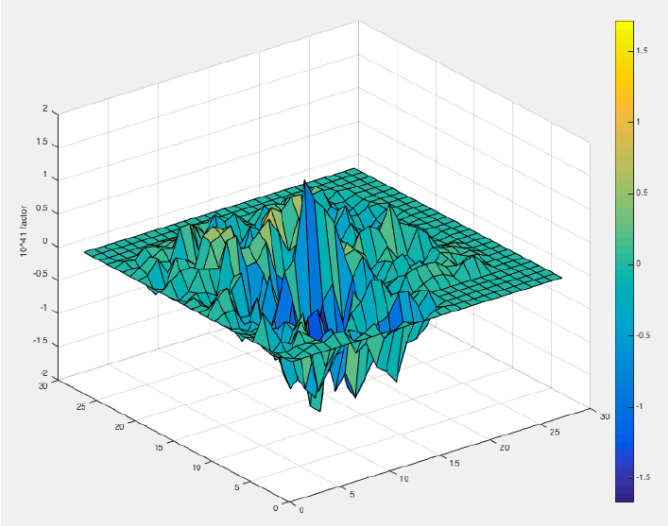


Fig. 1: Saliency map of a 784-dimensional input to the LeNet model. The 784 input dimensions are arranged to correspond to the 28x28 image (MNIST) pixel alignment. [6]

This smarter approach is known Jacobian Saliency Map Approach (JSMA) and is another technique for generating adversarial examples for acyclic feed-forward DNNs [6]. The JSMA approach does a source-target misclassification, where it specifically distorts the source input image from a particular class, s, to misclassify it to a chosen target class, t.

Unfortunately, the non-convexity and non-linearity properties of DNNs make it difficult to identify the set of optimal perturbations required to misclassify the input to the target class. [22] Consequently, this approach aims to find a suitable heuristic in order to enable efficient exploration of the adversarial-sample search space and find the most effective set of perturbations that will lead to the targeted misclassification. *So what heuristic does JSMA employ to overcome the challenge of non-convexity and non-linearity in DNNs?*

The JSMA strategically handles this challenge by exploiting the *forward derivatives* of the learned network. It focuses on the DNNs output with respect to changes in the input which yields the required forward derivatives. The forward derivatives of the model M, learned by this network with n input features, are defined by the Jacobian matrix which can be formulated as follows:

$$\nabla \mathbf{M}(\mathbf{X}) = [\frac{\partial \mathbf{M}(\mathbf{X})}{\partial \mathbf{x_1}} ... \frac{\partial \mathbf{M}(\mathbf{X})}{\partial \mathbf{x_n}}] \tag{6}$$

Forward derivatives assist the adversary by highlighting input features unlikely to produce adversarial examples. This helps the adversary to focus on features with larger forward derivative values that would yield the misclassification with a smaller degree of overall distortion.

To further develop the idea, the matrix in equation 6 is then utilized to form the *adversarial saliency map*. The adversarial saliency map assists the adversary A in finding the most efficient way to produce the targeted misclassification by indicating what features to perturb. Figure 1 provides a helpful visualization of an adversarial saliency map.

To understand how this approach is helpful, recall the *toy example* introduced at the beginning of our tutorial. We know that our model M outputs a vector with the class scores as shown in Equation 1. Then, the generated saliency map aids in increasing the class score of the target class, denoted as $M_{target}(X)$, while decreasing it for all M_i where $i \neq target$ until $target = argmax_i M_i(X)$. This strategy facilitates in finding the set of relevant features to perform perturbations on, such that it will lead to maximizing $M_{target}(X)$ and finally misclassifying the input to the target class.

Additionally, the JSMA has two hyperparameters that govern its performance in different settings. The amount by which selected features are perturbed in each iteration is regulated by θ. Additionally, the Υ controls the maximum distortion/max number of iterations allowed on a sample and limits the number of features that are perturbed to form the adversarial example. The value of these hyperparameters should change with respect to the data being handled by the DNN, as they keep the total distortion in check to avoid human detectability.

