FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Věra Kumová

Creating Adversarial Examples in Machine Learning

Department of Theoretical Computer Science and Mathematical Logic

Supervisor of the master thesis:	Mgr. Martin Pilát, Ph.D.
Study programme:	Computer Science
Study branch:	Artificial Intelligence

Prague 2021

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources. It has not been used to obtain another or the same degree.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In .... date ....

Author's signature

I would like to thank my thesis supervisor, Mgr. Martin Pilát, Ph.D., for his helpful comments and advice as well as tremendous support during this work. I would also like to thank my two great friends for providing me computational resources to successfully complete my experiments.

Title: Creating Adversarial Examples in Machine Learning

Author: Věra Kumová

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: Mgr. Martin Pilát, Ph.D., Department of Theoretical Computer Science and Mathematical Logic

Abstract: This thesis examines adversarial examples in machine learning, specifically in the image classification domain. State-of-the-art deep learning models are able to recognize patterns better than humans. However, we can significantly reduce the model's accuracy by adding imperceptible, yet intentionally harmful noise. This work investigates various methods of creating adversarial images as well as techniques that aim to defend deep learning models against these malicious inputs. We choose one of the contemporary defenses and design an attack that utilizes evolutionary algorithms to deceive it. Our experiments show an interesting difference between adversarial images created by evolution and images created with the knowledge of gradients. Last but not least, we test the transferability of our created samples between various deep learning models.

Keywords: Adversarial examples; Deep Learning; Image classification; Evolutionary Algorithms

Contents

Introduction

Machine learning has achieved many significant successes in recent years. However, it turns out that most models are sensitive to so-called adversarial examples. This work focuses on adversarial examples in image classification, where these samples represents slightly modified patterns so that humans are not able to recognize this perturbation, but even state of the art deep learning models classify such a pattern incorrectly.

The question of adversarial examples in image processing rightfully draws an attention since the deep learning models are used in many industries and adversarial attacks on these models can pose a security threat. Therefore, many researchers investigate either the methods of defending neural networks against adversarial attacks or methods of creating new and more sophisticated adversarial perturbations which deceive current defenses. The goal of this work is the second aspect of adversarial examples research, thus finding adversarial examples that will be able to defeat modern methods of defense. As we will see later, adversarial examples can be crafted with various amount of information about the model we aim to deceive. We chose to design a black box attack which uses minimum knowledge and utilizes evolutionary algorithms.

This work aims to summarize the methods of adversarial attacks with the focus on evolutionary techniques. The next and main goal is to use the knowledge and results from previous works and design adversarial attack capable of deceiving one of the modern defenses. Last, we aim to test the transferability of our designed adversarial attack between models, including an adversarially trained network.

This work is organized as follows. First chapter describes the fundamental knowledge about neural networks, deep learning and evolutionary algorithms, which is necessary for understanding the basic idea of adversarial examples. Second chapter defines the adversarial examples, describes their taxonomy and summarizes the methods of attacks and defenses. Since the topic of adversarial examples is very actual, there are inexhaustible amount of attacks and defenses. We do not aim to cover all of them. Instead, we chose methods which we consider as influencing, impressive or important for our further experiments. Third chapter describes the approach we designed to defeat one of the modern defenses against adversarial images. In last chapter, we report the result of our experiments and analyze them.

1. Theoretical background

This chapter outlines the fundamental knowledge required for the understanding of the following theory as well as experiments. In the next sections, we will discuss a phenomenon called adversarial examples in image processing. Therefore it is necessary to explain basic approaches to this problem. We will also utilize evolutionary algorithms, which we also describe.

1.1 Basics of Neural Networks

Image processing, as we know it today, is possible thanks to models called neural networks. Even though neural networks are now in tremendous use, the idea behind them is relatively old. We can learn from the book of Šíma and Neruda [1996] that the origin of this field was in the 1940s with a basic mathematical model of the neuron created by McCulloch and Pitts. In 1957 Rosenblatt [1958] invented *perceptron*, which was the generalization of the previous model. He also proposed a learning algorithm and proved its convergence if the solution exists. The interest in neural nets has then retreated to the background. In 1986 an article was published by Rumelhart et al. [1986] about a learning algorithm called *backpropagation*, although the algorithm itself was created 16 years earlier by Linnainmaa [1976]. This algorithm was designed for multilayer neural networks and is still the most popular and most used algorithm for learning neural nets.

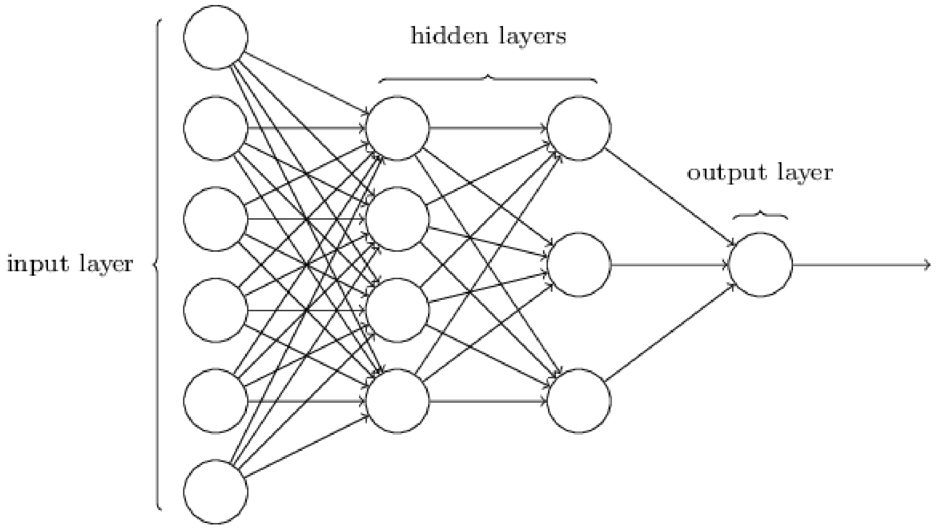


Figure 1.1: An example architecture of a multilayer perceptron by LeDell [2016].

An example architecture of a *multilayer perceptron* is displayed in Figure 1.1. Each circle represents a neuron, and all neurons are organized into layers. There are an input layer, hidden layers, and an output layer. All neurons are interconnected in a specific manner so that the information flows from the input layer through hidden layers to the output layer. Each connection has a *weight* w, a parameter, which needs to be learned. Neurons have one or more inputs and one output. The output y of a neuron depends on its inputs $\boldsymbol{x}$, its *activation function* f, and its specific number υ called *bias*:

$$y = f(\sum_{i=1}^{n} x_i w_i + \upsilon)$$

It is obvious, that a neural network computes a compound function $h(\boldsymbol{x}) = \boldsymbol{y}$, where $\boldsymbol{x} \in \mathbb{R}^m$ is an input and $\boldsymbol{y} \in \mathbb{R}^k$ is an output. All network parameters, weights and biases, are denoted as $\boldsymbol{\theta}$.

In general, there are two types of learning: *supervised* and *unsupervised*. In unsupervised learning, we want to find some hidden structures and connections in data. An example is a *k-means* algorithm, which finds clusters in data based on some measure of similarity. On the other hand, supervised learning aims to create a model, which learns to produce specific output based on *training data*[1]. Each example in the training data consists of one or more input values and the output value, which can be a class (a classification problem) or a number (a regression problem). We will consider only the classification setting because the goal of adversarial examples is to confuse the model so that it wrongly classifies the input pattern. In the classification scenario, the network represents an *m-class classifier*. Its output is a vector of probabilities, whose length is m. It is computed using the softmax function defined by the formula

$$\sigma(\boldsymbol{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}},$$

where $\boldsymbol{z} \in \mathbb{R}^k$. The inputs to the softmax function are called *logits*. The class with the highest probability is then chosen as the final classification of the input pattern:

$$C(\boldsymbol{x}) = \arg\, \max_i h(\boldsymbol{x})_i$$

The backpropagation algorithm was designed for supervised learning and applies gradient descent to minimize an error function. The idea behind this algorithm is crucial because it is applied for creating adversarial examples. The error function, which instantly comes to mind and is often used, computes the sum of squares and is described by Equation 1.1, where p is a number of patterns in training data, j is a number of classes, y is an actual output of the neural network and d is the desired output.

$$E = \frac{1}{2} \sum_p \sum_j (y_{j,p} - d_{j,p})^2 \tag{1.1}$$

With every step of the algorithm, we want to get closer to the minimum of the error function. Each weight needs to be changed by a small bit in a good direction. This direction is given by a gradient $\Delta_E w_{i,j}(t)$, where $w_{i,j}$ is a weight from neuron i to neuron j and t represents time. The whole adaptation step looks like this:

$$w_{i,j}(t+1) = w_{i,j}(t) + \Delta_E w_{i,j}(t)$$

The gradient is computed with respect to the weight, so the chain rule is applied:

$$\Delta_E w_{i,j} = -\frac{\partial E}{\partial w_{i,j}} = -\frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial \xi_j}\frac{\partial \xi_j}{\partial w_{i,j}}$$

[1]During a training process, *validation data* for hyperparameters setting and *testing data* for evaluation of the network's performance are used.

The result depends, among other things, on the layer location. For more details, see Rojas [1996], chapter 7.

In a classification problem, the output of the neural network is a probability distribution. Therefore there is a better choice for an error function, which draws inspiration from information theory. It is called *cross-entropy loss*, or *log loss*, and is defined in Equation 1.2.

$$\mathcal{L} = -\sum_i p_i \log q_i \tag{1.2}$$

Minimizing the cross-entropy corresponds to minimizing the *Kullback-Leibler (KL) divergence*, which measures how two probability distributions differ. The KL divergence is defined as

$$D_{KL}(P \parallel Q) = \mathbb{E}_{\boldsymbol{X}\sim P}\left[\log \frac{P(\boldsymbol{x})}{Q(\boldsymbol{x})}\right] = \mathbb{E}_{\boldsymbol{X}\sim P}\left[\log P(\boldsymbol{x}) - \log Q(\boldsymbol{x})\right],$$

where, in our case, $P(\boldsymbol{x})$ is the desired probability distribution represented by a one-hot vector with 1 in a place of true class, and $Q(\boldsymbol{x})$ is the output probability distribution of the neural network. For more information about the KL divergence and its properties, see Goodfellow et al. [2016].

