The Pennsylvania State University
The Graduate School

ON THE INTEGRITY OF DEEP LEARNING SYSTEMS IN ADVERSARIAL SETTINGS

A Thesis in
Computer Science and Engineering
by
Nicolas Papernot

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

May 2016

The thesis of Nicolas Papernot was reviewed and approved* by the following:

Patrick D. McDaniel
Distinguished Professor of Computer Science and Engineering
Thesis Advisor

Adam D. Smith
Associate Professor of Computer Science and Engineering

Mahmut Taylan Kandemir
Professor of Computer Science and Engineering
Chair of Graduate Program - Department of Computer Science and Engineering

*Signatures are on file in the Graduate School.

Abstract

Deep learning takes advantage of large datasets and computationally efficient training algorithms to outperform other approaches at various machine learning tasks. However, imperfections in the training phase of deep neural networks make them, like other machine learning techniques, vulnerable to *adversarial samples*: inputs crafted by adversaries with the intent of causing machine learning algorithms to misclassify. In this work, we formalize the space of adversaries against deep neural networks (DNNs) and introduce a novel class of algorithms to craft adversarial samples based on a precise understanding of the mapping between inputs and outputs of DNNs. In an application to computer vision, we show that our algorithms can reliably produce samples correctly classified by human subjects but misclassified in specific targets by a DNN with a 97% adversarial success rate while only modifying on average 4.02% of the input features per sample. We then evaluate the vulnerability of different sample classes to adversarial perturbations by defining a hardness measure. Finally, we describe preliminary work outlining defenses against adversarial samples by defining a predictive measure of distance between a benign input and a target classification.

Table of Contents

List of Figures

Acknowledgments

I would like to take the time to thank all of those who have made these first two years at the Pennsylvania State University a meaningful learning experience.

Most importantly, I would like to acknowledge the never failing support of my advisor Patrick McDaniel. I could not hope for better inspiration than his passion for computer security research. He fulfills his advising role marvelously: giving me the right amount of guidance by allowing me to research my ideas freely but being present when needed. His personal and professional devotion to his graduate students is outstanding.

I am thankful for all the hard work of my collaborators and co-authors (by alphabetical order): Z. Berkay Celik, Matt Fredrikson, Ian Goodfellow, Somesh Jha, Ananthram Swami, and Xi Wu.

I would also like to warmly thank Damien Octeau for insightful discussions about the work presented in this manuscript, but also for his friendship, help, and invaluable guidance as I joined the lab.

I am also fortunate to share my office with wonderful labmates: Diman, Lily, Nan, Noor, and Stefan. A special thank to Stefan for the countless coffee breaks taken discussing research.

Finally, I would like to thank my parents, siblings, and family for their support even though I was away from home. They have made being far from home a little less painful, giving me the strength to pursue my dreams abroad.

Last but not least, I would like to thank Adam Smith for accepting to read this work and sit on my committee.

This research would not have been possible without the generous financial support of the United States Army Research Laboratory.

Chapter 1

Introduction

Large neural networks, recast as *deep neural networks* (DNNs) in the mid 2000s, altered the machine learning landscape by outperforming other approaches in many tasks. This was largely made possible by advances reducing the computational complexity of training [32]. Thus, *Deep Learning* (DL) can now take advantage of large datasets together with the learning of hierarchical representations modeled by deep neural networks to achieve accuracy rates higher than previous classification techniques [36, 3]. Deep learning based approaches allowed for breakthroughs in classical machine learning tasks such as classification [27, 41, 30, 69, 68, 42], as well as cutting-edge semi-supervised and unsupervised tasks like automatic feature extraction [64, 63, 22, 70, 19], dimensionality reduction [33, 4], input denoising [75, 74], and reinforcement learning [51, 52, 15, 14, 50]. In short, deep learning is transforming computational processing of data in many domains such as vision [41, 73, 13], speech recognition [18, 62], language processing [16], financial fraud detection [39], financial market modeling [12, 23], human genome modeling [1, 47], malware detection [17, 60], and function recognition in binaries [66].

This increasing use of deep learning is creating incentives for adversaries to manipulate deep neural networks so as to force misclassification of inputs. For instance, applications of deep learning use image classifiers to distinguish inappropriate from appropriate content, and text and image classifiers to differentiate between SPAM and non-SPAM email. An adversary able to craft misclassified inputs would profit from evading detection–indeed such attacks occur today on non-DL classification systems [6, 7, 35]. In the physical domain, consider a driverless car that uses deep learning to identify traffic signs [13]. If slightly altering "STOP" signs causes DNNs to misclassify them, the car will not stop, thus subverting the car's safety.

An *adversarial sample* is an input crafted to cause learning algorithms to misclassify. Although adversarial samples can be crafted for various machine learning techniques [29], we only consider the case of deep neural networks in this manuscript. Note that adversarial samples are created at test time, after the DNN has been trained by the defender, and do not require any alteration of the training process. Figure 1.1 shows examples of adversarial samples taken from our validation experiments. It shows how an image originally showing a digit can be altered to force a DNN to classify it as another digit. Adversarial samples are created from benign samples by adding distortions exploiting the imperfect generalization learned by DNNs from finite training sets [3], and the underlying linearity of most components used to build DNNs [29]. Previous work explored DNN properties that could be used to craft adversarial samples [29, 57, 72]. Simply put, these techniques exploit gradients computed by training algorithms: instead of using these gradients to update DNN parameters as would normally be done, gradients are used to update the original input itself, which is subsequently misclassified by DNNs.

In this manuscript, we describe a new class of algorithms for adversarial sample creation against any feedforward DNN [61] and formalize the threat model space of deep learning with respect to the integrity of output classification. Unlike previous approaches mentioned above, we compute a direct mapping from the input to the output to achieve an explicit adversarial goal. Furthermore, our approach only alters a (frequently small) fraction of input features leading to reduced perturbation of the source inputs. It also enables adversaries to apply heuristic searches to find perturbations leading to targeted misclassifications (perturbing inputs to result in a specific output classification).

More formally, a DNN models a multidimensional function $\mathbf{F} : \mathbf{X} \mapsto \mathbf{Y}$ where $\mathbf{X}$ is a (raw) feature vector and $\mathbf{Y}$ is an output vector. We construct an adversarial sample $\mathbf{X}^*$ from a benign sample $\mathbf{X}$ by adding a perturbation vector $\delta_{\mathbf{X}}$ solving the following optimization problem:

$$\arg\min_{\delta_{\mathbf{X}}} \|\delta_{\mathbf{X}}\| \text{ s.t. } \mathbf{F}\big(\mathbf{X} + \delta_{\mathbf{X}}\big) = \mathbf{Y}^* \tag{1.1}$$

where $\mathbf{X}^* = \mathbf{X} + \delta_{\mathbf{X}}$ is the adversarial sample, $\mathbf{Y}^*$ is the desired adversarial output, and $\|\cdot\|$ a norm appropriate to compare the DNN inputs. Solving this problem is non-trivial, as properties of DNNs make it non-linear and non-convex [43]. Thus, we craft adversarial samples by constructing a mapping from input perturbations to DNN output variations. Note that all research mentioned above took the opposite approach: they used output variations to find corresponding input perturbations. Our understanding of how changes

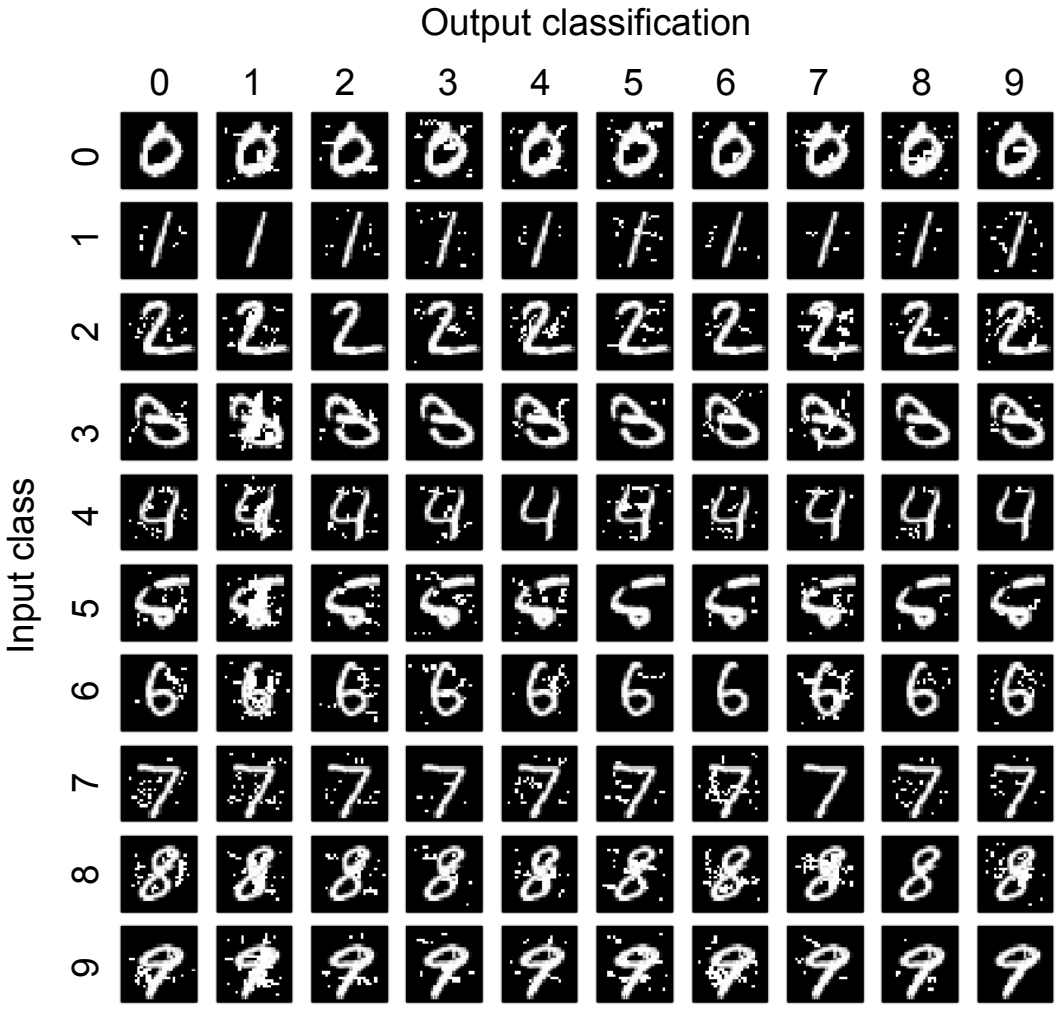


Figure 1.1. Adversarial sample generation: distortion is added to input samples to force the DNN to output adversary-selected classification (min distortion: 0.26%, max distortion: 13.78%, and average distortion: 4.06%).

made to inputs affect a deep neural network's output stems from the *forward derivative*: a matrix we introduce and define as the Jacobian of the function learned by the DNN. The forward derivative is used to construct *adversarial saliency maps* indicating input features to include in perturbation $\delta_{\mathbf{X}}$ in order to produce adversarial samples inducing the desired output from the DNN.

Approaches based on the forward derivative are much more powerful than gradient descent techniques used in prior systems. They are applicable to both supervised and unsupervised architectures and allow adversaries to generate information for broad families of adversarial samples. Indeed, adversarial saliency maps are versatile tools based on the forward derivative and designed with adversarial goals in mind, giving greater control to adversaries with respect to the choice of perturbations, often drastically reducing the proportion of components perturbed. In our work, we consider the following questions to formalize the security of deep learning: (1) "What is the minimal knowledge required to perform attacks against deep neural networks?", (2) "How can vulnerable or resistant samples be identified?", and (3) "How are adversarial samples perceived by humans?".

The adversarial sample generation algorithms are validated using the widely studied *LeNet* architecture (a pioneering DNN used for hand-written digit recognition [45]) and MNIST dataset [46]. We show that any input sample from any source class can be perturbed to be misclassified as any target class given by the adversary with 97.10% success while perturbing on average 4.02% of the input features per sample. The computational costs of the sample generation are modest; samples were each generated in less than a second in our setup. Lastly, we study the impact of our algorithmic parameters on distortion and human perception of samples. This manuscript describes contributions published in [58], which are the following:

- We formalize the space of adversaries against classifier DNNs with respect to adversarial goal and capabilities. Here, we provide a better understanding of how attacker capabilities constrain attack strategies and goals.

- We introduce a new class of algorithms for crafting adversarial samples solely by using knowledge of the DNN architecture. These algorithms (1) exploit *forward derivatives* that inform the learned behavior of DNNs, and (2) build *adversarial saliency maps* enabling efficient explorations of the adversarial-samples space.

- We validate the algorithms using a widely used computer vision DNN. We measure sample distortion and source-to-target hardness, and explore defenses against adversarial samples. We conclude with a study of how human perceive these samples.

Chapter 2

About Deep Learning

This chapter provides a brief overview of deep neural networks and the machine learning tasks they can solve. It also introduces a deep neural network architecture used in the remainder of this manuscript to illustrate our findings.

2.1 Deep Learning and Machine Learning

Deep learning, like any machine learning technique [53, 11, 20, 76], can be partitioned in two categories, depending on whether deep neural networks are trained in a *supervised* or *unsupervised* manner [36, 3]. Supervised training yields models that map unseen samples to a predefined set of outputs using a function inferred from labeled training data. The output data nature yields different supervised problems: classification [41, 17, 13, 27], pattern recognition [11, 20, 65], or regression [26, 56]. On the contrary, unsupervised training learns representations of *unlabeled* training data, and resulting models can be used to generate new samples, or to automate feature engineering by acting as a pre-processing layer for other models [40, 49, 21].

We restrict ourselves to the problem of learning multi-class classifiers in supervised settings using deep neural networks. These DNNs are given an input $\mathbf{X}$ and output a class probability vector $\mathbf{Y}$. The probability vector $\mathbf{Y}$ encodes the DNN's belief of the input sample $\mathbf{X}$ to be in each of the predefined classes. Note that our work remains valid for unsupervised DNNs, and leaves a detailed study of this issue for future work.

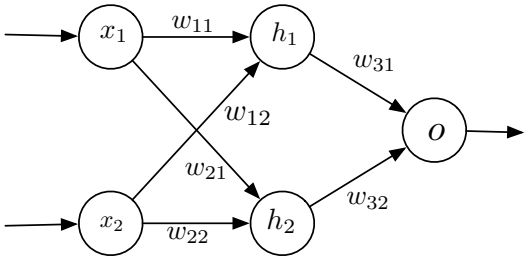


Figure 2.1. Simplified Multi-Layer Perceptron with input $\mathbf{X} = (x_1, x_2)$, hidden layer (h_1, h_2), and output o.