Now that we have seen that the adversary A could be smarter in producing its adversarial examples, the reader might believe that its classifier is always at risk. So that leaves us with this question: ***Is there a way to defend our machine learning model against such smart adversaries?***

C. Adversarial Training

In order to give an intuition about the first defense mechanism introduced, recall that adversarial examples do not reflect naturally occurring images but confront the classifier with artificially crafted worst-case perturbations which expose unlearned "blindspots" in the learned model [2]. Thus it is a first instinctive approach to use those examples and confront the model with them again to force correct classification of the corrupted input.

The concept of adversarial training develops this idea further to a systematic training procedure for neural networks where adversarial examples are used for augmentation of the training set. This data augmentation method differs from other known data augmentation schemes such as translating or rotating legitimate examples, which reflects the reproduction of legitimate inputs in different shapes. However, similar to natural data set augmentation, it is aimed to create a regularization effect where the classifier learns to generalize better using the augmented dataset. This regularization can be achieved by adding an additional term when calculating the loss function. The modified loss function has the form:

$$L_{new}(w,x,y) = \alpha L(w,x,y) + (1-\alpha)L(w,x+\varepsilon sign(\nabla_x L(w,x,y)),y) \quad (7)$$

Balanced by the factor α, the loss of the adversarial example is added to the loss of its original example, so that correct classification of both, the legitimate and corrupted input is forced.

The procedure of adversarial training is visualized[1] in Figure 2 and can be understood as an iterative approach. When training a Neural Network, adversarial examples are created after each training epoch and added to the data set for future training. Note then that the new loss function shown in equation 7 simulates the process of generating adversarial examples using the Fast Gradient Sign method and injecting it into the training set.

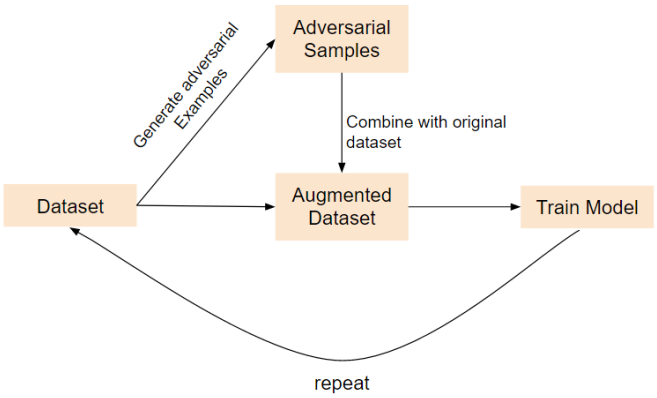


Fig. 2: Principle of Adversarial Training

D. Defensive Distillation

Defensive distillation is another method to mitigate the effect of adversarial examples developed in [9]. It builds on the concept of distilling knowledge from deep architectures introduced in [8] where a learner consisting of a small architecture was used to mimic the output of a large, computational expensive model. However, defensive distillation uses the interaction between two model architectures differently. Here, it is aimed to train a model to reproduce probabilistic labels created by the initial model, which makes the second, distilled model more resilient against adversarial examples. The principle of defensive distillation[2] is presented in Figure 3. The single steps leading to a distilled network architecture can be summarized as follows:

- Train an initial neural network using discrete labels (one single non-zero element in output vector Y which corresponds to the correct class)
- Use a softmax layer which outputs probability scores for for each training sample belonging to a particular class (e.g. P(car) = 0.95, P(ship) = 0.03, P(tree) = 0.02 ...)
- Train a second neural network of identical architecture using the same training set with the difference of assigning previously received probability scores as new labels for each training sample

[1]Own visualization created after [2]
[2]Own visualization created after [9]

Using the probabilistic training labels instead of discrete labels has the following beneficial effect: The weights of the distilled model are prevented from fitting too tightly to the trained data. Defensive distillation thus contributes to a better generalization, which is resilient against the small worst-case perturbations of adversarial examples resulting in correctly classifying them with a high accuracy.