1.2 Image Processing with Neural Networks

Neural networks used for image processing are called convolutional because they apply an operation called *convolution*[2]. According to Goodfellow et al. [2016], the modern convolutional networks were developed by LeCun in 1989. They were some of the first deep neural networks trained with the backpropagation algorithm and further some of the first networks utilized for commercial applications, e.g., reading checks in the 1990s.

The basic building blocks of convolutional networks are layers, which apply a so-called *kernel* or *filter*. Because we speak only about image processing, we can limit ourselves to 2-D convolutions. The kernel represents a sliding window of some smaller size, looking for a specific feature throughout a whole image. In this manner, a series of kernels produces a 2-D map of particular features in the input image, each kernel extracting one kind of feature. Considering that detecting a feature in an image is positionally insignificant, this approach with a convolutional kernel is more computationally efficient than applying a fully connected (FC) layer with significantly more parameters.

In Figure 1.2, there is an example of convolutional operation with a 2×2 kernel and a 3×4 input. The resulting output has a 2×3 size because we compute a convolution only in positions where the kernel fully lies within the input image. This is called a *valid padding.* Padding is a method of how to deal with image edges.

[2]As a matter of fact, a mathematical operation applied in convolutional networks is the cross-correlation. However, because it resembles a convolution, the name convolutional networks were adopted. Probably a more appropriate name would be a positionally invariant locally activated networks. For more detailed information, see Goodfellow et al. [2016] or Straka [2018].

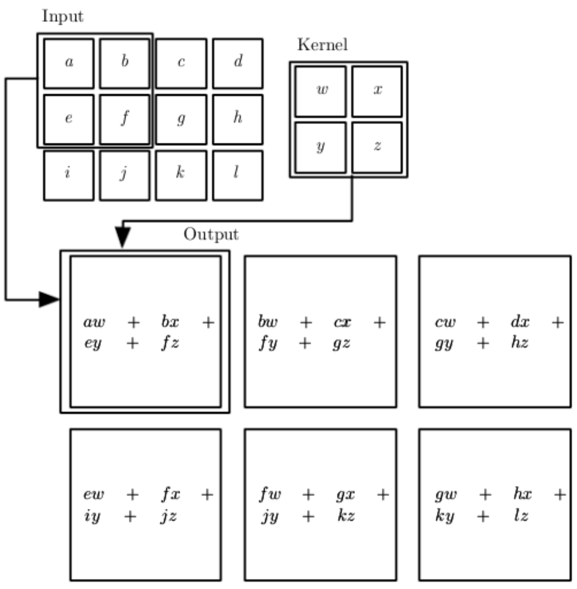


Figure 1.2: An example of 2-D convolution by Goodfellow et al. [2016].

A convolution operation produces a linear activation. Then a nonlinear activation is applied, most often rectified linear unit (ReLU), defined as

$$f(x) = max(0, x),$$

less often hyperbolic tangent

$$\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}.$$

Goodfellow et al. [2016] refer to this as the detector stage. A *pooling layer* follows, which replaces the output of nonlinear activation with some summary statistics. The pooling operation is also used for shrinking the output and, therefore, improving the whole network's computational efficiency. Figure 1.3 shows how max pooling along with downsampling works.

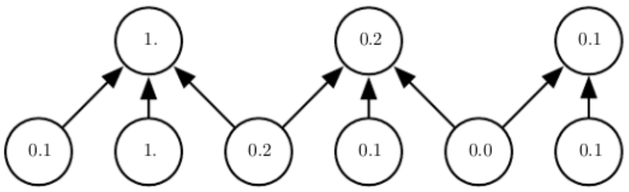


Figure 1.3: An example of a pooling operation with downsampling by Goodfellow et al. [2016].

Several FC layers follow a series of convolutional and pooling layers because the information from the map of features needs to be connected together. Then, the softmax activation function follows for classification purposes. Different architectures vary in size of kernels, activation functions, and depth of the whole network, but some also adopt specific improvements.

1.2.1 Architectures

In Chapter 3, we use three architectures, VGG, ResNet and GoogLeNet (Inception). Let us introduce them very briefly.

Simonyan and Zisserman [2015] introduced VGG networks in 2014 as a model with the generic layout described above but with numerous convolutional layers. The architecture used in Chapter 3 has 16 convolutional and 3 FC layers. Therefore, it is referred to as VGG-19. Unlike architectures used before, VGG applies a smaller kernel size, mainly 3×3. A stack of two 3×3 kernels has the same receptive field as one 5×5 kernel but fewer parameters. When we compare a stack of three 3×3 kernels with one 7×7 kernel, the receptive field is still the same, but the difference in the number of parameters is even more vigorous.

Unfortunately, even with smaller kernels, the convolutional network cannot be deepened indefinitely because of a *degradation* problem when adding more layers leads to accuracy saturation and consequently higher training error. He et al. [2016] solve the degradation by a *deep residual learning.* They utilize shortcut connections displayed in Figure 1.4 as identity. These residual connections represent no extra parameters nor computational complexity, and their output is element-wise added to the outputs of at least two stacked weighted layers, either convolutional or fully connected.

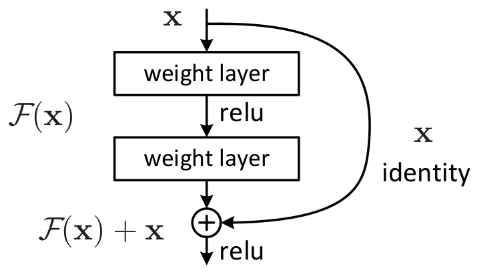


Figure 1.4: A building block of ResNet architecture by He et al. [2016].

The deepest ResNet architecture has 152 layers and besides residual connections exploits a *bottleneck* building block. The bottleneck represents three stacked layers with 1×1, 3×3 and 1×1 convolutions. In Chapter 3 we use ResNet101 and ResNet152.

Last architecture we use is Inception-v3 by Szegedy et al. [2016]. Its advantage is lower computational cost thanks to fewer parameters than VGG. On the other hand, it has a more complex architecture because it utilizes special modules with various kernels. One such module is depicted in Figure 1.5. It applies at most 3×3 kernel, same as VGG. On the other hand, it does not have as many fully connected layers at its end. Inception implements global pooling instead, which spares parameters. It is not as deep as ResNet architectures, which we use, as it has only 48 layers.

Inception-v3 adopts two more techniques, which we did not see in previous architectures. First of them are *auxiliary classifiers* implemented in lower layers for regularization and additional gradients in backpropagation. Second is *label smoothing* for the model to be less confident and therefore more adaptive.

Last note to image processing concerns standard deep learning datasets. Most used datasets for computer vision tasks are MNIST, CIFAR-10 and ImageNet.

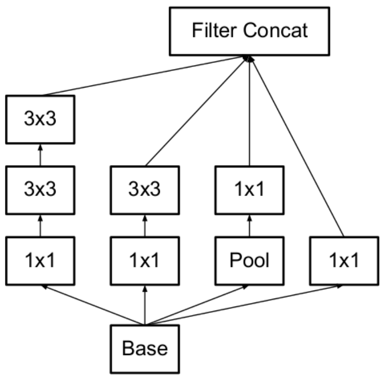


Figure 1.5: A module of Inception architecture by Szegedy et al. [2016].

MNIST by LeCun and Cortes [1999] contains handwritten single digits. The images are in grayscale and only 28×28 pixels small, so they do not correspond to today's real-world scenarios. CIFAR-10 by Krizhevsky et al. [2009] contains color and slightly bigger images of 32×32 pixels. Images are in 10 classes. There is also CIFAR-100 with 100 classes. ImageNet by Deng et al. [2009] is the most extensive database of millions of color images in full resolution. Since 2010, there is a contest called the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), where authors compete to provide the best performing model for image processing. As the database is so enormous, there is always only a subset of it. In Chapter 3, we do experiments also on a subset of ImageNet.

1.3 Evolutionary Algorithms

Evolutionary algorithms (EA) are optimization methods motivated by Darwin's theory. The basic idea is to find a solution to the problem using *populations* of potential solutions called *individuals* and stochastic operations inspired by evolution such as *selection*, *mutation*, or *crossover*. We describe a simple genetic algorithm and then a method we are experimenting with in Chapter 3, called differential evolution.

1.3.1 Simple Genetic Algorithm

Simple genetic algorithm (SGA) is an original version of evolutionary algorithms proposed by Holland in the 1970s. Although we discuss adversarial examples in more detail in Chapter 2, we explain the concept of SGA on them to better understand it.

Suppose we have an image from the MNIST dataset. It is a grayscale image of 28×28 pixels; therefore, we can represent it in a vector of 784 real numbers. We aim to modify the image in such a way that some neural network classifies the image incorrectly. We can achieve this by adding noise to the image that can also be represented in a vector of length 784, but this time there are zeros and ones. Ones mean we add a small number to the pixel, e.g., 0.1. Zeros mean we do not perform any change in that position. Thus, all possible vectors of zeros and ones of length 784 are candidate solutions to the problem, which is 1.02×10^{236}

potential solutions in total. This search space is enormous, so SGA provides a heuristic to speed up computation.

To compare two candidate solutions, we need a so-called *fitness function.* Fitness should reflect the quality of individuals, and often we aim to maximize it. In our example, a good fitness could be a probability of the true class returned by a neural network. As the probability should be the lowest possible, we can turn it into a maximization problem by subtracting the probability from one. Also, the algorithm can be implemented as a minimization problem.