2.2 Shallow Neural Networks

Figure 2.1 illustrates an example shallow feedforward neural network. A shallow neural network is a small neural network that operates (albeit at a smaller scale) identically to the DNNs considered throughout. Traditionally, neural networks with more than one hidden layer are considered to be *deep*. The network has two input neurons x_1 and x_2, a hidden layer with two neurons h_1 and h_2, and a single output neuron o. In other words, it is a simple multi-layer perceptron. Both input neurons x_1 and x_2 take real values in $[0, 1]$ and correspond to the network input: a feature vector $\mathbf{X} = (x_1, x_2) \in [0, 1]^2$. Hidden layer neurons each use the logistic sigmoid function $\phi : x \mapsto \frac{1}{1+e^{-x}}$ as their activation function. This function is frequently used in neural networks because it is continuous (and differentiable), demonstrates linear-like behavior around 0, and saturates as the input goes to $\pm\infty$. Neurons in the hidden layers apply the sigmoid to the weighted input layer: for instance, neuron h_1 computes $h_1(\mathbf{X}) = \phi\left(z_{h_1}(\mathbf{X})\right)$ with $z_{h_1}(\mathbf{X}) = w_{11}x_1 + w_{12}x_2 + b_1$ where w_{11} and w_{12} are weights and b_1 a bias. Similarly, the output neuron applies the sigmoid function to the weighted output of the hidden layer where $z_o(\mathbf{X}) = w_{31}h_1(\mathbf{X}) + w_{32}h_2(\mathbf{X}) + b_3$. Weight and bias values are determined during training. Thus, the overall behavior of the network learned during training can be modeled as a function: $\mathbf{F} : \mathbf{X} \to \phi\left(z_o(\mathbf{X})\right)$.

2.3 Deep Neural Networks

Deep neural networks are large neural networks organized into *layers* of neurons, corresponding to successive representations of the input data [36, 3]. A *neuron* is an individual computing unit transmitting to other neurons the result of the application of its *activation function* on its input. Neurons are connected by links with different *weights* and

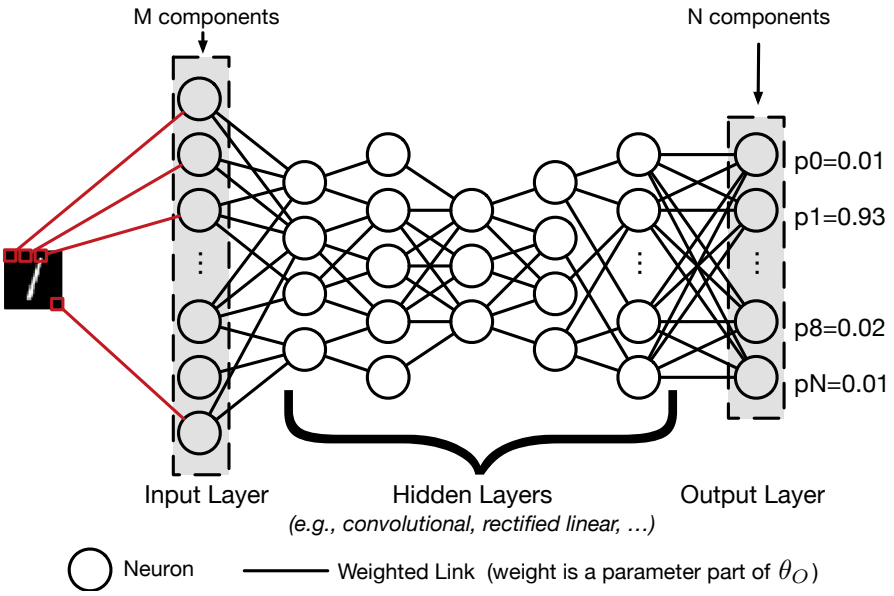


Figure 2.2. Deep Neural Network Classifier Architecture Example (adapted from [59]).

biases characterizing the strength between neuron pairs. Weights and biases can be viewed as deep neural network parameters used for information storage. We define a deep neural network *architecture* to include knowledge of the neural network topology, neuron activation functions, as well as weight and bias values. Weights and biases are determined during *training* by finding values that minimize a *cost function* c evaluated over the training dataset T. Deep Neural Network training is traditionally done by gradient descent using techniques derived from *backpropagation* [61].

In our manuscript we illustrate our findings using classifiers modeled by deep neural networks: for a given input $\mathbf{X}$, the neural network produces a probability vector $\mathbf{F}(\mathbf{X})$ encoding its belief of input $\mathbf{X}$ being in each of the predefined classes. An instance of such a deep neural network classifier is illustrated in its simplest form in Figure 2.2. The architecture illustrated maps images of digits with probability vectors indicating the digit identified in the image. The weight parameters $\theta_{\mathbf{F}}$ hold the knowledge acquired by the model during training. Ideally, the learning algorithm used during the training phase should allow the model to generalize so as to make accurate predictions for inputs outside of the domain explored during training. However, this is not the case in practice as shown by previous attacks manipulating DNN outputs using adversarial samples [29, 58, 72].

Chapter 3

A Taxonomy of Threat Models in Deep Learning

Classical threat models enumerate the goals and capabilities of adversaries in a target domain [37]. This section taxonimizes threat models in deep learning systems and positions several previous works with respect to the strength of the modeled adversary. We begin by providing an overview of DNNs highlighting their inputs, outputs, and function. We then consider the taxonomy presented in Figure 3.1.

3.1 Adversarial Goals

Threats are defined with a specific function to be protected/defended. In the case of deep learning systems, the *integrity* of the classification is of paramount importance. Specifically, an adversary of a deep learning system seeks to provide an input $\mathbf{X}^*$ that results in an incorrect output classification. The nature of the incorrectness represents the adversarial goal, as identified in the X-axis of Figure 3.1. Consider four goals that impact classifier output integrity:

1. **Confidence reduction** - reduce the output confidence classification (thereby introducing class ambiguity)

2. **Misclassification** - alter the output classification to any class different from the *original class*

3. **Targeted misclassification** - produce inputs that force output classification into a specific *target class*. Continuing the example illustrated in Figure 1.1, the adver-

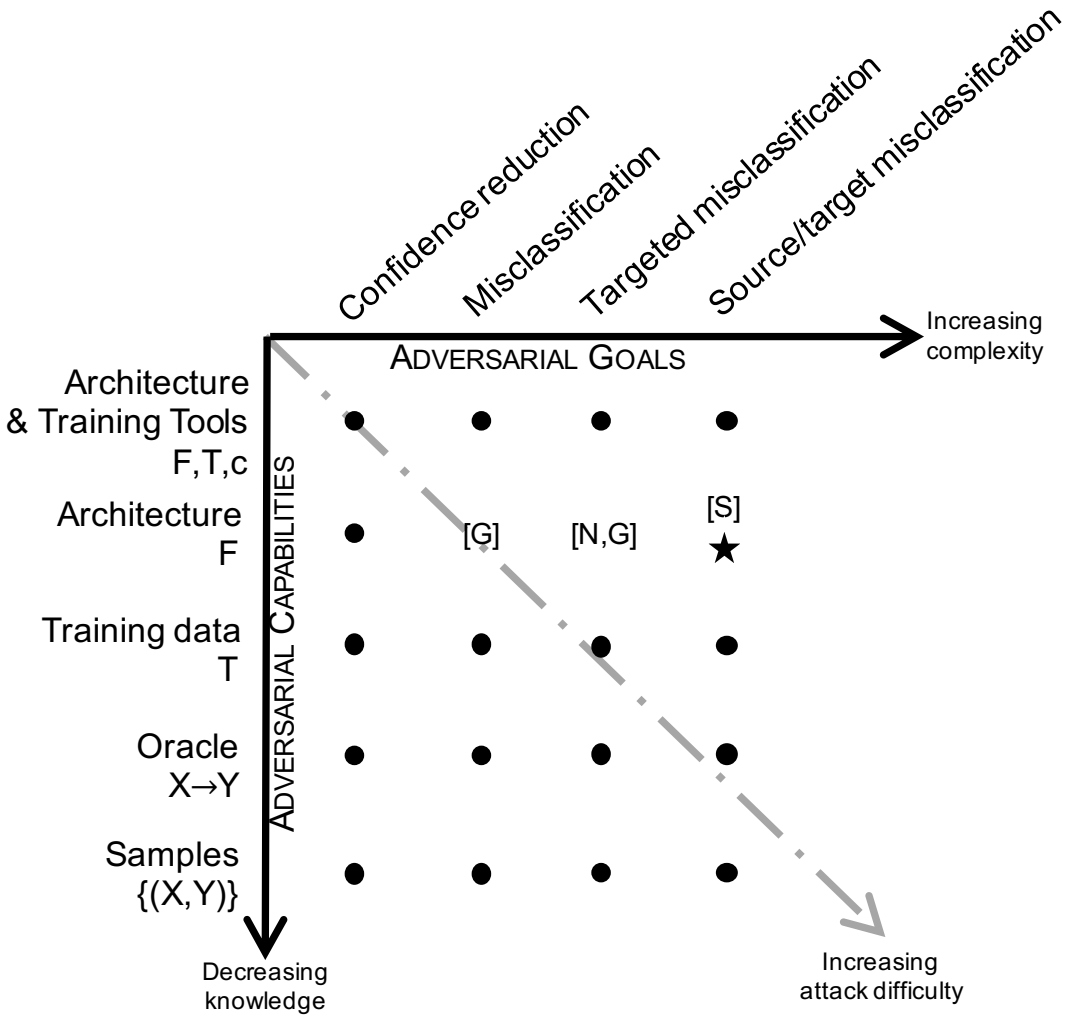


Figure 3.1. Threat Model Taxonomy: our class of algorithms operates in the threat model indicated by the star. [N] refers to [57], [G] to [29], and [S] to [72].

sary would create a set of speckles classified as a digit.

4. **Source/target misclassification** - force the output classification of a specific input to be a specific *target class*. Continuing the example from Figure 1.1, adversaries take an existing image of a digit and add a small set of speckles to classify the resulting image as another digit.

The scientific community recently started exploring adversarial deep learning. Previous work on other machine learning techniques is referenced later in Section 7.

Szegedy et al. introduced a system that generates adversarial samples by perturbing inputs in a way that creates source/target misclassifications [29, 72]. The perturbations made by their work, which focused on a computer vision application, are not distinguishable by humans – for example, small but carefully-crafted perturbations to an image of a vehicle resulted in the DNN classifying it as an ostrich. The authors named this modified input an *adversarial image*, which can be generalized as part of a broader definition of *adversarial samples*. When producing adversarial samples, the adversary's goal is to generate inputs that are correctly classified (or not distinguishable) by humans or other classifiers, but are misclassified by the targeted DNN.

Another example is due to Nguyen et al., who presented a method for producing images that are unrecognizable to humans, but are nonetheless labeled as recognizable objects by DNNs [57]. For instance, they demonstrated how a DNN will classify a noise-filled image constructed using their technique as a television with high confidence. They named the images produced by this method *fooling images*. Here, a fooling image is one that does not have a source class but is crafted solely to perform a targeted misclassification attack.

3.2 Adversarial Capabilities

Adversaries are defined by the information and capabilities at their disposal. The following (and the Y-axis of Figure 3.1) describes a range of adversaries loosely organized by decreasing adversarial strength (and increasing attack difficulty). Note that we only consider attacks conducted at test time: any tampering of the training procedure is outside the scope of this paper and left as future work.

Training data and network architecture - This adversary has perfect knowledge of the DNN used for classification. The attacker has access to the training data T, functions and algorithms used for network training, and is able to extract knowledge about the

DNN's architecture **F**. This includes the number and type of layers, the activation functions of neurons, as well as weight and bias. It also knows which algorithm was used for network training, including the associated loss function c. This is the strongest adversary that can analyze the training data and simulate the DNN *in toto*.

Network architecture - This adversary has knowledge of the network architecture **F** and its parameter values. For instance, this corresponds to an adversary who can collect information about both (1) the layers and activation functions used to design the neural network, and (2) the weights and biases resulting from the training phase. This gives the adversary enough information to simulate the network. Our algorithms assume this threat model, and show a new class of algorithms that generate adversarial samples for supervised and unsupervised feedforward DNNs.

Training data - This adversary is able to collect a *surrogate* dataset, sampled from the same distribution as the original dataset used to train the DNN. However, the attacker is not aware of the architecture used to design the neural network. Thus, typical attacks conducted in this model would likely include training commonly deployed deep learning architectures using the surrogate dataset to approximate the model learned by the legitimate classifier.

Oracle - This adversary has the ability to use the neural network (or a proxy of it) as an "oracle". Here the adversary can obtain output classifications from supplied inputs (much like a chosen-plaintext attack in cryptography). This enables differential attacks, where the adversary can observe the relationship between changes in inputs and outputs (continuing with the analogy, such as used in differential cryptanalysis) to adaptively craft adversarial samples. This adversary can be further parameterized by the number of absolute or rate-limited input/output trials they may perform.

Samples - This adversary has the ability to collect pairs of input and output related to the neural network classifier. However, it cannot modify these inputs to observe the difference in the output. To continue the cryptanalysis analogy, this threat model would correspond to a known plaintext attack. These pairs are labeled output data, and intuition states that they would most likely only be useful when available in very large quantities.

Chapter 4

Attacking Deep Neural Network Integrity at Test Time

In this section, we present a general algorithm for modifying samples so that a DNN yields any desired adversarial output. We later validate this algorithm by having a classifier misclassify samples from a *source class* into a chosen *target class*. This algorithm captures adversaries crafting samples in the setting corresponding to the upper right-hand corner of Figure 3.1. We show that knowledge of the architecture and weight parameters is sufficient to derive adversarial samples against acyclic feedforward DNNs. This requires evaluating the DNN's *forward derivative* in order to construct an *adversarial saliency map* that identifies the set of input features relevant to the adversary's goal. Perturbing the features identified in this way quickly leads to the desired adversarial output, for instance, misclassification. Although we describe our approach with supervised DNNs used as classifiers, it also applies to unsupervised architectures.

4.1 Studying a Simple Neural Network

Recall the simple architecture introduced previously in section 3 and illustrated in Figure 2.1. Its low dimensionality allows us to better understand the underlying concepts behind our algorithms. We indeed show *how small input perturbations found using the forward derivative can induce large variations of the neural network output*. Assuming that input biases b_1, b_2, and b_3 are null, we train this toy network to learn the Boolean `AND` function: the desired output is $\mathbf{F}(\mathbf{X}) = x_1 \wedge x_2$ with $\mathbf{X} = (x_1, x_2)$. Note that non-integer inputs are rounded up to the closest integer, thus we have for instance $0.7 \wedge 0.3 = 0$ or

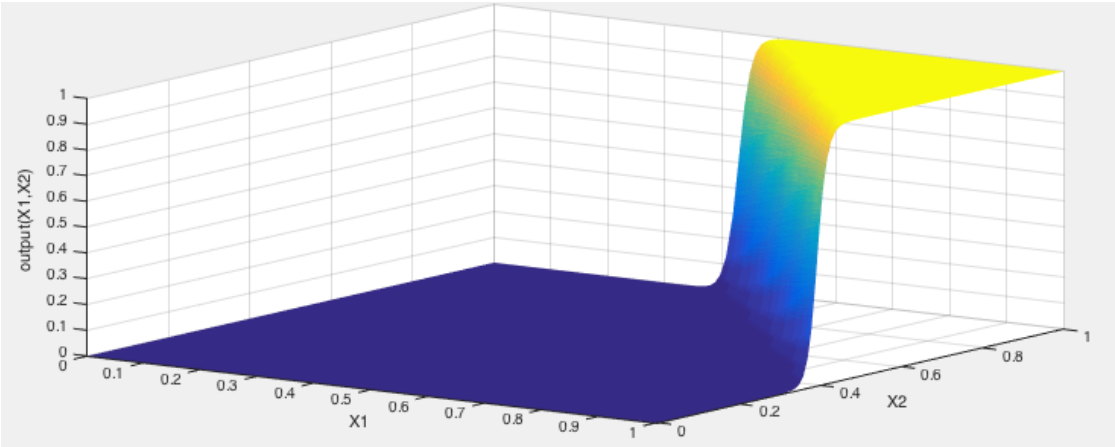


Figure 4.1. The output surface of our simplified Multi-Layer Perceptron for the input domain $[0,1]^2$. Blue corresponds to a 0 output while yellow corresponds to a 1 output.