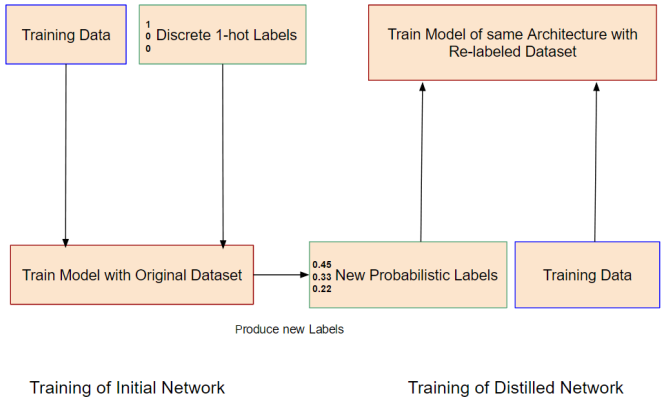


Fig. 3: Principle of Defensive Distillation

V. Methods in Action

In this section we give readers an simulation of a run through of the attack methods outlined above on various network architectures. We then follow up with an implementation of the Adversarial Training defense mechanism to demonstrate that it is possible to defend your machine against such attacks. The first section below gives an overview of the datasets and neural network architectures we used to simulate our methods. The attack and defense mechanisms above are implemented using the *cleverhans* library [20] with *tensorflow* [21] as the backend.

A. Overview of Datasets and Architectures

In this section we give readers the opportunity to learn more about the datasets and the neural network architectures we used when giving practical examples of the attack and defense methods:

1) Dataset:

- MNIST: This dataset consists of 28x28 pixel black and white images of handwritten digits [16]. There are 10 classes, each representing a digit from 0 to 9. There are 60 000 labeled images in the training set and 10 000 labeled images in the test set.
- CIFAR10: This dataset consists of 32x32 RGB images spread over 10 categories [17]. The train set consists of 50 000 labeled images and the test set consists of 10 000 labeled images.
- STL10: This dataset is similar to the CIFAR10 dataset but it has different classes and it contains images that are bigger in resolution - 96 x 96 pixels [18]. The training set contains 5 000 labeled images and the test set contains 8 000 labeled images. For this tutorial, we combine the original training and testing sets to build a dataset consisting 13 000 labeled images. We then perform a 90-10 split to obtain a training dataset consisting of 11 700 samples and a testing set with 1 300 samples, with the distribution of classes being very similar in the training and test sets.

2) Neural Network Architectures: We explore two different neural network architectures. The first one is a simple deep convolutional neural network architecture that we built ourselves. The second one is a pre-trained VGG19 model [19]. This is a much more complex neural network architecture with 19 layers. We decided to use more than one architecture in this tutorial is because we wanted to illustrate through examples that vulnerability to adversarial examples is not specific to a specific architecture of neural networks. Along this line of reasoning, we decided to use a simple architecture along with a much more complicated one. A summary of the architecture of the simpler network can be seen in Table III in the appendix.

B. Fast Gradient Sign Method

In order to give an example of an implementation to the readers, we simulated a run of this algorithm to generate adversarial examples from various datasets. This implementation follows the pseudo code outlined in figure 4. On the CIFAR10 and MNIST dataset we trained our own neural network architecture for 10 epochs. On the STL10 dataset, a more complicated model was required to achieve a reasonable level of accuracy so we trained the VGG19 network on this dataset for 20 epochs. We outline the statistics of the attacks in table I.

```
Algorithm FGSM(M,X,Y, epsilon)
lossFunction ← compute loss of M's predictions of X against Y
gradient ← compute gradient of lossFunction with respect to X
sign ← sign of gradient
perturbation ← epsilon * sign
return X + perturbation
```

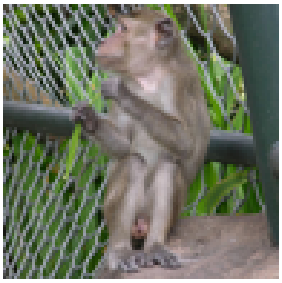
Fig. 4: Algorithm for Generating Adversarial Examples using FGSM where **M** is the model, **X** the input, **Y** the set of labels and **epsilon** the degree of perturbation applied.

