A decade of adversarial examples: a survey on the nature and understanding of neural network non-robustness

A.V. Trusov[1,2,3], E.E. Limonova[1,2], V.V. Arlazarov[1,2]
[1]Federal Research Center "Computer Science and Control" of the Russian Academy of Sciences, 119333, Russia, Moscow, Vavilova 44, kor.2;
[2]Smart Engines Service LLC, 117312, Russia, Moscow, pr. 60-letiya Oktyabrya 9,
[3]Moscow Institute of Physics and Technology, 141701, Russia, Dolgoprudny, Institutskiy per. 9

Abstract

Adversarial examples, in the context of computer vision, are inputs deliberately crafted to deceive or mislead artificial neural networks. These examples exploit vulnerabilities in neural networks, resulting in minimal alterations to the original input that are imperceptible by humans but can significantly impact the network's output. In this paper, we present a thorough survey of research on adversarial examples, with a primary focus on their impact on neural network classifiers. We closely examine the theoretical capabilities and limitations of artificial neural networks. After that, we explore the discovery and evolution of adversarial examples, starting from basic gradient-based techniques and progressing toward the recent trend of employing generative neural networks for this purpose. We discuss the limited effectiveness of existing countermeasures against adversarial examples. Furthermore, we emphasize that the adversarial examples originate the misalignment between human and neural network decision-making processes. That can be attributed to the current methodology for training neural networks. We also argue that the commonly used term "attack on neural networks" is misleading when discussing adversarial deep learning. Through this paper, our objective is to provide a comprehensive overview of adversarial examples and inspire further researchers to develop more robust neural networks. Such networks will align better with human decision-making processes and enhance the security and reliability of computer vision systems in practical applications.

Keywords: adversarial examples, adversarial deep learning, neural networks, neural network security.

Citation: Trusov AV, Limonova EE, Arlazarov VV. A decade of adversarial examples: a survey on the nature and understanding of neural network non-robustness. Computer Optics 2025; 49(2): 222-252. DOI: 10.18287/2412-6179-CO-1494.

Introduction

In recent years deep neural networks (DNNs) have become an essential tool for many computer vision (CV) tasks, such as image recognition [1], matching [2], semantic segmentation [3], object detection [4], object localization [5], and many others. They are widely used in complex software applications, including but not limited to person re-identification [6], document recognition [7, 8], and autonomous driving systems [9]. Such systems often process private information and significantly affect people, so their security is essential.

Unfortunately, DNNs are not ideal tools. They have limitations that can be exploited to influence their behavior. For example, adding a small specifically generated perturbation to an image can lead to misclassification of this image, even if this perturbation is invisible to the human eye [10]. Over the last decade, the methods for generating such perturbations have evolved greatly, and many applications have been suggested. Deliberately prepared and printed stickers in real-world scenes can fool neural network classifiers [11]. Special make-up [12], make-up-like image transformations [13], different types of masks [14], and even special infrared lighting devices [15] have been proposed to evade facial recognition systems. Clothing with computer-generated prints [16] and patches [17] can also be used to evade surveillance systems. Even shadows of specific shapes cast on physical objects may lead to its missrecognition [18]. Figure 1 presents some of the above examples.

Moreover, information about the architecture and parameters of DNNs has significant commercial value. This information can be reverse-engineered and stolen if an adversary has physical access to the hardware on which a neural network is deployed [19, 20, 21]. Furthermore, if a malicious party has access to the training data (even to a small part of it), they can poison it by inserting specific examples [22]. Training on poisoned data is dangerous because the resulting DNN may have backdoors [23] or may not work correctly for certain inputs [24]. Finally, data privacy is also a substantial risk, as the training data can be successfully retrieved from a DNN during its inference [25] or distributed training [26].

All the threats to DNNs are usually referred to as *adversarial deep learning* or *adversarial attacks* on DNNs. The exploitation of neural network vulnerabilities has become a growing concern in recent years, with an increasing number of approaches and methods developed

for this purpose. In a comprehensive survey of adversarial attacks in computer vision published in 2018, 12 different attacks were identified [27]. However, a more recent survey from 2021 has already identified 33 distinct attacks [28], while a 2022 paper has organized over 50 attacks according to their taxonomy [29]. This taxonomy is complex and rapidly evolving, as new methods for attacking deep neural networks are constantly being proposed. In response, a variety of defenses have been developed, with the 2021 study citing over 60 defenses [28], which can be roughly divided into two types: heuristic and provable (certified) [30].

Fig. 1. Images that "fool" a neural network. Left image: digital perturbations, that make DNN recognize all the images as an "ostrich" from study [10]. Middle image: "invisibility" sweatshirt from study [16], which makes a person in front undetectable by DNN (in contrast to all the other people). Right image: shadow of a specific shape makes DNN misinterpret the speed limit as 35 instead of 25 as described in study [18]

Heuristic defenses are various techniques aimed at preventing certain types of adversarial attacks (e.g. by including adversarial samples into dataset [31], or adding random noise before each layer of a network [32]). Unfortunately, any heuristic defense that performs well against specific attacks can be easily broken by others [33].

On the other hand, provable defenses guarantee that if an input permutation is small enough, the result will not change dramatically. Still, such defenses have several major disadvantages: decreasing DNN quality [34], being computationally challenging, and only guaranteeing robustness to small perturbations under predefined metrics, while attacks use many different metrics [30]. All in all, the race between adversarial attacks and defenses is far from being over.

Apart from CV tasks adversarial deep learning may be used to temper with speech recognition [35], text analysis [36], reinforcement learning [37], instruction detection in Internet of Things (IoT) systems [38] and any other area where DNNs are applied. However, each area has its own specifics, and in this work, we will mainly consider image classification.

In this article, we try to take a broader view of the problem of adversarial attacks. We argue that the term "attack" is misleading. When one considers the vulnerability of neural networks to various kinds of *adversarial perturbations* (inputs generated to attack DNNs), he is generally facing the problem of a neural network being undertrained or incorrectly trained to solve a particular task. The fact that human perception is not affected by adversarial perturbations proves that the corresponding recognition task can be solved correctly. Thus, there exists a certain function that maps the input to the correct output, and this function is not affected by adversarial perturbations. Neural networks, as universal approximators, can theoretically approximate this function to any desired degree of accuracy [39]. However, as we can see from the DNNs used in practice, they do not. The main problem is that the correct function is not known. The network learns it from the training data, according to some algorithm. And that is why there is a possibility of adversarial deep learning.

There are many other recent surveys dedicated to adversarial examples [40, 28, 29, 41, 42]. Their authors try to present the entire variety of techniques for generating such examples, provide new taxonomy, or explore all the possible applications. In this work, we are not repeating them. Instead, we focus on a deeper understanding of adversarial examples. We mainly consider the image classification task and investigate it in detail.

Starting from the definition of an artificial neural network and determining its theoretical strength and drawbacks (Section 1) we move to the discovery of adversarial perturbation (Section 2). We provide a detailed description of the most remarkable methods for generating such perturbations and present their visual comparison (Section 2.1); demonstrate that perturbations can be generated even without full access to a neural network or its input (i.e. in black-box settings) and that they transfer between different networks (Section 2.2); and an overview of the most recent trends in applying specifically trained neural networks to generate almost imperceptible adversarial examples (Section 2.3). Then, we review the existing countermeasures namely heuristic (Section 3.1) and certified (Section 3.2) defenses against adversarial examples, and confirm that they are insufficient to fully prevent neural networks from being "fooled". Finally, in Section 4 we reflect on the previous

sections and attempt to take a broader look at the phenomenon of adversarial examples. We show how it originates from the misalignment of inductive bias of neural networks and human perception, argue why calling adversarial examples "attacks" is misleading, and determine future directions of research toward training robust and secure neural networks.

1. What a neural network is?

To better comprehend the mechanism of adversarial attacks on DNNs and defenses against them, we first need to understand what a neural network is and how it operates. That is why, in this section, we will review the main theoretical concepts related to DNNs. We will begin with a brief overview of neural network elements such as neurons and layers. Then, we will discuss the training procedure and the *backpropagation* algorithm, which is de facto the main tool for modern neural network training. After that, we will present the theoretical background regarding the ability of neural networks to solve tasks. Finally, we will consider difficulties that arise during neural network training and ways to overcome them. We will also recall an important concept of *inductive bias* in neural network architectures.

1.1. What a neural network consists of

First artificial neural networks were inspired by their biological counterparts. Their developers (e.g. McCulloch and Pitts [43], or Rosenblatt [44]) built a mathematical model of such neurons. Those neurons were able to react to the input signal by switching states from "not active" to "active". Stacked together multiple neurons formed a network. Those models allowed for impressive theoretical and practical results for their time (the 50-th – 70-th years of the 20-th century). However as time passed and the model evolved and now, when we are speaking about a neuron of an artificial neural network (ANN) we are usually considering the following function:

$$y = \sigma(\sum_i w_i x_i + b), \tag{1}$$

where x is an input of a neuron, w is a vector of weights (trainable parameters of a neuron), b is called *bias* and it is also a trainable parameter, σ is non-linear (usually differentiable or piecewise differentiable) function and y is an output. That neuron can be represented as demonstrated in Figure 2. The values of x, w, and b are usually considered to be real, represented by floating-point types on computing devices. However, for the sake of computational efficiency, they can be integer as well [45]. Of course, that is not the only possible type of neuron. For example, in recent years several multiplication-free neurons have been proposed [46, 47].

ANN is a structure, that consists of neurons, which we described above. The output of a neuron is passed to another neuron or is used as an output of a network. Thus, a neural network can be viewed as a computing graph, where each node is a neuron. However, that representation is not very convenient. Usually, researchers work with *layers* of neurons. A layer is a set of neurons with the same input, which are not connected to each other. So, a layer can be viewed as a function that maps an input vector to an output vector. It is usually considered, that all the neurons in a layer have the same non-linear function σ (also known as activation function). Then, if all neurons of a layer have the connections to all the inputs, such layer is called *fully-connected* and can be computed as:

$$y_j = \sigma(\sum_i w_{ji} x_i + b_j), \tag{2}$$

where the notation is the same as in (1), except that w is now a matrix of weights and b is a bias vector. As we can see from (2), a fully connected layer can be interpreted as two vector transformations: the first consists of matrix-to-vector multiplication and vector-to-vector addition, and the second is element-wise non-linear *activation*. So, a fully-connected layer can be viewed as two sequentially connected layers: the first performing matrix multiplication and addition, the second – computing activation function (see Figure 3).

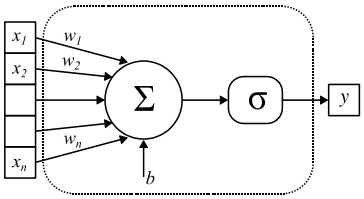


Fig. 2. An artificial neuron, as a mapping function of a vector x to a number y

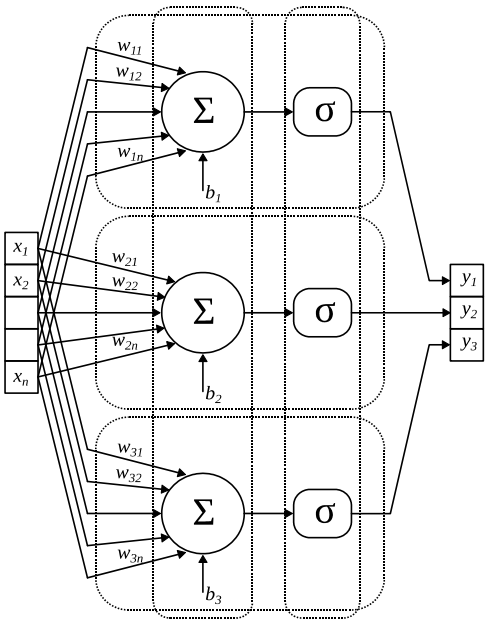


Fig. 3. A fully-connected layer of a neural network interpreted as Linear and Activation layers

Thus, a neural network can be viewed as a computing graph, where each node takes a vector (or several vectors) as an input, and outputs another vector (or several

vectors). This interpretation is used in popular neural-network frameworks like PyTorch [48] and Keras [49].

One of the simplest architectures of an artificial neural network is *Multilayer perceptron* (MLP). MLP consists of several fully-connected layers, which are sequentially stacked together (see Figure 4). MLP is one of the first ANN models, known since Rosenblatt [44], yet it is universal in a sense, that all *feedforward neural networks* (i.e. networks, where connections between neurons do not form a cycle) can be represented as MLP. Up to this day, MLP remains the main object of study in theoretical works (e.g. convergence of training [50, 51], scaling laws [52] and generalization bounds [53]) as well as empirical studies of the general properties of neural networks [54].

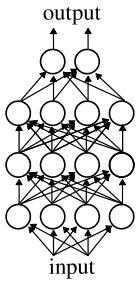


Fig. 4. An example of a multilayer perceptron. Circles denote neurons, and arrows – connections

More complex ANN architectures of neural networks were developed to solve difficult tasks like image recognition. We will describe some of them later in this article.

1.2. How a neural network works

Let us consider what ANN (particularly MLP) is from a mathematical point of view. The MLP consists of layers, which consist of neurons, which compute input transformations according to equation (1). Each layer has *trainable parameters*: weights (w), and biases (b) shown in equation (2). These parameters are adjusted when a network learns to solve a particular task. This process is called *training*. On the contrary, the number of layers and the number of neurons in each layer are *hyper-parameters* of MLP. They are usually selected and fixed prior to training. In more complex neural networks there may be more hyper-parameters, i.e. non-trainable values that determine network structure or behaviour.

Thus, ANN is a specific parametric vector function of many variables, and its training is just an optimization of some function, that measures how well the network solves its task, using the current set of parameters. This function is called *loss-function*. Its computation usually requires knowledge of correct answers for a given input.

The set of pairs of inputs and correct outputs is called *training set* $D = \{(X_i, y_i)\}_{i=1\ldots N}$. During training ANN (let us call it F_θ), computes the outputs (predictions) for inputs in a training set, given the current value of parameters θ, and the loss function L measures the inaccuracy of the prediction. To train a network is to choose parameters that minimize

$$L(F_\theta(X), y) \to \min_\theta, \tag{3}$$

for all samples (X, y) of a training set D. Usually, the loss function over a dataset is considered to be a sum (or an average) of the loss functions over particular points of the dataset, and then the training can be defined as an optimization problem:

$$\theta^* = \operatorname{argmin}_\theta \left(\sum_{i=1}^{N} L(F_\theta(X_i), y_i)\right). \tag{4}$$

F_{θ^*} is called trained neural network. It can be used to make a prediction. If a network is well-trained, its prediction will be accurate even on samples, that do not belong to the training set, but belong to the same distribution.

Now the question arises: how to solve (4) and train a neural network? Since this is a very high dimensional (modern neural networks contain up to billions of parameters) non-convex optimization problem, the exact solution is unknown. Yet there exist many methods to achieve sub-optimal solutions from evolutionary algorithms [55] to searching for high-quality sub-network in a randomly initialized neural network without any change of parameters (except for those that are zeroed-out by sub-network extraction) [56]. However, the main tools for training DNNs are methods based on stochastic gradient descent, which rely on the *backpropogation* algorithm [57].

The backpropagation algorithm is a method of computing gradients of loss function w.r.t. parameters of a neural network. It is based on Leibniz's chain rule for computing the derivatives of the composition of functions. It is applied as follows: first, the gradient of a loss function w.r.t. to the output of a layer is computed, then the gradient of a layer second to last, based on the gradient of the last layer, and so on (hence the name backpropagation).

Using the backpropagation we can compute the gradient for any given input. Thus we can use stochastic gradient descent to iteratively search for the minimum of a loss function (3). Of course, because DNN training is a nonconvex optimization problem, convergence to the global minimum is not guaranteed. Despite that, gradient descent-based methods have proven to be a good practical tool, especially the variations with momentum (heavy ball method) [57], and adaptive learning rate (gradient step size) like AdaGrad [58], RMSProp [59], and Adam [60] algorithms.

1.3. Why a neural network works

From the previous sections, we can conclude that a neural network is a parametric machine learning model. Given a training set, a loss function, and a training procedure this model can learn to solve a task. But the theoretical question remains open: can it solve any task or are there any principal limitations?

Universal approximation theorems answer this question. The first versions of such theorems for MLPs were independently proven in 1989 by G. Cybenko [61] and K. Hornik [39]. But before describing them let us first consider the earlier fundamental result: Kolmogorov's representation theorem [62].

Theorem 1 (Kolmogorov's representation theorem) *For any integer $n \geq 2$, there exist a set of real-valued continuous functions $\psi^{pq}(x)$ with domain $[0, 1]$, such as that for any real-valued continuous function $f(x_1, \ldots, x_n)$ with cube domain $[0, 1]^n$ can be represented as:*

$$f(x_1,...,x_n) = \sum_{q=1}^{2n+1} \chi_q \left(\sum_{p=1}^{n} \psi^{pq}(x_p) \right), \tag{5}$$

where functions $\chi_q(y)$ are real-valued continuous functions.

The equation (5) can be interpreted as a 3-layer neural network where all weights are either 1 or 0 (if there is no connection), and each neuron has its own activation function (see Figure 5). This network has a fixed number of neurons, and this number grows rapidly (as $O(n^2)$) with an increase of input dimensions n. Also, since functions ψ^{pq} do not depend on the function f, we can simply add new neurons to the second and third layers to increase the number of output dimensions (make f a vector-function), without changing the first layer.

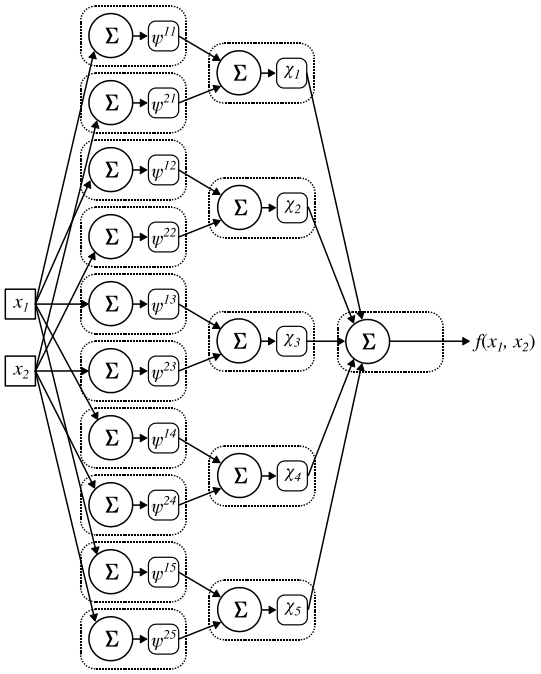


Fig. 5. Kolmogorov's representation theorem for a 2-variable (n = 2) function interpreted as a neural network

Kolmogorov's theorem of representation has one significant disadvantage: it describes a network, where each neuron has its own activation function. In real ANNs, those functions are usually the same (at least for the neurons of the same layer). That was empathised in works of G. Cybenko [61] and K. Hornik [39]. They proved that shallow (few-layer) MLPs with sigmoid-like functions can work as universal approximators. Let us present the theorem proposed by Cybenko.