$0.8 \wedge 0.6 = 1$. Using backpropagation on a set of 1,000 samples, corresponding to each case of the function ($1 \wedge 1 = 1$, $1 \wedge 0 = 0$, $0 \wedge 1 = 0$, and $0 \wedge 0 = 0$), we train for 100 epochs using a learning rate $\eta = 0.0663$. The overall function learned by the neural network is plotted on Figure 4.1 for input values $\mathbf{X} \in [0,1]^2$. The horizontal axes represent the 2 input dimensions x_1 and x_2 while the vertical axis represents the network output $\mathbf{F}(\mathbf{X})$ corresponding to $\mathbf{X} = (x_1, x_2)$.

We are now going to demonstrate how to craft adversarial samples on this neural network. The adversary considers a *legitimate* sample $\mathbf{X}$, classified as $\mathbf{F}(\mathbf{X}) = Y$ by the network, and wants to craft an *adversarial sample* $\mathbf{X}^*$ very similar to $\mathbf{X}$, but misclassified as $\mathbf{F}(\mathbf{X}^*) = Y^* \neq Y$. Recall, that we formalized this problem as:

$$\arg\min_{\delta_\mathbf{X}} \|\delta_\mathbf{X}\| \text{ s.t. } \mathbf{F}(\mathbf{X} + \delta_\mathbf{X}) = \mathbf{Y}^*$$

where $\mathbf{X}^* = \mathbf{X} + \delta_\mathbf{X}$ is the adversarial sample, $\mathbf{Y}^*$ is the desired adversarial output, and $\|\cdot\|$ is a norm appropriate to compare points in the input domain. Informally, the adversary is searching for small perturbations of the input that will induce a modification of the output into $\mathbf{Y}^*$. Finding these perturbations can be done using optimization techniques or even brute force. However such solutions are hard to implement for deep neural networks because of non-convexity and non-linearity [43]. Instead, we propose a systematic approach stemming from the *forward derivative*.

We define the forward derivative as the Jacobian matrix of the function $\mathbf{F}$ learned by the neural network during training. For this example, the output of $\mathbf{F}$ is one dimensional,

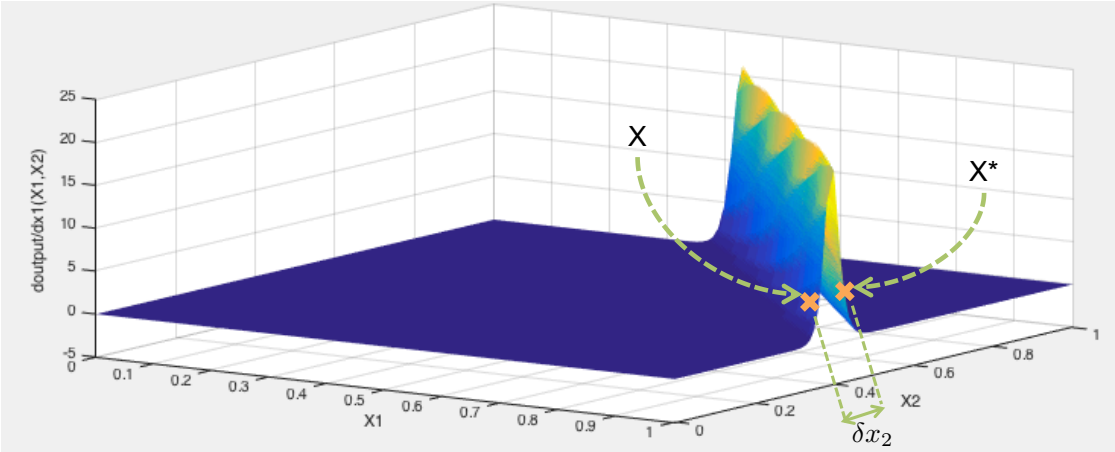


Figure 4.2. Forward derivative of our simplified Multi-Layer Perceptron according to input neuron x_2. Sample $\mathbf{X}$ is benign and $\mathbf{X}^*$ is adversarial, crafted by adding $\delta_{\mathbf{X}} = (0, \delta x_2)$.

the matrix is therefore reduced to a vector:

$$J_{\mathbf{F}}(\mathbf{X}) = \left[\frac{\partial \mathbf{F}(\mathbf{X})}{\partial x_1}, \frac{\partial \mathbf{F}(\mathbf{X})}{\partial x_2}\right] \tag{4.1}$$

Both components of this vector are computable using the adversary's knowledge, and later we show how to compute this term efficiently. The forward derivative for our example network is illustrated in Figure 5, which plots the gradient for the second component $\frac{\partial \mathbf{F}(\mathbf{X})}{\partial x_2}$ on the vertical axis against x_1 and x_2 on the horizontal axes. We omit the plot for $\frac{\partial \mathbf{F}(\mathbf{X})}{\partial x_1}$ because $\mathbf{F}$ is approximately symmetric on its two inputs, making the first component redundant for our purposes. This plot makes it easy to visualize the divide between the network's two possible outputs in terms of values assigned to the input feature x_2: 0 to the left of the spike, and 1 to its right. Notice that this aligns with Figure 4.1, and gives us the information needed to achieve our goal: find input perturbations that drive the output closer to a desired value.

Consulting Figure 4.2 alongside our example network, we can confirm this intuition by looking at a few sample points. Consider $\mathbf{X} = (1, 0.37)$ and $\mathbf{X}^* = (1, 0.43)$, which are both located near the spike in Figure 4.2. Although they only differ by a small amount ($\delta x_2 = 0.05$), they cause a significant change in the network's output, as $\mathbf{F}(\mathbf{X}) = 0.11$ and $\mathbf{F}(\mathbf{X}^*) = 0.95$. Recalling that we round the inputs and outputs of this network so that it agrees with the Boolean `AND` function, we see that X* is an adversarial sample: after rounding, $\mathbf{X}^* = (1, 0)$ and $\mathbf{F}(\mathbf{X}^*) = 1$. Just as importantly, the forward derivative tells us which input regions are unlikely to yield adversarial samples, and are thus more immune to adversarial manipulations. Notice in Figure 4.2 that when either input is

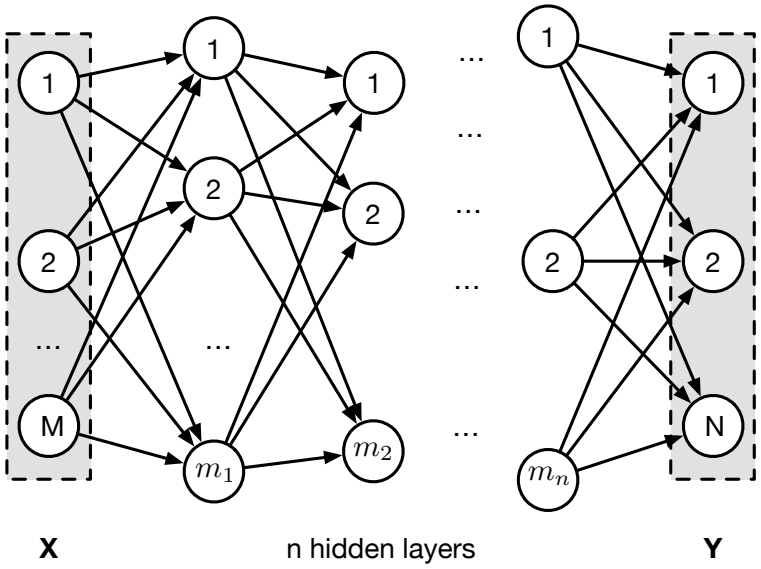
close to 0, the forward derivative is small. This aligns with our intuition that it will be more difficult to find adversarial samples close to $(1, 0)$ than to $(1, 0.4)$. This tells the adversary to focus on features corresponding to larger forward derivative values in a given input when constructing a sample, making its search more efficient and ultimately leading to smaller overall distortions.

The takeaways of this example are thereby: *(1) small input variations can lead to extreme neural network output variations, (2) not all regions from the input domain are conducive to find adversarial samples, and (3) the forward derivative reduces the adversarial-sample search space.*

4.2 Generalizing to Deep Neural Networks

We now generalize this approach to any feedforward DNN, using assumptions and adversary model identical to Section 4.1. The only assumptions made on the architecture are that its neurons form an acyclic DNN, and each uses a differentiable activation function. Note that this last assumption is not limiting because differentiability is required for training. In Figure 4.3, we give an example of a feedforward deep neural network architecture and define some notations used throughout the remainder of the manuscript. Most importantly, the N-dimensional function $\mathbf{F}$ learned by the DNN during training assigns an output $\mathbf{Y} = \mathbf{F}(\mathbf{X})$ when given an M-dimensional input $\mathbf{X}$. We denote by n the number of hidden layers. Layers are indexed by $k \in 0..n+1$ such that $k = 0$ is the index of the input layer, $k \in 1..n$ corresponds to hidden layers, and $k = n + 1$ indexes the output layer.

Algorithm 1 shows our process for constructing adversarial samples. As input, the algorithm takes a benign sample $\mathbf{X}$, a *target adversarial output* $\mathbf{Y}^*$, an acyclic feedforward DNN $\mathbf{F}$, a *maximum distortion* parameter Υ, and a *feature variation* parameter θ. It returns new adversarial sample $\mathbf{X}^*$ such that $\mathbf{F}(\mathbf{X}^*) = \mathbf{Y}^*$, and proceeds in three basic steps: (1) compute the forward derivative $J_{\mathbf{F}}(\mathbf{X}^*)$, (2) construct a saliency map S based on the forward derivative, and (3) modify an input feature i_{max} by θ. This process is repeated until the network outputs $\mathbf{Y}^*$ or the maximum distortion Υ is reached. We now successively detail each step.



Notation
F: function learned by neural network during training
X: input of neural network; M=dim(X)
Y: output of neural network; N=dim(Y)
$W_{k,p}$: row vector of weights from layer k-1 to p-th neuron in layer k
$b_{k,p}$: bias term for p-th neuron in layer k
n: number of hidden layers in neural network
f: activation function of a neuron
H_k : column vector of neuron outputs at hidden layer k

Indices
k: index for layers (between 0 and n+1)
i: index for input X component (between 0 and M)
j: index for output Y component (between 0 and N)
p: index for neurons (between 0 and m_k for any layer k)

Figure 4.3. Example architecture of a feedforward deep neural network with notation used.

Algorithm 1 Crafting adversarial samples: $\mathbf{X}$ is the benign sample, $\mathbf{Y}^*$ is the target network output, $\mathbf{F}$ is the function learned by the network during training, Υ is the maximum distortion, and θ is the change made to features. This algorithm is applied to a specific DNN architecture and dataset in Algorithm 2.

Require: $\mathbf{X}$, $\mathbf{Y}^*$, $\mathbf{F}$, Υ, θ
1: $\mathbf{X}^* \leftarrow \mathbf{X}$
2: $\delta_{\mathbf{X}} \leftarrow \vec{0}$
3: $\Gamma = \{1 \dots |\mathbf{X}|\}$
4: **while** $\mathbf{F}(\mathbf{X}^*) \neq \mathbf{Y}^*$ and $||\delta_{\mathbf{X}}|| < \Upsilon$ **do**
5: Compute forward derivative $J_{\mathbf{F}}(\mathbf{X}^*)$
6: $S(\mathbf{X}, \mathbf{Y}^*) = \texttt{saliency\_map}\,(J_{\mathbf{F}}(\mathbf{X}^*), \Gamma, \mathbf{Y}^*)$
7: $i_{max} \leftarrow \arg\max_i S(\mathbf{X}, \mathbf{Y}^*)[i]$
8: Modify $\mathbf{X}^*_{i_{max}}$ by θ
9: $\delta_{\mathbf{X}} \leftarrow \mathbf{X}^* - \mathbf{X}$
10: **end while**
11: **return** $\mathbf{X}^*$

4.2.1 Forward Derivative of a Deep Neural Network

The first step is to compute the forward derivative for the given sample $\mathbf{X}$. As introduced previously, this is given by:

$$J_{\mathbf{F}}(\mathbf{X}) = \frac{\partial \mathbf{F}(\mathbf{X})}{\partial \mathbf{X}} = \left[\frac{\partial \mathbf{F}_j(\mathbf{X})}{\partial x_i}\right]_{i \in 1..M, j \in 1..N} \tag{4.2}$$

This is essentially the Jacobian of the function corresponding to what the neural network learned during training. The forward derivative computes gradients that are similar to those computed for backpropagation, but with two important distinctions: we take the derivative of the network directly, rather than of its cost function, and we differentiate with respect to the input features rather than the network parameters. As a consequence, instead of propagating gradients backwards, we choose in our approach to propagate them forward, as this allows us to find input components that lead to significant changes in network outputs.

Our goal is to express $J_{\mathbf{F}}(\mathbf{X}^*)$ in terms of $\mathbf{X}$ and constant values only. To simplify our expressions, we now consider one element $(i, j) \in [1..M] \times [1..N]$ of the $M \times N$ forward derivative matrix defined in Equation 4.2: that is the derivative of one output neuron $\mathbf{F}_j$ according to one input dimension x_i. Of course our results are true for any matrix element. We start at the first hidden layer of the DNN. We can differentiate the output of this first hidden layer in terms of the input components. We then recursively

differentiate each hidden layer $k \in 2..n$ in terms of the previous one:

$$\frac{\partial \mathbf{H}_k(\mathbf{X})}{\partial x_i} = \left[\frac{\partial f_{k,p}(\mathbf{W}_{k,p} \cdot \mathbf{H}_{k-1} + b_{k,p})}{\partial x_i}\right]_{p \in 1..m_k} \tag{4.3}$$

where $\mathbf{H}_k$ is the output vector of hidden layer k and $f_{k,j}$ is the activation function of neuron j in layer k. Each neuron p on a hidden or output layer indexed $k \in 1..n+1$ is connected to the previous layer $k-1$ using weights defined in vector $\mathbf{W}_{k,p}$. By defining the weight matrix accordingly, we can define fully or sparsely connected interlayers, thus modeling a variety of architectures. Similarly, we write $b_{k,p}$ the bias for neuron p of layer k. By applying the chain rule, we can write a series of formulae for $k \in 2..n$:

$$\begin{aligned} \left.\frac{\partial \mathbf{H}_k(\mathbf{X})}{\partial x_i}\right|_{p \in 1..m_k} &= \left(\mathbf{W}_{k,p} \cdot \frac{\partial \mathbf{H}_{k-1}}{\partial x_i}\right) \times \\ & \frac{\partial f_{k,p}}{\partial x_i}(\mathbf{W}_{k,p} \cdot \mathbf{H}_{k-1} + b_{k,p}) \end{aligned} \tag{4.4}$$

We are thus able to express $\frac{\partial \mathbf{H}_n}{\partial x_i}$. We know that output neuron j computes the following expression:

$$\mathbf{F}_j(\mathbf{X}) = f_{n+1,j}\left(\mathbf{W}_{n+1,j} \cdot \mathbf{H}_n + b_{n+1,j}\right)$$

Thus, we apply the chain rule again to obtain $J_{\mathbf{F}}[i,j](\mathbf{X})$:

$$\begin{aligned} \frac{\partial \mathbf{F}_j(\mathbf{X})}{\partial x_i} &= \left(\mathbf{W}_{n+1,j} \cdot \frac{\partial \mathbf{H}_n}{\partial x_i}\right) \times \\ & \frac{\partial f_{n+1,j}}{\partial x_i}(\mathbf{W}_{n+1,j} \cdot \mathbf{H}_n + b_{n+1,j}) \end{aligned} \tag{4.5}$$

In this formula, according to our threat model, all terms are known but one: $\frac{\partial \mathbf{H}_n}{\partial x_i}$. This is precisely the term we computed recursively. By plugging these results for successive layers back in Equation 4.5, we get an expression for component (i,j) of the DNN's forward derivative. Hence, the forward derivative $J_{\mathbf{F}}(\mathbf{X})$ of a network $\mathbf{F}$ can be computed for any input $\mathbf{X}$ by successively differentiating layers from the input layer to the output layer. We later discuss in our methodology evaluation the computability of $J_{\mathbf{F}}(\mathbf{X})$ for state-of-the-art DNN architectures. Notably, the forward derivative can be computed using symbolic differentiation.