In Figure 5, we can see one example of an image and its adversarial version that was generated using this approach. The image is from the STL10 dataset. Here the distortion is clearly visible. However, the adversarial sample still largely contains the characteristics of the original image.

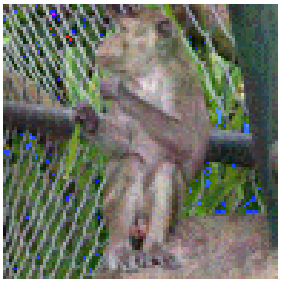
C. Jacobian Saliency Map Approach

Now we perform an implementation of the JSMA algorithm discussed previously to generate adversarial examples for the three datasets defined in Section V-A, where the source input image was distorted to misclassify it to the target class.

We used the algorithm shown in Figure 6 for our implementation of JSMA. For each dataset, the algorithm for JSMA was adapted to work with the various resolutions and color channels that were available. The algorithm then



(a) Original Image



(b) Adversarial Image

Fig. 5: Example of an image from the STL10 dataset and its adversarial version generated using the fast gradient sign method ε=0.04.

(a) Original Image

(b) Adversarial Image

Fig. 7: Example of an CIFAR10 image and its adversarial version generated using the JSMA approach with Υ=0.1 and θ =+1.

chooses the input features, i.e. pixel intensities in case of image classification problems, to be perturbed. For example, in the case of the CIFAR10 dataset, the number of input rows, input columns and color channels was set to 64, 64 and 3, respectively. Furthermore, the sizes of the filters in the pooling layer of the network were also adjusted to work well with this dataset because the default values were set to work with the MNIST dataset.

```
GenerateUsingJSMA(X, Y*, M, ϒ, θ):
X* = X
Features = {1…..|X|} //feature search domain
num_of_pixels = image_width * image_height*number_of_color_chanels
max_iterations = [(ϒ.num_of_pixels)/(2.100)]
s = argmax_i M(X*)_i //choosing the source
t = argmax_i Y*_i //choosing the target
while ((M(X*) ≠ Y*) and (iter < max_iterations) and (features ≠ ∅)) do
  ∇M(X*) = compute_forward_derivative(M(X*))
  (p_1, p_2) = form_saliency_map(∇M(X*), features, Y*, X*)
  Modify the chosen p_1, p_2 in X* by θ
  Remove p_1 from features if p_1 = 0 or 1
  Remove p_2 from features if p_2 = 0 or 1
  s = argmax_i M(X*)_i
  iter = iter + 1
end while
return X*
```

Fig. 6: Algorithm for Generating Adversarial Examples using JSMA where **X** is original image, **Y*** is target network output, **M** is the model, Υ is maximum distortion, and θ is amount of perturbation applied to chosen features.

Once the input parameters were chosen for CIFAR10, the algorithm ran a while loop where it calculated the forward derivative of the network with respect to the input image, formed an adversarial saliency map and then used it to choose the image pixels it wants to perturb. It then perturbs these chosen features by θ. The algorithm stops when it either finds the perturbation which produces the misclassification or reaches the distortion limit set by Υ.

Figure 7 shows the adversarial example generated by the JSMA on a sample image X, taken from the CIFAR10 dataset. Originally, X was correctly classified as a cat (i.e. $M(X)$= cat), but the adversarial example X^*, was misclassified as a ship (i.e. $M(X^*)$= ship).

Furthermore, the algorithm used different combinations of hyperparameter values for different datasets. Depending on the resolution of the input image, the Υ parameter adjusts the maximum number of iterations to increase proportionally with the number of pixels in the input image. The reason for this is because Υ monitors the percentage by which the image could be distorted to craft the adversarial example. As the resolution gets better, the algorithm can afford to change a higher percentage while avoiding human detection. This phenomenon is further validated by the Figure 7, where lower resolution of the CIFAR10 images limits the value of Υ, which makes it harder to form an good adversarial example that produces a misclassification, while being imperceptible to the human eye.