Theorem 2 (Cybenko's universal approximation theorem) *Let σ be any continuous sigmoidal function. Then for any $\varepsilon > 0$, there is a finite sum of the form*

$$G(x) = \sum_{j=1}^{N} a_j \sigma(y_j x + b_j), \tag{6}$$

so that for any $f \in C(I_n)$

$$|G(x) - f(x)| < \varepsilon \quad \text{for all} \quad x \in I_n.$$

Also, for any $\varepsilon > 0$ and description function f, there is a finite sum of the above form and a set $D \subset I_n$ so that $m(D) \geq 1 - \varepsilon$ and

$$|G(x) - f(x)| < \varepsilon \quad \text{for all} \quad x \in D.$$

In the formulation of the above theorem *sigmoidal* are continuous real-valued functions of a single real argument such as that:

$$\sigma(t) \to \begin{cases} 1 \text{ as } t \to +\infty, \\ 0 \text{ as } t \to -\infty, \end{cases}$$

I_n denotes unit cube $[0, 1]^n$, $C(I_n)$ is a set of continuous functions on that unit cube, description function is such function that enumerates partitions of I_n ($I_n = P_1 \cup P_2 \cup \ldots \cup P_k$, and $f(x) = j \Leftrightarrow x \in P_j$), a, b and y are real-valued parameters of neural network.

We can see from Cybenko's theorem for a given class of activation functions, 2-layered MLPs can serve as universal approximators of any continuous or partially-constant function (or Borrel measurable functions [39]). Later K. Hornik obtained a similar result for arbitrary bounded and non-constant activation functions [63].

The early theoretical work was focused on the shallow (few-layer) MLP-s, while in practice DNNs have multiple layers, and different architectures (the manners in which layers are connected). So, recently similar approximation theorems were proven for residual neural networks [64], and for MLPs, where the number of neurons in each layer does not exceed the maximum between the length of the input and the output of a network [65].

From the above theorems, we can conclude that,

Corollary 1 *With enough neurons (i.e. large enough depth or width, i.e. neurons per layer) a neural network can approximate any (reasonable) function to any degree of accuracy.*

That is a fundamental property of ANNs, which explains why they can be used in a wide range of tasks.

1.4. Why a neural network does not work

From the previous sections, we have learned how to construct a neural network called MLP, and how to train it. We discussed that it can theoretically approximate any

given function, and thus correctly solve any task for which there exists a deterministic solution. However, as we will show in Sections 2 and 3 a neural network sometimes performs unexpectedly, and even a small change in a network's input can lead to a significant change of an output (even though should not, from a human point of view). That small change of an input is called *adversarial perturbation* and the methods, that are used to generate such perturbations are called *adversarial deep learning*.

But before considering the complex phenomenon of adversarial deep learning let us present other problems that an ANN may face.

In 1989, long before the recent growth of the popularity of neural networks, in a work, that proved one formulation of the universal approximation theorem K. Hornik stated, that "*... standard multilayer feedforward networks are capable of approximating any measurable function to any desired degree of accuracy, in a very specific and satisfying sense. We have thus established that such "mapping" networks are universal approximators. This implies that any lack of success in applications must arise from inadequate learning, insufficient numbers of hidden units, or the lack of a deterministic relationship between input and target*" [39]. In case of a lack of deterministic relationship between input and target, there is no algorithm that can map them, so it is not surprising that neural networks can not. We will not consider such a case.

The lack of hidden units (neurons), which was a problem at the end of the 20th century because of the limited computational capabilities of computers of that time, has been overcome by technical progress. For example, networks with 70 billion parameters are now feasible and applied to tasks like language modelling [66]. In the case of modern neural networks, computational complexity limitations exist only if such networks are applied in real-time systems and/or on resource-constrained devices (e.g. smartphones and embedded devices) [67].

The only remaining issue is inadequate learning. This is a complex problem that has many manifestations with respect to neural networks. Some of the learning limitations are common for all machine learning (ML) algorithms, some are specific to ANNs. Let us describe the most important ones.

1.4.1. Overfitting

The first and one of the most well-studied drawbacks of the ANNs is *overfitting* [68]. It is a general flaw of many ML algorithms, which arises from the minimization of error on the training set according to equation (4).

Let us consider a network trained to classify objects using the gradient descent method and cross-entropy loss (or similar loss function). It will learn to assign higher class probabilities to the correct classes according to the training set. After that a network can be used on arbitrary input, and the class with the highest probability is an answer of a network for this input. If we take two inputs, for which a network predicts different classes, and gradually "move" (interpolate) the input from one point to another, we will see that at some point the decision of a neural network changes. All the points at which the answer of a classifying network changes are called *decision boundary*. It is important to understand, that simple loss functions (like cross-entropy loss) do not give any guarantees on how this boundary will form. Thus, it is possible, that despite correct answers for the points of the training set, a neural network will predictions will not be useful for any other inputs (see Figure 6). This phenomenon is called *overfiting*, and it can usually be noticed by performing test runs of a network on a part of the dataset that was not used in the training (*test set*). When the quality of a network on the test set is significantly lower than on the *training set* – the network is overfitted. If a neural network has some adjustable hyper-parameters, the initial training set can be split into two parts. The first part is used for training, the second (*validation set*) – for unbiased quality validation and hyper-parameter estimation [69].

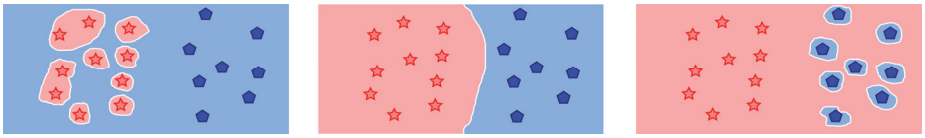


Fig. 6. Examples of overfitted (left and right) and not overfitted (middle) classifiers on the same training set

Methods aimed at preventing the model from overfitting can be divided into two categories: those that change the model and those that change the training process [70]. The first category includes pruning, model construction, and network architecture search. The pruning [71] is removing excess neurons from a network, thus simplifying it, making the decision boundary simpler and the network less prone to overfitting. Model construction is a less popular method. This is the opposite of pruning: it starts with an underparametrized (too simple) neural network and then adds neurons to it until the result has a satisfying quality. The intuition here is that the result is not overparameterized and thus not overfitted. Network architecture search (NAS) combines different techniques for automated network architecture selection with minimal human interaction [72]. The NAS aims to find the best model for solving a given task under the constraints of a computational budget. Therefore, if the validation set is used to evaluate quality during an architecture search, it may reduce overfitting. NAS techniques vary greatly. Simple ones are based on reinforcement learning [73] or evolution algorithms [74].

Unfortunately, they involve a lot of re-training of neural networks with different architectures during the search. It makes such methods very computationally demanding. More advanced NAS techniques leverage modular network structures [75], differentiable search [76], and other approaches that make NAS more feasible. Currently, NAS is in its early stages of development [72], and its goal – completely human-free optimal network architecture selection – is far from being reached.

There are other methods to constrain a given neural network, make it more robust and less prone to overfitting, and only change the training process (not affecting model architecture or inference). These methods are called *regularization* [68].

They apply one (or several) of the following methods:

- Modify input data in some way, enlarging and enhancing the training set. For example, in the case of image classification, it can be simple augmentation (like translation, rotation, or cropping of an input image) or more advanced methods like CutMix [77] and AugMix [78].
- Modify internal representation (input of internal layers of a network) to enhance the robustness of a network. Probably the most well-known method in this category is dropout regularization [79]. At every forward pass during training, it removes some neurons (zeroing out the values in the input vector), thus ensuring that a network learns multiple different features. There are many advanced variants of dropout, for instance, DropBlock [80] and LocalDrop [81]. The internal representation can be changed in a more complex way; for example, Manifold mixup [82] fuses the representations of different input samples.
- Modify labels. For example, neural network classifiers usually apply one-hot encoding of a label. However, this encoding can be replaced by a smoother one to prevent network overconfidence and overfitting. This smoothing can be simply uniform [83] or more complex, such as structural label smoothing [84].
- Modify loss function. Additional penalties, such as weighted l_1 of l_2 norms of the network's parameters, can be added to a loss function to make the network parameter distribution more sparse (l_1 norm) or more dense (l_2 norm) [85]. By constraining the parameters of a network, the parameter penalty (also known as weight decay) makes a network less prone to overfitting.

Various regularization techniques can often successfully reduce network overfitting, thus increasing the network quality on the test set. Still, there is no universal regularization protocol that would be suitable for all networks and all tasks [68].

1.4.2. Underfitting

Underfitting is the opposite of overfitting. It is a phenomenon that happens when an ML model is not able to capture the correct relation between the input and the output. That can be caused by the insignificant model complexity (i.e. incorrect selection of a model for the task), or too strong model regularization [85].

The common solution to the underfitting problem is to use more complex models (e.g. neural networks with more layers and trainable parameters) or to soften the regularization constraints.

1.4.3. Curse of dimensionality

"Curse of dimensionality" is rather a vague term that refers to the problems that arise when processing high-dimensional data and which do not exist in lower dimensions. These problems include the peaking phenomenon, distance problems, and data sparsity.

The peaking phenomenon is as follows: when training an ML model on a finite set of data, with an increase in the data dimensions (number of features), the quality of the model starts decreasing (instead of increasing). This means that more data features "confuses" the model and it struggles to find underlying dependency. However, recent studies indicate that the peaking phenomenon may occur (or not occur) in different ways, depending on the cumulative efficacy of additional data dimensions [86]. Neural networks can often avoid peaking phenomenon [87]. In addition, increasing the size of the dataset may help to overcome this limitation. Therefore, this particular variant of the "curse of dimensionality" is not crucial for modern neural networks.

Moreover, in high-dimensional spaces, for many reasonable distributions of the points of the dataset, the variance of distances between the points of the dataset is low [88]. This means that the distance between any two points of the dataset becomes closer to the average value as the number of data dimensions increases. This makes the nearest neighbor search overcomplicated in high dimensions [88]. Although neural networks usually do not rely on nearest neighbors search, this may become a problem for metric neural networks, or for anomaly search [89].

Probably the most essential problem for neural network learning is the sparsity of a dataset in the case of high-dimensional data (like images and videos). That is why adversarial perturbations (small changes in the input that significantly affect the output) are possible [90].

1.4.4. Vanishing gradients

Apart from general machine learning limitations, neural network have their unique ones. One of them is the vanishing gradient problem, which arises from the very nature of backpropagation. It happens in some (especially very deep neural networks with saturated activation functions) when the gradient values become too small and gradually diminish as they propagate backward through the layers of the network [91]. This prevents the network from effectively updating its weights and can even halt the training process.

Many approaches attempt to mitigate vanishing gradient problems, including ReLU-like functions, batch normalization, residual connections, and gradient magnification. ReLU and other similar functions [92] are only constrained from one side or not constrained at all; thus, gradients do not become too low when activation saturates [93]. Batch normalization [94] normalizes the input data during each training iteration. By keeping the data within a reasonable range, it helps prevent gradients from becoming too small or too large. ResNet architectures [95] introduce the concept of residual connections. Such connections directly connect nonsequential layers. They enable better information flow through the network, both in the forward and backward direction, and alleviate vanishing gradients problem. A similar concept is used in DenseNets [96], where the input of a layer is concatenated with the output before passing to the next layer. Figure 7 illustrates the data flow in the ResNet and DeseNet networks.

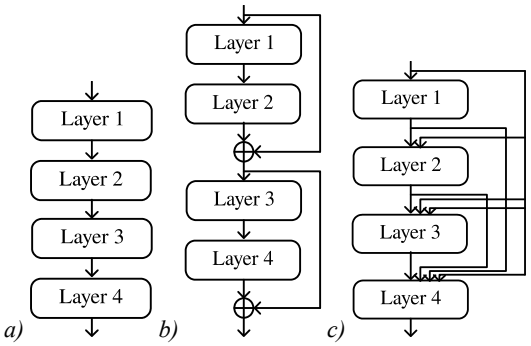


Fig. 7. Architectural solutions that avoid vanishing gradient problems of a standard feed-forward neural network (left). ResNet (middle) uses residual connections. DenseNet (right) concatenates the input with the output

Although the above methods mitigate the vanishing gradient problem, in general, it remains unsolved and constrains a set of neural network architectures that can be applied to practical tasks.

1.4.5. Exploding gradients

Exploding gradients are the opposite problem that arises when the gradients in the backward path become too high to maintain numerical stability or do not allow for gradient descent convergence. This may occur because of a specific network structure or incorrect network initialization [97].

To mitigate this drawback one may use batch normalization, control the network structure, use specific initialization techniques, or control the learning rate at the first iterations of training [98].

1.4.6. Hyper-parameter tuning

Another significant problem of neural networks is hyperparameter selection. ANNs usually have many hyper-parameters, including network structure (number of trainable parameters, number of layers, connectivity of those layers, etc.). The training process itself also has hyper-parameters: optimization algorithm (also known as optimizer), and parameters of this algorithm, such as batch size (number of data samples processed in a single iteration of the optimization algorithm) and learning rate (multiplier of step size of the optimizer).

Regarding network structure, NAS is a promising yet resource-demanding approach (see Section 1.4.1). Optimization algorithms may vary from a simple stochastic gradient descent (SGD) to more complex ones like Adam [60] and its variations [99]. The choice of the best optimizer for a given task is difficult. Sometimes simple SGD with momentum and proper training settings may outperform more advanced Adam (which is a typical default choice) [100]. Moreover, to achieve the best quality, the learning rate should differ according to the stages of training. State-of-the-art algorithms involve “warmup” at a low learning rate at the beginning of training, followed by a rapid increase in the learning rate, which is then decreased gradually [101]. Strategies with multiple increases and decreases in the learning rate are also possible and may be applied in some tasks to prevent suboptimal convergence of a neural network [102].

Of course, there are significantly more obstacles in ANN training than those listed above. For example, quantized neural networks (networks with integer weights and activations) face gradient mismatch problem [103]. This is due to the discrepancy between the continuous nature of the gradients and the discrete nature of the quantized values. Addressing these additional challenges is crucial for further advancements in neural network training and achieving their full potential in various applications. Researchers continue to explore various methods and algorithms to pave the way for more robust and efficient artificial neural networks.

1.5. Why there are so many networks

From Section 1.3 it may seem that a simple MLP (a feed-forward network with fully-connected layers) may be used as a universal architecture for almost all tasks (or at least for those, that can be reformulated as a classification). However, in practice, there are many different neural networks. Therefore, one may ask: why? In this section, we will try to answer this question.

We have already mentioned above (see Section 1.4.4), that some modification to a network structure, like residual connections, unconstrained activation functions, or batch normalization, may help to avoid the vanishing gradient problem. But of course that is not the main reason for the existing variety of neural networks. The main reason is the variety of tasks they solve. Here we come to an important concept of deep learning known as *inductive bias*.

An inductive bias in machine learning is a set of factors that makes the algorithm choose one inductive hypothesis over another, other than strict consistency with the training data itself [104, 105]. In other words, biases are fundamental properties of ML algorithms that

allow generalization. The only unbiased ML model is a lookup table, which searches for a given input in a training set, and if one is found, outputs corresponding results [104]. Such a model is practically useless because it is unable to produce output for an input, which it has not seen in a training set. Therefore, any practical ML model, including artificial neural networks, has some sort of inductive bias.

The same inductive biases can be good or bad depending on the particular task that a model is trying to solve. Let us demonstrate this using a convolutional neural network in the image classification task.

A convolutional neural network contains several layers that perform discrete convolution. They are usually followed by one or several fully-connected layers that produce the desired output (i.e. pseudo-probabilities of classes for multi-class classification). The input of a convolutional layer is a three-dimensional feature map (height, width, and channels), which is convolved with a four-dimensional kernel (which can be viewed as a set of three-dimensional filters) to produce the output feature map. Filters are usually small (1×1, 3×3, or 5×5 multichannel pixel windows). Therefore, the total number of trainable parameters in the convolutional neural network is significantly lower than in MLP. For example, at the end of the 20-th century, Yann LeCun proposed a convolutional network with as few as 60 000 trainable parameters, which could successfully recognize handwritten digits better than any other classifier of that time [106]. Since then convolutional neural networks (CNNs) have gone a long way and are widely used in many computer vision tasks.

By applying a convolutional layer, we add the following biases to a network:

1) translation-invariance (because convolution operation itself is translation-invariant);

2) data locality (since convolution operates on a small window, it extracts local features of an image, to produce a feature map). Those are desired properties or *explicit inductive bias*. However, the same operation produces undesired properties or *implicit inductive bias*. For example, unlike humans, convolutional neural networks are prone to recognize texture rather than shape [107]. The authors of this research avoid this particular drawback with the help of specific augmentation of the training set, i.e., with other explicit biases introduced directly into the training set. Moreover, sensitivity to local features is not only a strength but also a weakness of convolutional neural networks because they are unable to grasp global information. This disadvantage can be avoided with the help of global feature map transformations, for example, with Fourier [108] or Hough [109] transform.

The inductive bias of convolutional neural networks is very strong (meaning that it significantly constrains a model). Therefore, recent research has focused on models with weaker bias: visual transformers [110] and MLP-mixers [111]. Such models are less constrained and thus have greater generalization capabilities. However, there is a price to that potential–they are data-hungry and only outperform convolutional networks on very large datasets or with the help of specific regularization techniques [112] (i.e. additional inductive bias). It is also important to remember that visual transformers and MLP-mixers have inductive biases of their own [113]. Currently, both convolutional and transformer-based networks are applied in various computer-vision tasks [114].

2. Adversarial examples, or how a network hallucinates

Now that we have reviewed the main concepts, strengths, and weaknesses of modern neural networks, let us consider one particular drawback in detail: *adversarial examples*. As mentioned above, adversarial examples are inputs that confuse a neural network and thus modify its behavior in a way that an *adversary* wants. The process of generating adversarial examples is called *adversarial deep learning*. The widely used term *adversarial attacks* is rather misleading in this context because an adversary does not influence a neural network directly. He only manipulates its inputs to compromise the result and somehow influence the system, in which the neural network is used (e.g. avoid automatic recognition systems [115]). Although we believe that the term *attack* is misleading, it is widely used in literature, so in this paper, we will apply it, where necessary, but try not to abuse it.

Usually, adversary achieves his goals by the addition of a relatively small and almost unnoticeable signal [10] to the input called *adversarial perturbation*, by replacing part of a signal with *adversarial patch* [11], or by generating a new image that is only similar to the original image from a human perspective [116].

In this section, we consider different ways to generate adversarial examples in detail. We will mainly focus on adversarial images, that are used against neural network classifiers because that is the original and most developed area of adversarial deep learning. However, it is important to remember that similar approaches can be used in image detection [41], image segmentation [117], audio recognition [118] and natural language processing [119].

2.1. Generating adversarial examples for a given network

Let us consider the following situation. An adversary has a neural network f, and some image x. The network correctly classifies x as belonging to class c. However, an adversary wants to find a very similar (from a human perspective) image x', such as the neural network either

- classifies the image x' as belonging to the class $l \neq c$. Then l is called *target* class, and the process of finding such x' is called *targeted attack* on network f.
- classifies the image x' as any other image except for c. This is called *untargeted attack*.

Both targeted and untreated attacks that aim at fooling a neural network are often referred to as “evasion attacks” [40].

In this section, we will consider several classic examples of such attacks in detail, to better understand how they exploit the vulnerabilities of neural networks.

2.1.1. First adversarial perturbation

The rapid development of DNNs in computer vision began more than a decade ago when Krizhevsky et al. [93] successfully applied the convolutional neural network AlexNet to a difficult 1000-class image classification problem on ImageNet [120] dataset. Only a year later Szegedy et al. [10] demonstrated that such a network can be "fooled" using small image perturbations. Let us demonstrate how they did it.