With having a measure to select the best solution, we can choose individuals according to their fitness function values to apply genetic operators. Roulette wheel selection is often used, in which the probability of being chosen is proportional to the quality of an individual's fitness. Both crossover and mutation then happen only with some specific probability. Crossover is an operation for creating two new individuals, *offspring*, by crossing two selected old individuals, *parents.* The basic kind of crossover is a one-point crossover, where we randomly select a point in an individual. One child is then created by copying a part of one parent to that point and copying the rest from the second parent. Second child has opposite contributions from parents. This approach can be generalized also to n-point crossover. By mutation, we alter a single individual by some minor change. In this case, it can be flipping one bit from zero to one or the other way around. The whole algorithm in pseudocode looks like this:

```
$t \leftarrow 0$;
$P_t \leftarrow$ first population of $n$ individuals generated at random;
while condition not met do
    compute fitness value for each individual in $P_t$;
    with repetition select $n$ parents based on their fitness;
    $O_t \leftarrow$ create offspring from parents by applying crossover with
      probability $p_c$ and mutation with probability $p_m$;
    $P_{t+1} \leftarrow O_t$;
    $t \leftarrow t + 1$;
end
```

Algorithm 1: Simple Genetic Algorithm

The condition for ending the while loop could be reaching a predefined number of generations or fulfilling some criteria for the best individual. In our example, evolution is successful if the adjusted image fools the classifier. Also, the algorithm can be stopped if the best fitness for a given number of generations does not change. For more detailed information, see Šíma and Neruda [1996] or Pilát [2020].

1.3.2 Differential Evolution

Continuous optimization where an individual is a vector of real numbers is better suited for adversarial examples crafting. We can apply the same approach for continuous optimization as in SGA, but with modified genetic operations. Mutation can be done by adding a small number to each position with a small probability. Apart from the n-point crossover, we can use arithmetic crossover where a child is a weighted average of parents. The problem of this approach is the inability to optimize non-separable or highly conditioned functions. Therefore, in Chapter 3, we use differential evolution.

In differential evolution, each individual undergoes mutation and crossover, and the mutation looks quite differently. For every individual $\boldsymbol{x}$ in a population, we randomly choose three other individuals $\boldsymbol{x}_a$, $\boldsymbol{x}_b$ and $\boldsymbol{x}_c$, and we create a donor:

$$\boldsymbol{v} = \boldsymbol{x}_a + F(\boldsymbol{x}_b - \boldsymbol{x}_c),$$

where F is a mutation parameter from range $\langle 0, 2 \rangle$. Donor $\boldsymbol{v}$ is then uniformly crossed with parent $\boldsymbol{x}$. It means that for each position, we take value from the donor with probability p_c or from the parent probability $1 - p_c$. Furthermore, the crossover is done so that at least one element comes from the parent. The resulting individual goes into the next generation if it has better fitness than the parent. Otherwise, the parent stays for the next generation. Selection is thus deterministic. This is differential evolution in pseudocode:

```
$t \leftarrow 0$;
$P_t \leftarrow$ first population of $n$ individuals generated at random;
while condition not met do
    $P_{t+1} \leftarrow \emptyset$;
    for each individual $\boldsymbol{x}$ in $P_t$ do
        $\boldsymbol{x}_a, \boldsymbol{x}_b, \boldsymbol{x}_c \leftarrow$ individuals selected from $P_t$ at random;
        $\boldsymbol{v} \leftarrow \boldsymbol{x}_a + F(\boldsymbol{x}_b - \boldsymbol{x}_c)$;
        SP $\leftarrow$ number of safe position generated at random;
        for each position $i$ in $\boldsymbol{x}$ do
            RN $\leftarrow$ real number generated at random from $\langle 0, 1 \rangle$;
            if $RN \leq p_c$ and $i \neq SP$ then
                $\boldsymbol{x}[i] \leftarrow \boldsymbol{v}[i]$;
            end
        end
        if fitness($\boldsymbol{x}$) $\leq$ fitness($\boldsymbol{v}$) then
            add $\boldsymbol{v}$ to $P_{t+1}$;
        end
        else
            add $\boldsymbol{x}$ to $P_{t+1}$;
        end
    end
    $t \leftarrow t + 1$;
end
```

Algorithm 2: Differential Evolution

In the next chapter, we describe creating adversarial examples using other approaches inspired by evolution, such as particle swarm optimization or evolutionary strategies. As we do not use these methods in experiments in Chapter 3, we do not describe them in more details. For more information about these methods, see original papers cited with the approach or see Eiben and Smith [2015].

2. Adversarial examples: existing approaches

This chapter summarizes the theoretical background of adversarial examples. As it is a prevalent topic, there are many approaches to creating adversarial examples or defending against them. Therefore, we describe only methods that are influencing, ingenious, or important for our experiments.

2.1 Pioneering works

The first paper, which referred to adversarial examples, was written by Szegedy et al. and published in 2014. The authors made a breakthrough discovery that input–output mapping learned by deep neural networks is not as continuous as we would expect it to be.

During the training of computer vision models, adjusted images are used for better generalization and robustness of the models. These adjustments might include flipping the image, introduction of the Gaussian noise, color distortion, etc. Adjustments of this kind are well recognized by humans, and of course, there should be no change in predicted class. However, Szegedy et al. [2014] showed that when we use a specific distortion, even such that is not visible by the human eye, the predicted label will change dramatically, and the model might be fairly confident with the wrong prediction.

The important property of this specific distortion is non-randomness. In this case, the perturbation is applied in such a manner that the prediction error is maximal. Authors created their adversarial examples by solving a box-constrained optimization problem, where having a classifier $C : \mathbb{R}^m \rightarrow \{1 \dots k\}$, input image $\boldsymbol{x} \in \mathbb{R}^m$ and given a label $l \in \{1 \dots k\}$, the aim is to find a minimal $\|\boldsymbol{\eta}\|_2$ such that these conditions hold:

1. $C(\boldsymbol{x} + \boldsymbol{\eta}) = l$
2. $\boldsymbol{x} + \boldsymbol{\eta} \in [0, 1]^m$

In other words we are looking for a ”minimum distortion” function $D(\boldsymbol{x}, l)$ which finds such $\boldsymbol{\eta}$ that when label of $\boldsymbol{x}$ is k then $\boldsymbol{x} + \boldsymbol{\eta}$ is classified by f as $l \neq k$ and $\boldsymbol{x} + \boldsymbol{\eta}$ is closest image to $\boldsymbol{x}$ as possible. As the exact computation of $D(\boldsymbol{x}, l)$ is a hard problem, authors computed an approximation using a box-constrained L-BFGS [1].

Authors made a series of experiments with different architectures trained on different datasets and found out important properties of adversarial examples. The first one is *cross model generalization*, which means that when adversarial examples are created against one neural network, then a relatively large portion of them is also misclassified by a different network that was trained from scratch with different hyperparameters such as a number of layers, activations or regularization. Second significant property is *cross training-set generalization.* This

[1]Limited-memory Broyden–Fletcher–Goldfarb–Shanno algorithm, which uses an linear search for finding the optimal value.

means that a relatively large portion of adversarial examples created against one neural network is also misclassified by a different network trained on a completely distinct dataset.

It is worth emphasizing that adversarial examples and distorted images used during training, as mentioned above, have distinct properties from a statistical point of view. Distorted images are highly correlated and drawn from the same distribution as the images from the training set. On the other hand, adversarial examples are extremely low probable images, so-called ”pockets” in the network's manifold. Nevertheless, an adversarial image can be found for every test example, which means that they are dense despite their low probability.

One year later, a second major paper was published, which sheds a different light on how and why adversarial examples are made. According to Goodfellow et al. [2015], the vulnerability of neural networks to adversarial examples is caused by their linearity. It is an entirely different point of view from Szegedy et al. [2014] as they thought they are caused by nonlinearity and overfitting.

The basic idea behind linearity causing adversarial examples lies in a high dimension of input space and a vast amount of parameters in a single neural network. As there is a limited precision of the features, the input $\boldsymbol{x}$ seems to be for the neural network same as the $\boldsymbol{x}^* = \boldsymbol{x} + \boldsymbol{\eta}$, as long as every element of the perturbation $\boldsymbol{\eta}$ is smaller than the precision of the features. More rigorously $||\boldsymbol{\eta}||_\infty < \epsilon$ where ϵ is below the distinguishing ability of the classifier. When we compute a dot product of an adversarial example $\tilde{\boldsymbol{x}}$ and a weight vector of the neural network $\boldsymbol{w}$ we get

$$\boldsymbol{w}^\top \boldsymbol{x}^* = \boldsymbol{w}^\top \boldsymbol{x} + \boldsymbol{w}^\top \boldsymbol{\eta}.$$

Then $\boldsymbol{w}^\top \boldsymbol{\eta}$ is a difference in activation between the original and the perturbed input. As elements of $\boldsymbol{\eta}$ are very small the maximal increase of the activation function can be reached by assigning $\boldsymbol{\eta} = sign(\boldsymbol{w})$. Since $||\boldsymbol{\eta}||_\infty < \epsilon$ and if we expect the average value of one element of the vector $\boldsymbol{w}$ to be n then the activation of the neural network will grow by ϵmn where m is a number of elements of the weight vector $\boldsymbol{w}$. From this, we can see that even though a single change of one element can be infinitesimal, if we add many of them, the final difference could be large enough to change the classifier's output completely.