D. Evaluating the Attack Methods

By this point, we have provided an in-depth overview of two methods used to generate adversarial examples to fool our DNN models. Now it would be wise to evaluate how well these methods perform in producing adversarial examples and if they solve the issues we pointed out with our adversary's strategy in our *toy example* in Section II.

1) Fast Gradient Sign Method: In Table I we can see the statistics of our attack. This method consistently produced adversarial examples that were largely misclassified from all the datasets that we trained our networks on. We can see this as testament to the reliability of this simple approach. Note that our selection of ε stems directly from our empirical observation based on the richness of features in the images within each dataset - for example, the STL10 dataset requires a higher ε value compared to the other datasets.

TABLE I: Statistics on the attack using the Fast Gradient Sign Method

PARAMETERS	CIFAR10	STL10	MNIST
Error rate on legitimate samples	33.2%	37.2%	2.5%
epsilon, ε	0.04	0.07	0.04
Error rate on adversarial samples	83.7%	83.5%	97.2%

2) Jacobian Based Saliency Approach: In Table II, we can observe the statistics received by using the JSMA. It can be seen that it successfully produces good adversarial examples for all three datasets, that tricks the learned models to misclassify to the chosen target classes. The adversarial examples effectively tricked the DNN models producing high error rates of 97.2% and 98.0% for MNIST and STL datasets, respectively.

TABLE II: Statistics on attacking the simple architecture using the Jacobian Saliency Map Approach

PARAMETERS	CIFAR10	STL10	MNIST
Error rate on legitimate samples	33.2%	36.7%	2.5%
Maximum iterations w.r.t Υ	153	1382	39
Feature Variation, θ	+1	+2	+1
Error rate on adversarial samples	94.3%	98.0%	99.2%
Average perturbed features	4.0%	1.0%	2.9%

Figure 8 shows that adversarial attacks produced by JSMA induce higher error rates than FGSM on all 3 datasets. This can be explained as it uses the heuristic of an adversarial saliency map to only perturb relevant features while keeping the overall distortion in check. In contrast, all pixels are perturbed by ϵ in FGSM which leads to an overall higher distortion. However, this comes at the cost of higher computational time for JSMA as it needs to search through the adversarial sample space, compared to FGSM which does not. For example, it took over 48 hours on a machine with a GPU to produce adversarial examples from the STL10 dataset using JSMA. This makes it unfavourable to use JSMA for datasets with much higher image resolutions.

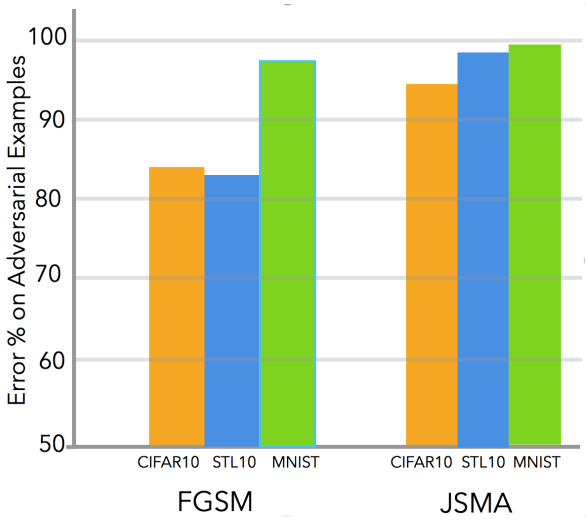


Fig. 8: Comparing the Error % induced on DNNs by Adversarial Examples, using FGSM and JSMA on all 3 datasets