Let $f(x)$ be the network classification result for the input image x. Let $l \neq f(x)$ be a class that the adversary wants a network to "see" instead of $f(x)$. The adversary wants to find perturbation δ such as that $f(x+\delta)=l$. The input should be valid, so $x+\delta \in [0,1]^N$, where N is the number of pixels in an image. The adversary also wants his perturbation to be small, so he poses the following optimization problem:

- **Minimize:** $\|\delta\|_2$, subject to
  (a) $f(x+\delta)=l$,
  (b) $x+\delta \in [0,1]^N$.

Because it is difficult to precisely solve this problem, the authors propose the following approximation:

- **Minimize:** $L_f(x+\delta, l)+\lambda$, subject to
  (a) $x+\delta \in [0,1]^N$,

where $L_f(x+\delta, l)$ is a loss function for a network f, with input $x+\delta$ w.r.t. desired class l; $\lambda > 0$ is the coefficient that controls the magnitude of the perturbation. When λ is low, the algorithm may find a perturbation that significantly changes the image. When λ is large, the perturbations are small, and it may be difficult for the algorithm to find an adversarial example. The authors apply line-search to find such maximal λ that the algorithm can actually find such δ that $f(x+\delta)=l$.

The solution to the latter problem can be easily found using backpropagation and gradient descent. Thus, adversarial perturbations are generated. The authors demonstrated that these perturbations are small and noise-like.

2.1.2. Fast gradient sign method

As it turned out, the first method to generate adversarial perturbations was not the simplest. A year later Goodfellow et al. [121] proposed the Fast Gradient Sign Method (FGSM). Using the previous notation, the perturbation is computed as follows:

$$\delta = \varepsilon \cdot \operatorname{sign}(\nabla_{\bar{x}}(L_f(x,c))), \tag{7}$$

where c is a correct class for x, $\nabla_{\bar{x}}$ is a gradient w.r.t. input image (presented as a vector), ε is a small enough parameter, that controls the perturbation magnitude.

The FGSM algorithm computes the gradient of a loss function in x, thus determining the direction of the most rapid increase in this function. It then computes the sign of the gradient and multiplies it by ε. Therefore $\|\delta\|_\infty = \varepsilon$, meaning that δ is such a perturbation that does not change the value of each pixel by more than $\pm\varepsilon$. This perturbation is computed using a single step in the gradient of the gradient sign (computed coordinate-wise).

Despite its simplicity, FGSM is a very efficient tool for adversarial deep learning. For example, with relatively low $\varepsilon = 0.1$ it successfully "fools" convolutional neural network, trained on CIFAR-10 datast [121], in 87 % of cases. I.e. 87 % of images that are correctly classified before perturbation is added, are misclassified after. This percentage is also known as *success rate* of an attack.

The algorithm of Szegedy et al. [10] that we have described in the previous section (let us call it A) and FGSM (let us denote it as B) have several important differences. They are as follows:

- A is iterative and B is single-step. B only requires single forward and backward propagation through the network to compute. This makes B significantly easier to compute. It is especially important for *adversarial training* – a taring procedure that includes both usual images from the training set and images with adversarial perturbations, to enforce the robustness of a network to such perturbations [121]. We will discuss adversarial training in detail in Section 3.
- B specifies the maximal perturbation bounds in terms of l_∞ norm (parameter ε), and A is unbounded. However, because A is iterative and includes a penalty for the l_2 norm of perturbation, this perturbation is usually low.
- Algorithm A requires a specific target class l and finds an image that the network confuses with l. So A is *targeted attack* and B is *untargeted attack*.

It is relatively easy to reformulate the FGSM in terms of targeted attack. Instead of stepping "away" from the correct class c, we need to step toward the target class l, so equation (7) can be rewritten as follows:

$$\delta = -\varepsilon \cdot \operatorname{sign}(\nabla_{\bar{x}}(L_f(x,l))). \tag{8}$$

In addition, FGSM can be generalized to Fast Gradient Method (FGM), which constrains arbitrary vector norms (not l_∞, specifically):

$$\delta = \varepsilon \cdot \frac{\nabla_{\bar{x}}(L_f(x,c))}{\|\nabla_{\bar{x}}(L_f(x,c))\|}, \tag{9}$$

where $\|\cdot\|$ denotes specific norm. Such generalizations for l_1 and l_2 norms were introduced in a popular framework called "Adversarial Robustness Toolbox" [122].

2.1.3. Basic iterative method and projected gradient descent

The Basic Iterative Method [123] (BIM) is a straightforward extension of FGSM. It applies FGSM multiple times while keeping the perturbation under a certain threshold ε. BIM is formulated as follows:

$$x_{i+1} = \mathrm{Clip}_{x,\varepsilon}(x_i + \alpha \cdot \mathrm{sign}(\nabla_{\bar{x}}(L_f(x,c)))), \quad (10)$$

x_i is an adversarial image on the i-th iteration of the algorithm ($x_0 = x$); $\mathrm{Clip}_{x,\varepsilon}(x_k)$ clips the value of its argument, such as that $||x - x_k||_\infty \le \varepsilon$; α limits the perturbation at each step of this iterative algorithm; in the original research $\alpha = 1/255$; the rest is the same as in equation (7).

BIM can "fool" the targeted network more efficiently because of its iterative nature. At the same time, BIM preserves the important quality of FGSM: it guarantees that the perturbation does not exceed the threshold ε. In the original setting, the BIM algorithm is an untargeted attack. However, it can be modified to be targeted in the same way as FGSM by changing the sign of the gradient and the class label to the target label in equation (10) (refer to equations (7) and (8) for analogy).

In their work Madry et al. [31], suggested a more general approach to generate adversarial images: Projected Gradient Decent (PGD). This is a standard convex optimization method, which works the same as gradient descent, but projects the vector to a required set (in our case, neighborhood of x) after each iteration. Using the same notation as in equations (7) and (10) PGD can be formulated as:

$$x_{i+1} = \Pi_{S(x,\varepsilon)}(x_i + \alpha \cdot \mathrm{sign}(\nabla_{\bar{x}}(L_f(x,c)))), \quad (11)$$

where Π is projection operator to $S(x,\varepsilon)$, which is neighbourhood of x. Usually PGD implements l_1, l_2, or l_∞ bounded adversary perturbations [122].

BIM can be viewed as a special case of PGD with l_∞-bounded perturbation. Thus, later in this article, we will refer to both methods as PGD.

PGD-based is widely used in ANN robustness estimation. Recently, several improved versions of PGD have been proposed. For example, AutoPGD [124], which accelerates PGD computation using a controlled decent step size and an automatic balance between the number of iterations in PGD and the number of its restarts from different points in the neighborhood of x. Another example is the PGD variation, which uses the conjugate gradient method to further accelerate and improve the convergence of PGD to an adversarial example [125].

2.1.4. Jacobian saliency map attack and single pixel attacks

All the methods discussed above produce adversarial images by small changes of *all* the pixels in the image by a small amount. In contrast, Papernot et al. [126], proposed an algorithm that changes only a few pixels. This is achieved by computing the Adversarial Saliency Maps, which indicate the input features (i.e. pixels of a grayscale image, or channels of pixels of the colored image), that have the most impact on changing the result of classification towards the desired class (i.e. increase ANN confidence in the targeted class, while decreasing the confidences in all the other classes). This method is based on computing the Jacobian of a neural network w.r.t its input; therefore, it is known as the Jacobian Saliency Map Attack (or JSMA). Let us outline its main ideas.

The method is iterative, in each iteration JSMA, computes the forward derivative of its input, then searches for two components of input vector $x \in [0,1]^N$, which are the most relevant to changing the output. This is done as follows:

$$\begin{aligned} &i, j = argmax_{i,j\in\{1,\dots,N\}}(-\alpha(i,j)\beta(i,j)\times \\ &\times(\alpha(i,j) > 0)\cdot(\beta(i,j) < 0)), \\ &\alpha(i,j) = \frac{\partial F_l(x)}{\partial x_i} + \frac{\partial F_l(x)}{\partial x_j}, \\ &\beta(i,j) = \sum_{k\neq l}\left(\frac{\partial F_k(x)}{\partial x_i} + \frac{\partial F_k(x)}{\partial x_j}\right), \end{aligned} \quad (12)$$

where $F_m(x) \in R^n$ is the m-th *logit* of a neural network for input x, that is m-th output of the last layer of the neural network, before conversion to pseudo-probabilities (i.e. before softmax function); l is the target class. Basically $\alpha(i,j)$ indicates how much the i-th and the j-th components of x increase networks confidence in class l, and $-\beta(i,j)$ indicates how much those components decrease the confidence in all the other classes. After components i and j are selected the algorithm increases x_i and y_j by $\theta > 0$, which is a fixed parameter of the algorithm. Note, that it is possible to use negative θ as well, but it will require some modifications in equation (12).

JSMA applies the iterations described above until the network prediction changes to the desired class l, or until maximal l_0 norm (number of modified components) γ is reached. Thus, it is a greedy algorithm. The authors mention that they modify two pixels at a time, because *"selecting pixels one at a time is too strict, and very few pixels would meet the heuristic search criteria described in equation (12)"*. So in each iteration, JSMA performs a search in a high-dimensional space of $(N^2 - N)/2$ pairs of input features, and after that recomputes the Jacobian. That is why this algorithm is time- and resource-consuming.

As we mentioned above, JSMA treats each channel of the input image separately. So, from its point of view, the perturbation of a single channel in three different pixels and the perturbation of three channels in the single pixel of a colored image are the same in terms of l_0 metric. Carlini and Wagner [127] pointed out that it is a rather strange threat model. Also, the original JSMA can not both increase and decrease components of the input vector (because θ is either positive or negative). In this context, the research of Su et al.[128] is especially interesting. Su et al. have demonstrated that it is sometimes possible to "fool" a neural network using a single perturbed pixel. This is called One Pixel Attack.

One Pixel Attack finds the pixel to perturb using differential evolution [129] in a five-dimensional discrete search space (red, green, blue channels and x, y coordinates). Let us denote the initial candidates as $\chi_i(0)$. At each iteration of the algorithm, a new generation of candidates is computed as follows:

$$\chi_i(g+1) = \chi_{r_1}(g) + \frac{\chi_{r_2}(g) - \chi_{r_3}(g)}{2}, \quad (13)$$

where g is the number of the generation; r_1, r_2 and r_3 are three random indexes; χ_{r1}, χ_{r2}, and χ_{r3} are considered to be the parents of χ_i. A child survives for the next iteration, if it has a better score in terms of similarity to the target class for targeted attacks or dissimilarity to the correct class for untargeted attacks.

It is important to note that, unlike previous methods, this particular algorithm does not rely on the gradients of the network. Therefore, it can be computed if the adversary only has access to the input of a network and its predictions (pseudo-probabilities of classes). This makes One Pixel Attack a *black box* attack in contrast to *white box* attacks, which require full access to the ANN. We will discuss several black-box algorithms in Section 2.2.2. This method was further developed to few-pixel perturbation in [130]. However, the search space grows exponentially with an increase in the number of pixels, which may prevent convergence of the evolution algorithm.

2.1.5. DeepFool and NewtonFool

DeepFool [131] is another method that finds minimal adversarial perturbations that change the classification result of the ANN (i.e. it is an untargeted attack). To that end, DeepFool attempts to find the nearest decision boundary of the classifier (i.e. the hyperplane in the input space, which separates different classes) and project the input image to that decision boundary. In the case of a two-class affine classifier $f_a(x) = wx + b$ (which is also a single-layer neural network), the perturbation can be analytically computed as follows:

$$\delta = -\frac{f_a(x)}{\| w \|_2^2} w. \quad (14)$$

For a nonlinear classifier such as a DNN, the same result can be achieved iteratively by linearizing the function $f(x+\delta_i)$ in the neighborhood of $x+\delta_i$:

$$\delta_{i+1} = -\frac{f(x+\delta_i)}{\| \nabla f(x+\delta_i) \|_2^2} \nabla f(x+\delta_i). \quad (15)$$

These iterations are repeated until the sign of $f(x+\delta_i)$ changes, thus achieving another binary classification class decision. In the case of multi-class classification, the classifier is treated as a set of binary classifiers, and the equation (15) is generalized by replacing $f(x+\delta_i)$ with $f_c(x+\delta_i) - f_l(x+\delta_i)$, where f_c is the logit of the correct class and f_l is the logit of the incorrect class l. The step is taken toward the closest incorrect class.

The authors of DeepFool point out that it can be viewed as a generalization of Newton's iterative algorithm for finding the roots of a nonlinear system. DeepFool can find adversarial perturbations that are smaller than those of FSGM. It also works faster than another iterative algorithm of Szegedy et al. [10] because DeepFool does not require any hyperparameter search.

There is another very similar algorithm known as NewtonFool [132]. Its key difference is that it aims to find the closest decision boundary, not by stepping toward an incorrect class, but away from the correct (as in FSGM, but using Newton's algorithm for roots), thus further accelerating the computation.

2.1.6. Carlini and Wagner

N. Carlini and D. Wagner [127] investigated various models of adversarial attacks and a distillation defense [133], which used to be a popular countermeasure to adversarial deep learning. They proposed new l_0, l_2, and l_∞ constrained attacks, that completely break this defense. The reasoning leading to choosing the best objective function for the attack model as well as a set of practical tricks that allow to elevate non-differentiable constraints of l_0 and l_∞ algorithms is fascinating. That is why we invite interested readers to familiarize themselves with the original research [127]. Here we will briefly outline the l_2 version of Carlini and Wagner's attack (we will denote it as a C&W attack).

The adversary wants to solve the box-constrained optimization problem of searching the image, that ANN misclassifies as a targeted class (C&W attack is a targeted one). The formal statement is the same as in Section 2.1.1. Since this is a hard box-constrained non-differentiable optimization problem, it is approximate, similar to the method of Szegedy et al. [10]. However, here the authors use the following objective:

$$\begin{aligned} &\| \frac{1}{2}(\tanh(w)+1) - x \|_2^2 + \\ &+c \cdot \phi(\frac{1}{2}(\tanh(w)+1)) \to \min_w, \\ &\phi(x') = \max(\max_{i \neq l}(f_i(x')) - f_l(x'), -\kappa), \end{aligned} \quad (16)$$

where $w \in \mathbb{R}^N$ is an adjustable parameter, $f_i(x')$ is a logit of class i, predicted by a network for the input x', l is a target class, κ and c are parameters of the algorithm. Parameter κ is set manually, to set confidence with which misclassification occurs. Parameter c is either set manually [127] or automatically adjusted [122] to be small enough so that both terms in the objective (16) are minimized simultaneously.

Function $\phi(x')$ in the objective (16) measures the "goodness" of the adversarial image (the lower the better), by computing the difference between the current best logit of the network, other the l-th, and the l-th logit itself. If $\phi(x') > 0$, the attack fails because there exists a class at which the network is more confident than l. The

difference between logits is clipped by $-\kappa$ to prevent the algorithm from over-tuning to the target class while sacrificing the similarity to the source image. This is one of the main distinctive features of the C&W attack. Other important details include automatic box constraint: adversarial image $x' = (\tanh(w)+1)/2$ is always in the range [0, 1] and *multiple starting-point gradient descent*: the algorithm selects multiple random points in the neighborhood of x to search for adversarial example, to avoid being stuck in a suboptimal point.

C&W proved to be a very useful tool against many adversarial defenses (countermeasures to adversarial attacks). The price for its usefulness is computational complexity [40].

2.1.7. Shadow attack

All the above methods try to produce a small perturbation (measured by l_0, l_2, or l_∞ norms). Thus, if there is some guarantee that the given ANN classifier is robust against such small perturbation (refer to Section 3.2 for examples of such certified classifiers), then those methods will be unable to find adversarial perturbation. The authors of a ShadowAttack [134] point out, that from a human perspective, some perturbations may be insensible even if they are significant in teams of vector norms. They also notice that many l_2 norm-based perturbations are sensitive to Gaussian smoothing, or the addition of Gaussian noise (which is also a smoothing in some sense), which allows for countermeasures which adopt such strategies (e.g. [135]).

The Shadow Attack produces perturbations that

1. is imperceptible to the human eye,
2. is not sensitive to the added Gaussian nose, thus imperceptible to noise-based defenses.

This is achieved by minimizing the following objective:

$$L_f(x+\delta, l) + \lambda_c C(\delta) + \lambda_{tv} TV(\delta) + \\ + \lambda_s Dissim(\delta) \to \min_{\delta}, \tag{17}$$

where

- δ is the adversarial perturbation, that we are searching for.
- l is the target class (Shadow Attack is by design targeted, but can be generalized to untargeted attack).
- L_f is a loss function (i.e. cross-entropy), not only for a given point $x+\delta$ but also for its neighborhood; L_f is computed by averaging losses for several samples acquired by the addition of random Gaussian noise to $x+\delta$.
- Penalty $TV(\delta)$ is anisotropic total variation. It enforces perturbation to look smoother and more natural to the human eye.
- Penalty $C(\delta) = \| \operatorname{avg}|\delta_R|, \operatorname{avg}|\delta_G|, \operatorname{avg}|\delta_B| \|_2^2$ limits the change in the mean absolute value of each color channel of perturbation (δ_R, δ_G, and δ_B. Here $|\cdot|$ computes absolute values element-wise and avg(·) computes the average of the vector. $C(\delta)$ aims to prevent extreme changes in the color balance of the input image.
- Penalty $Dissim(\delta) = \|(\delta_R-\delta_G)^2 + (\delta_R-\delta_B)^2 + (\delta_G-\delta_B)^2\|_2$, is also aimed at preserving the original color of the source image. Thus, the image becomes lighter or darker (hence the name Shadow Attack), and not changing its color.
- λ_c, λ_{tv}, and λ_s are floating-point parameters that control the relative weight of the components of the objective (17). In the original research, they were set as follows: $\lambda_c = 1$, $\lambda_{tv} = 0.3$, and $\lambda_s = 0.5$.

The images produced by a shadow attack look very natural. This method is aimed at fooling the defense algorithms by design. This makes the Shadow attack almost intractable. The price of this intractability is the high computational cost.

2.1.8. AutoAttack

As we can see, there are many different methods for generating adversarial perturbations. We have only mentioned the few most popular ones. For example, another recent survey [29] names more than 50 methods. Therefore, researchers who want to assess the robustness of their model to adversarial examples have difficulty choosing the attack method.

To overcome this challenge AutoAttack was proposed [124]. First, the authors improve standard PGD [122], which results in the AutoPGD method mentioned earlier in Section 2.1.3. Then, they propose replacing the standard cross-entropy loss with a normalized version of C&W loss (refer to (16)). AutoAttack combines AutoPGD a new loss function with other methods such as fast adaptive boundary attack [136], and square attack [137]. Auto Attack is also automatic as it does not require any hyperparameter fine-tuning. Therefore, it is a strong and reliable baseline for ANN robustness evaluation.

2.1.9. Illustrations

It may be hard to understand what an adversarial perturbation is without a visual example. That is why we illustrate how the above algorithms generate adversarial perturbations in practice. We applied them to the well-known ResNet-50 model [95], trained on ImageNet dataset [120] according to the training methodology of Torchvision library [138]. In this section, we refer to the pseudo-probability of a class, which is predicted by a network, as the confidence of a network in this class and measure it in percentages.