4.2.2 Adversarial Saliency Maps

We extend saliency maps previously introduced as visualization tools [67] to construct *adversarial saliency maps*. These maps indicate which input features an adversary should perturb in order to effect the desired changes in network output most efficiently. They are thus versatile tools that allow adversaries to generate broad classes of adversarial samples.

Adversarial saliency maps are defined to suit problem-specific adversarial goals. For instance, we later study a network used as a classifier, its output is a probability vector across classes, where the final predicted class value corresponds to the component with the highest probability:

$$label(\mathbf{X}) = \arg\max_j \mathbf{F}_j(\mathbf{X}) \tag{4.6}$$

In our case, the saliency map is therefore based on the forward derivative, as this gives the adversary the information needed to cause the neural network to misclassify a given sample. More precisely, the adversary wants to misclassify a sample $\mathbf{X}$ such that it is assigned a target class $t \neq label(\mathbf{X})$. To do so, probability $\mathbf{F}_t(\mathbf{X})$ of target class t assigned by $\mathbf{F}$ must be increased while the probabilities $\mathbf{F}_j(\mathbf{X})$ of all other classes $j \neq t$ decrease, until $t = \arg\max_j \mathbf{F}_j(\mathbf{X})$. The adversary can accomplish this by *increasing* input features using the following saliency map $S(\mathbf{X}, t)$:

$$S(\mathbf{X}, t)[i] = \begin{cases} 0 \text{ if } J_{it}(\mathbf{X}) < 0 \text{ or } \sum_{j \neq t} J_{ij}(\mathbf{X}) > 0 \\ J_{it}(\mathbf{X}) \left| \sum_{j \neq t} J_{ij}(\mathbf{X}) \right| \text{ otherwise} \end{cases} \tag{4.7}$$

where i is an input feature, and $J_{ij}(\mathbf{X})$ denotes $J_{\mathbf{F}}[i, j](\mathbf{X}) = \frac{\partial \mathbf{F}_j(\mathbf{X})}{\partial \mathbf{X}_i}$. The condition specified on the first line rejects input components with a negative target derivative or an overall positive derivative on other classes. Indeed, $J_{it}(\mathbf{X})$ should be positive in order for $\mathbf{F}_t(\mathbf{X})$ to increase when feature $\mathbf{X}_i$ increases. Similarly, $\sum_{j \neq t} J_{ij}(\mathbf{X})$ needs to be negative to decrease or stay constant when feature $\mathbf{X}_i$ is increased. The product on the second line allows us to consider all other forward derivative components together in such a way that we can easily compare $S(\mathbf{X}, t)[i]$ for all input features. In summary, high values of $S(\mathbf{X}, t)[i]$ correspond to input features that will either increase the target class, or decrease other classes significantly, or both. By increasing these input features, the adversary eventually misclassifies the sample into the target class. A saliency map example is shown in Figure 4.4.

It is possible to define other adversarial saliency maps using the forward derivative,

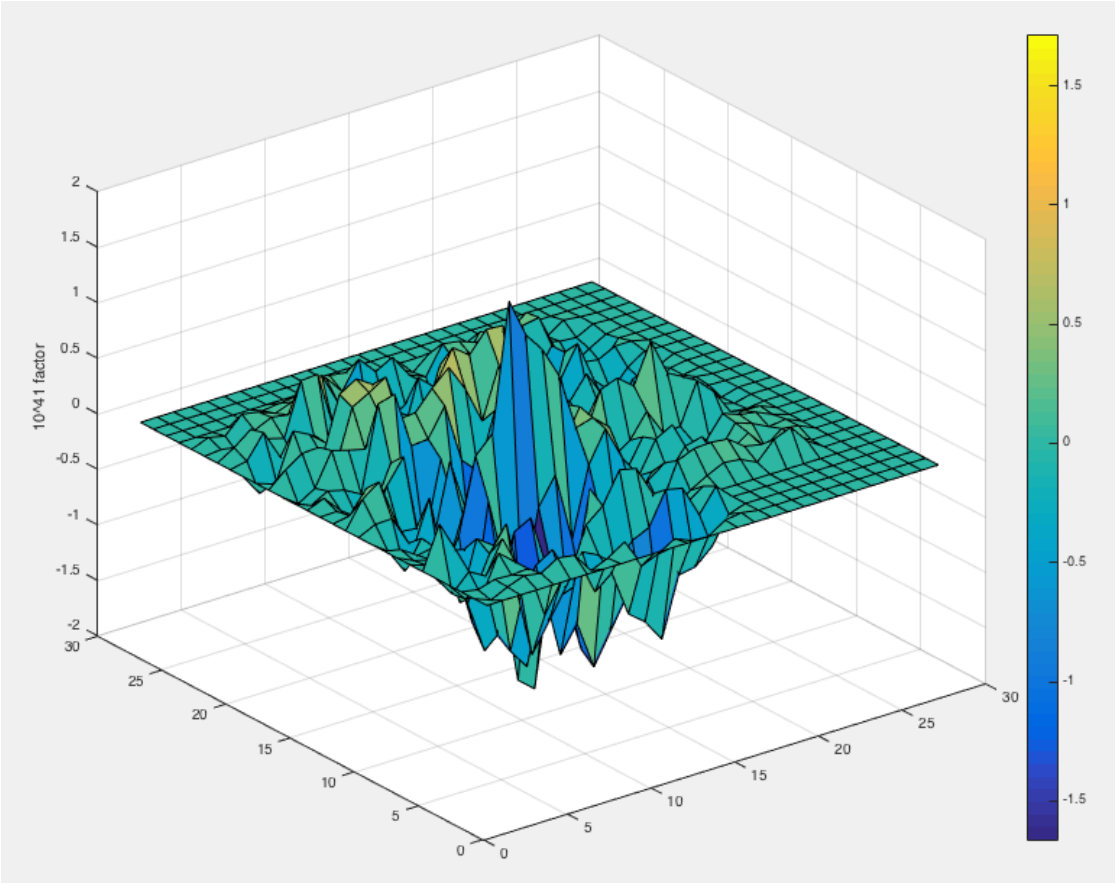


Figure 4.4. Saliency map of a 784-dimensional input to the LeNet architecture (cf. validation section). The 784 input dimensions are arranged to correspond to the 28x28 image pixel alignment. Large absolute values correspond to features with a significant impact on the output when perturbed.

and the quality of the map can have a large impact on the amount of distortion that Algorithm 1 introduces; we will study this in more detail later. Before moving on, we introduce an additional map that acts as a counterpart to the one given in Equation 4.7 by finding features that the adversary should *decrease* to achieve misclassification. The only difference lies in the constraints placed on the forward derivative values and the location of the absolute value in the second line:

$$\tilde{S}(\mathbf{X}, t)[i] = \begin{cases} 0 \text{ if } J_{it}(\mathbf{X}) > 0 \text{ or } \sum_{j \neq t} J_{ij}(\mathbf{X}) < 0 \\ |J_{it}(\mathbf{X})| \left(\sum_{j \neq t} J_{ij}(\mathbf{X}) \right) \text{ otherwise} \end{cases} \tag{4.8}$$

4.2.3 Modifying samples

Once an input feature has been identified by an adversarial saliency map, it needs to be perturbed to realize the adversary's goal. This is the last step in each iteration of Algorithm 1, and the amount by which the selected feature is perturbed (θ in Algorithm 1) is also problem-specific. We discuss in Section 5 how this parameter should be set in an application to computer vision. Lastly, the maximum number of iterations, which is equivalent to the *maximum distortion* allowed in a sample, is specified by parameter Υ. It limits the number of features changed to craft an adversarial sample and can take any positive integer value smaller than the number of features. Finding the right value for Υ requires considering the impact of distortion on humans' perception of adversarial samples – too much distortion or specific distortion patterns might cause adversarial samples to be easily identified by humans.

Chapter 5

Validation of the Attack

We formally described a class of algorithms for crafting adversarial samples misclassified by DNNs using three tools: the forward derivative, adversarial saliency maps, and the crafting algorithm. We now apply them to a DNN used for a vision classification task: handwritten digit recognition. We show that our algorithms successfully craft adversarial samples from any source class to any given target class, which for this application means that any digit can be perturbed so that it is misclassified as any other digit.

We investigate a DNN based on the well-studied LeNet architecture, which has proven to be an excellent classifier for handwritten digits [45]. Recent architectures like AlexNet [41] or GoogLeNet [71] heavily rely on convolutional layers introduced in the LeNet architecture, thus making LeNet a relevant DNN to validate our approach. We have no reason to believe that our method will not perform well on larger architectures. The network input is black and white images (28x28 pixels) of handwritten digits, which are flattened as vectors of 784 features, where each feature corresponds to a pixel intensity taking normalized values between 0 and 1. This input is processed by a succession of a convolutional layer (20 then 50 kernels of 5x5 pixels) and a pooling layer (2x2 filters) repeated twice, a fully connected hidden layer (500 neurons), and a softmax output layer (10 neurons). The output is a 10 class probability vector, where each class corresponds to a digit from 0 to 9, as shown in Figure 5.1. The deep neural network then labels the input image with the class assigned the maximum probability, as shown in Equation 4.6. We train our network using the MNIST training dataset of 60,000 samples [46].

We attempt to determine whether, using the framework introduced in previous sections, we can effectively craft adversarial samples misclassified by the DNN. For instance,

Figure 5.1. Samples taken from the MNIST test set: the respective output vectors are: $[0, 0, 0, 0, 0, 0, 0.99, 0, 0]$, $[0, 0, 0.99, 0, 0, 0, 0, 0, 0]$, and $[0, 0.99, 0, 0, 0, 0, 0, 0, 0]$, where values smaller than 10^{-6} have been rounded to 0.

if we have an image $\mathbf{X}$ of a handwritten digit 0 classified by the network as $label(\mathbf{X}) = 0$ and the adversary wishes to craft an adversarial sample $\mathbf{X}^*$ based on this image classified as $label(\mathbf{X}^*) = 7$, the source class is 0 and the target class is 7. Ideally, the crafting process must find the smallest perturbation $\delta_{\mathbf{X}}$ required to construct the adversarial sample $\mathbf{X}^* = \mathbf{X} + \delta_{\mathbf{X}}$. A perturbation is a set of pixel intensities – or input feature variations – that are added to $\mathbf{X}$ in order to craft $\mathbf{X}^*$. Note that perturbations introduced to craft adversarial samples must remain indistinguishable to humans.

5.1 Crafting algorithm

Algorithm 2 shows the crafting algorithm implemented in our experiments using Python. To train and use DNNs, we use Theano [5], a Python package designed to simplify large-scale scientific computing. Theano allows us to efficiently implement the network architecture, the training through back-propagation, and the forward derivative computation. We configure Theano to make computations with float32 precision, because they can then be accelerated using graphics processors. Indeed, all our experiments are facilitated using GPU acceleration on a machine equipped with a Xeon E5-2680 v3 processor and a Nvidia Tesla K5200 graphics processor.

Our deep neural network makes some simplifications, suggested in the Theano Documentation [48], to the original LeNet-5 architecture. Nevertheless, once trained on batches of 500 samples taken from the MNIST dataset [46] with a learning parameter of $\eta = 0.1$ for 200 epochs, the learned network exhibits a 98.93% accuracy rate on the MNIST training set and 99.41% accuracy rate on the MNIST test set, which are comparable to state-of-the-art accuracies.

It is based on Algorithm 1, but several details have been changed to accommodate our digit recognition problem. Given a network $\mathbf{F}$, Algorithm 2 iteratively modifies a sample $\mathbf{X}$ by perturbing two input features (i.e., pixel intensities) p_1 and p_2 selected by `saliency_map`. The saliency map is constructed and updated between each iteration of the algorithm using the DNN's forward derivative $J_{\mathbf{F}}(\mathbf{X}^*)$. The algorithm halts when one of the following conditions is met: (1) the adversarial sample is classified by the DNN with the target class t, (2) the maximum number of iterations `max_iter` has been reached, or (3) the feature search domain Γ is empty.

Algorithm 2 Crafting adversarial samples for LeNet-5: $\mathbf{X}$ is the benign image, $\mathbf{Y}^*$ is the target network output, $\mathbf{F}$ is the function learned by the network during training, Υ is the maximum distortion, and θ is the change made to pixels.

Require: $\mathbf{X}$, $\mathbf{Y}^*$, $\mathbf{F}$, Υ, θ
$\mathbf{X}^* \leftarrow \mathbf{X}$
$\Gamma = \{1 \dots |\mathbf{X}|\}$ ▷ search domain is all pixels
`max_iter` $= \lfloor \frac{784 \cdot \Upsilon}{2 \cdot 100} \rfloor$
$s = \arg\max_j \mathbf{F}(\mathbf{X}^*)_j$ ▷ source class
$t = \arg\max_j \mathbf{Y}^*_j$ ▷ target class
while $s \neq t$ & `iter` $<$ `max_iter` & $\Gamma \neq \emptyset$ **do**
    Compute forward derivative $J_{\mathbf{F}}(\mathbf{X}^*)$
    $p_1, p_2 =$ `saliency_map`$(J_{\mathbf{F}}(\mathbf{X}^*), \Gamma, \mathbf{Y}^*)$
    Modify p_1 and p_2 in $\mathbf{X}^*$ by θ
    Remove p_1 from Γ if $p_1 == 0$ or $p_1 == 1$
    Remove p_2 from Γ if $p_2 == 0$ or $p_2 == 1$
    $s = \arg\max_j \mathbf{F}(\mathbf{X}^*)_j$
    `iter` $++$
end while
return $\mathbf{X}^*$

The crafting algorithm is fine-tuned by three parameters:

- Maximum distortion Υ: this defines when the algorithm should stop modifying the sample in order to reach the adversarial target class. The maximum distortion, expressed as a percentage, corresponds to the maximum number of pixels to be modified when crafting the adversarial sample. Assuming two additional pixels are modified per iteration, the maximum number of iterations `max_iter` is as follows:

$$\texttt{max\_iter} = \left\lfloor \frac{784 \cdot \Upsilon}{2 \cdot 100} \right\rfloor$$

  where $784 = 28 \times 28$ is the dimension of a sample.