This reasoning gave rise to a computationally much more convenient method of creating adversarial examples than the L-BFGS algorithm used in the previous case. The method is called *fast gradient sign method* (FGSM) and as the name suggests, it is based on the gradient of the loss function $\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{x}, y)$ used for the training where $\boldsymbol{\theta}$ are the parameters of a neural network, $\boldsymbol{x}$ is the input and y is the target associated with it. The perturbation can be computed as

$$\boldsymbol{\eta} = \epsilon \mathrm{sign}(\boldsymbol{\nabla}_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{x}, y)).$$

As can be seen from the formula, the gradient is not computed with respect to the model's parameters as usual but with respect to the input image. The parameter ϵ affects how much the resulting adversarial image will differ from the original input. According to experiments regarding the transferability of adversarial examples in Papernot et al. [2017], the value above 0.4 was optimal as the higher value did not increase transferability much and, on the other hand, could lead to adversarial examples distinguishable by the human eye.

With this fast method for creating adversarial examples, it is possible to train the model adversarially and by that increase its robustness. Authors experimented with this regularization and found out that models trained adversarially have more interpretable and localized weights. It is worth mentioning that this adversarial training is different from adding random noise to the inputs because the expected dot product of noise vector with zero mean and zero covariance with any other vector is zero. Therefore the activation will not change, and the result is only harder training without any regularization effect. Adversarial training is described in more details in Section 2.3.1.

Much progress has been made since publishing the first papers about adversarial examples. Researchers suggested many types of defenses and also many new types of attacks. In next sections, we describe some of them to provide a basic overview.

2.2 Attacks

As discovered by Szegedy et al. [2014], adversarial images are transferable between models. This property leads to two basic types of attack: *white box attack* and *black box attack*. In a white box attack, an attacker can access the underlying model, its architecture, weights, hyper-parameters, training process, or training data. An example of a white box attack is FGSM mentioned earlier. On the other hand, a black box attack has no such knowledge, and generating adversarial examples takes advantage of this transferability or uses the attacked model only as an oracle, which provides a vector of probabilities.

The second categorization of attacks is *targeted* and *non-targeted*. In a targeted attack, an attacker crafts adversarial examples so that the fooled network classifies them as a requested class. It is used in multi-class classification problems. On the other hand, the aim of a non-targeted attack is for the network to classify the adversarial input to any class but the true one. According to Yuan et al. [2019], there are two options for creating a non-targeted attack. One of them is to lower the probability of the true class. The other one is to compute several targeted images and take the one with the smallest perturbation. This implies that non-targeted attacks are less powerful than targeted.

Another categorization of attacks is *single-step* and *multi-step (iterative)*. In a single-step attack, an adversarial example is crafted in one iteration; therefore a gradient needed for the attack is computed only once. These examples are faster to create, but they are not necessarily as successful as examples crafted in a multi-step attack.

We can also divide attacks based on the type of computation used for creating the adversarial input. Most common are gradient-based attack, but there are also attacks which exploit evolutionary approaches.

These types of attacks could be mixed. For example, we can have a non-targeted black box attack or multi-step white box attack. A very informative point of view on types of attacks can also be found in Papernot et al. [2016a].

Figure 2.1, which is slightly modified from the original, shows the dependence of adversarial goals on knowledge about the model, which should be deceived. The most manageable goal is just a reduction of the confidence level. Then, mere misclassification means a non-targeted attack, thus changing the model's output

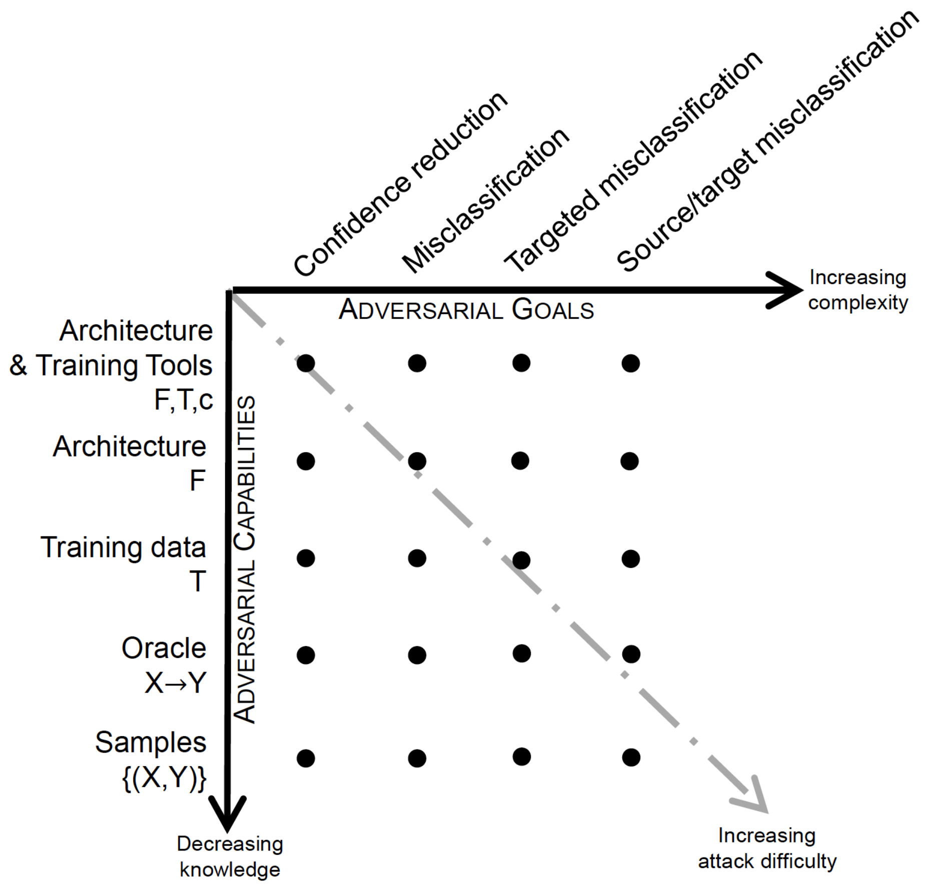


Figure 2.1: Taxonomy of threat models from Papernot et al. [2016a]

on any other class than the true one. It means reducing the confidence level of the true class so much that some other class will have higher confidence. Targeted misclassification means creating an input classified by the model as the chosen class different from the true one. Created input does not need to resemble any of the real classes to humans. Nguyen et al. [2015] refers to these inputs also as *fooling images*[2]. The last and most demanding adversarial goal is target/source misclassification, thus targeted attack, which means changing the output class of the real input to another specific output class different from the true one.

Regarding the knowledge of the original model, the best scenario for the attacker is the full knowledge of the model. It means the awareness of architecture F, training data T, and also loss function c used during the training process. Less informative is the knowledge of only the architecture. When the attacker has a first or second type of knowledge, it is considered a white box attack. Otherwise, it is a black box attack. If the attacker knows the distribution of training data, they can create a surrogate model based on a very similar dataset. The attacker can only have access to the model as an oracle and ask on any input and get the output assigned by the original model. The worst case, meaning the least informative, is having only samples. Samples are input-output pairs, but the attacker cannot change the input and see the difference in output.

Yuan et al. [2019] also suggest a taxonomy of adversarial attacks based on

[2]Fooling images can be created using evolution or gradient methods. As these images are not adversary images as defined before, we will not deal with them further.

perturbation and benchmark, but these types already mentioned are sufficient for our purpose.

Now let's see some particular examples of attacks.

2.2.1 White box attacks

An attacker using a white box attack has a significant advantage of knowing the architecture and weights of the target model. With this knowledge, the derivative can be computed, and the adversary image can be created based on this derivative.

FGSM and its extensions

We have already discussed FGSM method, which is non-targeted. According to Kurakin et al. [2017], it is usually not very successful, but its straightforward extension is an iterative version:

$$\boldsymbol{x}^*_{t+1} = Clip_{x,\epsilon}\{\boldsymbol{x}^*_t + \epsilon \text{sign}(\nabla_x \mathcal{L}(\boldsymbol{x}^*_t, y_{true}))\},$$

where $Clip_{x,\epsilon}$ is element-wise clipping. The iterative method produces more effective examples, but due to its greedy nature, it tends to end up in local optima, which leads to poor transferability. Therefore, Dong et al. [2018] integrated the momentum[3] introducing methods both for L_∞ and L_2 norm.

Another extension of FGSM is *fast gradient value method* (FGVM) described in Rozsa et al. [2016]. As the name of the method suggests, FGVM does not apply only the direction of the gradient but also takes into account the scaled magnitude. Perturbations in adversarial examples crafted with this method are more focused with less structural damage.

Basic target class method

Kurakin et al. [2017] describe an alternative to FGSM, which aims to maximize the probability of a target class. The least likely class, which means the class with the lowest probability assigned by the model, is suggested as a target class . The formula for computing such image is

$$\boldsymbol{x}^* = \boldsymbol{x} - \epsilon \text{sign}(\nabla_x \mathcal{L}(\boldsymbol{x}, y_{target})).$$

Again, there is an iterative version successful in more than 99% cases, according to Kurakin et al. [2017].

Saliency maps

Papernot et al. [2016a] describe an attack using the forward derivative and *saliency maps*. Saliency maps were described by Simonyan et al. [2014]. It is a tool used in image classification for identifying the influence of each pixel on the predicted class. Considering a model with a classification function $h_k(\boldsymbol{x})$, where $\boldsymbol{x}$ is an input image and k is a class, then computing an influence of each pixel of the input image is problematic given high non-linearity of the classification

[3]The method of momentum was designed to accelerate learning with backpropagation algorithm. For more details, see Goodfellow et al. [2016].

function. However, using a Taylor expansion, we can get a linear approximation in the neighborhood of input image $\boldsymbol{x}_0$:

$$h_k(\boldsymbol{x}) \approx \boldsymbol{w}^\top \boldsymbol{x}, \text{ where } \boldsymbol{w} = \left.\frac{\partial h_k}{\partial \boldsymbol{x}}\right|_{\boldsymbol{x}_0}.$$

According to Simonyan et al. [2014], the saliency map for the given input image is created in two steps. First, the derivative $\boldsymbol{w}$ is computed using the back-propagation algorithm. Then, as $\boldsymbol{w}$ is a vector, it is rearranged to correspond the dimensions of the input image. This rearrangement depends on whether the image is grey-scale or multi-channel.