We selected two images shown in Figure 8 from the validation set of the ImageNet dataset. Both images are 8-bit RGB images with 224×224 resolution. The network correctly classifies them as “Goldfish” with 65.6 % confidence, and “American chameleon” with 25.3 % confidence (the confidences in other classes are lower). We implemented algorithm of Szegedy et al. [10] (refer

to Section 2.1.1 for more details) using PyTorch framework [48]. For other algorithms, we used implementations provided by Adversarial Robustness Toolbox library [122].

*Fig. 8. Two selected images from the ImageNet dataset. Left: "Goldfish" (with **65%** confidence). Right: "American chameleon" (with **25%** confidence)*

Figure 9 illustrates the results of the algorithm of Szegedy et al. [10]. We set the regularization parameter $\lambda = 1$ and performed 100 iterations of the gradient descent. The network classifies the modified images as "Siamese cat" with 99.9% confidence. As we can see from Figure 9, the adversarial perturbations are very low and almost unnoticeable at first sight. However, adversarial perturbations in this case are noise-like and can be noticed in the background. Whereas the background in Figure 8 is smooth, in Figure 9 it is noisy. If we look closer, we may notice that this nose is not uniform and random but rather forms some patterns. This is how an expert can conclude that the image was tampered with.

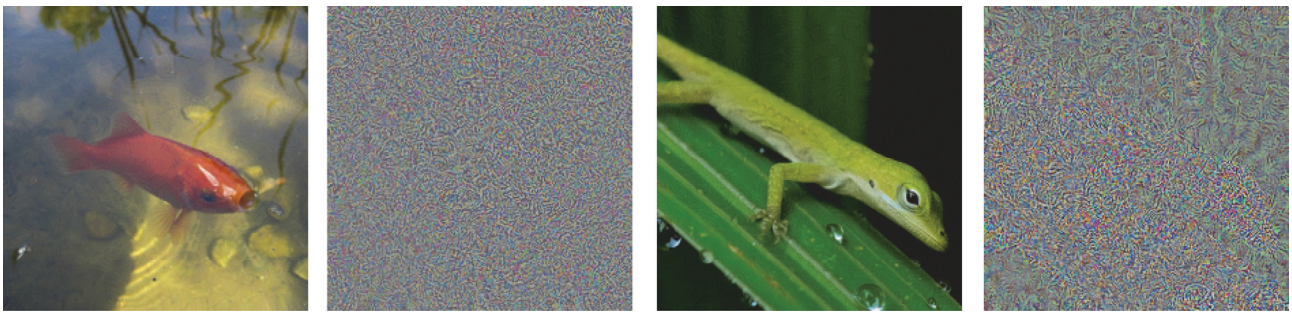


*Fig. 9. The images produced by the algorithm of Szegedy et al. [10] and corresponding adversarial perturbation magnified 10 times. Left: "Siamese cat" (with **99.9%** confidence). Right: "Siamese cat" (with **99.9%** confidence)*

Figure 10 demonstrates adversarial perturbations produced by FGSM [121] algorithm (refer to Section 2.1.2 for detailed description). We set parameter $\varepsilon = 5/255$. The input is an 8-bit image; thus, each channel of each pixel is changed by no more than ± 5 (if we consider integer values before scaling to the interval $[0, 1]$). The network classifies the left in Figure 10 as "Goldfish" with 25.8 % confidence. This is still a correct prediction, but the confidence is noticeably lower. The right image is classified as "Green lizard" with 17.5 % confidence. This prediction is incorrect, so FGSM succeeds in fooling the network. The less impressive results of the FGSM algorithm are due to its single-step nature compared to the iterative algorithm of Szegedy et al.

We can see the "noize" added by adversarial in Figure 10 is stronger, than that in Figure 9. However, the maximum absolute difference between the original and the modified images (l_∞ measure of the perturbation) is lower for the FGSM. This controversy arises because FGSM (as any other l_∞-bounded algorithm), noticeably changes the values of each pixel, which is more visible, then small perturbations of each pixel, and strong perturbations of some pixels in Figure 9.

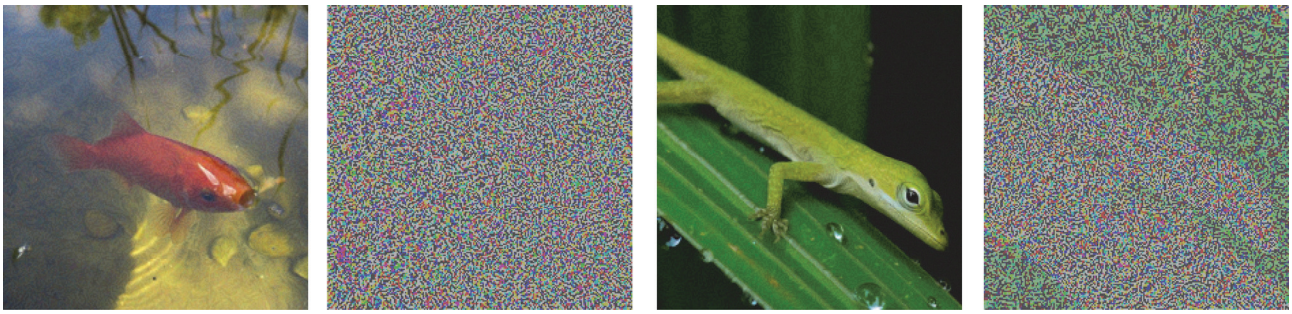


*Fig. 10. The images produced by the FGSM algorithm and corresponding adversarial perturbation magnified 10 times. Left: "Goldfish" (with **29%** confidence). Right: "Green lizard" (with **18%** confidence)*

Figure 11 shows the adversarial images produced by BIM algorithm [123] (see Section 2.1.3 for more details). It has the same perturbation threshold $\varepsilon = 5$, as FGSM in Figure 10. We set the maximum perturbation at each iteration $\alpha = 1/255$ as in the original research, and the total number of iterations was 100.

During those 100 iterations, BIM managed to produce images that completely fool ResNet: in the left image it predicts "Spider web" (not "Goldfish") with 99.9 % confidence, and in the right one "Green lizard" (not "American chameleon") with the same 99.9 % confidence. We can also observe that the perturbation

patterns of the BIM algorithm are more complex than those of the FGSM algorithm. Still, these perturbations can be noticed by the human eye as unnatural changes in the background.

The JSMA [126] algorithm (refer to Section 2.1.4) is very different from those illustrated above. It greedily tries to minimize the number of perturbed features and only increases their value. As we can see from Figure 12, JSMA modifies less than 5 % of pixels; however, such perturbations are very distinguishable. ResNet was successfully fooled by those images and once again "sees" "Siamese cat" (a target class) there.

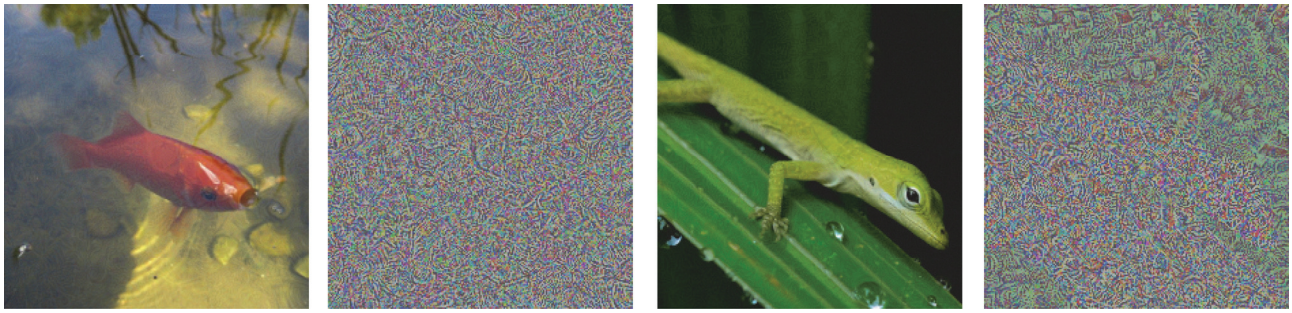


*Fig. 11. The images produced by the BIM algorithm and corresponding adversarial perturbation magnified 10 times. Left: "Spider web" (with **99.9%** confidence). Right: "Green lizard" (with **99.9%** confidence)*

*Fig 12. The images produced by the JSMA algorithm and corresponding adversarial perturbation magnified 10 times. Left: "Siamese cat" (with **24%** confidence). Right: "Siamese cat" (with **18%** confidence)*

On the contrary, DeepFool [131] is not aimed at minimizing the number of perturbed pixels. It minimizes the l_2 norm by design. However, we can see in Figure 13, that the found perturbation is very low and almost imperceptible. This is likely caused by the fact that DeepFool searches for the decision boundary and does not attempt to further perturb the image to increase network confidence in the incorrect class. In our case, the network is 14.7 % confident that the left image is "Coho salmon" and the left image is "Green lizard" with 16.3 % confidence.

*Fig. 13. The images produced by the DeepFool algorithm and corresponding adversarial perturbation magnified 10 times. Left: "Coho salmon" (with **15%** confidence). Right: "Green lizard" (with **16%** confidence)*

The results of C&W attack [127] (see Figure 14) are somewhat similar to those of DeepFoool. The perturbations are small, nearly invisible, and local, meaning that they do not change the entire image. The last property is not guaranteed, because this method minimizes the l_2 norm of the adversarial perturbation. In the left image, the network recognizes "Coho salmon" with 29.6 % confidence, and in the right – "Green lizard" with 19.9 % confidence.

Shadow Attack [134] (refer to Section 2.1.7) on the other hand produces smooth and natural-looking perturbations (see Figure 15). It is possible to find them if you know what you are looking for (e.g. have perturbation maps, demonstrated in the bottom row of Figure 15), but otherwise, they can be perceived as properties of the image. Unfortunately, in this particular case, the attack was unsuccessful, and only slightly lowered the confidence of the network (to 56 % and 25 % for the left and the right images respectively).

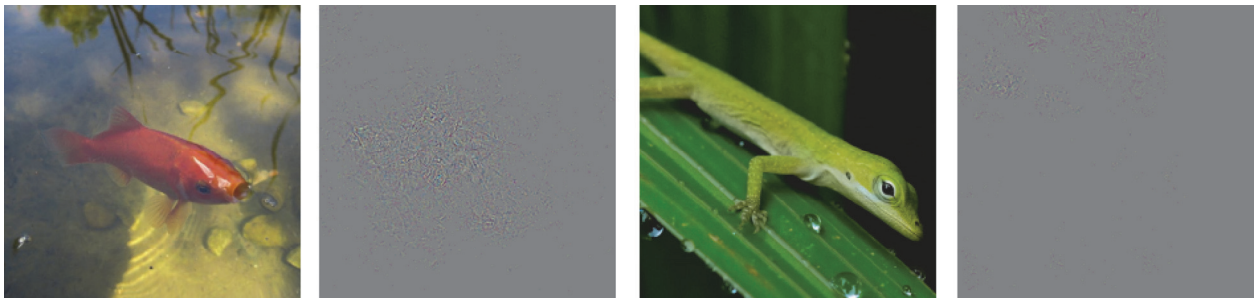


*Fig. 14. The images produced by the C&W algorithm and corresponding adversarial perturbation magnified 10 times. Left: "Coho salmon" (with **30%** confidence). Right: "Green lizard" (with **20%** confidence)*

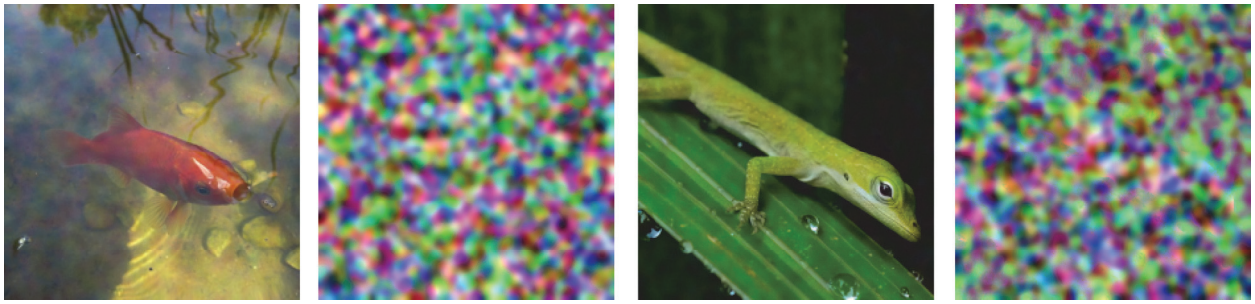


*Fig. 15. The images produced by the Shadow Attack algorithm and corresponding adversarial perturbation magnified 10 times. Left: "Goldfish" (with **56%** confidence). Right: "American chameleon" (with **25%** confidence)*

2.2. Obscurity does not mean security

Almost all the methods that we considered above required the adversary to have direct access to two key components:

- *source image*, that the adversary tries to modify;
- *neural network*, to compute its gradients w.r.t. the input or to evaluate it multiple times (in case of One Pixel Attack).

It turned out that this access is not essential for adversarial deep learning.

Therefore, the inability of the adversary to access a digital image (e.g., if it is directly captured from a camera) or the network (e.g., if it is computed on a secure server) does not guarantee that the vulnerability of the ANN to adversarial examples would not be exploited. In other words: *obscurity does not mean security*. In this section, we consider several examples that prove this point.

2.2.1. Adversary without source image

Let us first consider the following scenario: the adversary aims to produce adversarial for many input images but does not have computational resources (or time window) to compute perturbations for each image in the input. Instead, the adversary seeks to discover a perturbation that can be universally applied to *any* input image, thereby producing an adversarial example.

Moosavi-Dezfooli et al. [139] demonstrated that such perturbations exist and called them "Universal adversarial perturbations". Universal perturbations are generated similarly to the DeepFool algorithm (refer to Section 2.1.5 for more details), which iteratively pushes the current perturbation toward the nearest decision boundary, but with two major differences:

- at each iteration, a new input image is used (but the perturbation remains from the previous iteration);
- at each iteration, the perturbation is projected back to l_p (originally l_2 or l_∞) sphere with zero center, thus limiting the perturbation.

By design, generating universal perturbations requires some labeled dataset, but as the authors point out, the size of this dataset can be negligible compared with the one used for classifier training. Some adversarial perturbations for different CNNs trained on the ImageNet dataset are shown in Figure 16.

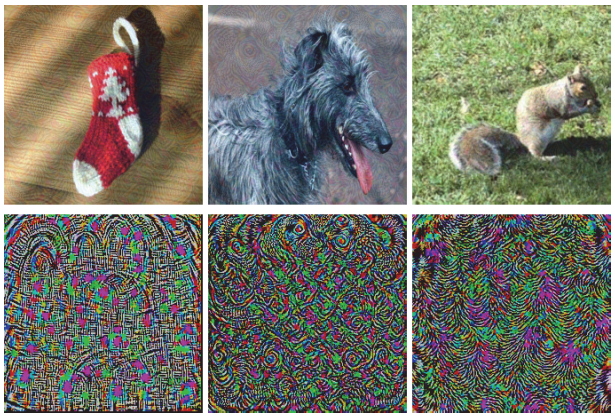


Fig. 16. The adversarial examples produced by universal perturbation (top row) and examples of universal adversarial perturbations for different CNNs trained on ImageNet. Perturbations are amplified for better visualization (bottom row). The images are from the original research [139]

Despite being generated for one particular CNN, the universal perturbation can "fool" other CNNs as well. This property is called *transferability*.

Universal perturbations are only possible in the digital domain. Incorporating such perturbations into the physical world requires the following challenges:

- the perturbation is constrained to the surface of a certain object;
- the properties of the perturbation (mainly its ability to affect the ANN's output) should be preserved if the point of observation of the perturbed image changes.

To overcome these obstacles, expectation over transformation (EOT) was introduced [140]. The adversarial objective is not simply computed for the given perturbation. On the contrary, it is averaged over many geometrical transformations that model the movement of an object with *adversarial texture* in the 2D or 3D space. The EOT was further used to generate Adversarial Patches [11] – relatively small image patches that can be printed, placed on the scene, and photographed. Later, if such a photo is passed to an ANN classifier, it will react to the patch and ignore all other objects in the scene (see Figure 17).

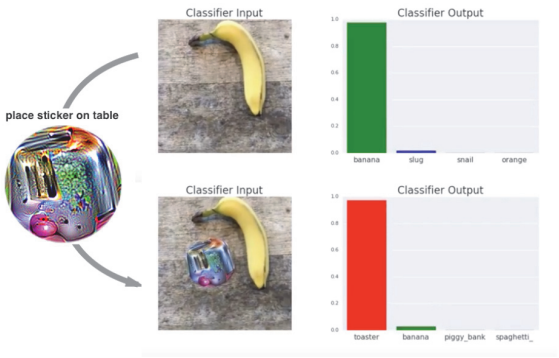


Fig. 17. The adversarial patch, put into the scene in the physical world, fools the ANN classifier; as illustrated in original work [11]

2.2.2. Adversary without direct access to the target network

Now let us consider the following setup: an adversary does not know what a neural network classifier is under attack but can query this ANN and obtain the classification result (the label of the class with the highest pseudo-probability, predicted by ANN) or vector of pseudo-probabilities of classes. Methods that operate in this setup are often called *black-box attacks*, in contrast to *white-box attacks*, which assume that an adversary knows the network.

Probably the most well-known black-box attack is Decision-Based Attack [141]. The core idea of this method is simple. Suppose we have two images of different classes (source image and some image of target class, which we want to assign to the source image). In that case, there is a decision boundary at which the ANN changes its classification result from one class to another. We can determine the image at the decision boundary by interpolating these two images linearly. We can then add small random perturbations to the image and control that

1. it becomes more and more similar to the source image (in terms of l_2 norm);
2. it remains misclassified (lies outside the decision boundary of the correct source class).

The Decision-Based Attack can achieve impressive results regarding the similarity of the adversarial image to the source image. However, its convergence requires tens of thousands of queries to the ANN. Thus, it is very computationally demanding and would be almost infeasible in a real-world scenario as an attack on a computer vision system based on an ANN classifier.

On the other hand, Square Attack [137] requires only about a hundred queries to the network (two orders of magnitude fewer queries than Decision-Based Attack) to generate an adversarial sample. However, it requires knowledge of the pseudo-probabilities of the classes, which is a weaker constraint. High query efficiency is achieved because the perturbation search space is limited to adding square-like patterns to the source image (while keeping the perturbation below the pre-defined l_2 or l_∞ threshold). The Square Attack is shown in Figure 18. Despite its simple pattern structure, the Square Attack achieves a success rate similar to white-box methods under the same constraints on the perturbation norm.

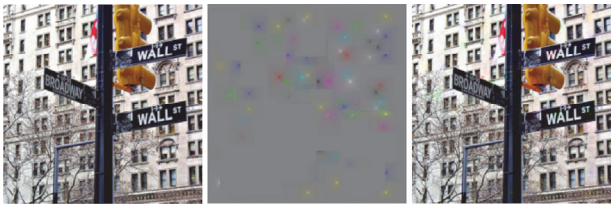


Fig. 18. Illustration of the l_2-bounded Square Attack, from the original paper [137]. The source image (left), the adversarial perturbation (middle), and adversarial example (right)

It is interesting to note that performing a black-box attack on a network that works in the physical world is possible. A recently proposed GRAPHITE [142] (acronym for Generating Robust Automatic PHysical Image Test Examples) is an example of such attacks.

2.2.3. Adversary without target network at all

Now, let us consider that the adversary does not have access to the network at all but knows which task it is designed to solve (or maybe even has access to the dataset it was trained on, e.g., this dataset is public). In this case, an adversary can leverage *attack transferability* to find adversarial perturbations.