- Saliency map: subroutine `saliency_map` generates a map defining which input

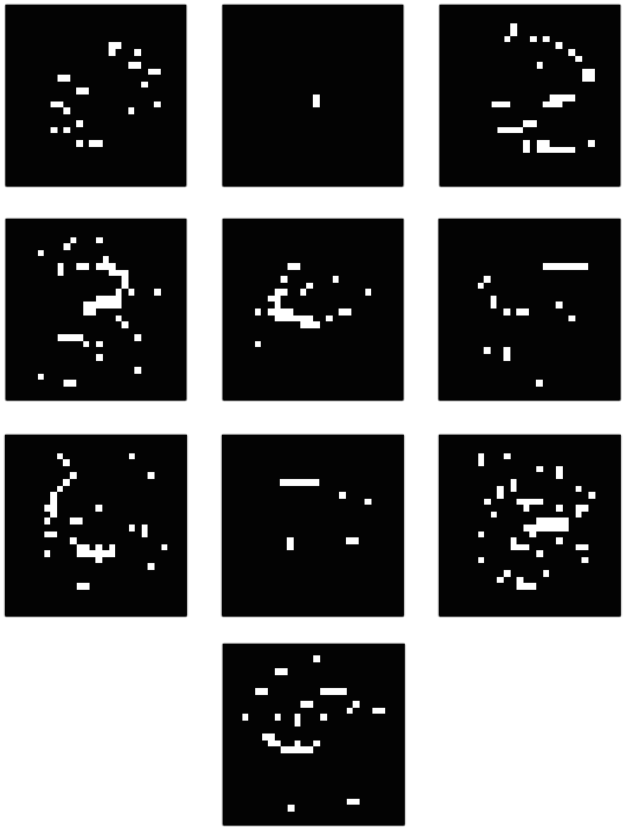


Figure 5.2. Adversarial samples generated by feeding the crafting algorithm an empty input. Each sample produced corresponds to one target class from 0 to 9. Interestingly, for classes $0, 2, 3$ and 5 one can clearly recognize the target digit.

features will be modified at each iteration. Policies used to generate saliency maps vary with the nature of the data handled by the considered DNN, as well as the adversarial goals. We provide a subroutine example later in Algorithm 3.

- Feature variation per iteration θ: once input features have been selected using the saliency map, they must be modified. The variation θ introduced to these features is another parameter that the adversary must set, in accordance with the saliency maps she uses.

The problem of finding good values for these parameters is a goal of our current evaluation, and is discussed later in Section 6. For now, note that human perception is a limiting factor as it limits the acceptable maximum distortion and feature variation introduced. We now show the application of our framework with two different adversarial strategies.

5.2 Crafting by increasing pixel intensities

The first strategy to craft adversarial samples is based on increasing the intensity of some pixels. To achieve this purpose, we consider 10 samples of handwritten digits from the MNIST test set, one from each digit class 0 to 9. We use this small subset of samples to illustrate our techniques. We scale up the evaluation to the entire dataset in Section 6. Our goal is to report whether we can reach any adversarial target class for a given source class. For instance, if we are given a handwritten 0, we increase some of the pixel intensities to produce 9 adversarial samples respectively classified in each of the classes 1 to 9. All pixel intensities changed are increased by $\theta = +1$. We discuss this choice in section 6. We allow for an unlimited maximum distortion $\Upsilon = \infty$. We simply measure for each of the 90 source-target class pairs whether an adversarial sample can be produced or not.

The adversarial saliency map used in the crafting algorithm to select pixel pairs that can be increased is an application of the map introduced in the general case of classification in Equation 4.7. The map aims to find pairs of pixels (p_1, p_2) using the following heuristic:

$$\arg\max_{(p_1,p_2)} \left(\sum_{i=p_1,p_2} \frac{\partial \mathbf{F}_t(\mathbf{X})}{\partial \mathbf{X}_i} \right) \times \left| \sum_{i=p_1,p_2} \sum_{j \neq t} \frac{\partial \mathbf{F}_j(\mathbf{X})}{\partial \mathbf{X}_i} \right| \tag{5.1}$$

where t is the index of the target class, the left operand of the multiplication operation

Algorithm 3 Increasing pixel intensities saliency map: $J_{\mathbf{F}}(\mathbf{X})$ is the forward derivative, Γ the features still in the perturbation search space, and t the target class

Require: $J_{\mathbf{F}}(\mathbf{X})$, Γ, t

```
max ← 0
for each pair (p, q) ∈ Γ do
    α = ∂F_t(X)/∂X_p + ∂F_t(X)/∂X_q
    β = Σ_{j≠t} ( ∂F_j(X)/∂X_p + ∂F_j(X)/∂X_q )
    if α > 0 and β < 0 and −α × β > max then
        p_1, p_2 ← p, q
        max ← −α × β
    end if
end for
return p_1, p_2
```

is constrained to be positive, and the right operand of the multiplication operation is constrained to be negative. This heuristic, introduced in the previous section of this manuscript, searches for pairs of pixels increasing the target class output while reducing the summed output of all other classes when simultaneously increased. The pseudocode of the corresponding subroutine `saliency_map` is given in Algorithm 3.

The saliency map considers pairs of pixels and not individual pixels because selecting pixels one at a time is too strict, and very few pixels would meet the heuristic search criteria described in Equation 4.7. Searching for pairs of pixels is more likely to match the condition: one pixel can compensate a minor flaw of the other pixel. Let's consider an example: p_1 has a target derivative of 5 but a sum of other class derivatives equal to 0.1, while p_2 as a target derivative equal to -0.5 and a sum of other classes derivatives equal to -6. Individually, these pixels do not match the saliency map's criteria stated in Equation 4.7, but combined, the pair does match the saliency criteria defined in Equation 5.1. One would also envision considering larger groups of input features to define saliency maps. However, this comes at increased computational costs as more combinations need to be considered when the group size is increased.

In our algorithm implementation, we compute the DNN forward derivative using the last hidden layer instead of the output probability layer. This is justified by the extreme variations introduced by the logistic regression computed between these two layers to ensure probabilities sum up to 1, leading to extreme derivative values. This reduces the quality of information on how the neurons are activated by different inputs and causes the forward derivative to loose accuracy when generating saliency maps. Better results are achieved when working with the last hidden layer, also made up of 10 neurons, each

corresponding to one digit class 0 to 9. This justifies enforcing constraints on the forward derivative. Indeed, as the output layer used for computing the forward derivative does not sum up to 1, increasing $\mathbf{F}_t(\mathbf{X})$ does not imply that $\sum_{j \neq t} \partial \mathbf{F}_j(\mathbf{X})$ will decrease, and vice-versa.

The algorithm is able to craft successful adversarial samples for all 90 source-target class pairs. Figure 1.1 shows the 90 adversarial samples obtained as well as the 10 original samples used to craft them. The original samples are found on the diagonal. A sample on row i and column j, when $i \neq j$, is a sample crafted from an image originally classified as source class i to be misclassified as target class j.

To verify the validity of our algorithms, and of our adversarial saliency maps, we run a simple experiment. We run the crafting algorithm on an empty input (all pixel intensities initially set to 0) and craft one adversarial sample for each class from 0 to 9. The different samples shown in Figure 5.2 demonstrate how adversarial saliency maps are able to identify input features relevant to classification in a class.

5.3 Crafting by decreasing pixel intensities

Instead of increasing pixel intensities to achieve the adversarial targets, the second adversarial strategy decreases pixel intensities by $\theta = -1$. The implementation is identical to the exception of adversarial saliency maps. The formula is the same as previously written in Equation 5.1 but the constraints are different: the left operand of the multiplication operation is now constrained to be negative, and the right operand to be positive. This heuristic, also introduced in Section 4, searches for pairs of pixels producing an increase in the target class output while reducing the sum of the output of all other classes when simultaneously decreased.

The algorithm is once again able to craft successful adversarial samples for all source-target class pairs. Figure 5.3 shows the 90 adversarial samples obtained as well as the 10 original samples used to craft them. One observation made is that the distortion introduced by reducing pixel intensities seems harder to detect by the human eye. We address the human perception aspect with a study later in Section 6.

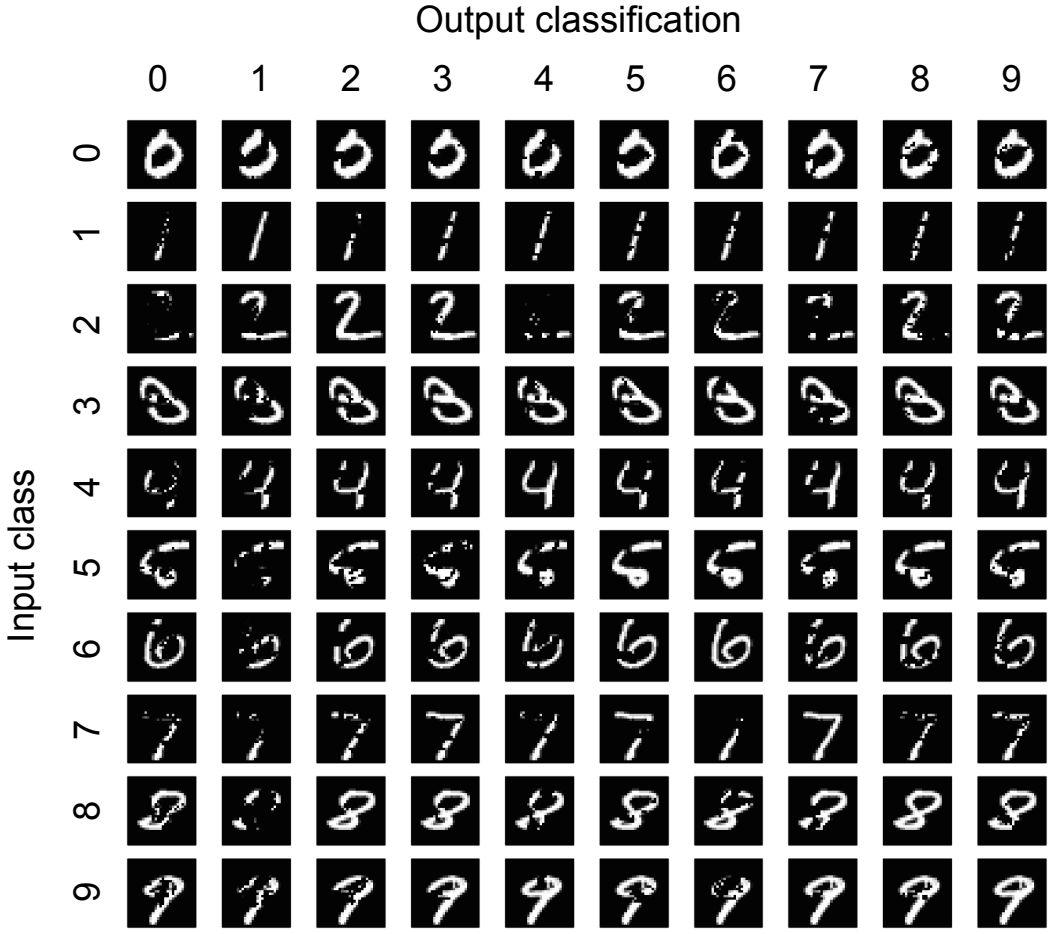


Figure 5.3. Adversarial samples obtained by decreasing pixel intensities: Original samples are on the diagonal, whereas adversarial samples are all non-diagonal elements. Each column corresponds to an output class from 0 to 9.

Chapter 6

Understanding the Attack to build Defense Mechanisms

We now use our experimental setup to answer the following questions: (1) "Can we exploit any sample?", (2) "How can we identify samples more vulnerable than others?" and (3) "How do humans perceive adversarial samples compared to DNNs?". Our primary result is that adversarial samples can be crafted reliably for our validation problem with a 97.10% success rate by modifying samples on average by 4.02%. We define a hardness measure to identify sample classes easier to exploit than others. This measure is necessary for designing robust defenses. We also found that humans cannot perceive the perturbation introduced to craft adversarial samples misclassified by the DNN: they still correctly classify adversarial samples crafted with a distortion smaller than 14.29%.

6.1 Crafting large amounts of adversarial samples

Now that we previously showed the feasibility of crafting adversarial samples for all source-target class pairs, we seek to measure whether the crafting algorithm can successfully handle large quantities of distinct samples of hand-written digits. That is, we now design a set of experiments to evaluate whether or not all legitimate samples in the MNIST dataset can be exploited by an adversary to produce adversarial samples. We run our crafting algorithm on three sets of 10,000 samples each extracted from one of the three MNIST training, validation, and test subsets[1]. For each of these samples, we

[1]Note that we extracted original samples from the dataset for convenience. Any sample can be used with the adversarial crafting algorithm.

craft 9 adversarial samples, each of them classified in one of the 9 target classes distinct from the original legitimate class. Thus, we generate 90,000 samples for each set, leading to a total of 270,000 adversarial samples. We set the maximum distortion to $\Upsilon = 14.5\%$ and pixel intensities are increased by $\theta = +1$. The maximum distortion was fixed after studying the effect of increasing it on the success rate τ. We found that 97.1% of the adversarial samples could be crafted with a distortion of less than 14.5% and observed that the success rate did not increase significantly for larger maximum distortions. Parameter θ was set to $+1$ after observing that decreasing it or giving it negative values increased the number of features modified, whereas we were interested in reducing the number of features altered during crafting. One will also notice that because features are normalized between 0 and 1, if we introduce a variation of $\theta = +1$, we always set pixels to their maximum value 1. This justifies why in Algorithm 2, we remove modified pixels from the search space at the end of each iteration. The impact on performance is beneficial, as we reduce the size of the feature search space at each iteration. In other words, our algorithm performs a best-first heuristic search without backtracking.

We measure the success rate τ and distortion of adversarial samples on the three sets of 10,000 samples. The *success rate* τ is defined as the percentage of adversarial samples that were successfully classified by the DNN as the adversarial target class. The *distortion* is defined to be the percentage of pixels modified in the legitimate sample to obtain the adversarial sample. In other words, it is the percentage of input features modified in order to obtain adversarial samples. We compute two average distortion values: one taking into account all samples and a second one, denoted by ε, only taking into account successful samples. Figure 6.1 presents the results for the three sets from which the original samples were extracted. Results are consistent across all sets. On average, the success rate is $\tau = 97.10\%$, the average distortion of all adversarial samples is 4.44%, and the average distortion of successful adversarial samples is $\varepsilon = 4.02\%$. This means that on average 32 out of 784 pixels are modified to craft a successful adversarial sample. The first distortion is higher because it includes unsuccessful samples, for which the crafting algorithm used the maximum distortion Υ, but was unable to induce a misclassification.

We also studied crafting of $9,000$ adversarial samples using the decreasing saliency map. We found that the success rate $\tau = 64.7\%$ was lower and the average distortion $\varepsilon = 3.62\%$ slightly lower. Again, decreasing pixel intensities is less successful at producing the desired adversarial behavior than increasing pixel intensities. Intuitively, this can be understood because removing pixels reduces the information entropy in an already

		All adversarial samples	Successful adversarial samples
Training	97.05%	4.45%	4.03%
Validation	97.19%	4.41%	4.01%
Test	97.05%	4.45%	4.03%

Figure 6.1. Results on larger sets of $10,000$ samples

sparse image, thus making it harder for DNNs to extract the information required to classify the sample. Greater absolute values of intensity variations are more confidently misclassified by the DNN.

6.2 Hardness and defense mechanisms

Looking at the previous experiment, about 2.9% of the $270,000$ adversarial samples were not successfully crafted. This suggests that some samples are harder to exploit than others. Furthermore, the distortion figures reported are averaged on all adversarial samples produced but not all samples require the same distortion to be misclassified. Thus, we now study the hardness of different samples in order to quantify these phenomena. Our aim is to identify which source-target class pairs are easiest to exploit, as well as similarities between distinct source-target class pairs. A class pair is a pair of a source class s and a target class t. This hardness metric allows us to lay ground for defense mechanisms.