This notion of saliency maps is a visualization tool, but Papernot et al. [2016a] extend it into adversarial saliency maps. The saliency maps used to find such input feature that its increase leads to the desired adversarial image is computed as follows:

$$S(\boldsymbol{x},t)[i] = \begin{cases} 0 \text{ if } \frac{\partial h_t(\boldsymbol{x})}{\partial x_i} < 0 \text{ or } \sum_{j \neq t} \frac{\partial h_t(\boldsymbol{x})}{\partial x_i} > 0 \\ \left(\frac{\partial h_t(\boldsymbol{x})}{\partial x_i}\right) \left| \sum_{j \neq t} \frac{\partial h_t(\boldsymbol{x})}{\partial x_i} \right| \text{ otherwise} \end{cases}$$

where $\boldsymbol{x}$ is an input image, i is an input feature, t is a target class such that $t \neq label(\boldsymbol{x})$ and h is the function computed by the model. As can be seen, the target derivative should be positive, and the overall derivative on other classes should be negative. Then the increase of the given input feature leads to higher output probability for the target class or lower output probability for other classes. The saliency map for finding features which need to be decreased is computed very similarly, where the only difference is in the first line of the equation in comparison symbols:

$$S(\boldsymbol{x},t)[i] = \begin{cases} 0 \text{ if } \frac{\partial h_t(\boldsymbol{x})}{\partial x_i} > 0 \text{ or } \sum_{j \neq t} \frac{\partial h_t(\boldsymbol{x})}{\partial x_i} < 0 \\ \left(\frac{\partial h_t(\boldsymbol{x})}{\partial x_i}\right) \left| \sum_{j \neq t} \frac{\partial h_t(\boldsymbol{x})}{\partial x_i} \right| \text{ otherwise} \end{cases}$$

The algorithm for computing adversary image needs five inputs: clean image $\boldsymbol{x}$, which will be turned into adversary $\boldsymbol{x}^*$, target class t, function h, the value of maximum allowed distortion Υ and the value for the change in one feature θ. The whole attack consists of three steps in a loop. First is computing the forward derivative $\nabla h(\boldsymbol{x}^*)$, then the saliency map S, and the last step is to modify the feature with the highest value in the map S. The loop is repeated until the adversarial input has the target class t; thus $h(\boldsymbol{x}^*) = t$, or until the maximum distortion is reached; thus $\|\boldsymbol{x} - \boldsymbol{x}^*\| \geq \Upsilon$. We can see that this attack is targeted and multi-step.

C&W's Attack

An attack described in Carlini and Wagner [2017] was created as a response to defensive distillation, which is one of the defenses specified later in this chapter. The authors used the gradient method and did a comprehensive search for the best settings of their attack. The ideas behind this powerful attack are as follows.

The authors started with the basic formulation of the problem, where we want to minimize the distance between benign and adversarial input, such that classification of the adversarial example is as requested, meaning $C(\boldsymbol{x}+\boldsymbol{\eta}) = t$, and the constraint $\boldsymbol{x}+\boldsymbol{\eta} \in [0,1]^m$ holds. As the first constraint is highly non-linear, the author reformulated the problem so that it could be better optimized. They defined a function f, such that $C(\boldsymbol{x}+\boldsymbol{\eta}) = t$ holds if and only if $f(\boldsymbol{x}+\boldsymbol{\eta}) \leq 0$. This change leads to an alternative formulation of the problem:

$$\text{minimize } \mathcal{D}(\boldsymbol{x}, \boldsymbol{x}+\boldsymbol{\eta}) + c \cdot f(\boldsymbol{x}+\boldsymbol{\eta})$$

$$\text{such that } \boldsymbol{x}+\boldsymbol{\eta} \in [0,1]^m$$

where c is a suitable constant. Regarding the function f, the authors tried seven possible choices, and the most effective one was

$$f(\boldsymbol{x}) = (\max_{i \neq t}(Z(\boldsymbol{x})_i) - (Z(\boldsymbol{x})_t)^+$$

where $Z(\boldsymbol{x})$ are the logits, so it is the output of the layer before the softmax, and $(e)^+$ means $\max(e, 0)$.

The second modification concerns the box constraint $\boldsymbol{x}+\boldsymbol{\eta} \in [0,1]^m$. Szegedy et al. [2014] solved this with L-BFGS, but Carlini and Wagner [2017] investigated three different methods, which allows them to use a wider range of optimization methods. For the L_2 attack, they chose a change of variables. This means that instead of optimizing over $\boldsymbol{\eta}$, they optimized over $\boldsymbol{w}$, such that $\eta_i = \frac{1}{2}(\tanh(w_i) + 1) - x_i$. The complete attack for L_2 metric is:

$$\text{minimize } \|\frac{1}{2}(\tanh(\boldsymbol{w}) + 1) - \boldsymbol{x}\|_2^2 + c \cdot f(\frac{1}{2}(\tanh(\boldsymbol{w}) + 1),$$

where f is defined as

$$f(\boldsymbol{x}) = \max(\max\{Z(\boldsymbol{x})_i : i \neq t\} - Z(\boldsymbol{x})_t, -\kappa).$$

This loss function is based on the function described before, and κ is used to control the confidence of the misclassification. In Chapter 3, we denote this function as the margin loss.

For more details about L_0 and L_∞ attacks, see Carlini and Wagner [2017].

2.2.2 Black box attacks

When attacking a model without any knowledge of its architecture, parameters or training set, the basic approach is to exploit transferability, thus obtain a different independent training set and train a new model with it. The new training set should come from the same distribution as the original one. This should lead to a new model with very similar decision boundaries as the original model. Adversarial examples are then crafted for the new model using some white box attack. The disadvantage of this approach is its time consumption as mostly it is needed to train on an extensive dataset to obtain a model most similar to the original one.

Jacobian-based Augmentation

Papernot et al. [2017] suggest a more elaborate method that uses synthetic dataset and should be possible to use in practical cases. Authors consider an attack where the only information an attacker has is the labels assigned by the original network to specific inputs. This original network, which ought to be deceived, is perceived as an oracle. The attacker is able to ask the oracle about any input and get a relevant label. Of course, the best substitute of the oracle can be trained when we ask on every single element from a relevant input domain of the given oracle, but this approach is impracticable. Instead, the authors suggest a method of producing such inputs, which leads to the best exploration of the oracle's decision boundaries. This method is called *Jacobian-based Dataset Augmentation*, and as the name suggests, it uses a Jacobian matrix of the substitute model.

The whole algorithm for training the substitute model M works in training epochs. In the beginning, we have to choose the architecture for model M, a number of training epochs, and initial training set T_0. The architecture details of M, such as the number or size of layers, do not significantly impact the performance of the resulting attack. The initial training set could be a randomly selected subset of the input domain. In each epoch, the current dataset is labeled with the oracle, and on the labeled dataset, the substitute model M is trained. The dataset is then augmented using a Jacobian matrix of M:

$$T_{t+1} \leftarrow \{\boldsymbol{x} + \lambda \cdot \operatorname{sgn}(J_M[\tilde{O}(\boldsymbol{x})]) : \boldsymbol{x} \in T_t\} \cup T_t,$$

where T_t is the current dataset, $\tilde{O}(\boldsymbol{x})$ is the label assigned by the oracle to the input $\boldsymbol{x}$, $J_M[\tilde{O}(\boldsymbol{x})]$ is Jacobian matrix dimension which corresponds to this label and λ is a parameter influencing how much the new input will be changed. With this augmentation method, the new dataset should contain inputs that are close to the decision boundaries. Therefore the final substitute M should mimic the behavior of the original oracle.

Zeroth Order Optimization

Another black box attack does not exploit the transferability of adversarial examples at all. Instead, the authors in Chen et al. [2017] used the zeroth order optimization method[4]. Their attack is based on white box C&W's attack, but in the practical black box setting, where an attacker has only access to inputs and outputs of the targeted network, two changes needed to be done.

First, the objective function in C&W's attack uses the logit layer representation, which is absent in the black box scenario. Instead, the authors modified the function, so it uses only the output of the targeted network and label of the desired class:

$$f(\boldsymbol{x}, t) = \max\{\max_{i \neq t} \log[h(\boldsymbol{x})]_i - \log[h(\boldsymbol{x})]_t, -\kappa).$$

The second change concerns the computation of an approximate gradient. Since standard backpropagation is not applicable, the authors use for gradient estimation the symmetric difference quotient:

$$\frac{\partial f(\boldsymbol{x})}{\partial x_i} \approx \frac{f(\boldsymbol{x} + c\boldsymbol{e}_i) - f(\boldsymbol{x} - c\boldsymbol{e}_i)}{c},$$

[4]These are so-called derivative-free methods where no derivation is computed.

where c is a small constant and $\boldsymbol{e}_i$ is a basis vector with one in i-th component. With only a small adjustment in the formula, they also compute Hessian approximation:

$$\frac{\partial^2 f(\boldsymbol{x})}{\partial x_i^2} \approx \frac{f(\boldsymbol{x} + c\boldsymbol{e}_i) - 2f(\boldsymbol{x}) + f(\boldsymbol{x} - c\boldsymbol{e}_i)}{c^2}$$

Relatively extensive computational cost was solved with the proposed algorithm based on Adam, a method for stochastic optimization by Kingma and Ba [2015], and stochastic coordinate descent.

Boundary Attack

The Boundary Attack described by Brendel et al. [2018] is a black box attack that uses only information about the decision of the target model without specific probabilities. The basic idea is to start with high perturbation, which is gradually decreased. The initial adversarial example is either a randomly generated input in case of a non-targeted attack or a sample of the target class in case of a targeted attack. The perturbation is being adjusted using a simple rejection sampling algorithm to be less and less visible. This approach leads to an adversarial example that is very close to the target model's decision boundary.