We have already mentioned above that Universal Adversarial Perturbation [139] is transferable in the sense that it is to some degree effective against ANNs other than the one it was designed to deceive. However, it is not an exclusive property of universal adversarial perturbation. Transferability of the adversarial examples was discovered along with the first adversarial images by Szegedy et al. [10]. Many

other white-box methods (e.g. FGSM [121] and DeepFool [131]) share this property.

Leveraging transferability, an adversary can compute adversarial perturbation using a surrogate ANN (basically any ANN trained to solve the same task) and any white-box perturbation algorithm. Many recent studies have focused on improving the transferability of adversarial examples: for example Xie et al. [143] enhanced input diversity by random resizing and padding the source image. They also attacked several ANNs simultaneously (averaging the optimization objective). Thus, they significantly improved the transferability of the generated adversarial images.

Wang et al. [144] applied MixUp-like augmentation [145] to the source image but without changing its label. It also resulted in more transferable attacks. Another interesting approach was suggested by Wu et al. [146], who restricted adversarial perturbations to the most crucial regions of the input image using the attention mechanism, thus ensuring that perturbation modifies the features that are likely to be used by any ANN classifier, regardless of the particular architecture.

Overall, protective actions like securing the network and its architecture or controlling its input complicate the adversary's task but certainly do not secure ANN entirely. Still, making the adversary's path to fooling the network as difficult as possible is always a good idea, so such simple security measures should not be ignored.

2.3. Neural networks that attack each other

All of the above methods for generating adversarial perturbations can be considered conventional. In these methods, the authors pose some optimization problems. If the problem turns out to be hard, they relax or approximate it. After that, they apply an optimization algorithm to look for a solution. However, such approaches are constrained by the search for similar images in the input vector space. The adversarial sample is considered similar to the source image if the norm of the difference between the source and adversarial images (i.e., the norm of adversarial perturbation) is low. However, this is different from how humans perceive similarity. For example, we can assume that the two signatures of the same person are similar. However, if we digitize and compare them pixel-wise, they will differ significantly. On the contrary, the small (from the vector norm perspective) perturbation of classical methods looks unnatural to the human eye (consider, for example, the noise of FGSM perturbations in Figure 10). Thus, we can conclude that no such images exist in "real world". Therefore, instead of them, we may want to generate adversarial samples that look natural. Of course, it is an ill-posed computer vision problem. Therefore, it is no surprise that only feasible solutions are provided by an ANN.

Zhao et al. [147] followed a similar chain of thought and proposed the generation of natural adversarial examples using a generative adversarial neural network (GAN) [148]. GAN comprises two parts: *generator* and *discriminator*. The generator takes a dense vector as an input and produces an image that should look like the one from the training dataset. The discriminator classifies whether its input image is from the training dataset or produced by the generator. If the generator and discriminator are trained together, the generator learns to mimic the train dataset distribution. The adversarial attack of Zhao et al. [147] works as follows:

1. The input image is encoded into the generator's dense input space (i.e., latent space) using a specifically trained encoder ANN.
2. Small amounts of random noise perturb this encoding.
3. The perturbed encoding is decoded using the generator.
4. If the prediction of the network does not change, the procedure is repeated with slightly higher noise.

Thus, perturbation in the input space is replaced by perturbation in the latent space of the generator. Since a well-trained generator always produces natural-looking images, the adversarial example would also be natural-looking.

The widely adopted baseline for ANN-generated adversarial perturbations is AdvGan [149]. It is also a GAN-based algorithm; however, in this case, the generator takes the source image as an input and directly produces adversarial perturbation. Thus, it is computationally more efficient (not requiring any search in the latent space).

Recently Ho et al. [150] demonstrated the superiority of diffusion-based ANNs to GANs in the tasks of high-resolution image generation. That is likely why the most recent ANNs for adversarial image generation are also diffusion-based [151, 116, 152]. For example, AdvDiffuser [116] allows one to "generate adversarial examples that are semantically close to the originals, yet containing shape-based adversarial perturbations exhibiting detailed diversity". To achieve that, the authors perform an interesting trick. The diffusion network itself is designed to perform iterative denoising operation [150]. Classic attacks like PGD [31] add noise-like perturbation to the image. Thus, authors of AdvDiffuser use PGD as a source of additional noise (and the adversarial perturbation) in the denoising process. However, since the diffusion network is trained to mimic the training data (like the generator in GAN), the final result will also look natural, although it will contain small differences from the original image (refer to Figure 19).

The ANN-generated perturbations are unconstrained by any norm in the input vector space and are aligned with the dataset distribution (if the generative ANN is well-trained). That makes such an attack almost imperceptible both to the human eye and defensive algorithms [152].

Fig. 19. Adversarial perturbations, produced by AdvDiffuser, as reported in original paper [116]

3. Why the fight against adversarial examples is not won yet

A decade has passed since the first adversarial examples were discovered. Of course, the scientific community was anxious about them, and researchers proposed many *adversarial defenses*, i.e., algorithms that aim to detect adversarial perturbations or train ANNs that are not vulnerable to such perturbations. Then, a reasonable question arises: why do adversarial perturbations remain a problem? Why don't we use some algorithm that produces adversarial-free ANNs?

The simple answer to this question is that there is no such algorithm. A more detailed answer is that adversarial defenses are not universal or free. When protecting the network against a particular class of attacks, we sacrifice computational resources for more complicated training. Often (yet not always), ANN quality on clean labels also decreases. Thus, in some cases, it may be more appropriate to use a better but not robust network than a worse but robust one. Now, let us consider several defensive approaches to prove this point.

Two main groups of adversarial defense methods exist. They are called *heuristic* and *certified*. Although heuristic methods do not guarantee protection, most are relatively simple and easy to implement and often show good empirical stability against specific adversarial perturbations. On the other hand, certified methods are based on formal mathematical concepts and offer strong security against perturbations under pre-defined assumptions that can be theoretically guaranteed. However, these methods are usually more computationally intensive. In this section, we briefly overview both groups of adversarial defenses.

3.1. Why hot fix is not enough

Let us start with heuristic defenses. It is almost impossible to overview all the variety of ad hoc methods that the researchers suggested for countering the adversarial examples, so we will focus on two main groups:

- augmentation-based defenses;
- adversarial training;

Input transformations include various filtering [153], noise addition [154], and obfuscation [155] (e.g. JPEG compression and total variance minimization) of the input image before it is passed to the network. However, these and similar approaches have two significant limitations. First, despite being efficient against specific attacks, they may lower the final quality of the ANN on clean data [156]. Second, as Carlini and Wagner (authors of C&W attack [127]) demonstrated, if the attacker knows what defensive mechanism is used in the system (e.g., in input preprocessing), this mechanism can easily be broken [157].

The idea behind adversarial training is to use adversarial samples as training inputs to improve robustness. It was introduced in the very first work regarding adversarial images [10] and enhanced in the following research [31]. This method has many modifications according to different attack procedures (e.g., incorporating momentum [158], FGSM-attacks [121], PGD-attacks [31], attacks with modified loss functions [159], and unusual architectural solutions like feature denoising [160]). The field of adversarial training is growing rapidly. For example, a recent survey [161] describes six main groups of modifications: adversarial regularization, curriculum-based adversarial training, ensemble adversarial training, adversarial training with adaptive ε, adversarial training with semi/unsupervised learning and efficient adversarial training, besides other variations.

Initially, adversary training was focused on single-perturbation robustness [10, 121] (i.e., robustness against one particular perturbation method). Such robustness is easily and successfully achieved by incorporating the perturbed samples into the dataset. Conversely, adversarial training for multi-perturbation robustness (considering a set of possible attacks) is more challenging. One of the first works in this area [162] considers three types: l_∞, l_0 and l_2-bounded attacks. Unfortunately, adversarial training suffers from *robust overfitting*, which decreases test accuracy during training [163]. That can be partially avoided with common augmentation schemes (refer to Section 1.4.1).

3.2. Why there are no hard guarantees

Provable (or certified) defense methods can theoretically guarantee robustness against attacks under some set of conditions. The problem with such guarantees is that they are either hard to obtain (because of computational demands) or easy to avoid (because the constraints are too strict).

In study [164], the authors show that formal verification of the model is an NP-hard problem. Therefore, verifying that an ANN is simultaneously sound, complete, and able to terminate training in a reasonable time is challenging. Thus, approaches usually guarantee two of these requirements.

Certified defenses can be roughly categorized into methods based on satisfiability modulo theories (SMT), optimization and relaxation-based bounds, and randomized smoothing [165]. This categorization is not the only one possible by any means (for example,

according to taxonomy of [42], there are seven different families of certified methods), but let us stick to it for the sake of clarity.

Most optimization-based methods make the certification procedure more tractable by either developing convex relaxation or directly bounding and analyzing the network over the set of inputs [165]. These methods may be based on linear, semi-definite, partitioned, or mixed-integer programming. However, these approaches suffer from high computational costs and do not scale well enough to larger ANNs (for example, ImageNet classifiers).

The main idea of Reluplex metod [164] is to find an exact minimal perturbation distance $r(x,f)$, where x is the sample and f is the model. Thus, the classifier is safe against perturbations with vector norm less than $r(x,f)$. Some methods approximate this exact value with some trainable certificates $C(x,f)$, for example, the upper bound of adversarial loss [166, 34] or lower bound of minimal perturbation [167]. These problems can be described as linear programming problems [167] or integral inequalities [166] solved with semi-definite programming (SDP) [168]. These studies mainly consider ANNs with one hidden layer.

Safety verification approach [169] detects failures of the classifier in some neighborhood of the given sample x with a finite exhaustive search. In [170], the verification of an ANN is formulated as an optimization problem for potential perturbed inputs. In [171], the authors compute an approximation to the adversarial polytope and use this approximation as a part of the loss function.

In addition, there is a distribution (or principled) adversarial training method that combines adversarial training and provable defense together [172]. Genuinely, the model trains on the worst-case perturbation examples and its certificates are derived from the Lagrangian duality of adversarial loss.

In Interval Bound Propagation (IBP) [173], a loss is defined to minimize an upper bound on the maximum difference between any pair of logits when the input can be perturbed within an l_∞ norm-bounded ball. This method was later improved in [174] by proposing a novel initialization procedure, regularization, and batch normalization. These changes allow for faster model training. In addition, a commonly known modification named CROWN-IBP [175] is delivered by combining the fast IBP bounds and a tight linear relaxation-based bound, CROWN. Unfortunately, it was shown in [176] that IBP can not achieve sufficient certified robustness on simple datasets for any standard ReLU network.

In the mixed-integer approach, the verification of piece-wise linear neural networks is formulated as a mixed-integer program [177]. In the partition-based approach, the input set is split into parts, and the optimization problem is solved separately for each part to obtain an overall global bound [178].

Counterintuitively, tighter relaxation usually results in worse model robustness. In the study [179], the authors formulate two necessary properties for certified training: continuity and sensitivity of convex relaxations used in the training. They have also shown that improving unfavorable properties often harms other properties, revealing a complex tradeoff. Another open problem is deriving optimization-based methods for general (non-ReLU) activations [165].

It is much easier to certify Lipshitz neural networks (i.e., networks for which Lipshitz constraint holds), for example, based on the output margin [180]. Early works (like [181]) can only handle the l_2-bounded perturbations under Lipschitz constraints. For l_∞ norm, these networks only provide a negligible certified radius [182]. It was shown in [183] that standard Lipschitz ReLU networks cannot represent certain simple functions; therefore, the GroupSort network was proposed [184]. Despite some investigations, its performance is still worse than that of the above relaxation-based approaches. The training procedure of the study [185] (based on l_p-relaxation) demonstrated practical success for l_∞-bounded perturbations; however, it lacks theoretical explanations.

Another approach for certified training is randomized smoothing. Randomized smoothing [135, 186] is a probabilistic method for improving classifier robustness with deterministic robustness guarantees. It intentionally corrupts the inputs with random noise and uses predictions for the noised and actual inputs to average out any potential danger. This method includes symmetric Gaussian (e.g. [135]), non-Gaussian (e.g. [187]) and input-dependent smoothing [188]. An open problem with this approach is the exact optimization of the smoothing distribution in a manner that is scalable to large datasets such as ImageNet [165]. Some theoretical estimations for this method (e.g., [189]) show that there are certain limitations for the perturbation radius.

The general problem with certified defenses is that the requirements of these methods can be easily violated in real-life conditions. Defense approaches claim they can guard against specific attacks while failing against others [165].

4. Discussion: step back and broader view

Now that we have learned what adversarial examples are, how to find them, and why avoiding them at present is infeasible, let us step back and reflect. In this section, we overview the phenomenon of adversarial examples in general, raise unsolved questions, and present our point of view. Some of our statements, like “adversarial examples should not be called attacks on neural networks,” are debatable, and readers may disagree with us and our arguments. Our intention here is to bring the remaining issues to readers’ attention and inspire further research in this field.

4.1. Whose bias is that?

From Section 1.3, we learned that ANNs are universal approximators. Therefore, they can mimic any function, including, theoretically, the human decision-making function, in image classification. When humans classify images, they essentially map input images to corresponding labels. That procedure can be performed by a single human or through a consensus of multiple humans (to avoid spontaneous individual errors). The mapping can be viewed as a deterministic function from the image to the label space. Thus, an ANN can theoretically approximate this decision-making function by learning from labeled training data. However, ANNs are susceptible to adversarial images, as demonstrated in Section 2.1. On the other hand, humans are generally unaffected by these adversarial images and can make accurate classifications.

One probable explanation for this mismatch is that ANNs and humans perceive the input image differently, thus reacting to different features. However, we have already seen a similar explanation in Section 1.5 when we discussed the inductive bias of CNNs. As we recall, CNNs tend to rely more on texture information rather than shape information when making classifications. In contrast, humans prefer shape over texture. To address this bias, researchers have used augmentation technique [107], introducing images with different textures to the dataset. Thus, explicit bias is added to the network via training. This explicit bias also helps the network learn to focus on shape information, thus improving its robustness and accuracy. Why not do the same for adversarial images? That is how we reinvent adversarial training, presented in Section 3.1. Unfortunately, it is not a universal solution and only works against certain adversarial examples. Therefore, we can further conclude that

- algorithms that search for adversarial examples actually find implicit biases in ANNs,
- there are many such biases,
- and it is impossible to group them all into a single model (which would have allowed for building a universal defense against such a threat).

The next reasonable question is: "Why do ANNs have such a bias?" In other words, why are they vulnerable to attacks? Many hypotheses have been proposed to answer this question. Let us present some of them.

Szegedy et al. [10] blamed *extreme non-linearity* of the ANNs. They supposed that the set of adversarial examples has extremely low probabilities (thus, not observed in the test set) but is dense (thus, for any input, we can easily find adversarial examples in its neighborhood). Goodfellow et al. [121] called that assumption speculative, and on the contrary blamed *linearity* of neural networks. They have shown that adversarial examples (at least those generated with the FGSM algorithm, discussed in Section 2.1.2) are the property of high-dimensional dot products and stated that ANNs preserve this property. ANNs are designed to be "sufficiently linear" to be trained with gradient-based optimization. Gilmer et al. [190] hypothesized that adversarial examples originate in the geometry of the high-dimensional manifold. They have also demonstrated (on a simple, specific example) that in order to increase robustness to small perturbations, the test error of an ANN should be significantly decreased. The authors have also stated that "any model which misclassifies a small constant fraction of the data manifold will be vulnerable to adversarial perturbations.". Finally, they provided the theoretical upper bound for such perturbations. Schmidt et al. [191] further elaborated on this point and theoretically proven (once again for a specific simple case) that "the sample complexity of robust generalization is significantly larger than that of standard generalization". In other words, we need significantly more training data to guarantee some adversarial robustness than we need to train an ANN, which would work fine on the test set.

We will not enter that discussion. Instead of blaming one particular aspect of the ANN training process for the existence of adversarial examples, let's overview it in general. Three components are essential for ANN training:

1. ANN itself,
2. training data,
3. and training algorithm (which includes optimization objective, optimization method, etc.).

Let us consider all three of them in the context of adversarial examples.

ANN, as a universal approximator, should be able to learn a human-like decision-making function that is not vulnerable to adversarial examples. This particular, and the fact that adversarial examples can be found for all current networks regardless of their architecture, suggests that the problem lies outside ANN as a model.

Regarding the training data, most theoretical models in ML assume that the training data is sampled from some unknown distribution and that when the model is trained, it will observe other data (test data) from the same distribution. The difference between the training and test datasets distributions is known as *dataset shift* [192]. A dataset shift may exist in the context of adversarial examples because adversarial perturbations may not appear in the training set. This property is used by a defense mechanism known as Deep k-nearest neighbors [193], which aims to find out-of-distribution inputs by analyzing hidden representations of the input produced by each layer of the ANN. The intuition here is that for "normal" input, the hidden representations will be close to the hidden representations of the samples of the same class (from the test set) and far from the hidden representations of the samples of other classes. For an out-of-distribution adversarial image, this property will

not hold. However, there are three major limitations to these methods in practice:

- The first is increased computational complexity. Anomaly detection (using nearest neighbors or not) is a much more complex task than image classification.
- When we (as humans) see an adversarial example (consider Figures 9-15), we may say "I see an image of a goldfish, with some unnatural noise on it". However, we will not say "I suspect that this image is inconsistent with all the images I have seen before, so I refuse to classify it". Thus, when designing a computer vision system, we may want its behavior to align more with ours.
- Last but not least is the fact that there are in-distribution adversarial examples. Section 2.3 shows that such examples can be generated using GANs and diffusion ANNs. The study [190] also theoretically predicts that any classifier has in-distribution adversarial examples, except for the ideal one.

We should also consider the size of the training set. As discussed above, adversarial robustness requires a significant increase in the dataset size even in the simplest cases of l_∞-bounded perturbations [191] (and as we remember, perturbations may be unbounded as demonstrated in Section 2.3). Such an increase in the dataset size is almost infeasible in practice. In any case, considering the training dataset from the inductive bias perspective, we can conclude that it is not the source of any bias (if the dataset indeed represents real data distribution). The problem is that it does not contain enough data that an ANN can learn enough to model some prior knowledge about input data, which would have allowed for adversarial-robust training.

As we have seen so far, neither an ANN nor the training data are the source of the inductive bias that leads to adversarial examples. Thus, we can conclude that the source of the inductive bias should be in the training algorithm. This conclusion corresponds to that of Goodfellow et al. [121], who blamed the SGD-like algorithm for the linearity of neural networks.

However, another point of view is that bias is not induced at all. In fact, the problem with adversarial examples is that ANNs do not have the same prior information about input data that we (as humans) do (i.e., we have different inductive biases).

To support that, we would like to cite Schmidt et al. [191]: "The conventional wisdom usually is not to bias the neural network with our priors. Our work suggests that there are trade-offs with robustness here and that adding more prior information could help to learn more robust classifiers".

The last two conclusions are not necessarily contradictory. We can summarize them as follows. Training ANNs robust to adversarial perturbations will require inducing corresponding prior knowledge (i.e., bias) about the input data through the training algorithm. Such a training algorithm will be more complex when using simple SGD-like approaches. Significant research efforts would be required to understand what prior information is essential for adversarial-robust training and how to incorporate it into the training algorithm properly. We believe that these will be the future directions of adversarial-robust ANN training research.

4.2. Adversarial examples are not attacks

The "attack"-"defense" terminology is widely used in the field of adversarial deep learning. We believe that when it comes to generating adversarial examples, or making an ANN more robust this terminology is misleading.