6.2.1 Class pair study

From this experiment, we obtain a deeper understanding of the crafting success rate and average distortion for different source-target class pairs. We use the 90,000 adversarial samples crafted in the previous experiments from the 10,000 samples of the MNIST test set.

We break down the success rate τ reported in Figure 6.1 by source-target class pairs. This allows us to know, for a given source class, how many samples of that class were successfully misclassified in each of the target classes. In Figure 6.2, we draw the success rate matrix indicating which pairs are most successful. Darker shades correspond to higher success rates. Rows correspond to success rates per source class while columns

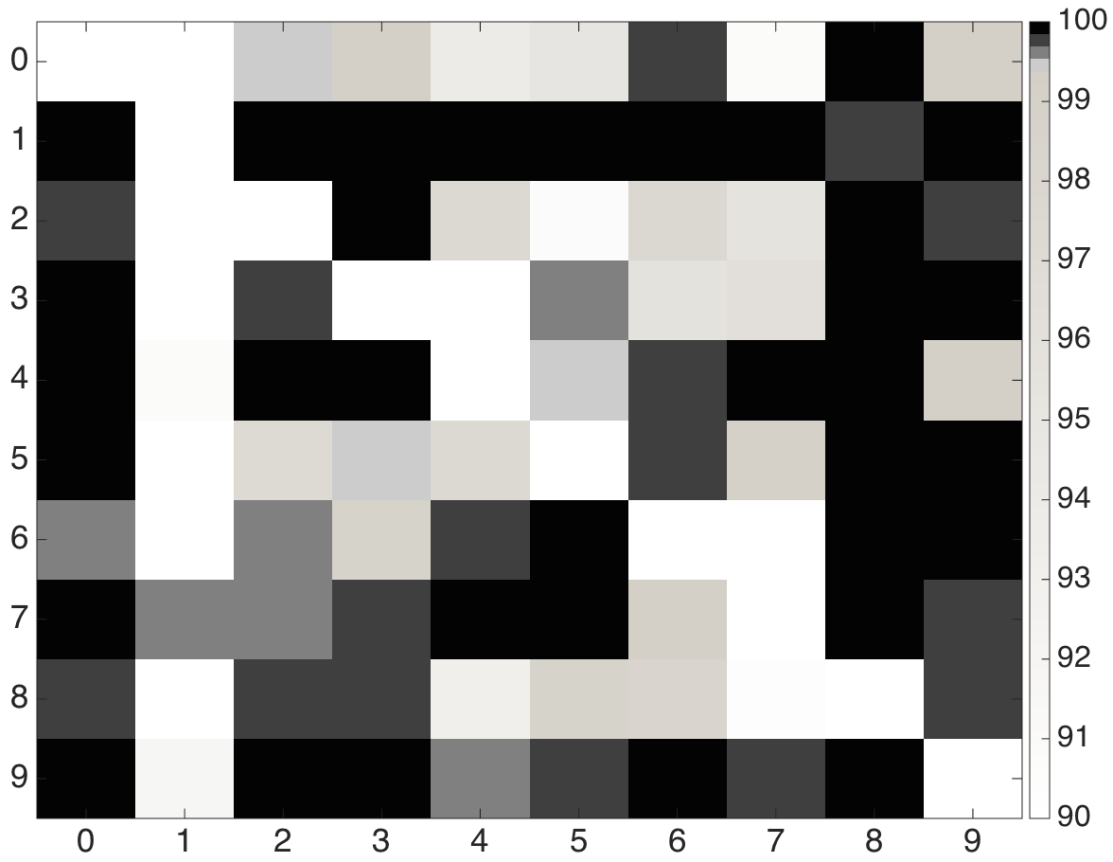


Figure 6.2. Success rate per source-target class pair.

correspond to success rates per target class. If one reads the matrix row-wise, it can be perceived that classes 0, 2, and 8 are hard to start with, while classes 1, 7, and 9 are easy to start with. Similarly, reading column-wise, one can observe that classes 1 and 7 are hard to make, while classes 0, 8, and 9 are easy to make.

In Figure 6.3, we report the average distortion ε of successful samples by source-target class pair, thus identifying class pairs requiring the most distortion to successfully craft adversarial samples. As expected, classes requiring lower distortions correspond to classes with higher success rates in Figure 6.2. For instance, the column corresponding to class 1 contains the highest distortions, and it was the column with the least success rates in Figure 6.2. Indeed, the higher the average distortion of a class pair is, the more likely samples in that class pair are to reach the maximum distortion, and thus produce unsuccessful adversarial samples.

To better understand why some class pairs were harder to exploit, we tracked the evolution of class probabilities during the crafting process. We observed that the distortion required to leave the source class was higher for class pairs with high distortions whereas the distortion required to reach the target class, once the source class had been left, remained similar. This correlates with the fact that some source classes are more confidently classified by the DNN then others.

6.2.2 Hardness measure

Results indicating that some source-target class pairs are not as easy as others lead us to question the existence of a measure quantifying the distance between two classes. This is relevant to a defender seeking to identify which classes of a DNN are most vulnerable to adversaries. We name this measure the *hardness* of a target class relatively to a given source class. It normalizes the average distortion of a class pair (s, t) relatively to its success rate:

$$H(s,t) = \int_{\tau} \varepsilon(s,t,\tau)\mathrm{d}\tau \tag{6.1}$$

where $\varepsilon(s,t,\tau)$ is the average distortion of a set of samples for the corresponding success rate τ. In practice, these two quantities are computed over a finite number of samples by fixing a set of K maximum distortion parameter values Υ_k in the crafting algorithm where $k \in 1..K$. The set of maximum distortions gives a series of pairs (ε_k, τ_k) for $k \in 1..K$. Thus, the practical formula used to compute the hardness of a source-destination class

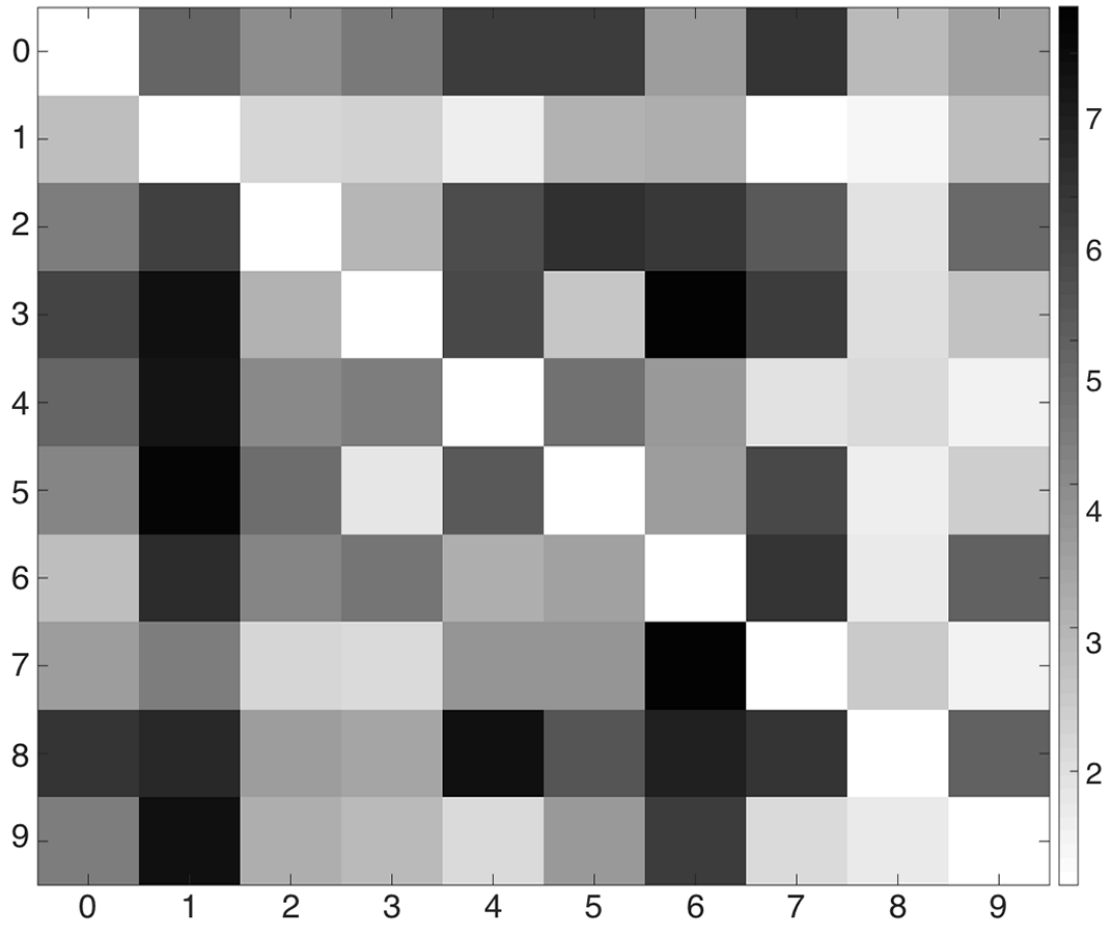


Figure 6.3. Average distortion ε of successful samples per source-target class pair. The scale is a percentage of pixels.

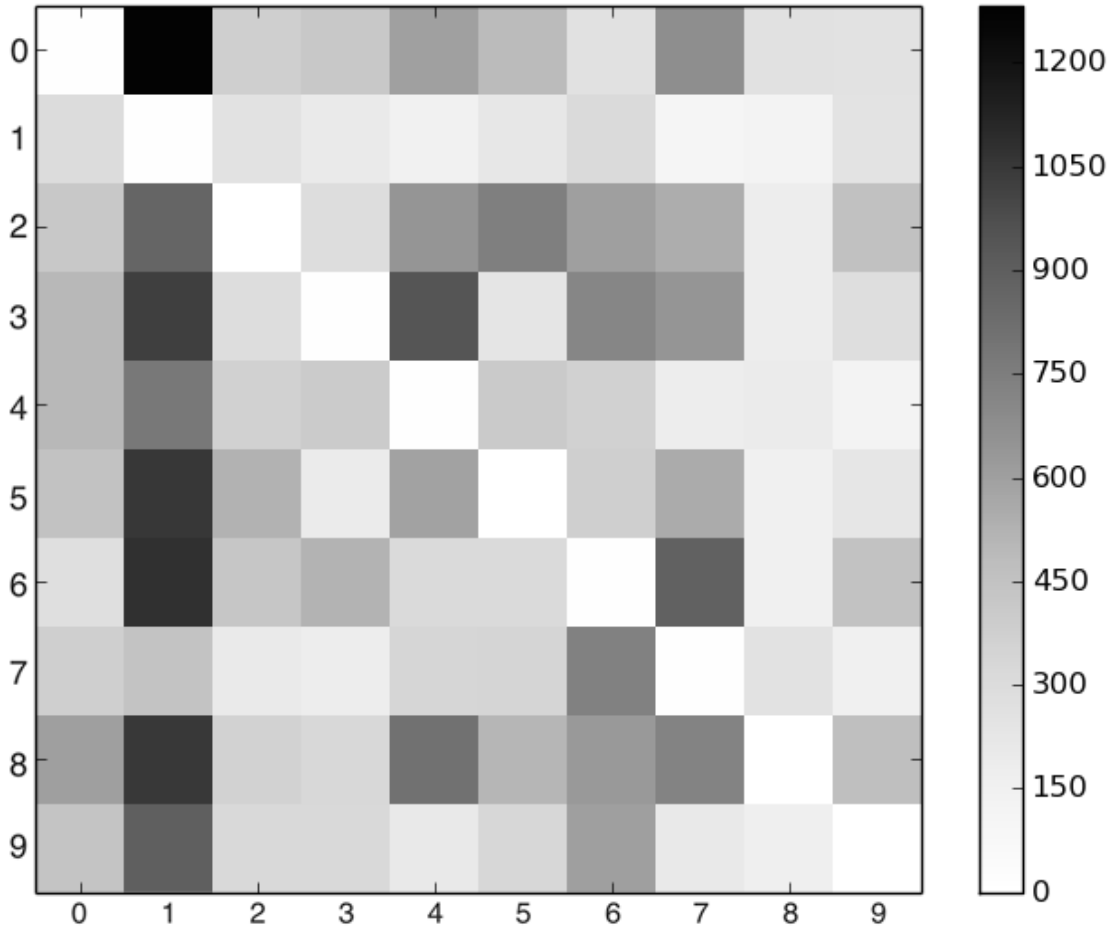


Figure 6.4. Hardness matrix of source-target class pairs. Darker shades correspond to harder to achieve pairs.

pair can be derived from the trapezoidal rule:

$$H(s,t) \approx \sum_{k=1}^{K-1} (\tau_{k+1} - \tau_k) \frac{\varepsilon(s,t,\tau_{k+1}) + \varepsilon(s,t,\tau_k)}{2} \tag{6.2}$$

We computed the hardness values for all classes using a set of $K = 9$ maximum distortion values $\Upsilon \in \{0.3, 1.3, 2.6, 5.1, 7.7, 10.2, 12.8, 25.5, 38.3\}\%$ in the algorithm. Average distortions ε and success rates τ are averaged over 9,000 adversarial samples for each maximum distortion value Υ. Figure 6.4 shows the hardness values $H(s,t)$ for all pairs $(s,t) \in \{0..9\}^2$. Note that the matrix has a shape similar to the average distortion matrix plotted on Figure 6.3. However, the hardness measure is more accurate because it is plotted using a series of maximum distortions.

6.2.3 Adversarial distance

The measure introduced lays ground towards finding defenses against adversarial samples. Indeed, if the hardness measure were to be predictive instead of being computed after adversarial crafting, the defender could identify vulnerable inputs. Furthermore, a predictive measure applicable to a single sample would allow a defender to evaluate the vulnerability of specific samples as well as class pairs. We investigated several complex estimators including convolutional transformations of the forward derivative or Hessian matrices. However, we found that simply using a formula derived from the intuition behind adversarial saliency maps gave good accuracy for predicting the hardness of samples in our experimental setup.