E. Adversarial Training

We analyzed adversarial training on our simple network structure to give an illustration about the effect of this defense method. Figure 9, representing training on the STL10 dataset and Figure 10, using the CIFAR10 dataset show the evolution of validation accuracy for both, original (legitimate) examples and their adversarial counterparts after each epoch. It can be clearly seen in Figure 9 that that accuracy scores increase for both sample types during the training process. Especially the increased accuracy for corrupted images testifies the increasing robustness of the classifier against adversarial examples using adversarial training. Figure 10 does not reveal the increasing robustness against adversarially corrupted images but only shows an increased accuracy on legitimate examples which reflects a normal training process of neural network structures. However, before starting the training process, the validation accuracy for adversarial examples appeared to be extremely low, ranging at 18 percent accuracy. Thus, an increase of robustness against adversarial examples, not visible in the graph, has been achieved after the first training epoch.

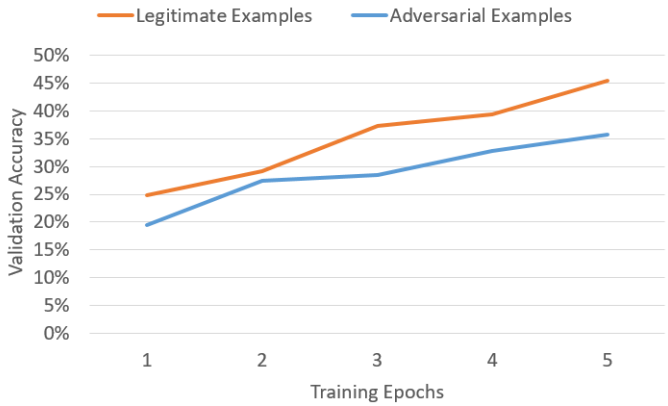


Fig. 9: Comparison of Validation Accuracy for Legitimate and Adversarial Examples when performing Adversarial Training on the STL10 Dataset

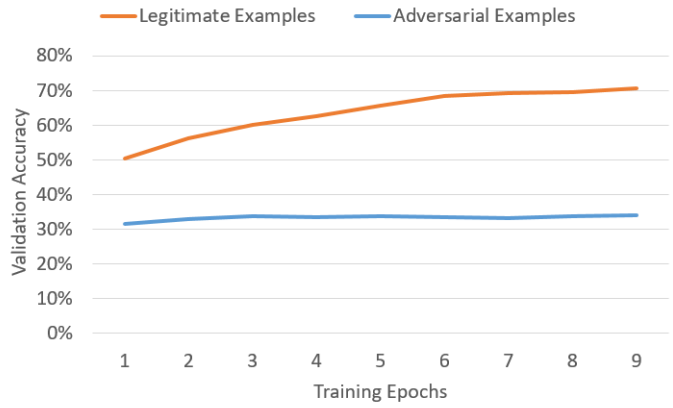


Fig. 10: Comparison of Validation Accuracy for Legitimate and Adversarial Examples when performing Adversarial Training on the CIFAR10 Dataset

VI. Discussion

A. Limitations of Methods to Generate Adversarial Examples

At first glance, it looks like the attack methods we discussed work very well against neural networks that are developed with no particular adversary in mind. It is also remarkable how similar to the original images the samples produced by these methods are. Even the relatively simple Fast Gradient Sign method very successfully attacks neural networks with very similar looking adversarial samples.

It is important to note that these attack methods all rely on one rather big assumption: the adversary has information about the network's architecture, parameters and training data! While allowing for this assumption to be made can produce very interesting theoretical results, it is unrealistic to assume that, for example, when an adversary wants to attack a proprietary machine learning system, they will have all the information they need about the network to carry out these simple attacks. It is in fact more realistic to assume that adversaries in most malicious attacks know very little about the machine learning models that they are attacking. Often the only piece of information the adversary has about the network is the machine learning task it is designed to accomplish. This then begs the question: *does that mean that it is impossible to attack a network without knowing everything there is to know about a network?*

The answer to the question we posed above is, unsurprisingly, no! We refer back to the "Black-Box" methods family we briefly mentioned in III-B. It has been shown that an adversary can easily train a substitute model that behaves in a very similar way and has similar gradients and decision boundaries to the target model and use this substitute model to generate adversarial examples using the methods we discussed in section IV [10]. This can easily be done to circumvent a situation where an adversary does not have access to all the information.