As Section 2 demonstrates, an adversarial example does not break a neural network, it does not affect its state, and ability to recognize other examples. It just exploits a flaw that already exists in the ANN by design, to compromise an application, in which this network is applied.

From Section 3.1 we have learned that applying "defense" as a countermeasure against a specific method of generation adversarial perturbation does not guarantee the robustness of ANN to other adversarial perturbations. Moreover, it is impossible to train neural networks with complete provable robustness to all sorts of adversarial examples (see Section 3.2).

All in all, current "defense" methods do not solve the general problem of misalignment of ANN and human decision-making processes. We hope that in the future, more work will be focused in this direction and we will learn how to train ANNs that are not fooled so easily.

In our opinion, the term "attack on neural network" would be more suitable for data poisoning [24], which we mentioned briefly in the introduction. In that case an adversary adds specifically generated images to the dataset so that any ANN trained on such "poisoned" dataset will have a predefined vulnerability (e.g., react to some trigger when it is present on the image, and ignore the rest of the image). However data poisoning and methods preventing it lie in another area of research and deserve their own survey.

4.2.1. Does anyone attack neural networks with adversarial examples?

Another good reason not to call the generation of adversarial examples an "attack" is that, at present, real adversaries do not use neural networks' vulnerability to adversarial examples.

The question arises: if there is a well-known vulnerability, why is it not exploited in practice? This question was partially answered in the study of Apruzzese [194] called "Real Attackers Don't Compute Gradients". Let us briefly consider the main ideas of that study.

Unlike the researchers, actual attackers usually have little information about the attacked system and have very restricted economic and time resources. So, they

use cheap and straightforward methods to create adversarial images. For example, phishing pages can fool the phishing detectors by making small changes like distortions, masking, replacing letters, changing logotypes, etc. [194]. Obviously, attackers do not have to use gradient methods to create such images. Also, the system can be fooled by adding things the system is susceptible to like human silhouettes for autonomous driving cars [195].

Usually, ANNs do not work alone – they are parts of the systems that consist of many components, such as preprocessors, postprocessors, security checkers, etc. Thus, to fool the system, it is not necessary to fool the ANN [196]. Potentially, any error in the system can cause a fault, and attackers can find such vulnerabilities using testing techniques such as fuzzing [197] or random testing [198].

To summarize, the practical attack methods on computer vision systems differ from the adversarial example generation described in the previous sections. Real attackers use more straightforward techniques or target non-ML system components to achieve their goals.

4.3. Where are the boundaries of adversarial examples?

Future researchers of adversarial robust training would have to answer yet another hard question: Where are the boundaries of the phenomenon that we call adversarial examples?

In their study, Kotyan et al. [130] introduced the concept of *false adversarial examples*. Those are examples that even a human observer cannot recognize. Kotyan et al. have shown that such examples can be generated by l_1 and l_2-bounded perturbations (see example in Figure 20).

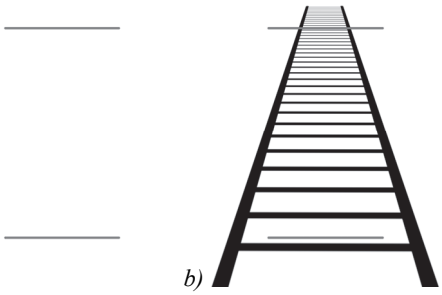
Fig. 20. An example of a false adversarial image from the study [130]. In the left image (source), we can see an airplane in the sky. In the right image, there is no plane; therefore, these are two different images from the human point of view, although the l_2 norm of the perturbation is low. Thus, the right image cannot be considered a true adversarial example

Kotyan et al. [130] suggested that detecting these false adversarial examples requires a thorough inspection. They proposed using attacks solely based on l_0 and l_∞ norm to avoid creating such samples. However, as we observed in Section 2, the variety of methods for generating adversarial examples is not limited to l_0- and l_∞-bounded. Moreover, some of them are unbounded by any norm yet preserve the main property: the ability to "fool" a network without being noticed by humans. We believe that a deeper understanding of the nature of true and false adversarial examples will help future researchers develop robust and high-quality ANNs.

4.4. Humans are not ideal recognizers

In all the previous sections, we considered a human as a reference to ANN classifiers. However, it is essential to remember that any recognizer with generalization capability has its own bias [104], and that includes humans as well. We need only a few images with some object to be able to recognize it in different images. That is because we have strong inductive bias. For example, we can distinguish an object and background (that is closely related to figure-ground perception, which plays a vital role in Gestalt psychology [199]). When recognizing an image, we are focused on the object. On the contrary, for an ANN, all an image's pixels are equal information sources.

Unfortunately, humans are not ideal recognizers as well. The way our perception works makes us vulnerable to many optical illusions. For example, M. Ponzo [200] observed that the perception of an object depends on the background. His famous illusion is illustrated in Figure 21.



a) *b)*

Fig. 21. Ponzo illusion. In the left image, there are two identical lines. In the right one, there are the same lines but with the railways-like pattern on the background; the bottom line appears to be shorter

Moreover, human perception is a complex process. For example, when looking at ambiguous images (images can provide two or more different perceptions), we switch between the alternatives but do not perceive them all simultaneously [201]. Figure 22 presents two famous examples of such images. We leave the reader with a philosophical question about how a hypothetical ideal ANN classifier should process ambiguous images.

b)

Fig. 22. Ambiguous images. "My Wife and My Mother-in-Law" by William Ely Hill (left) and "Rabbit and Duck" image, published in Fliegende Blätter in 1892 (right)

Conclusion

Adversarial examples have been a topic of interest in the computer vision research community for over a decade. Despite numerous efforts and advancements in this field, the existence of adversarial examples remains an unsolved problem.

In this survey, we focused primarily on adversarial examples against ANN classifiers. We provide a detailed examination of the discovery of adversarial examples and the evolution of methods for generating them. We explored various techniques, from simple perturbations to more complex and advanced approaches, such as the recently proposed generative models. We also briefly discuss countermeasures suggested to mitigate the impact of adversarial examples. However, these measures are insufficient in providing robust defense against a whole variety of adversarial "attacks".

Through our analysis, we identified the fundamental concept of inductive bias in ANNs as a key factor contributing to the vulnerability of these classifiers to adversarial perturbations. While both humans and ANNs exhibit biases (as it is a fundamental property of all models, capable of generalization), they differ in nature, which poses a significant challenge in addressing the problem of adversarial examples.

By providing a detailed overview of the research on adversarial examples, we hope that this survey will serve as a valuable resource for future researchers who attempt to reconsider and tackle this persistent problem. We hope that this work will inspire the development of new ANNs that are more resilient to adversarial perturbations and ultimately enhance the security and reliability of computer vision systems.

References

[1] Lynchenko A, Sheshkus A, Arlazarov VL. Document image recognition algorithm based on similarity metric robust to projective distortions for mobile devices. Proc SPIE 2019; 11041: 110411K. DOI: 10.1117/12.2523152.

[2] Demidova YA, Sheshkus AV, Arlazarov VL. Method for training a compact discrete neural network descriptor. Proc SPIE 2022; 12084: 1208411. DOI: 10.1117/12.2623166.

[3] Bezmaternykh PV, Ilin DA, Nikolaev DP. U-Net-bin: hacking the document image binarization contest. Computer Optics 2019; 43(5): 825-832. DOI: 10.18287/2412-6179-2019-43-5-825-832.

[4] Shariff W, Farooq MA, Lemley J, Corcoran P. Event-based yolo object detection: proof of concept for forward perception system. Proc SPIE 2023; 12701: 127010A. DOI: 10.1117/12.2679341.

[5] Ilin DA. Fast words boundaries localization in text fields for low quality document images. Proceedings of the Institute for Systems Analysis Russian Academy of Sciences (ISA RAS) 2018; 68(S1): 192-198. DOI: 10.14357/20790279180522.

[6] Ye M, Shen J, Lin G, Xiang T, Shao L, Hoi SCH. Deep learning for person re-identification: A survey and outlook. IEEE Trans Pattern Anal Mach Intell 2022; 44(6): 2872-2893. DOI: 10.1109/TPAMI.2021.3054775.

[7] Andreeva EI, Arlazarov VV, Gayer AV, Dorokhov EP, Sheshkus AV, Slavin OA. Document recognition method based on convolutional neural network invariant to 180 degree rotation angle. Journal of Information Technologies and Computing Systems 2019; 4: 87-93. DOI: 10.14357/20718632190408.

[8] Arlazarov VV, Andreeva EI, Bulatov KB, Nikolaev DP, Petrova OO, Savelev BI, Slavin OA. Document image analysis and recognition: a survey. Computer Optics 2022; 46(4): 567-589. DOI: 10.18287/2412-6179-CO-1020.

[9] Yang B, Cao X, Xiong K, Yuen C, Guan YL, Leng S, Qian L, Han Z. Edge intelligence for autonomous driving in 6g wireless system: Design challenges and solutions. IEEE Wirel Commun 2021; 28(2): 40-47. DOI: 10.1109/MWC.001.2000292.

[10] Szegedy C, Zaremba W, Sutskever I, Bruna J, Erhan D, Goodfellow I, Fergus R. Intriguing properties of neural networks. arXiv Preprint. 2013. Source: <https://arxiv.org/abs/1312.6199>. DOI: 10.48550/arXiv.1312.6199.

[11] T. B. Brown, D. Mané, A. Roy, M. Abadi, J. Gilmer, Adversarial patch. arXiv Preprint. 2017. Source: <https://arxiv.org/abs/1312.6199>. DOI: 10.48550/arXiv.1712.09665

[12] Lin C-S, Hsu C-Y, Chen P-Y, Yu C-M. Real-world adversarial examples via makeup. 2022 IEEE Int Conf on Acoustics, Speech and Signal Processing (ICASSP) 2022: 2854-2858. DOI: 10.1109/ICASSP43922.2022.9747469.

[13] Hu S, Liu X, Zhang Y, Li M, Zhang LY, Jin H, Wu L. Protecting facial privacy: Generating adversarial identity masks via style-robust makeup transfer. IEEE/CVF Conf on Computer Vision and Pattern Recognition (CVPR) 2022: 15014-15023. DOI: 10.1109/CVPR52688.2022.01459.

[14] Zolfi A, Avidan S, Elovici Y, Shabtai A. Adversarial mask: Real-world universal adversarial attack on face recognition models. In Book: Amini MR, Canu S, Fischer A, Guns T, Kralj NP, Tsoumakas G, eds. Machine learning and knowledge discovery in databases. European Conference, ECML PKDD 2022, Grenoble, France, September 19-23, 2022, Proceedings, Part III. Cham, Switzerland: Springer Nature Switzerland AG; 2023, pp. 304-320. DOI: 10.1007/978-3-031-26409-2_19.

[15] Zhou Z, Tang D, Wang X, Han W, Liu X, Zhang K. Invisible mask: Practical attacks on face recognition with infrared, arXiv Preprint. 2018. Source: <https://arxiv.org/abs/1803.04683>. DOI: 10.48550/arXiv.1803.04683.

[16] Wu Z, Lim S-N, Davis LS, Goldstein T. Making an invisibility cloak: Real world adversarial attacks on object detectors, In Book: Vedaldi A, Bischof H, Brox T, Frahm JM, eds. Computer vision – ECCV 2020. 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part IV. Cham, Switzerland: Springer Nature Switzerland AG; 2020: 1-17. DOI: 10.1007/978-3-030-58548-8_1.

[17] Thys S, Van Ranst W, Goedemé T. Fooling automated surveillance cameras: Adversarial patches to attack person detection. 2019 IEEE/CVF Conf on Computer Vision and Pattern Recognition Workshops (CVPRW) 2019: 49-55. DOI: 10.1109/CVPRW.2019.00012.

[18] Zhong Y, Liu X, Zhai D, Jiang J, Ji X. Shadows can be dangerous: Stealthy and effective physical-world

adversarial attack by natural phenomenon. 2022 IEEE/CVF Conf on Computer Vision and Pattern Recognition (CVPR) 2022: 15345-15354. DOI: 10.1109/CVPR52688.2022.01491.

[19] Hong S, Davinroy M, Kaya Y, Locke SN, Rackow I, Kulda K, Dachman-Soled D, Dumitraş T. Security analysis of deep neural networks operating in the presence of cache side-channel attacks. arXiv Preprint. 2018. Source: <https://arxiv.org/abs/1810.03487>. DOI: 10.48550/arXiv.1810.03487.

[20] Oh SJ, Schiele B, Fritz M. Towards reverse-engineering black-box neural networks. In Book: Samek W, Montavon G, Vedaldi A, Hansen L, Müller KR, eds. Explainable AI: Interpreting, explaining and visualizing deep learning. Cham, Switzerland: Springer Nature Switzerland AG; 2019: 121-144. DOI: 10.1007/978-3-030-28954-6_7.

[21] Chmielewski L, Weissbart L. On reverse engineering neural network implementation on GPU. In Book: Zhou J, et al, eds. Applied cryptography and network security workshops. ACNS 2021 Satellite Workshops, AIBlock, AIHWS, AIoTS, CIMSS, Cloud S&P, SCI, SecMT, and SiMLA, Kamakura, Japan, June 21-24, 2021, Proceedings. Cham, Switzerland: Springer Nature Switzerland AG; 2021: 96-113. DOI: 10.1007/978-3-030-81645-2_7.

[22] Goldblum M, Tsipras D, Xie C, Chen X, Schwarzschild A, Song D, Madry A, Li B, Goldstein T. Dataset security for machine learning: Data poisoning, backdoor attacks, and defenses. IEEE Trans Pattern Anal Mach Intell 2023; 45(2): 1563-1580. DOI: 10.1109/TPAMI.2022.3162397.

[23] Gu T, Dolan-Gavitt B, Garg S. BadNets: Identifying vulnerabilities in the machine learning model supply chain. arXiv Preprint. 2017. Source: <https://arxiv.org/abs/1708.06733>. DOI: 10.48550/arXiv.1708.06733.

[24] Shafahi A, Huang WR, Najibi M, Suciu O, Studer C, Dumitras T, Goldstein T. Poison frogs! targeted clean-label poisoning attacks on neural networks. NIPS'18: Proc 32nd Int Conf on Neural Information Processing Systems 2018: 6106-6116.

[25] Fredrikson M, Jha S, Ristenpart T. Model inversion attacks that exploit confidence information and basic countermeasures. CCS '15: Proc 22nd ACM SIGSAC Conf on Computer and Communications Security 2015: 1322-1333. DOI: 10.1145/2810103.2813677.

[26] Wang Y, Deng J, Guo D, Wang C, Meng X, Liu H, Ding C, Rajasekaran S. SAPAG: A self-adaptive privacy attack from gradients. arXiv Preprint. 2020. Source: <https://arxiv.org/abs/2009.06228>. DOI: 10.48550/arXiv.2009.06228.

[27] Akhtar N, Mian A. Threat of adversarial attacks on deep learning in computer vision: A survey. IEEE Access 2018; 6: 14410-14430. DOI: 10.1109/ACCESS.2018.2807385.

[28] Machado GR, Silva E, Goldschmidt RR. Adversarial machine learning in image classification: A survey toward the defender's perspective. ACM Comput Surv (CSUR) 2021; 55(1): 8. DOI: 10.1145/3485133.

[29] Long T, Gao Q, Xu L, Zhou Z. A survey on adversarial attacks in computer vision: Taxonomy, visualization and future directions. Computers & Security 2022; 121: 102847. DOI: 10.1016/j.cose.2022.102847.

[30] Ren K, Zheng T, Qin Z, Liu X. Adversarial attacks and defenses in deep learning. Engineering 2020; 6(3): 346-360. DOI: 10.1016/j.eng.2019.12.012.

[31] Madry A, Makelov A, Schmidt L, Tsipras D, Vladu A. Towards deep learning models resistant to adversarial attacks. arXiv Preprint. 2017. Source: <https://arxiv.org/abs/1706.06083>. DOI: 10.48550/arXiv.1706.06083.

[32] Liu X, Cheng M, Zhang H, Hsieh C-J. Towards robust neural networks via random self-ensemble. In Book: Ferrari V, Hebert M, Sminchisescu C, Weiss Y, eds. Computer Vision – ECCV 2018. 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part VII. Cham, Switzerland: Springer Nature Switzerland AG; 2018: 381-397. DOI: 10.1007/978-3-030-01234-2_23.

[33] Athalye A, Carlini N, Wagner D. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. Int Conf on Machine Learning (PMLR) 2018: 274-283.

[34] Wong E, Kolter Z. Provable defenses against adversarial examples via the convex outer adversarial polytope. Int Conf on Machine Learning (PMLR) 2018: 5286-5295.

[35] Zhang X, Zhang X, Sun M, Zou X, Chen K, Yu N. Imperceptible black-box waveform-level adversarial attack towards automatic speaker recognition. Complex & Intelligent Systems 2023; 9: 65-79. DOI: 10.1007/s40747-022-00782-x.

[36] Kwon H, Lee S. Ensemble transfer attack targeting text classification systems. Computers & Security 2022; 117: 102695. DOI: 10.1016/j.cose.2022.102695.

[37] Mo K, Tang W, Li J, Yuan X. Attacking deep reinforcement learning with decoupled adversarial policy. IEEE Trans Dependable Secure Comput 2023; 20(1): 758-768. DOI: 10.1109/TDSC.2022.3143566.

[38] Zhou X, Liang W, Li W, Yan K, Shimizu S, Wang KI-K. Hierarchical adversarial attacks against graph-neural-network-based iot network intrusion detection system. IEEE Internet Things J 2022; 9(12): 9310-9319. DOI: 10.1109/JIOT.2021.3130434.

[39] Hornik K, Stinchcombe M, White H. Multilayer feedforward networks are universal approximators. Neural Netw 1989; 2(5): 359-366. DOI: 10.1016/0893-6080(89)90020-8.

[40] Akhtar N, Mian A, Kardan N, Shah M. Advances in adversarial attacks and defenses in computer vision: A survey. IEEE Access 2021; 9: 155161-155196. DOI: 10.1109/ACCESS.2021.3127960.

[41] Mi J-X, Wang X-D, Zhou L-F, Cheng K. Adversarial examples based on object detection tasks: A survey. Neurocomputing 2023; 519: 114-126. DOI: 10.1016/j.neucom.2022.10.046.

[42] Serban A, Poll E, Visser J. Adversarial examples on object recognition: A comprehensive survey. ACM Comput Surv (CSUR) 2020; 53(3): 66. DOI: 10.1145/3398394.

[43] McCulloch WS, Pitts W. A logical calculus of the ideas immanent in nervous activity. Bull Math Biophys 1943; 5: 115-133. DOI: 10.1007/BF02478259.

[44] Rosenblatt F. The perceptron: A probabilistic model for information storage and organization in the brain. Psychol Rev 1958; 65(6): 386. DOI: 10.1037/H0042519.

[45] Gholami A, Kim S, Dong Z, Yao Z, Mahoney MW, Keutzer K. A survey of quantization methods for efficient neural network inference. arXiv Preprint. 2021. Source:<https://arxiv.org/abs/2103.13630>. DOI: 10.48550/arXiv.2103.13630.

[46] H. Chen, Y. Wang, C. Xu, B. Shi, C. Xu, Q. Tian, C. Xu, AdderNet: Do we really need multiplications in

deep learning? 2020 IEEE/CVF Conf on Computer Vision and Pattern Recognition (CVPR) 2020: 1465-1477. DOI: 10.1109/CVPR42600.2020.00154.