We name this predictive measure the *adversarial distance* of sample $\mathbf{X}$ to class t and write it $A(\mathbf{X}, t)$. Simply put, it estimates the distance between a sample $\mathbf{X}$ and a target class t. We define the distance as:

$$A(\mathbf{X}, t) = 1 - \frac{1}{M} \sum_{i \in 1..M} 1_{S(\mathbf{X},t)[i]>0} \tag{6.3}$$

where 1_E is the indicator function for event E (i.e., is 1 if E is true). In a nutshell, $A(\mathbf{X}, t)$ is the normalized number of non-zero elements in the adversarial saliency map of $\mathbf{X}$ computed during the first crafting iteration in Algorithm 2. The closer the adversarial distance is to 1, the more likely sample $\mathbf{X}$ is going to be harder to misclassify in target class t. Figure 6.5 confirms that this formula is empirically well-founded. It illustrates the value of the adversarial distance averaged over source-destination class pairs, making it easy to compare the average value with the hardness matrix computed previously after

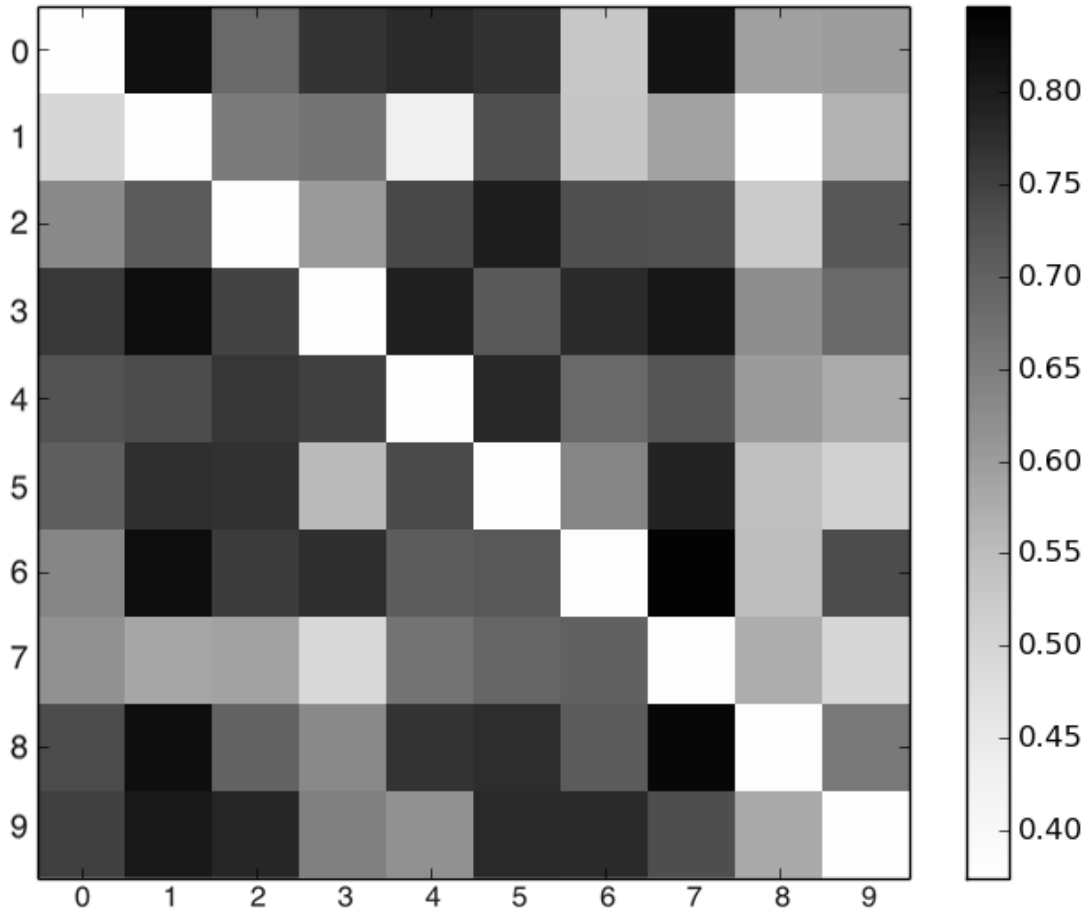


Figure 6.5. Adversarial distance averaged per source-destination class pairs computed with 1000 samples.

crafting samples. To compute it, we altered Equation 6.3 to sum over pairs of features, reflecting the observations made during our validation process.

This notion of distance between classes intuitively defines a metric for the *robustness* of a DNN $\mathbf{F}$ against adversarial perturbations. We suggest the following definition:

$$R(\mathbf{F}) = \min_{(\mathbf{X},t)} A(\mathbf{X}, t) \tag{6.4}$$

where the set of samples $\mathbf{X}$ considered is sufficiently large to represent the input domain of the network. A good approximation of robustness can be computed with the training dataset. Note that the min operator used here can be replaced by other relevant operators, like the statistical expectation. The study of various operators is left as future work.

6.3 Human perception of adversarial samples

Recall that adversarial samples must not only be misclassified as the target class by DNNs, but also visually appear (be classified) as the source class by humans. To evaluate this property, we ran an experiment using 349 human participants on the Mechanical Turk online service. We presented 3 original or adversarially altered samples from the MNIST dataset to human participants. To paraphrase, participants were asked for each sample: (a) 'is this sample a numeric digit?', and (b) 'if yes to (a) what digit is it?'. These two questions were designed to determine how distortion and intensity rates effected human perception of the samples.

The first experiment was designed to identify a baseline perception rate for the input data. The 74 participants were presented 3 of 222 unaltered samples randomly picked from the original MNIST data set. Respondents identified 97.4% as digits and classified correctly 95.3% of the samples.

Shown in Figure 6.6, a second experiment attempted to evaluate how distortion (ε) impacts human perception. Here, 184 participants were presented with a total of 1707 samples with varying levels of distortion (and features altered with an intensity increase $\theta = +1$). The experiments showed that below a threshold distortion ($\varepsilon = 14.29\%$), participants were able to identify samples as digits (95%) and correctly classify them (90%) only slightly less accurately than the unaltered samples. The classification rate dropped dramatically (71%) at distortion rates above the threshold.

A final set of experiments evaluate the impact of intensity variations (θ) on perception, as shown Figure 6.7. The 203 participants were accurate at identifying $5,355$

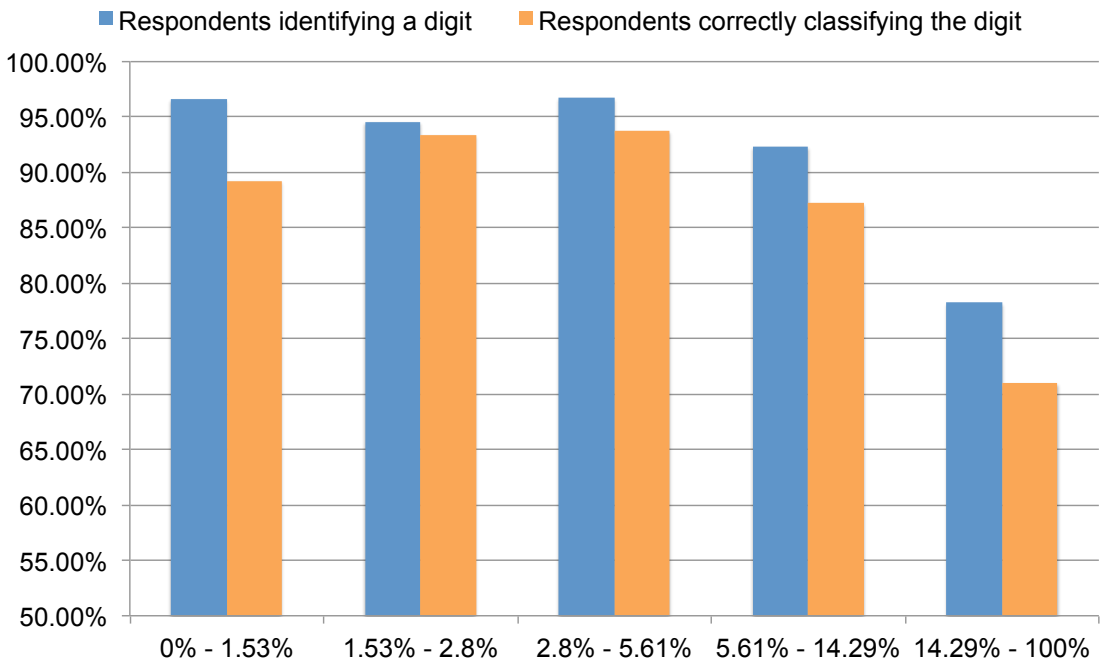


Figure 6.6. Human perception of different distortions ε.

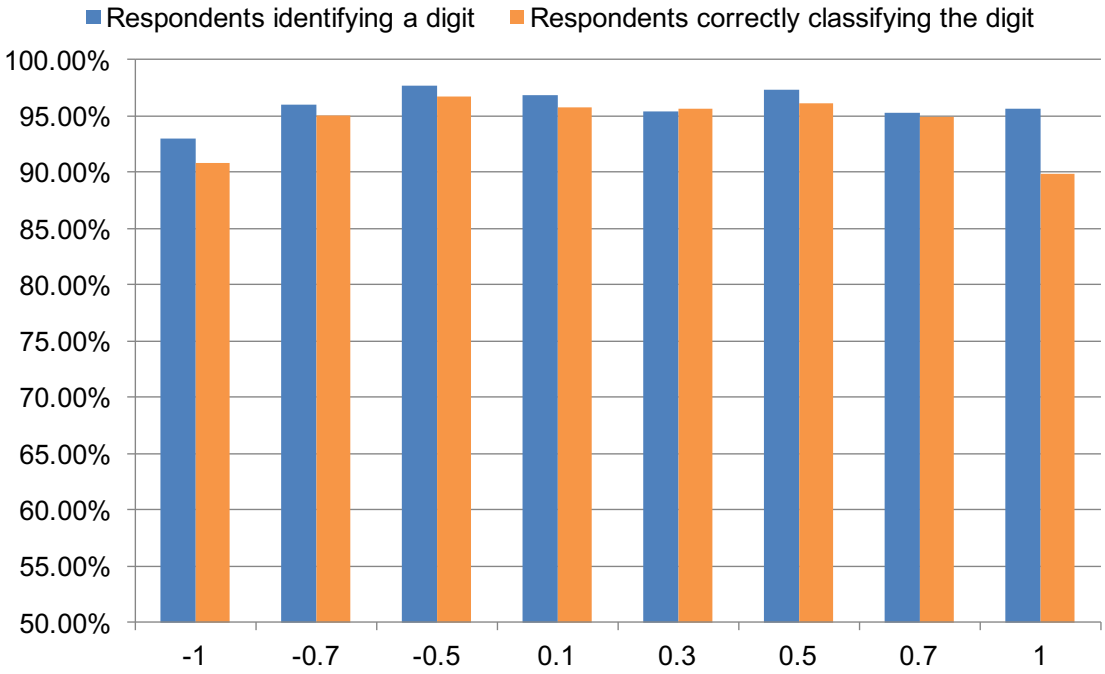


Figure 6.7. Human perception of intensity variations θ.

samples as digits (96%) and classifying them correctly (95%). At higher absolute intensities ($\theta = -1$ and $\theta = +1$), specific digit classification decreased slightly (90.5% and 90%), but identification as digits was largely unchanged.

While preliminary, these experiments confirm that the overwhelming number of generated samples retain human recognizability. Note that because we can generate samples with less than the distortion threshold for almost all of the input data, ($\varepsilon \leq 14.29\%$ for roughly 97% in the MNIST data), we can produce adversarial samples that humans will not detect—thus meeting our adversarial goal. Furthermore, limiting intensity variations provides even better results: at $-0.7 \leq \theta \leq +0.7$, humans classified the sample data at essentially the same rates as the original sample data.

Chapter 7

Related Work

The security community has actively researched the implications of deploying machine learning in adversarial settings. Huang et al. [35] provide a description of the field of *adversarial machine learning* building on a first formalization by Barenno et al. [2]. Specifically, their work first presents the taxonomy introduced by Barreno et al. in [2] that discriminates attacks against machine learning using three properties: *influence*, *security violations*, and *specificity*. The *influence* of an attack is either *causative* if its purpose it to alter the model learned by the machine learning technique during training or *exploratory* if the purpose is to gather evidence on the algorithms and training data used or to force a misbehavior of the technique without an impact on the model. Finally, the taxonomy characterizes attacks by their influence spectrum on the dataset, i.e. *specificity*. This taxonomy was later extended quantitatively in [44]. Huang et al. then discuss causative, exploratory, and privacy attacks in depth in the remainder of their paper [35]. Generally speaking, attacks against machine learning can be separated into two categories, depending on whether they are executed at training or test time.

7.1 Training Time Attacks

Attacks at training time—*poisoning attacks* [2]—alter the training set by adding or removing points with the intent of modifying the decision boundaries of the targeted model [38, 55, 54]. Poisoning attacks are particularly relevant in settings where models use online training to improve the accuracy of their predictions. Biggio et al. introduced a class of attacks against two-class Support Vector Machines at training time [8, 9]. In [8], they considered an adversary capable of manipulating training set labels. The adversary

follows one of the following two strategies: (1) flipping a random subset of training set labels, or (2) flipping a subset of training set labels selected using an heuristic that, simply put, identifies samples classified with high confidence.

7.2 Test Time Attacks

Attacks at test time—*exploratory attacks* [2]—do not tamper with the targeted model but instead either cause it to directly misbehave or simply use the attack to collect evidence about the model characteristics. This is the type of attack we described in the previous chapters of this manuscript. Biggio et al. [6] introduced an attack against binary classifiers (e.g., used for detection) at test time and consider adversaries with the goal of having malicious samples classified as benign. The adversary is assumed to have (at least partial) knowledge of the target classifier. They formulate the adversary's attack strategy as a minimization problem to find the input classified as benign which is closest to the adversarial target (classified as malicious). To help with the non-linearity and potential non-convexity, they use a heuristic that favors attack points in denser regions of the set of legitimate points (referred to as *mimicry*). They perform attacks on linear classifiers, SVMs, and low dimensional neural networks with a single hidden layer. Also assuming knowledge of the target classifier, Fogla et al. detail a polymorphic blending attack to evade network anomaly detection systems [24, 25]. Their attack is based on modeling the intrusion detection system as a regular grammar and finding mutated attack instances accepted by this grammar. Generating such instances is a NP-complete problem but the authors show that one can find near optimal solution using a reduction to the SAT satisfiability problem and its associated solvers.

7.2.1 Deep Learning and Adversarial Samples

Prior work on adversarial sample crafting against DNNs developed a simple technique corresponding to the *Architecture and Training Tools* threat model, based on gradients used for DNN training [29, 57, 72]. This approach creates adversarial samples by defining an optimization problem based on the DNN's cost function. In other words, instead of computing gradients to update DNN weights, one computes gradients to update the input, which is then misclassified as the target class by a DNN. The alternative approach proposed in this paper is to identify input regions that are most relevant to its classification by a DNN. This is accomplished by computing the saliency map of a given input, as described by Simonyan et al. in the case of DNNs handling images [67]. We extended

this concept to create adversarial saliency maps highlighting input regions that need to be perturbed to accomplish the adversarial goal.

Previous work by Yosinki et al. investigated how features are transferable between DNNs [77], while Szegedy et al. showed that adversarial samples can indeed be misclassified across models [72]. They report that once an adversarial sample is generated for a given neural network architecture, it is also likely to be misclassified in neural networks designed differently, which explains why the attack is successful. However, the effectiveness of this kind of attack depends on (1) the quality and size of the surrogate dataset collected by the adversary, and (2) the adequateness of the adversarial network used to craft adversarial samples.

Chapter 8

Conclusions

Broadly speaking, this manuscript has explored adversarial behavior in deep learning systems. In addition to exploring the goals and capabilities of DNN adversaries, we introduced a new class of algorithms to craft adversarial samples based on computing *forward derivatives*. This technique allows an adversary with knowledge of the DNN architecture to construct *adversarial saliency maps* identifying features of the input that most significantly impact DNN outputs. These algorithms can reliably produce samples correctly classified by human subjects but misclassified in specific targets by a DNN with a 97% adversarial success rate while only modifying on average 4.02% of the input features per sample.