B. Effectiveness of Current Defense Mechanisms

Defending against adversarial attacks is a hard problem. The fact that most machine learning models are not designed and trained with the assumption that an adversary is present also does not help alleviate this problem.

The defense mechanisms so far seem to handle the attacks that we described above quite well. We showed that a premise as simple as Adversarial Training can achieve promising results when it comes to warding off against adversarial examples.

However, most defense mechanisms so far aim to handle attack methods carried out under specific paradigms. For example, adversarial training handles FGSM attacks well and defensive distillation handles JSMA attacks well. Goodfellow and Papernot refer to this as playing a "game of whack-a-mole" [23]. That is to say, defense mechanisms proposed so far address only specific types of vulnerabilities but not all of them. These defense mechanisms are not adaptive to different types of attacks. It also seems that as researchers discover supposedly more promising defense mechanisms, others find more and more novel attack methods to go around these defenses. It, therefore, remains to be discovered what the most optimal paradigm of designing a defense mechanism would be.

VII. Conclusion

Acquiring a better understanding of adversarial examples in general would offer many insights into the theoretical properties of the models that we are attacking. With the ubiquity of deep learning models in multiple intelligent systems and increased dependence on neural networks, it is indeed concerning that models that important systems rely on can be fooled with the simple methods we showed in this tutorial. There is currently much research, not only in finding out the best way to defend against adversarial examples but also in determining if finding such an approach is at all possible [23]. In this tutorial, we have given readers an overview of some well known attack and defense methods currently. While we did not seek to be exhaustive in our description, we hope we have given readers a sufficient insight into this field of research. We encourage inclined readers to read further on this topic and take some of the knowledge from this tutorial in solving this problem.

VIII. Statement of Contributions

A. Christian

Evaluation of defense mechanisms as well as describing and visualizing them in the report. Added the section Background Information to the report and contributed to other sections.

B. Arun

Responsible for writing the description of the Jacobian Based Saliency Approach. Implemented and simulated the JSMA on three different datasets. Created some of the graphs and images used in the paper. Also contributed to the Background Information, Discussion sections of this paper.

C. Andi

Chiefly responsible for the simulation of the fast gradient sign method and parts of the paper pertaining to it. Generated results for Adversarial Training. Contributed in writing of the toy example and discussion on attack and defense methods.

D. All

Made the paper fit the description of a tutorial better as opposed to a regular report. Proof-read and performed many revisions of the paper.

We hereby state that all the work presented in this report is that of the Authors.

REFERENCES

[1] Szegedy, C., Zaremba, W., Sutskever, I. et al. (2014). *Intriguing properties of neural networks.*. ICLR, abs/1312.6199, 2014b. URL http://arxiv.org/abs/1312.6199.

[2] Goodfellow, I., Shlens, J. and Szegedy, C. (2015). *Explaining and harnessing adversarial examples.*. In Proceedings of the International Conference on Learning Representations.

[3] Krizhevsky, A., Sutskever, I. and Hinton, G. (2012). *Imagenet classification with deep convolutional neural networks*. Advances in Neural Information Processing Systems 25, pages 11061114.

[4] Goodfellow, I., Bengio, Y. and Courville, A. (2016). *Deep Learning* MIT Press, url:http://www.deeplearningbook.org.

[5] Deng, J., Dong, W., Socher, R. et al. (2009). *Imagenet: A large-scale hierarchical image database.* In Computer Vision and Pattern Recognition. CVPR 2009, IEEE Conference on, pages 248255.

[6] Papernot, N., McDaniel, P., Jha, S. et al (2016). *The limitations of deep learning in adversarial settings* In Security and Privacy (EuroS&P), 2016 IEEE European Symposium.