[47] Limonova EE, Alfonso DM, Nikolaev DP, Arlazarov VV. Bipolar morphological neural networks: Gate-efficient architecture for computer vision. IEEE Access 2021; 9: 97569-97581. DOI: 10.1109/ACCESS.2021.3094484.

[48] Paszke A, Gross S, Massa F, et al. PyTorch: An imperative style, high-performance deep learning library. Proc 33rd Int Conf on Neural Information Processing Systems 2019: 8026-8037.

[49] Keras. Simple. Flexible. Powerfull. 2024. Source: <https://keras.io>.

[50] Li Y, Yuan Y. Convergence analysis of two-layer neural networks with relu activation. NIPS'17: Proc 31st Int Conf on Neural Information Processing Systems 2017: 597-607.

[51] Zou D, Cao Y, Zhou D, Gu Q. Gradient descent optimizes over-parameterized deep relu networks. Mach Learn 2020; 109: 467-492. DOI: 10.1007/s10994-019-05839-6.

[52] Bahri Y, Dyer E, Kaplan J, Lee J, Sharma U. Explaining neural scaling laws. arXiv Preprint. 2021. Source: <https://arxiv.org/abs/2102.06701>. DOI: 10.48550/arXiv.2102.06701.

[53] Jacot A, Gabriel F, Hongler C. Neural tangent kernel: Convergence and generalization in neural networks. NIPS'18: Proc 32nd Int Conf on Neural Information Processing Systems 2018: 8580-8589.

[54] Bachmann G, Anagnostidis S, Hofmann T. Scaling MLPs: A tale of inductive bias. arXiv Preprint. 2023. Source: <https://arxiv.org/abs/2306.13575>. DOI: 10.48550/arXiv.2306.13575.

[55] Ding S, Li H, Su C, Yu J, Jin F. Evolutionary artificial neural networks: A review. Artif Intell Rev 2013; 39(3): 251-260. DOI: 10.1007/s10462-011-9270-6.

[56] Zhou H, Lan J, Liu R, Yosinski J. Deconstructing lottery tickets: Zeros, signs, and the supermask. Proc 33rd Int Conf on Neural Information Processing Systems 2019: 3597-3607.

[57] Rumelhart DE, Hinton GE, Williams RJ. Learning representations by back-propagating errors. Nature 1986; 323(6088): 533-536. DOI: 10.1038/323533a0.

[58] Duchi J, Hazan E, Singer Y. Adaptive subgradient methods for online learning and stochastic optimization. J Mach Learn Res 2011; 12(7): 2121-2159. DOI: 10.5555/1953048.2021068.

[59] Tieleman T, Hinton G, et al. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning 2012; 4(2): 26-31.

[60] Kingma DP, Ba J. Adam: A method for stochastic optimization. arXiv Preprint. 2014. Source: <https://arxiv.org/abs/1412.6980>. DOI: 10.48550/arXiv.1412.6980.

[61] Cybenko G. Approximation by superpositions of a sigmoidal function. Mathematics of Control, Signals and Systems 1989; 2(4): 303-314. DOI: 10.1007/BF02551274.

[62] Kolmogorov AN. On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition. Doklady Akademii Nauk SSSR 1957; 114(5): 953-956.

[63] Hornik K. Approximation capabilities of multilayer feedforward networks. Neural Netw 1991; 4(2): 251-257. DOI: 10.1016/0893-6080(91)90009-T.

[64] Tabuada P, Gharesifard B. Universal approximation power of deep residual neural networks through the lens of control. IEEE Trans Autom Control 2023; 68(5): 2715-2728. DOI: 10.1109/TAC.2022.3190051.

[65] Cai Y. Achieve the minimum width of neural networks for universal approximation. arXiv Preprint. 2022. Source: <https://arxiv.org/abs/2209.11395>. DOI: 10.48550/arXiv.2209.11395.

[66] Hoffmann J, Borgeaud S, Mensch A, et al., Training compute-optimal large language models. arXiv Preprint. 2022. Source: <https://arxiv.org/abs/2203.15556>. DOI: 10.48550/arXiv.2203.15556.

[67] Alam M, Samad MD, Vidyaratne L, Glandon A, Iftekharuddin KM. Survey on deep neural networks in speech and vision systems. Neurocomputing 2020; 417: 302-321. DOI: 10.1016/j.neucom.2020.07.053.

[68] Santos CFGD, Papa JP. Avoiding overfitting: A survey on regularization methods for convolutional neural networks. ACM Comput Surv (CSUR) 2022; 54(10s): 213. DOI: 10.1145/3510413.

[69] Xu Y, Goodacre R. On splitting training and validation set: A comparative study of cross-validation, bootstrap and systematic sampling for estimating the generalization performance of supervised learning. J Anal Test 2018; 2(3): 249-262. DOI: 10.1007/s41664-018-0068-2.

[70] Bejani MM, Ghatee M. A systematic review on overfitting control in shallow and deep neural networks. Artificial Intelligence Review 2021; 54(8): 6391-6438. DOI: 10.1007/s10462-021-09975-1.

[71] Blalock D, Gonsales Ortiz JJ, Frankle J, Guttag J. What is the state of neural network pruning? Proceedings of Machine Learning and Systems 2020; 2: 129-146.

[72] Ren P, Xiao Y, Chang X, Huang P-Y, Li Z, Chen X, Wang X. A comprehensive survey of neural architecture search: Challenges and solutions. ACM Comput Surv (CSUR) 2021; 54(4): 76. DOI: 10.1145/3447582.

[73] Zoph B, Le QV. Neural architecture search with reinforcement learning. arXiv Preprint. 2016. Source: <https://arxiv.org/abs/1611.01578>. DOI: 10.48550/arXiv.1611.01578.

[74] Real E, Moore S, Selle A, Saxena S, Suematsu YL, Tan J, Le QV, Kurakin A. Large-scale evolution of image classifiers, in: International conference on machine learning. ICML'17: Proc 34th Int Conf on Machine Learning 2017; 70: 2902-2911.

[75] Zoph B, Vasudevan V, Shlens J, Le QV. Learning transferable architectures for scalable image recognition. 2018 IEEE/CVF Conf on Computer Vision and Pattern Recognition (CVPR) 2018: 8697-8710. DOI: 10.1109/CVPR.2018.00907.

[76] Liu H, Simonyan K, Yang Y. DARTS: Differentiable architecture search. arXiv Preprint. 2018. Source: <https://arxiv.org/abs/1806.09055>. DOI: 10.48550/arXiv.1806.09055.

[77] Yun S, Han D, Oh SJ, Chun S, Choe J, Yoo Y. CutMix: Regularization strategy to train strong classifiers with localizable features. 2019 IEEE/CVF Int Conf on Computer Vision (ICCV) 2019: 6022-6031. DOI: 10.1109/ICCV.2019.00612.

[78] Hendrycks D, Mu N, Cubuk ED, Zoph B, Gilmer J, Lakshminarayanan B. AugMix: A simple data processing method to improve robustness and uncertainty. arXiv Preprint. 2019. Source: <https://arxiv.org/abs/1912.02781>. DOI: 10.48550/arXiv.1912.02781.

[79] Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R. Dropout: a simple way to prevent neural networks from overfitting. J Mach Learn Res 2014; 15(1): 1929-1958. DOI: 10.5555/2627435.2670313.
[80] Ghiasi G, Lin T-Y, Le QV. DropBlock: A regularization method for convolutional networks. NIPS'18: Proc 32nd Int Conf on Neural Information Processing Systems 2018: 10750-10760.
[81] Lu Z, Xu C, Du B, Ishida T, Zhang L, Sugiyama M. LocalDrop: A hybrid regularization for deep neural networks. IEEE Trans Pattern Anal Mach Intell 2021; 44(7): 3590-3601. DOI: 10.1109/TPAMI.2021.3061463.
[82] Verma V, Lamb A, Beckham C, Najafi A, Mitliagkas I, Lopez-Paz D, Bengio Y. Manifold mixup: Better representations by interpolating hidden states. International Conference on Machine Learning (ICML) 2019: 6438-6447.
[83] Szegedy C, Vanhoucke V, Ioffe S, Shlens J, Wojna Z. Rethinking the inception architecture for computer vision. 2016 IEEE Conf on Computer Vision and Pattern Recognition (CVPR) 2016: 2818-2826. DOI: 10.1109/CVPR.2016.308.
[84] Li W, Dasarathy G, Berisha V. Regularization via structural label smoothing. 23rd Int Conf on Artificial Intelligence and Statistics 2020: 1453-1463.
[85] Moradi R, Berangi R, Minaei B. A survey of regularization strategies for deep models. Artif Intell Rev 2020; 53(6): 3947-3986. DOI: 10.1007/s10462-019-09784-7.
[86] Zollanvari A, James AP, Sameni R. A theoretical analysis of the peaking phenomenon in classification. J Classif 2020; 37: 421-434. DOI: 10.1007/s00357-019-09327-3.
[87] Poggio T, Mhaskar H, Rosasco L, Miranda B, Liao Q. Why and when can deep-but not shallow-networks avoid the curse of dimensionality: A review. Int J Autom Comput 2017; 14(5): 503-519. DOI: 10.1007/s11633-017-1054-2.
[88] Beyer K, Goldstein J, Ramakrishnan R, Shaft U. When is “nearest neighbor” meaningful? In Book: Beeri C, Buneman P, eds. Database Theory – ICDT'99. 7th International Conference, Jerusalem, Israel, January 10-12, 1999, Proceedings. Berlin, Heidelberg, New York: Springer-Verlag; 1999: 217-235. DOI: 10.1007/3-540-49257-7_15.
[89] Nicolau M, McDermott J, et al. Learning neural representations for network anomaly detection. IEEE Trans Cybern 2018; 49(8): 3074-3087. DOI: 10.1109/TCYB.2018.2838668.
[90] Chattopadhyay N, Chattopadhyay A, Gupta SS, Kasper M. Curse of dimensionality in adversarial examples. 2019 Int Joint Conf on Neural Networks (IJCNN) 2019: 1-8. DOI: 10.1109/IJCNN.2019.8851795.
[91] Basodi S, Ji C, Zhang H, Pan Y. Gradient amplification: An efficient way to train deep neural networks. Big Data Mining and Analytics 2020; 3(3): 196-207. DOI: 10.26599/BDMA.2020.9020004.
[92] He K, Zhang X, Ren S, Sun J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. IEEE Int Conf on Computer Vision 2015: 1026-1034. DOI: 10.1109/ICCV.2015.123.
[93] Krizhevsky A, Sutskever I, Hinton GE. ImageNet classification with deep convolutional neural networks. NIPS'12: Proc 25th Int Conf on Neural Information Processing Systems 2012; 1: 1097-1105.
[94] Ioffe S, Szegedy C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. ICML'15: Proc 32nd Int Conf on International Conference on Machine Learning 2015; 37: 448-456.
[95] He K, Zhang X, Ren S, Sun J. Deep residual learning for image recognition. 2016 IEEE Conf on Computer Vision and Pattern Recognition (CVPR) 2016: 770-778. DOI: 10.1109/CVPR.2016.90.
[96] Huang G, Liu Z, Van Der Maaten L, Weinberger KQ. Densely connected convolutional networks. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) 2017: 4700-4708. DOI: 10.1109/CVPR.2017.243.
[97] Hanin B. Which neural net architectures give rise to exploding and vanishing gradients? NIPS'18: Proc 32nd Int Conf on Neural Information Processing Systems 2018: 580-589.
[98] Arpit D, Campos V, Bengio Y. How to initialize your network? Robust initialization for WeightNorm & ResNets. Proc 33rd Int Conf on Neural Information Processing Systems 2019: 10902-10911.
[99] Loshchilov I, Hutter F. Decoupled weight decay regularization. arXiv Preprint. 2017. Source: <https://arxiv.org/abs/1711.05101>. DOI: 10.48550/arXiv.1711.05101.
[100] Desai CG. Comparative analysis of optimizers in deep neural networks. International Journal of Innovative Science and Research Technology 2020; 5(10): 959-962.
[101] Nakamura K, Derbel B, Won K-J, Hong B-W. Learning-rate annealing methods for deep neural networks. Electronics 2021; 10(16): 2029. DOI: 10.3390/electronics10162029.
[102] Loshchilov I, Hutter F. SGDR: Stochastic gradient descent with warm restarts. arXiv Preprint. 2016. Source: <https://arxiv.org/abs/1608.03983>. DOI: 10.48550/arXiv.1608.03983.
[103] Yang J, Shen X, Xing J, Tian X, Li H, Deng B, Huang J, Hua X-s. Quantization networks. 2019 IEEE/CVF Conf on Computer Vision and Pattern Recognition (CVPR) 2019: 7308-7316. DOI: 10.1109/CVPR.2019.00748.
[104] Mitchell TM. The need for biases in learning generalizations. Rutgers CS Tech Report CBM-TR-117. New Brunswick, NJ: Rutgers University; 1980. Source: <http://dml.cs.byu.edu/cgc/docs/mldm_tools/Reading/Need%20for%20Bias.pdf>.
[105] Gordon DF, Desjardins M. Evaluation and selection of biases in machine learning. Machine Learning 1995; 20: 5-22. DOI: 10.1023/A:1022630017346.
[106] Lecun Y, Bottou L, Bengio Y, Haffner P. Gradient-based learning applied to document recognition. Proc IEEE 1998; 86(11): 2278-2324. DOI: 10.1109/5.726791.
[107] Geirhos R, Rubisch P, Michaelis C, Bethge M, Wichmann FA, Brendel W. ImageNet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness. Int Conf on Learning Representations 2019. Source: <https://openreview.net/forum?id=Bygh9j09KX>.
[108] Rao Y, Zhao W, Zhu Z, Lu J, Zhou J. Global filter networks for image classification. 35th Conf on Neural Information Processing Systems (NeurIPS 2021) 2021: 980-993.
[109] Sheshkus A, Ingacheva A, Arlazarov V, Nikolaev D. HoughNet: Neural network architecture for vanishing points detection. 2019 Int Conf on Document Analysis and Recognition (ICDAR) 2020: 844-849. DOI: 10.1109/ICDAR.2019.00140.

[110] Dosovitskiy A, Beyer L, Kolesnikov A, Weissenborn D, Zhai X, Unterthiner T, Dehghani M, Minderer M, Heigold G, Gelly S, Uszkoreit J, Houlsby N. An image is worth 16x16 words: Transformers for image recognition at scale. Int Conf on Learning Representations 2021. Source: <https://openreview.net/forum?id=YicbFdNTTy>.

[111] Tolstikhin IO, Houlsby N, Kolesnikov A, et al. MLP-mixer: An all-MLP architecture for vision. 35th Conf on Neural Information Processing Systems (NeurIPS 2021) 2021: 24261-24272.

[112] Liu Y, Sangineto E, Bi W, Sebe N, Lepri B, Nadai M. Efficient training of visual transformers with small datasets. NIPS'21: Proc 35th Int Conf on Neural Information Processing Systems 2021: 23818-23830. Source: <https://proceedings.neurips.cc/paper_files/paper/2021/file/c81e155d85dae5430a8cee6f2242e82c-Paper.pdf>.

[113] Morrison K, Gilby B, Lipchak C, Mattioli A, Kovashka A. Exploring corruption robustness: Inductive biases in vision transformers and MLP-mixers. arXiv Preprint. 2021. Source: <https://arxiv.org/abs/2106.13122>. DOI: 10.48550/arXiv.2106.13122.

[114] Feng P, Tang Z. A survey of visual neural networks: Current trends, challenges and opportunities. Multimed Syst 2023; 29(2): 693-724. DOI: 10.1007/s00530-022-01003-8.

[115] Zhang J, Sang J, Zhao X, Huang X, Sun Y, Hu Y. Adversarial privacy-preserving filter. MM '20: Proc 28th ACM Int Conf on Multimedia 2020: 1423-1431. DOI: 10.1145/3394171.3413906.

[116] Chen X, Gao X, Zhao J, Ye K, Xu C-Z. AdvDiffuser: Natural adversarial example synthesis with diffusion models. 2023 IEEE/CVF Int Conf on Computer Vision (ICCV) 2023: 4562-4572. DOI: 10.1109/ICCV51070.2023.00421.

[117] Metzen JH, Kumar MC, Brox T, Fischer V. Universal adversarial perturbations against semantic image segmentation. 2017 IEEE Int Conf on Computer Vision (ICCV) 2017: 2755-2764. DOI: 10.1109/ICCV.2017.300.

[118] Tan H, Wang L, Zhang H, Zhang J, Shafiq M, Gu Z. Adversarial attack and defense strategies of speaker recognition systems: A survey. Electronics 2022; 11(14): 2183. DOI: 10.3390/electronics11142183.

[119] Zhang WE, Sheng QZ, Alhazmi A, Li C. Adversarial attacks on deep-learning models in natural language processing: A survey. ACM Transactions on Intelligent Systems and Technology (TIST) 2020; 11(3): 24. DOI: 10.1145/3374217.

[120] Deng J, Dong W, Socher R, Li L-J, Li K, Fei-Fei L. ImageNet: A large-scale hierarchical image database, 2009 IEEE Conf on Computer Vision and Pattern Recognition 2009: 248-255. DOI: 10.1109/CVPR.2009.5206848.

[121] Goodfellow IJ, Shlens J, Szegedy C. Explaining and harnessing adversarial examples. arXiv Preprint. 2014. Source: <https://arxiv.org/abs/1412.6572>. DOI: 10.48550/arXiv.1412.6572.

[122] Nicolae M-I, Sinn M, Tran MN, et al. Adversarial robustness toolbox v1.0.0. arXiv Preprint. 2018. Source: <https://arxiv.org/abs/1807.01069>. DOI: 10.48550/arXiv.1807.01069.

[123] Kurakin A, Goodfellow I, Bengio S. Adversarial machine learning at scale. arXiv Preprint. 2016. Source: <https://arxiv.org/abs/1611.01236>. DOI: 10.48550/arXiv.1611.01236.

[124] Croce F, Hein M. Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks. ICML'20: Proc 37th Int Conf on Machine Learning 2020: 2206-2216.

[125] Yamamura K, Sato H, Tateiwa N, Hata N, Mitsutake T, Oe I, Ishikura H, Fujisawa K. Diversified adversarial attacks based on conjugate gradient method. Proc 39th Int Conf on Machine Learning (ICML 2022) 2022: 24872-24894.

[126] Papernot N, McDaniel P, Jha S, Fredrikson M, Celik ZB, Swami A. The limitations of deep learning in adversarial settings. 2016 IEEE European Symposium on Security and Privacy (EuroS&P) 2016: 372-387. DOI: 10.1109/EuroSP.2016.36.

[127] Carlini N, Wagner D. Towards evaluating the robustness of neural networks. 2017 IEEE Symposium on Security and Privacy (SP) 2017: 39-57. DOI: 10.1109/SP.2017.49.

[128] Su J, Vargas DV, Sakurai K. One pixel attack for fooling deep neural networks. IEEE Trans Evol Comput 2019; 23(5): 828-841. DOI: 10.1109/TEVC.2019.2890858.

[129] Storn R, Price K. Differential evolution – A simple and efficient heuristic for global optimization over continuous spaces. J Glob Optim 1997; 11: 341-359. DOI: 10.1023/A:1008202821328.

[130] Kotyan S, Vargas DV. Adversarial robustness assessment: Why in evaluation both L_0 and L_∞ attacks are necessary. PloS One 2022; 17(4): e0265723. DOI: 10.1371/journal.pone.0265723.