The advantage of this attack is an amount of required knowledge about the target model and the fact that it does not rely on transferability. It is additionally robust against the defenses based on gradient masking. On the other hand, this kind of black box attack requires more computation, which is the price for its benefits.

Limited queries and imperfect gradient estimation

The last two mentioned and other attacks based on a similar principle of crafting adversarial examples with querying the model deal with numerous queries, representing a problem either time-wise or even regarding actual monetary cost for the queries themselves. Several papers deal with limited queries and try to effectively approximate gradients using evolutionary algorithms. We will discuss these works in the next section.

Ilyas et al. [2019] go further, and besides effective estimation of gradients, they also use prior information about gradients, which brings further improvement of the black box attack. The authors show that for creating a successful adversarial example, the gradient or its estimation is genuinely needed, but this estimation does not need to be perfect. According to their experiment, it suffices to have only 20 % of randomly selected coordinates with an actual gradient to have 60 % success in fooling the classifier with one step PGD. Then, besides the gradient estimation with the least square method, they exploit prior knowledge about the gradient, namely time-dependent and data-dependent priors. Time-dependent priors benefit from the correlation of successive gradients. Data-dependent priors utilize a spatially local similarity in images, which allows incorporating of "tiles". The gradient does not need to be computed for each pixel, but the average of gradients for each tile of size k is enough, which reduces dimensionality by k^2. The whole gradients estimation is built on the framework of bandit optimization, which is a technique for convex optimization based on reinforcement learning.

Discrete optimization

Moon et al. [2019] also aim to effectively craft adversarial examples in the sense of the number of queries. Their approach is although different in the formulation of the problem; instead of continuous optimization, they suggest a discrete surrogate:

$$\max_{\boldsymbol{x}^*} f(\boldsymbol{x}^*) \quad \text{subject to } \boldsymbol{x}^* - \boldsymbol{x} \in \{\epsilon, -\epsilon\}^p$$

where ϵ is the maximum allowed noise in one pixel and p is the number of pixels in the image. This problem can be reformulated as finding the pixels in which the perturbation should be ϵ, and the rest of them should have the perturbation $-\epsilon$, which is NP-Hard. Thus, the authors exploit the accelerated greedy algorithm Lazy-Greedy with a hierarchical approach inspired by tiling.

2.2.3 Evolutionary attacks

Evolutionary attacks differ from previous attacks in the overall approach. White box attacks use exact mathematical formulas to create the adversarial input. Their utilization of gradient methods leads to faster computation but requires knowledge about the model's internals. Evolutionary attacks are suitable for the black box scenario as no gradient needs to be computed. However, as in some of the previous black box attacks, the evolutionary attacks can suffer from high computational cost, which grows with the image size.

One Pixel Attack

Su et al. [2019] describe the evolutionary attack, which differs from others in the strength and focus of perturbation. When considering L_2 attacks, most of them construct adversarial inputs such that total distortion from the original image is minimal but does not restrict the number of changed pixels. Therefore many pixels can be modified in a way that humans cannot notice. On the other hand, one pixel attack aims to modify only one pixel but does not limit the strength of the modification. Actually, paper by Su et al. [2019] is not the first one about an attack where only one pixel is perturbed. Narodytska and Kasiviswanathan [2017] introduced a black box attack, where only one or a few pixels were changed. First, they used completely random perturbation, and for perturbing more pixels, they used a greedy local search. They also showed a connection between pixels found with their local-search procedure and saliency maps.

Nevertheless, this change in the restriction effectively tackles the handicap of evolution regarding the computation cost. Su et al. [2019] utilize differential evolution and further compare the evolutionary approach with a random attack. The evolution gave better results.

Their methodology is the following. An individual consists of a fixed number of perturbations where one perturbation modifies one pixel. Each perturbation contains five elements: coordinates in two-dimensional space and RGB values. There are 400 individuals in each population, and a new generation is produced based on the formula:

$$\boldsymbol{x_i}(t+1) = \boldsymbol{x_a}(t) + F \cdot (\boldsymbol{x_b}(t) + \boldsymbol{x_c}(t)),$$

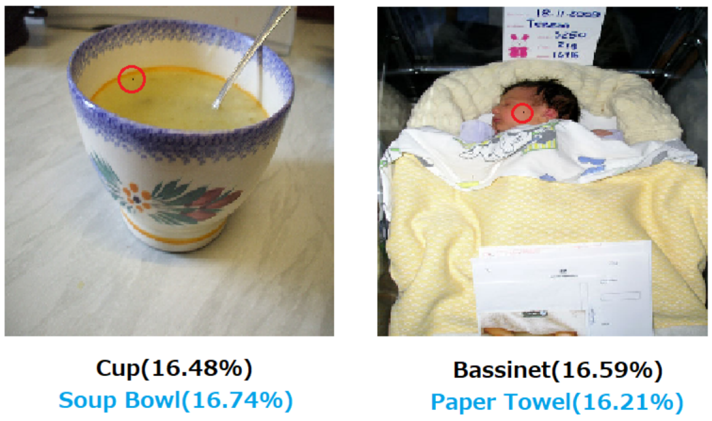
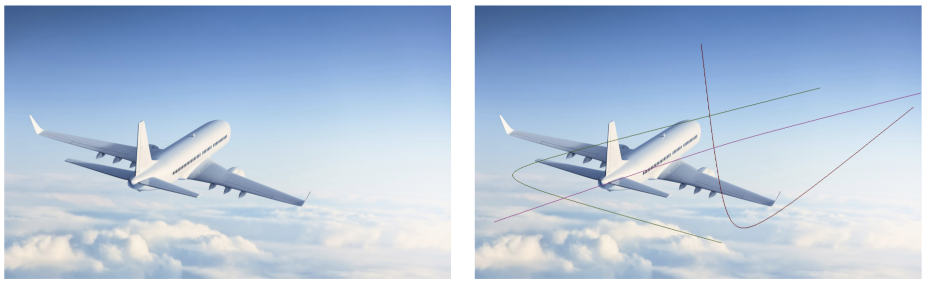


Figure 2.2: Examples of adversarial images created by One Pixel Attack.

where $a \neq b \neq c$, t is the index of the current generation, x_i is an element of one individual, a, b and c are random numbers, and parameter F is set to 0.5. The maximum number of generations is 100, but the authors included early stopping criteria based on the type of attack. In the targeted attack, the evolution stopped when the desired class had more than 50 % probability. In the non-targeted attack, the stopping criterion was a probability of less than 5 % for the true class. This implies that the fitness function was either the probability of the desired or true class. The authors additionally did not use the crossover. Figure 2.2 shows examples of this attack.

Adversarial scratches

The work of Jere et al. [2019] can be seen as an extension of the previous One Pixel Attack. The authors crafted adversarial examples using scratches in the form of simple geometric shapes, either lines or second-degree Bézier curves. Figure 2.3 shows an adversarial example created by this method.



(a) Original Image. (b) Adversarial Image

Figure 2.3: Original image and image perturbed by adversarial scratches technique.

Their L_0 black box attack has two scenarios which are in *network domain* or *image domain* setup. The image domain setup holds a restriction for pixel values, which must lie in the range [0,1] for RGB images. Also, the whole scratch was created in a single color. For this scenario, they used differential evolution. The network domain scenario forces no such restriction; therefore, covariance matrix adaptation evolution strategy was used.

The disadvantage of the high dimensionality of evolutionary black box attacks is, in this case, surpassed by parameterization of the adversarial scratches and focusing on perturbing only a small part of the image. For example, in the image domain setup, the individual is either a 10 or 7-dimensional vector depending on the type of scratch. In the network domain setup, the individual is even smaller as there is no restriction on the pixel's color. In both setups the same fitness functions were used. For targeted attacks, authors wanted to simultaneously increase the confidence of the target class and reduce the confidence of the true class. The logarithm was used for numerical stability and the whole fitness function is this:

$$fit(\boldsymbol{x}) = \alpha \cdot \log(h(\boldsymbol{x})_t) - \beta \cdot \log(h(\boldsymbol{x})_s),$$

where t is a target class, s is a true class and parameters α and β were set to 1 and 50. For non-targeted attacks, the fitness function aimed to maximize the entropy of the predictions:

$$fit(\boldsymbol{x}) = H(\boldsymbol{x}) = -\sum_{i=0}^{K} h(\boldsymbol{x})_i \cdot \log(h(\boldsymbol{x})_i). \tag{2.1}$$

GenAttack

GenAttack is a targeted black box attack of Alzantot et al. [2019], which does not aim to modify only a small fraction of pixels or approximate the gradients, but utilizes basic genetic algorithms with adaptive parameter scaling and dimensionality reduction.

The initialized population consists of examples around the given benign input. Thus, an individual i is computed as $\boldsymbol{x}_{orig} + \mathcal{U}(-\epsilon_{max}, \epsilon_{max})$, where added noise is generated from uniform distribution bounded by maximal possible distortion. The fitness function is similar to the targeted fitness function in adversarial scratches attack by Jere et al. [2019], but instead of minimizing only the probability of the true class, all other probabilities besides the target one are minimized together, therefore

$$fit(\boldsymbol{x}) = \log(h(\boldsymbol{x})_t) - \log \sum_{i=0, i\neq t}^{K} (h(\boldsymbol{x})_i).$$

The authors use roulette selection with the elitism technique of one elite member[5]. The crossover is uniform, where the parent with better fitness has a greater chance to be copied into the child. Precisely, the probability of selecting a value from parent1 is p, the probability of selecting a value from parent2 is $1-p$, where

$$p = \frac{fitness(parent1)}{fitness(parent1) + fitness(parent2)}.$$

[5]Elitism is a strategy where a few best individuals are automatically passed on to the next generation.