[7] Moosavi-Dezfooli S., Fawzi A. and Frossard P. (2015). *Deepfool: a simple and accurate method to fool deep neural networks.*. CoRR (2015) vol. abs/1511.04599.

[8] Hinton G. , Vinyals O., and Dean J. (2014). *Distilling the knowledge in a neural network.* In Deep Learning and Representation Learning Workshop at NIPS 2014. arXiv preprint arXiv:1503.02531, 2014.

[9] Papernot, N., McDaniel, P., Wu, X. et al (2016). *Distillation as a defense to adversarial perturbations against deep neural networks.* In Security and Privacy (SP), 2016 IEEE Symposium on (pp. 582-597). IEEE.

[10] Papernot, N., McDaniel, P., Goodfellow, I. et al. (2017). *Practical Black-Box Attacks against Machine Learning.* ASIA CCS '17 Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, Pages 506-519.

[11] Laskov, P. and Lippmann, R. (2010) *Machine learning in adversarial environments.* Mach Learn (2010) 81: 115. doi:10.1007/s10994-010-5207-6.

[12] Huang, L., Joseph, A. D., Nelson, B. et al. (2011). *Adversarial machine learning.* In Proceedings of the 4th ACM workshop on Security and artificial intelligence (pp. 43-58). ACM.

[13] Dalvi, N., Domingos, P., Sanghai, S., et al. (2004). *Adversarial classification.* In Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 99108. ACM.

[14] N. Papernot, P. McDaniel, and I. Goodfellow. (2016). *Transferability in Machine Learning: from Phenomena to Black-Box Attacks using Adversarial Samples.*ArXiv e-prints, May 2016b. URL http://arxiv.org/abs/1605.07277.

[15] Bengio, Y. (2009). *Learning deep architectures for AI.* Foundations and trends in Machine Learning 2.1 (2009): 1-127.

[16] LeCun, Y., Cortes, C. and Burges, C.J. (1998). *The MNIST database of handwritten digits.*

[17] Krizhevsky, A., and Hinton, G. (2009). *Learning multiple layers of features from tiny images.*

[18] Coates, A., Lee, H. and Ng, A.Y., (2010). *An analysis of single-layer networks in unsupervised feature learning*. Ann Arbor, 1001(48109), p.2.

[19] Simonyan, K. and Zisserman, A., (2014). *Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.*

[20] Papernot, N., Goodfellow, I., Sheatsley, R. et al. (2016). *cleverhans v1. 0.0: an adversarial machine learning library*. arXiv preprint arXiv:1610.00768.

[21] Martn A., Ashish A., Paul A., et al. *TensorFlow: Large-scale machine learning on heterogeneous systems*, 2015. Software available from tensorflow.org.

[22] Larochelle, H., Bengio Y.,Louradour J. et al. *Exploring strategies for training deep neural networks.* Journal of Machine Learning Research 10, no. Jan (2009): 1-40.

[23] Goodfellow, I., Papernot, N. (2017, February 15). *Is attacking machine learning easier than defending it?* [Blog post]. Retrieved from http://www.cleverhans.io/security/privacy/ml/2017/02/15/why-attacking-machine-learning-is-easier-than-defending-it.html

APPENDIX A
ARCHITECTURE OF CUSTOM NEURAL NETOWKR

TABLE III: Architecture of the custom neural network used in for the MNIST and CIFAR10 datasets

Convolutional, 64 filters, 3x3 filter size, ReLU
Convolutional, 128 filters, 3x3 filter size, ReLU
Convolutional, 128 filters, 3x3 filter size, ReLU
MaxPool 2x2 filter size
Dropout 0.25
Convolutional, 64 filters, 3x3 filter size, ReLU
Convolutional, 128 filters, 3x3 filter size, ReLU
Convolutional, 128 filters, 3x3 filter size, ReLU
MaxPool 2x2 filter size
Dropout 0.5
Fully-Connected, 512 nodes
Fully Connected, 10 nodes, softmax activation