[131] Moosavi-Dezfooli S-M, Fawzi A, Frossard P. DeepFool: a simple and accurate method to fool deep neural networks. 2016 IEEE Conf on Computer Vision and Pattern Recognition (CVPR) 2016: 2574-2582. DOI: 10.1109/CVPR.2016.282.

[132] Jang U, Wu X, Jha S. Objective metrics and gradient descent algorithms for adversarial examples in machine learning. ACSAC '17: Proc 33rd Annual Computer Security Applications Conf 2017: 262-277. DOI: 10.1145/3134600.3134635.

[133] Papernot N, McDaniel P, Wu X, Jha S, Swami A. Distillation as a defense to adversarial perturbations against deep neural networks. 2016 IEEE Symposium on Security and Privacy (SP) 2016: 582-597. DOI: 10.1109/SP.2016.41.

[134] Ghiasi A, Shafahi A, Goldstein T. Breaking certified defenses: Semantic adversarial examples with spoofed robustness certificates. arXiv Preprint. 2020. Source: <https://arxiv.org/abs/2003.08937>. DOI: 10.48550/arXiv.2003.08937.

[135] Lecuyer M, Atlidakis V, Geambasu R, Hsu D, Jana S. Certified robustness to adversarial examples with differential privacy. 2019 IEEE Symposium on Security and Privacy (SP) 2019: 656-672. DOI: 10.1109/SP.2019.00044.

[136] Croce F, Hein M. Minimally distorted adversarial examples with a fast adaptive boundary attack. ICML'20: Proc 37th Int Conf on Machine Learning 2020: 2196-2205.

[137] Andriushchenko M, Croce F, Flammarion N, Hein M. Square attack: a query-efficient black-box adversarial attack via random search. In Book: Vedaldi A, Bischof H, Brox T, Frahm J-M, eds. Computer Vision – ECCV 2020. 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXIII. Cham, Switzerland: Springer Nature Switzerland AG; 2020: 484-501. DOI: 10.1007/978-3-030-58592-1_29.

[138] How to train state-of-the-art models using TorchVision's latest primitives. 2023. Source: <https://pytorch.org/blog/how-to-train-state-of-the-art-models-using-torchvision-latest-primitives>.
[139] Moosavi-Dezfooli S-M, Fawzi A, Fawzi O, Frossard P. Universal adversarial perturbations. 2017 IEEE Conf on Computer Vision and Pattern Recognition (CVPR) 2017: 1765-1773. DOI: 10.1109/CVPR.2017.17.
[140] Athalye A, Engstrom L, Ilyas A, Kwok K. Synthesizing robust adversarial examples. Int Conf on Machine Learning 2018: 284-293.
[141] Brendel W, Rauber J, Bethge M. Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. arXiv Preprint. 2017. Source: <https://arxiv.org/abs/1712.04248>. DOI: 10.48550/arXiv.1712.04248.
[142] Feng R, Mangaokar N, Chen J, Fernandes E, Jha S, Prakash A. GRAPHITE: Generating automatic physical examples for machine-learning attacks on computer vision systems. 2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P) 2022: 664-683. DOI: 10.1109/EuroSP53844.2022.00047.
[143] Xie C, Zhang Z, Zhou Y, Bai S, Wang J, Ren Z, Yuille AL. Improving transferability of adversarial examples with input diversity. 2019 IEEE/CVF Conf on Computer Vision and Pattern Recognition (CVPR) 2019: 2730-2739. DOI: 10.1109/CVPR.2019.00284.
[144] Wang X, He X, Wang J, He K. Admix: Enhancing the transferability of adversarial attacks. 2021 IEEE/CVF Int Conf on Computer Vision (ICCV) 2021: 16158-16167. DOI: 10.1109/ICCV48922.2021.01585.
[145] Zhang H, Cisse M, Dauphin YN, Lopez-Paz D. mixup: Beyond empirical risk minimization. Int Conf on Learning Representations 2018. Source: <https://openreview.net/forum?id=r1Ddp1-Rb>.
[146] Wu W, Su Y, Chen X, Zhao S, King I, Lyu MR, Tai Y-W. Boosting the transferability of adversarial samples via attention. 2020 IEEE/CVF Conf on Computer Vision and Pattern Recognition (CVPR) 2020: 1161-1170. DOI: 10.1109/CVPR42600.2020.00124.
[147] Zhao Z, Dua D, Singh S. Generating natural adversarial examples. Int Conf on Learning Representations 2018. Source: <https://openreview.net/forum?id=H1BLjgZCb>.
[148] Goodfellow I, Pouget-Abadie J, Mirza M, Xu B, Warde-Farley D, Ozair S, Courville A, Bengio Y. Generative adversarial nets. NIPS'14: Proc 27th Int Conf on Neural Information Processing Systems 2014; 2: 2672-2680.
[149] Xiao C, Li B, Zhu J-Y, He W, Liu M, Song D. Generating adversarial examples with adversarial networks. arXiv Preprint. 2018. Source: <https://arxiv.org/abs/1801.02610>. DOI: 10.48550/arXiv.1801.02610.
[150] Ho J, Jain A, Abbeel P. Denoising diffusion probabilistic models. NIPS'20: Proc 34th Int Conf on Neural Information Processing Systems 2020: 6840-6851.
[151] Wang J, Lyu Z, Lin D, Dai B, Fu H. Guided diffusion model for adversarial purification. arXiv Preprint. 2022. Source: <https://arxiv.org/abs/2205.14969>. DOI: 10.48550/arXiv.2205.14969.
[152] Chen J, Chen H, Chen K, Zhang Y, Zou Z, Shi Z. Diffusion models for imperceptible and transferable adversarial attack. arXiv Preprint. 2023. Source: <https://arxiv.org/abs/2305.08192>. DOI: 10.48550/arXiv.2305.08192.
[153] Li X, Li F. Adversarial examples detection in deep networks with convolutional filter statistics. 2017 IEEE Int Conf on Computer Vision (ICCV) 2017: 5764-5772. DOI: 10.1109/ICCV.2017.615.
[154] Zantedeschi V, Nicolae M-I, Rawat A. Efficient defenses against adversarial attacks. AISec '17: Proc 10th ACM Workshop on Artificial Intelligence and Security 2017: 39-49. DOI: 10.1145/3128572.3140449.
[155] Guo C, Rana M, Cisse M, Van Der Maaten L. Countering adversarial images using input transformations. arXiv Preprint. 2017. Source: <https://arxiv.org/abs/1711.00117>. DOI: 10.48550/arXiv.1711.00117.
[156] Raff E, Sylvester J, Forsyth S, McLean M. Barrage of random transforms for adversarially robust defense. 2019 IEEE/CVF Conf on Computer Vision and Pattern Recognition (CVPR) 2019: 6528-6537. DOI: 10.1109/CVPR.2019.00669.
[157] Carlini N, Wagner D. Adversarial examples are not easily detected: Bypassing ten detection methods. AISec '17: Proc ACM Workshop on Artificial Intelligence and Security 2017: 3-14. DOI: 10.1145/3128572.3140444.
[158] Dong Y, Liao F, Pang T, Su H, Zhu J, Hu X, Li J. Boosting adversarial attacks with momentum. 2018 IEEE/CVF Conf on Computer Vision and Pattern Recognition 2018: 9185-9193. DOI: 10.1109/CVPR.2018.00957.
[159] Mosbach M, Andriushchenko M, Trost T, Hein M, Klakow D. Logit pairing methods can fool gradient-based attacks. arXiv Preprint. 2018. Source: <https://arxiv.org/abs/1810.12042>. DOI: 10.48550/arXiv.1810.12042.
[160] Xie C, Wu Y, Maaten Lvd, Yuille AL, He K. Feature denoising for improving adversarial robustness. 2019 IEEE/CVF Conf on Computer Vision and Pattern Recognition (CVPR) 2019: 501-509. DOI: 10.1109/CVPR.2019.00059.
[161] Bai T, Luo J, Zhao J, Wen B, Wang Q. Recent advances in adversarial training for adversarial robustness. arXiv Preprint. 2021. Source: <https://arxiv.org/abs/2102.01356>. DOI: 10.48550/arXiv.2102.01356.
[162] Schott L, Rauber J, Bethge M, Brendel W. Towards the first adversarially robust neural network model on MNIST. arXiv Preprint. 2018. Source: <https://arxiv.org/abs/1805.09190>. DOI: 10.48550/arXiv.1805.09190.
[163] Rebuffi S-A, Gowal S, Calian DA, Stimberg F, Wiles O, Mann TA. Data augmentation can improve robustness. NIPS'21: Proc 35th Int Conf on Neural Information Processing Systems 2021: 29935-29948.
[164] Katz G, Barrett C, Dill DL, Julian K, Kochenderfer MJ. Reluplex: An efficient smt solver for verifying deep neural networks. In Book: Majumdar R, Kunčak V, eds. Computer aided verification. 29th Int Conf, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I. Cham, Switzerland: Springer International Publishing AG; 2017: 97-117. DOI: 10.1007/978-3-319-63387-9_5.
[165] Anderson BG, Gautam T, Sojoudi S. An overview and prospective outlook on robust training and certification of machine learning models. arXiv Preprint. 2022. Source: <https://arxiv.org/abs/2208.07464>. DOI: 10.48550/arXiv.2208.07464.
[166] Raghunathan A, Steinhardt J, Liang P. Certified defenses against adversarial examples, arXiv preprint arXiv:1801.09344 (2018).

[167] Hein M, Andriushchenko M. Formal guarantees on the robustness of a classifier against adversarial manipulation. NIPS'17: Proc 31st Int Conf on Neural Information Processing Systems 2017: 2263-2273.
[168] Raghunathan A, Steinhardt J, Liang PS. Semidefinite relaxations for certifying robustness to adversarial examples. NIPS'18: Proc 32nd Int Conf on Neural Information Processing Systems 2018: 10900-10910.
[169] Huang X, Kwiatkowska M, Wang S, Wu M. Safety verification of deep neural networks. In Book: Majumdar R, Kunčak V, eds. Computer aided verification. 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I. Cham, Switzerland: Springer International Publishing AG; 2017: 3-29. DOI: 10.1007/978-3-319-63387-9_1.
[170] Dvijotham K, Stanforth R, Gowal S, Mann TA, Kohli P. A dual approach to scalable verification of deep networks. Conf on Uncertainty in Artificial Intelligence 2018; 1: 550-559.
[171] Mirman M, Gehr T, Vechev M. Differentiable abstract interpretation for provably robust neural networks. Int Conf on Machine Learning 2018: 3578-3586.
[172] Sinha A, Namkoong H, Volpi R, Duchi J. Certifying some distributional robustness with principled adversarial training. arXiv Preprint. 2017. Source: <https://arxiv.org/abs/1710.10571>. DOI: 10.48550/arXiv.1710.10571.
[173] Gowal S, Dvijotham K, Stanforth R, Bunel R, Qin C, Uesato J, Arandjelovic R, Mann T, Kohli P. On the effectiveness of interval bound propagation for training verifiably robust models. arXiv Preprint. 2018. Source: <https://arxiv.org/abs/1810.12715>. DOI: 10.48550/arXiv.1810.12715.
[174] Shi Z, Wang Y, Zhang H, Yi J, Hsieh C-J. Fast certified robust training with short warmup. NIPS'21: Proc 35th Int Conf on Neural Information Processing Systems 2021: 18335-18349.
[175] Zhang H, Chen H, Xiao C, Gowal S, Stanforth R, Li B, Boning D, Hsieh C-J. Towards stable and efficient training of verifiably robust neural networks. arXiv Preprint. 2019. Source: <https://arxiv.org/abs/1906.06316>. DOI: 10.48550/arXiv.1906.06316.
[176] Mirman M, Baader M, Vechev M. The fundamental limits of interval arithmetic for neural networks. arXiv Preprint. 2021. Source: <https://arxiv.org/abs/2112.05235>. DOI: 10.48550/arXiv.2112.05235.
[177] Tjeng V, Xiao K, Tedrake R. Evaluating robustness of neural networks with mixed integer programming. arXiv Preprint. 2017. Source: <https://arxiv.org/abs/1711.07356>. DOI: 10.48550/arXiv.1711.07356.
[178] Xiang W, Johnson TT. Reachability analysis and safety verification for neural network control systems. arXiv Preprint. 2018. Source: <https://arxiv.org/abs/1805.09944>. DOI: 10.48550/arXiv.1805.09944.
[179] Jovanović N, Balunović M, Baader M, Vechev M. On the paradox of certified training. arXiv Preprint. 2021. Source: <https://arxiv.org/abs/2102.06700>. DOI: 10.48550/arXiv.2102.06700.
[180] Zhang B, Jiang D, He D, Wang L. Rethinking lipschitz neural networks and certified robustness: A boolean function perspective. NIPS'22: Proc 36th Int Conf on Neural Information Processing Systems 2022: 19398-19413.
[181] Cisse M, Bojanowski P, Grave E, Dauphin Y, Usunier N. Parseval networks: Improving robustness to adversarial examples. ICML'17: Proc 34th Int Conf on Machine Learning 2017: 854-863.
[182] Tsuzuku Y, Sato I, Sugiyama M. Lipschitz-margin training: Scalable certification of perturbation invariance for deep neural networks. NIPS'18: Proc 32nd Int Conf on Neural Information Processing Systems 2018: 6542-6551.
[183] Huster T, Chiang C-YJ, Chadha R. Limitations of the lipschitz constant as a defense against adversarial examples. In Book: Alzate C, Monreale A, Assem H, et al., eds. ECML PKDD 2018 Workshops. Nemesis 2018, UrbReas 2018, SoGood 2018, IWAISe 2018, and Green Data Mining 2018, Dublin, Ireland, September 10-14, 2018, Proceedings. Cham, Switzerland: Springer Nature Switzerland AG; 2019: 16-29. DOI: 10.1007/978-3-030-13453-2_2.
[184] Anil C, Lucas J, Grosse R. Sorting out Lipschitz function approximation. Int Conf on Machine Learning 2019: 291-301.
[185] Zhang B, Jiang D, He D, Wang L. Boosting the certified robustness of l-infinity distance nets. arXiv Preprint. 2021. Source: <https://arxiv.org/abs/2110.06850>. DOI: 10.48550/arXiv.2110.06850.
[186] Li B, Chen C, Wang W, Carin L. Certified adversarial robustness with additive noise. Proc 33rd Int Conf on Neural Information Processing Systems 2019: 9464-9474.
[187] Erdemir E, Bickford J, Melis L, Aydore S. Adversarial robustness with non-uniform perturbations. NIPS'21: Proc 35th Int Conf on Neural Information Processing Systems 2021: 19147-19159.
[188] Wang L, Zhai R, He D, Wang L, Jian L. Pretrain-to-finetune adversarial training via sample-wise randomized smoothing. 2021. Source: <https://openreview.net/forum?id=Te1aZ2myPIu>.
[189] Yang G, Duan T, Hu JE, Salman H, Razenshteyn I, Li J. Randomized smoothing of all shapes and sizes. ICML'20: Proc 37th Int Conf on Machine Learning 2020: 10693-10705.
[190] Gilmer J, Metz L, Faghri F, Schoenholz SS, Raghu M, Wattenberg M, Goodfellow I. Adversarial spheres. arXiv Preprint. 2018. Source: <https://arxiv.org/abs/1801.02774>. DOI: 10.48550/arXiv.1801.02774.
[191] Schmidt L, Santurkar S, Tsipras D, Talwar K, Madry A. Adversarially robust generalization requires more data. NIPS'18: Proc 32nd Int Conf on Neural Information Processing Systems 2018: 5019-5031.
[192] Quinonero-Candela J, Sugiyama M, Schwaighofer A, Lawrence ND. Dataset shift in machine learning. Mit Press; 2008.
[193] Papernot N, McDaniel P. Deep k-nearest neighbors: Towards confident, interpretable and robust deep learning. arXiv Preprint. 2018. Source: <https://arxiv.org/abs/1803.04765>. DOI: 10.48550/arXiv.1803.04765.
[194] Apruzzese G, Anderson HS, Dambra S, Freeman D, Pierazzi F, Roundy K. "Real attackers don't compute gradients": Bridging the gap between adversarial ml research and practice. 2023 IEEE Conf on Secure and Trustworthy Machine Learning (SaTML) 2023: 339-364. DOI: 10.1109/SaTML54575.2023.00031.
[195] Nassi B, Mirsky Y, Nassi D, Ben-Netanel R, Drokin O, Elovici, Y. Phantom of the ADAS: Securing advanced

driver-assistance systems from split-second phantom attacks. CCS '20: Proc 2020 ACM SIGSAC Conf on Computer and Communications Security 2020: 293-308. DOI: 10.1145/3372297.3423359.
[196] Xiao Q, Chen Y, Shen C, Chen Y, Li K. Seeing is not believing: Camouflage attacks on image scaling algorithms. SEC'19: Proc 28th USENIX Conf on Security Symposium 2019: 443-460.
[197] Miller BP, Fredriksen L, So B. An empirical study of the reliability of UNIX utilities. Commun ACM 1990; 33(12): 32-44. DOI: 10.1145/96267.96279.
[198] Hamlet R. Random testing. Encyclopedia of Software Engineering. New York: Wiley; 1994.
[199] Wagemans J, Elder JH, Kubovy M, Palmer SE, Peterson MA, Singh M, Von der Heydt R. A century of gestalt psychology in visual perception: I. perceptual grouping and figure-ground organization. Psychol Bull 2012; 138(6): 1172-1217. DOI: 10.1037/a0029333.
[200] Ponzo M. Intorno ad alcune illusioni nel campo delle sensazioni tattili, sull'illusione di Aristotele e fenomeni analoghi. Wilhelm Engelmann; 1910.
[201] Wimmer MC, Doherty MJ, Collins WA. The development of ambiguous figure perception. Monogr Soc Res Child Dev 2011; 76(1): vii, 1-130. DOI: 10.1111/j.1540-5834.2011.00589.x.

Authors' information

Anton Vsevolodovich Trusov (b. 1997) completed Master's degree in Applied Mathematics and Physics at Moscow Institute of Physics and Technology in 2021, since then he has been a Ph.D. student at this institute. He also works as a programmer at FRC "Computer Science and Control" and at Smart Engines Service LLC. Research interests: computer vision algorithms, artificial neural networks, high-performance computing, and adversarial deep learning. E-mail: *trusov.av@smartengines.com*

Elena Evgenevna Limonova (b. 1993) completed and Master's degree (2017) in Applied Mathematics and Physics at Moscow Institute of Physics and Technology. Received her Ph.D. degree in Computer Science in 2023 at FRC "Computer Science and Control" RAS. Currently, she works as a researcher at the FRC "Computer Science and Control" RAS and as a programmer at Smart Engines Service LLC. Research interests: computer vision, artificial neural networks, image recognition on mobile devices. E-mail: *limonova@smartengines.com*

Vladimir Viktorovich Arlazarov (b. 1976) studied at Moscow Institute of Steel and Alloys where he completed his Specialist degree in 1999. Received his Ph.D. degree in Computer Science in 2005, and Doctor of Sciences degree in computer science in 2023. Currently he works as a lead researcher and head of a department at the FRC "Computer Science and Control" RAS. Since 2016, he has been a general director of Smart Engines Service LLC. Research interests: computer vision and document analysis systems. E-mail: *vva@smartengines.com*

Received January 10, 2024. The final version – June 07, 2024.