Mutation was represented by adding random noise, again uniformly sampled from the range $(-\alpha\epsilon_{max}, \alpha\epsilon_{max})$, and clipping the pixel values to ensure maximal distortion restriction. The probability of mutation and the mutation range α were adjusted during the evolution with an annealing scheme using exponential decay, which led to a lower number of queries.

Dimensionality reduction was achieved by a bilinear resizing technique where the noise is computed in the lower dimension and then scaled up before adding to the original image. This technique is similar to "tiling" by Ilyas et al. [2019]. In both techniques, the adversarial noise vector is condensed, and one value of the adversarial noise vector perturbs more values of the original image, which improves query efficiency. Alzantot et al. [2019] also deal with the number of queries by adjusting the population size in the genetic algorithm. They describe the trade-off between population size and query efficiency, and to achieve a smaller number of queries, they used the population size of six individuals.

Tiling and evolution strategies

Meunier et al. [2019] also apply dimensionality reduction to overcome the weakness of black box evolutionary attacks. They exploit the tiling trick from Ilyas et al. [2019] and found that convolution networks are not robust even to tiled random noise. They optimized continuous and discrete problems for which they combined tiling with evolution strategies, namely the $(1 + 1)$-evolution strategy with one-fifth rule using Cauchy distributions instead of Gaussian sampling. They further used the covariance matrix adaptation evolution strategy with approximating the covariance matrix by a diagonal one for improving computational complexity. They utilized the implementation of evolution strategies from Nevergrad by Rapin and Teytaud [2018] and, according to their experiments, achieved better results in query efficiency than Ilyas et al. [2019] and Moon et al. [2019].

Luo et al. [2019] also applied CMA-ES for the targeted black box attack. Their fitness function was the distance between the benign and adversarial example, and the evolution took until the classification of the adversarial input was as required.

Gradients approximation

One of the black box techniques for attacking neural networks is crafting adversarial examples using gradients approximation. This approach is well suited also for evolutionary algorithms. We can find an effective attack from Ilyas et al. [2018] who applied Natural Evolution Strategies (NES) to specifically tackle the problem of a large number of queries in black box attacks. After they obtained the estimation of gradients, they performed the white box attack, specifically a projected gradient descent (PGD) with momentum[6].

Another black box attack exploiting an evolutionary algorithm for gradients approximation can be found in a paper by Lin et al. [2020]. In this case, the authors utilize differential evolution and iterative FGSM with momentum. For better performance, they propose a *double step size* method for preventing the risk of being stuck in a local optimum, and a *candidate reuse* method for simulating the momentum in the evolutionary setting. The population consists of

[6]Projected gradient descent is a standard method for constrained optimization. In this context, the L_∞ version of projected gradient descent is essentially iterative FGSM.

individuals who represent gradients approximation. Thus, an individual has the same dimensions as input images, and consists of 1 and -1 with the first generation initialized randomly. In contrast with Su et al. [2019], the authors used crossover. The only difference from standard differential evolution is the application of sign function as they are interested only in gradients signs:

$$\boldsymbol{x_i}(t+1) = \operatorname{sign}(\boldsymbol{x_a}(t) + F \cdot (\boldsymbol{x_b}(t) - \boldsymbol{x_c}(t)))$$

Like in previous attacks, the fitness function aims to decrease the probability of the ground-truth label, but this time the authors did not use logarithm and used max instead of sums:

$$fit(\boldsymbol{x}^*) = \begin{cases} h_t(\boldsymbol{x}^*) - \max_{i \neq t}\{h_i(\boldsymbol{x}^*)\} & \text{non-targeted} \\ \max_{i \neq q}\{h_i(\boldsymbol{x}^*)\} - h_q(\boldsymbol{x}^*) & \text{targeted} \end{cases}$$

where $\boldsymbol{x}^*$ is the input image plus the perturbation. The aim is to minimize $fit(\boldsymbol{x}^*)$.

BANA

A (B)lack-box (A)ttack on (N)eural Networks Based on Swarm Evolutionary (A)lgorithm by Liu et al. [2020] exploits the swarm optimization algorithm with the information about the probability of the labels. The fitness function is the combination of the L_p distance between adversary and original input and the loss function $\mathcal{L}$:

$$fit(\boldsymbol{x}^*) = D(\boldsymbol{x}, \boldsymbol{x}^*) + \kappa \cdot \mathcal{L}(\boldsymbol{x}^*),$$

where the loss function depends on the type of attack:

$$\mathcal{L}(\boldsymbol{x}^*) = \begin{cases} \max([h(\boldsymbol{x}^*)]_r - \max_{i \neq r}[h(\boldsymbol{x}^*)]_i, 0) & \text{non-targeted} \\ \max(\max_{i \neq t}[h(\boldsymbol{x}^*)]_i - [h(\boldsymbol{x}^*)]_t, 0) & \text{targeted} \end{cases}$$

and κ is a positive number much larger than $D(\boldsymbol{x}, \boldsymbol{x}^*)$, r is the real label and t is the target label. With this form of the fitness function, the loss is first being pushed towards zero, and then adversarial inputs are crafted to be most similar to the original input.

The length of individuals in a population depends on the input resolution and color as one gene corresponds to one pixel value of one color channel. The initial population was created by adding a small random noise to the original input. With the lower values of the fitness being better, the authors used tournament selection[7]. Further, they applied uniform crossover and Gaussian mutation[8]. The authors also constrained the mutation to speed up the convergence since the final adversarial example has to be very close to the original one. They additionally did not change pixels with 0 value based on the assumption that it is the background.

[7]A selection method where two or more individuals are selected at random, and the best one is passed on to the next generation.

[8]With Gaussian mutation, a random number generated from a normal distribution is added to the mutated value in the individual.

2.3 Defenses

Yuan et al. [2019] divide defenses into two types: reactive and proactive. In reactive techniques, the training process is not adjusted, and defense consists of detecting and then rejecting adversarial samples. On the other hand, proactive techniques modify the training process to make neural networks robust against adversarial inputs, which leads to correct classification. This section focuses on two proactive defense methods, which adversaries try to break most often. Then we describe one reactive method, which is relatively new, and combine two crucial insights to detect adversarial samples.

2.3.1 Proactive defenses

Adversarial training

A method that immediately comes in mind when considering making the neural network more robust against adversarial examples is adversarial training. The essence of the method lies in the augmentation of the training set by constantly adding new adversarial examples. Szegedy et al. [2014] used this method in such a manner that they created adversarial inputs for each layer. As mentioned before, Goodfellow et al. [2015] also applied this technique by modifying the loss function with FGSM:

$$\tilde{\mathcal{L}}(\boldsymbol{\theta}, \boldsymbol{x}, y) = \alpha \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{x}, y) + (1-\alpha)\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{x} + \epsilon \operatorname{sign}(\nabla_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{x}, y))).$$

This led to reducing an error rate from 89.4 % to 17.9 %. Kurakin et al. [2017] scaled adversarial training with batch normalization to large models and datasets, also using FGSM.

Unfortunately, adversarially trained networks are not very robust against black box attacks and even against white box multi-step attacks. Therefore Tramèr et al. [2018] introduced *Ensemble Adversarial Training*, which injects adversarial samples created not only on the trained network but also transferred from other pre-trained models. Xie et al. [2019] suggest another improvement. Besides adversarial training with PGD, they use *denoising blocks*, which are added to the convolutional architecture.

Defensive distillation

Another proactive defense aims against white box attacks as it masks the gradients. It was proposed by Papernot et al. [2016b] and is inspired by the distillation technique used for knowledge transfer between different networks.

Distillation was designed to distill information encoded in parameters of a more extensive network into the parameters of a network with smaller architecture. It is performed in two steps. In first step, a large network with a softmax output layer is trained in a usual way. It is important that the output vector of the softmax layer is computed using a distillation temperature T:

$$h(\boldsymbol{x}) = \left[\frac{e^{z_i(\boldsymbol{x})/T}}{\sum_{l=0}^{N-1} e^{z_l(\boldsymbol{x})/T}} \right]_{i \in 0 \ldots N-1}$$

The output vectors of probabilities are used as *soft labels* for training the smaller network in the second step of the process. Soft labels encode knowledge of the higher capacity model. Higher temperature increases the entropy and thus the information in the soft labels. The temperature should therefore be larger than one. It holds that probabilities in the output vector converge to $1/N$ as $T \to \infty$.

Defensive distillation does not aim to compress the network but to make it more robust. Therefore, the same architecture is trained in both steps of the process. The resulting distilled model generalizes better around training samples. Even though the authors have promising results with their approach, some of the described attacks succeeded in cracking it. For example Carlini&Wagner attack was crafted especially for this defense. Also, the black box attack BANA broke defensive distillation.

2.3.2 Reactive defenses

In this section, we describe one reactive defense characterized by the ability to detect adversarial inputs even though the adversary has full knowledge of the defense. We have also chosen this defense as the one that we aim to deceive due to its interesting approach, so we describe its implementation in detail and also the authors' results.

Yu et al. [2019] draw on two properties of natural images, which may seem to be contradicting. Firstly, natural images are robust to random noise, as pointed out by Szegedy et al. [2014]. It means that if we add i.i.d. Gaussian noise to the image, it should not change the classification. Secondly, natural images also probably lie near the decision boundaries. According to Yu et al. [2019], adversarial images mostly violate at least one of these attributes. This assumption leads to the defense based on the testing of both of them. The authors designed the defense against white box and gray box[9] attacks as they say they are more powerful than black box attacks.

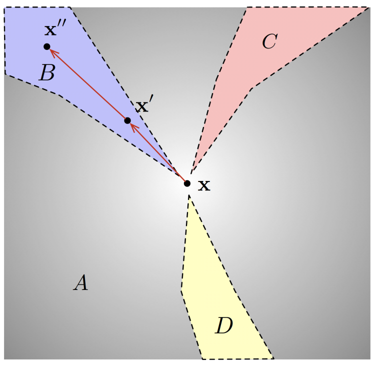


Figure 2.4: Schema of adversarial pockets near a benign image $\boldsymbol{x}$.