8.1 Discussion

We introduced a new class of algorithms that systematically craft adversarial samples so as to cause a DNN to misclassify the sample, assuming that the adversary possesses knowledge of the DNN architecture. One of our key results is reducing the distortion—the number of features altered—to craft adversarial samples, compared to previous work. We believe this makes adversarial crafting much easier for input domains like malware executables, which are not as easy to perturb as images [10, 24]. This distortion reduction comes with a performance cost. Indeed, more elaborate but accurate saliency map formulae are more expensive to compute for the attacker. We would like to emphasize that our method's high success rate can be further improved by adversaries only interested in crafting a limited number of samples. Indeed, to lower the distortion of one particular sample, an adversary can use adversarial saliency maps to fine-tune the per-

turbation introduced. On the other hand, if an adversary wants to craft large amounts of adversarial samples, performance is important. In our evaluation, we balanced these factors to craft adversarial samples against the DNN in less than a second. As far as our algorithm implementation was concerned, the most computationally expensive steps were the matrix manipulations required to construct adversarial saliency maps from the forward derivative matrix. The complexity is dependent on the number of input features. These matrix operations can be made more efficient, notably by making better use of GPU-accelerated computations.

8.2 Future Work: Defenses against Adversarial Samples

Our efforts so far represent a first but meaningful step towards mitigating adversarial samples: the hardness and adversarial distance metrics lay out bases for defense mechanisms. Designing such defenses is outside the scope of this manuscript but we outline two approaches: (1) adversarial sample detection and (2) DNN robustness improvements.

Developing techniques for adversarial sample detection is a reactive solution. In our experimental setup, we noticed that adversarial samples can for instance be detected by evaluating the regularity of samples. More specifically, in our application example, the sum of the squared difference between each pair of neighboring pixels is always higher for adversarial samples than for benign samples. However, there is no a priori reason to assume that this technique will reliably detect adversarial samples in different settings, so extending this approach is one avenue for future work. Another approach was proposed in [31], but it is unsuccessful as by stacking the denoising auto-encoder used for detection with the original DNN, adversarial samples can again be crafted.

The second class of solutions seeks to improve training to increase the robustness of DNNs. Interestingly, the problem of adversarial samples is closely linked to training. Notably, the universal approximation theorem formulated by Hornik et al. states one hidden layer is sufficient to represent arbitrarily accurately a function [34]. Thus, one can conceive that DNNs have the capacity to model a classification function resistant to adversarial samples. Work on generative adversarial networks showed that a two player game between two DNNs can lead to the generation of new samples from a training set [28]. Furthermore, adding adversarial samples to the training set can act like a regularizer [29]. We also observed in our experiments that training with adversarial samples makes crafting additional adversarial samples harder. Indeed, by adding 18,000 adversarial samples to the original MNIST training dataset, we trained a new instance

of our DNN. We then crafted a set of 9,000 adversarial samples with this newly trained network. Preliminary analysis of these samples crafted showed that the success rate was reduced by 7.2% while the average distortion increased by 37.5%, suggesting that training with adversarial samples makes DNNs more robust. A different avenue to address the lack of robustness of deep neural networks to adversarial samples is to use knowledge transfer to smooth the models learned by DNNs and thus reduce their sensitivity to small perturbations of their inputs [59].

8.3 Future Work: Extending the Attack

Future work also includes extending the technique described in this manuscript to adapt it to DNNs trained in an unsupervised manner as well as recurrent neural networks. Although we focused our work on DL techniques used in the context of classification and trained with supervised methods, our approach is also applicable to unsupervised architectures. Instead of achieving a given target class, the adversary achieves a target output $\mathbf{Y}^*$. Because the output space is more complex, it might be harder or impossible to match $\mathbf{Y}^*$. In that case, Equation 1.1 would need to be relaxed with an acceptable distance between the network output $\mathbf{F}(\mathbf{X}^*)$ and the adversarial target $\mathbf{Y}^*$. Thus, the only remaining assumption made in this manuscript is that DNNs are feedforward. In other words, we did not consider recurrent neural networks, as the forward derivative must be adapted to accommodate such cyclical DNNs. Also, as most models of our taxonomy have yet to be researched, this leaves room for further investigation of deep learning in various adversarial settings.

Bibliography

[1] Alipanahi, B., Delong, A., Weirauch, M. T., and Frey, B. J. (2015). Predicting the sequence specificities of dna-and rna-binding proteins by deep learning. *Nature biotechnology*.

[2] Barreno, M., Nelson, B., Sears, R., Joseph, A. D., and Tygar, J. D. (2006). Can machine learning be secure? In *Proceedings of the 2006 ACM Symposium on Information, computer and communications security*, pages 16–25. ACM.

[3] Bengio, Y. (2009). Learning deep architectures for AI. *Foundations and trends in Machine Learning*, 2(1):1–127.

[4] Bengio, Y., Courville, A., and Vincent, P. (2013). Representation learning: A review and new perspectives. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(8):1798–1828.

[5] Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., Warde-Farley, D., and Bengio, Y. (2010). Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for scientific computing conference (SciPy)*, volume 4, page 3. Austin, TX.

[6] Biggio, B., Corona, I., Maiorca, D., Nelson, B., Šrndić, N., Laskov, P., Giacinto, G., and Roli, F. (2013). Evasion attacks against machine learning at test time. In *Machine Learning and Knowledge Discovery in Databases*, pages 387–402. Springer.

[7] Biggio, B., Fumera, G., and Roli, F. (2014a). Pattern recognition systems under attack: Design issues and research challenges. *International Journal of Pattern Recognition and Artificial Intelligence*, 28(07):1460002.

[8] Biggio, B., Nelson, B., and Laskov, P. (2011). Support vector machines under adversarial label noise. In *ACML*, pages 97–112.

[9] Biggio, B., Nelson, B., and Pavel, L. (2012). Poisoning attacks against support vector machines. In *Proceedings of the 29th International Conference on Machine Learning*.

[10] Biggio, B., Rieck, K., Ariu, D., Wressnegger, C., Corona, I., Giacinto, G., and Roli, F. (2014b). Poisoning behavioral malware clustering. In *Proceedings of the 2014 Workshop on Artificial Intelligent and Security Workshop*, pages 27–36. ACM.

[11] Bishop, C. M. (2006). Pattern recognition. *Machine Learning*.

[12] Cao, W., Hu, L., and Cao, L. (2015). Deep modeling complex couplings within financial markets. In *AAAI*, pages 2518–2524.

[13] Cireşan, D., Meier, U., Masci, J., et al. (2012). Multi-column deep neural network for traffic sign classification. *Neural Networks*, 32:333–338.

[14] Cireşan, D., Meier, U., Masci, J., and Schmidhuber, J. (2011). A committee of neural networks for traffic sign classification. In *Neural Networks (IJCNN), The 2011 International Joint Conference on*, pages 1918–1921. IEEE.

[15] Ciresan, D., Meier, U., and Schmidhuber, J. (2012). Multi-column deep neural networks for image classification. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3642–3649. IEEE.

[16] Collobert, R. and Weston, J. (2008). A unified architecture for natural language processing: Deep neural networks with task learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM.

[17] Dahl, G. E., Stokes, J. W., Deng, L., and Yu, D. (2013). Large-scale malware classification using random projections and neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3422–3426. IEEE.

[18] Dahl, G. E., Yu, D., et al. (2012). Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *IEEE Transactions on Audio, Speech, and Language Processing*, 20(1):30–42.

[19] Dosovitskiy, A., Springenberg, J. T., Riedmiller, M., and Brox, T. (2014). Discriminative unsupervised feature learning with convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 766–774.

[20] Duda, R. O., Hart, P. E., and Stork, D. G. (2012). *Pattern classification*. John Wiley & Sons.

[21] Erhan, D., Bengio, Y., Courville, A., Manzagol, P.-A., Vincent, P., and Bengio, S. (2010). Why does unsupervised pre-training help deep learning? *The Journal of Machine Learning Research*, 11:625–660.

[22] Farabet, C., Couprie, C., Najman, L., and LeCun, Y. (2013). Learning hierarchical features for scene labeling. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(8):1915–1929.

[23] Fehrer, R. and Feuerriegel, S. (2015). Improving decision analytics with deep learning: The case of financial disclosures. *arXiv preprint arXiv:1508.01993*.

[24] Fogla, P. and Lee, W. (2006). Evading network anomaly detection systems: formal reasoning and practical techniques. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 59–68. ACM.

[25] Fogla, P., Sharif, M. I., Perdisci, R., Kolesnikov, O. M., and Lee, W. (2006). Polymorphic blending attacks. In *USENIX Security*.

[26] Freedman, D. A. (2009). *Statistical models: theory and practice*. cambridge university press.

[27] Glorot, X., Bordes, A., and Bengio, Y. (2011). Domain adaptation for large-scale sentiment classification: A deep learning approach. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 513–520.

[28] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., et al. (2014). Generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2672–2680.

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

[30] Goodfellow, I. J., Warde-Farley, D., Mirza, M., Courville, A., and Bengio, Y. (2013). Maxout networks. *arXiv preprint arXiv:1302.4389*.

[31] Gu, S. and Rigazio, L. (2015). Towards deep neural network architectures robust to adversarial examples. In *Proceedings of the 2015 International Conference on Learning Representations*. Computational and Biological Learning Society.

[32] Hinton, G., Osindero, S., and Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554.

[33] Hinton, G. E. and Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507.

[34] Hornik, K., Stinchcombe, M., et al. (1989). Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366.

[35] Huang, L., Joseph, A. D., Nelson, B., Rubinstein, B. I., and Tygar, J. (2011). Adversarial machine learning. In *Proceedings of the 4th ACM workshop on security and artificial intelligence*, pages 43–58. ACM.

[36] Ian Goodfellow, Y. B. and Courville, A. (2016). Deep learning. Book in preparation for MIT Press.

[37] Kaufman, C., Perlman, R., and Speciner, M. (2002). *Network security: private communication in a public world*. Prentice Hall Press.

[38] Kloft, M. and Laskov, P. (2007). A poisoning attack against online anomaly detection. In *NIPS Workshop on Machine Learning in Adversarial Environments for Computer Security*.

[39] Knorr, E. (2015). How paypal beats the bad guys with machine learning. http://www.infoworld.com/article/2907877/machine-learning/how-paypal-reduces-fraud-with-machine-learning.html.

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

[41] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

[42] Larochelle, H. and Bengio, Y. (2008). Classification using discriminative restricted boltzmann machines. In *Proceedings of the 25th international conference on Machine learning*, pages 536–543. ACM.

[43] Larochelle, H., Bengio, Y., Louradour, J., and Lamblin, P. (2009). Exploring strategies for training deep neural networks. *The Journal of Machine Learning Research*, 10:1–40.

[44] Laskov, P. and Kloft, M. (2009). A framework for quantitative security analysis of machine learning. In *Proceedings of the 2nd ACM workshop on Security and artificial intelligence*, pages 1–4. ACM.

[45] LeCun, Y., Bottou, L., et al. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

[46] LeCun, Y. and Cortes, C. (1998). Mnist handwritten digit database. *AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*.

[47] Leung, M. K., Xiong, H. Y., Lee, L. J., and Frey, B. J. (2014). Deep learning of the tissue-regulated splicing code. *Bioinformatics*, 30(12):i121–i129.

[48] LISA lab (2010). http://deeplearning.net/tutorial/lenet.html.

[49] Masci, J., Meier, U., Cireşan, D., and Schmidhuber, J. (2011). Stacked convolutional auto-encoders for hierarchical feature extraction. In *Artificial Neural Networks and Machine Learning–ICANN 2011*, pages 52–59. Springer.

[50] Mnih, V., Heess, N., Graves, A., et al. (2014). Recurrent models of visual attention. In *Advances in Neural Information Processing Systems*, pages 2204–2212.

[51] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

[52] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

[53] Murphy, K. P. (MIT 2012). *Machine learning: a probabilistic perspective*.

[54] Nelson, B., Barreno, M., Chi, F. J., Joseph, A. D., Rubinstein, B. I., Saini, U., Sutton, C. A., Tygar, J. D., and Xia, K. (2008). Exploiting machine learning to subvert your spam filter. *LEET*, 8:1–9.

[55] Nelson, B. and Joseph, A. D. (2006). Bounding an attack?s complexity for a simple learning model. In *Proc. of the First Workshop on Tackling Computer Systems Problems with Machine Learning Techniques (SysML), Saint-Malo, France*. Citeseer.

[56] Neter, J., Kutner, M. H., Nachtsheim, C. J., and Wasserman, W. (1996). *Applied linear statistical models*, volume 4. Irwin Chicago.

[57] Nguyen, A., Yosinski, J., and Clune, J. (2015). Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *IEEE Computer Vision and Pattern Recognition*.

[58] Papernot, N., McDaniel, P., Jha, S., Fredrikson, M., Celik, Z. B., and Swami, A. (2016a). The limitations of deep learning in adversarial settings. In *Proceedings of the 1st IEEE European Symposium on Security and Privacy*. IEEE.

[59] Papernot, N., McDaniel, P., Wu, X., Jha, S., and Swami, A. (2016b). Distillation as a defense to adversarial perturbations against deep neural networks. In *Proceedings of the 37th IEEE Symposium on Security and Privacy*. IEEE.

[60] Pascanu, R., Stokes, J. W., Sanossian, H., Marinescu, M., and Thomas, A. (2015). Malware classification with recurrent networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*, pages 1916–1920. IEEE.

[61] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1988). Learning representations by back-propagating errors. *Cognitive modeling*, 5.

[62] Sak, H., Senior, A., and Beaufays, F. (2014). Long short-term memory recurrent neural network architectures for large scale acoustic modeling. In *Proceedings of the Annual Conference of International Speech Communication Association (INTERSPEECH)*.

[63] Salakhutdinov, R. and Hinton, G. (2012). An efficient learning procedure for deep boltzmann machines. *Neural computation*, 24(8):1967–2006.

[64] Salakhutdinov, R. and Hinton, G. E. (2009). Deep boltzmann machines. In *International Conference on Artificial Intelligence and Statistics*, pages 448–455.

[65] Schürmann, J. (1996). *Pattern classification: a unified view of statistical and neural approaches*. Wiley Online Library.

[66] Shin, E. C. R., Song, D., and Moazzezi, R. (2015). Recognizing functions in binaries with neural networks. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 611–626.

[67] Simonyan, K., Vedaldi, A., and Zisserman, A. (2013). Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*.

[68] Snoek, J., Rippel, O., Swersky, K., Kiros, R., Satish, N., Sundaram, N., Patwary, M., Ali, M., Adams, R. P., et al. (2015). Scalable bayesian optimization using deep neural networks. *arXiv preprint arXiv:1502.05700*.

[69] Springenberg, J. T., Dosovitskiy, A., Brox, T., and Riedmiller, M. (2014). Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*.

[70] Srivastava, N., Salakhutdinov, R. R., and Hinton, G. E. (2013). Modeling documents with deep boltzmann machines. *arXiv preprint arXiv:1309.6865*.

[71] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2014a). Going deeper with convolutions. *arXiv preprint arXiv:1409.4842*.

[72] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. (2014b). Intriguing properties of neural networks. In *Proceedings of the 2014 International Conference on Learning Representations*. Computational and Biological Learning Society.

[73] Taigman, Y., Yang, M., et al. (2014). Deepface: Closing the gap to human-level performance in face verification. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1701–1708.

[74] Vincent, P., Larochelle, H., Bengio, Y., and Manzagol, P.-A. (2008). Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM.

[75] Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., and Manzagol, P.-A. (2010). Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *The Journal of Machine Learning Research*, 11:3371–3408.

[76] Webb, A. R. (2003). *Statistical pattern recognition*. John Wiley & Sons.

[77] Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. (2014). How transferable are features in deep neural networks? In *Advances in Neural Information Processing Systems*, pages 3320–3328.