Robustness to random noise of the input $\boldsymbol{x}$ is tested by sampling $\boldsymbol{\epsilon} \sim N(0, \sigma^2 I)$ of the same dimension, where σ is a hyperparameter, and then computing the change in classification after adding $\boldsymbol{\epsilon}$ to $\boldsymbol{x}$. The authors denote this value as Δ:

$$\Delta = \|h(\boldsymbol{x}) - h(\boldsymbol{x} + \boldsymbol{\epsilon})\|_1 \tag{2.2}$$

[9]The adversary is not aware of the defense mechanism but has full access to the classifier.

The input $\boldsymbol{x}$ is rejected as adversarial if Δ is high enough. Figure 2.4 shows a schematic illustration of a decision boundary near an input $\boldsymbol{x}$ in high dimensional space. By creating an adversarial sample $\boldsymbol{x}^*$ classified as class B, we get an image whose classification can be corrected by random perturbation in most directions. However, if there was only this criterion, an adversary could change a loss function and create a sample $\boldsymbol{x}^n$ deeper in the pocket and less sensitive to random noise.

Therefore, the authors chose close proximity to decision boundary as the second criterion. It is tested with some first-order iterative attack algorithm $\mathcal{A}$ (the authors used PGD and C&W), where a defender records the minimum number of steps K needed to perturb $\boldsymbol{x}$. Again, the input $\boldsymbol{x}$ is rejected as adversarial if K is high enough. Furthermore, this part can be divided into targeted and non-targeted attacks depending on whether we measure a number of steps to a specific class K_t or not K_u.

The whole whole defense mechanism can be described by the following algorithm:

```
Input: An image $\boldsymbol{x}$
$\Delta \leftarrow$ change in classification defined by Eq. 2.2;
$K_t \leftarrow$ number of targeted steps required to adversarially perturb $\boldsymbol{x}$;
$K_u \leftarrow$ number of untargeted steps required to adversarially perturb $\boldsymbol{x}$;
if $\Delta \geq T_\Delta$ or $K_t \geq T_{K_t}$ or $K_u \geq T_{K_u}$ then
    reject $\boldsymbol{x}$ as adversarial;
end
else
    accept $\boldsymbol{x}$ as benign ;
end
```

Algorithm 3: Detection Algorithm

T_Δ, T_{K_t} and T_{K_u} are respective thresholds. Thus, the image is rejected as adversarial if it exceeds at least one of these thresholds.

Implementation The authors implemented their solution in Python 3 using PyTorch and numpy packages[10]. Their gray box attack is implemented as a standard white box PGD or C&W attack with their basic losses denoted as $\mathcal{L}_1$. For the white box attack, they created "best effort adversary" by adding three more losses to bypass both noise detection ($\mathcal{L}_2$) and the number of steps detection for targeted ($\mathcal{L}_3$) and non-targeted ($\mathcal{L}_4$) attacks:

$$\mathcal{L}_1 = \mathcal{L}(h(\boldsymbol{x}^*), \boldsymbol{p}^{adv}),$$

$$\mathcal{L}_2 = \mathbb{E}_{\epsilon \sim N(0,\sigma^2 I)}\left[\|h(\boldsymbol{x}^*) - h(\boldsymbol{x}^* + \boldsymbol{\epsilon})\|_1\right],$$
$$\mathcal{L}_3 = \mathbb{E}_{y' \sim \text{Uniform}, y' \neq y_t}\left[\mathcal{L}(h(\boldsymbol{x}^* - \alpha \boldsymbol{\delta}_{y'}), y')\right],$$
$$\mathcal{L}_4 = -\mathcal{L}(h(\boldsymbol{x}^* + \alpha \boldsymbol{\delta}_{y_t}), y_t),$$

where $\mathcal{L}(\cdot,\cdot)$ is the cross-entropy loss in PGD and the margin loss in CW attack, $\boldsymbol{p}^{adv}$ is a probability vector identical to $h(\boldsymbol{x})$ in every dimension but the true class

[10]Implementation of the defense is available on `https://github.com/s-huu/TurningWeaknessIntoStrength`.

y_t and target class y', whose probabilities are swapped. $\boldsymbol{\delta}_{y'}$ denotes the gradient of $\mathcal{L}$ with respect to $\boldsymbol{x}^*$, $(\boldsymbol{x}^* - \alpha\boldsymbol{\delta}_{y'})$ is the one-step move towards class y' at the step size α and $(\boldsymbol{x}^* + \alpha\boldsymbol{\delta}y_t)$ is one gradient step away from the true class y_t. The complete adversarial loss becomes

$$\mathcal{L}^* = \lambda\mathcal{L}_1 + \mathcal{L}_2 + \mathcal{L}_3 + \mathcal{L}_4, \tag{2.3}$$

where λ controls the domination of $\mathcal{L}_1$ so that the adversarial example generation would be successful.

They used two datasets for their experiments: CIFAR-10 test set and ImageNet validation set from 2012. CIFAR-10 was used with the VGG-19 model, which authors trained by themselves. ImageNet was used with two models pre-trained in PyTorch: ResNet-101 and Inception-v3. Images from ImageNet were resized to 256 × 256 pixels and then cropped at the center, so the final dimensions are 224 × 224 pixels. The authors used L_∞-bound $\tau = 0.1$ for all attacks, which determines the maximally allowed distortion for one pixel. 0.1 distortion is relatively high and recognizable by the human eye.

Results As with many statistical tests and machine learning models, there is a competition between *true positive rate (TPR)* and *false positive rate (FPR)*. This defense could have been 100 % effective in detecting adversarial examples if it would have rejected all images. This would have mean 100 % TPR, but also 100 % FPR. As the goal is to have 0 % FPR, or the smallest possible percentage, the authors tested two levels of FPR, 10 % and 20 %. They experimented with attack settings, but the most significant impact on the defense effectivity had the learning rate used in loss function $\mathcal{L}_1$ during the attack. Table 2.1 shows the best and the worst detection rates for all combination of models, attacks and FPR.

Model	Attack	FPR	Detection Rate			
			Best	(LR)	Worst	(LR)
VGG-19	PGD	0.2	100 %	(0.001)	66 %	(0.1)
VGG-19	PGD	0.1	97 %	(0.001)	19 %	(0.1)
VGG-19	CW	0.2	74 %	(0.1)	37 %	(0.001)
VGG-19	CW	0.1	57 %	(0.1)	26 %	(0.001)
ResNet-101	PGD	0.2	61 %	(0.1)	49 %	(0.03)
ResNet-101	PGD	0.1	38 %		26 %	(0.03)
ResNet-101	CW	0.2	81 %		69 %	(0.01)
ResNet-101	CW	0.1	60 %		48 %	(0.03)
Inception-v3	PGD	0.2	76 %	(0.01)	47 %	(0.1)
Inception-v3	PGD	0.1	52 %		20 %	
Inception-v3	CW	0.2	79 %		48 %	
Inception-v3	CW	0.1	64 %		20 %	

Table 2.1: Best and worst detection rates with their relevant learning rates from the paper. The detection rate depends on the model, attack and false positive rate.

The authors had to determine the thresholds for the maximal change in prediction Δ defined by Equation 2.2, and the maximal number of steps towards or away from a required class T_{K_t} and T_{K_u}. The thresholds are in Table 2.2. The maximal number of steps for untargeted attacks is set to 10000 for all attacks with the comment in the code that it is possible to lower the number to 1000 or neglect it at all to save computation time.

Model	FPR	T_Δ	T_{K_t}
VGG-19	0.2	0.0006	89
VGG-19	0.1	0.009	119
ResNet-101	0.2	1.77	22
ResNet-101	0.1	1.90	35
Inception-v3	0.2	1.83	26
Inception-v3	0.1	1.95	57

Table 2.2: Original thresholds for the monitored metrics depending on the model and false positive rate.

We can see a significant difference in thresholds between models trained on ImageNet and VGG-19 trained on CIFAR-10. Yu et al. [2019] claim that models trained on CIFAR-10 are less robust to added random noise because of low diversity of training images. Furthermore, due to lower dimensionality, natural images are farther from the decision boundary than samples from ImageNet. Also, the criterion for the number of steps in untargeted attacks was ineffective because VGG-19 over-saturates predicted probabilities for natural images. These reasons lead to worse performance of the defense and realization that not all images satisfy the authors' hypotheses about natural samples. On this basis, we do not perform evolutionary attacks on CIFAR-10 dataset because creating adversarial images on ImageNet should be more challenging in all aspects.

One last note to original results concerns gray box attacks performed on ImageNet. With the L_∞-bound $\tau = 0.03$, the authors reached a detection rate 97.6% on PGD and 98.1% on CW at 5% FPR. These are very good results compared to white box scenario. Also, the detection rate is better for $\tau = 0.03$ than $\tau = 0.1$. This may seem surprising at first sight, but we have to understand that although humans are able to visually distinguish images with higher adversarial noise, when we create an adversarial example with stricter noise limitation, the algorithm can more easily break one of the two hypotheses about natural images. We will see this effect later in the experiments.

3. Evolutionary Approach to Creating Adversarial Examples

This chapter describes our approach that applies evolutionary algorithms to craft adversarial examples. Our goal is not only to fool a clean model without any defense but also to evade the chosen detection mechanism specified in Section 2.3.2. Since the chosen defense counts the number of gradient steps towards the decision boundary, which is not a differentiable problem, evolutionary algorithms are very suitable for acquiring a solution. We perform the black box attack because we use only output probabilities of the attacked model.

The previous work by Meunier et al. [2019] used tiling trick and evolutionary strategies. We decided to apply differential evolution explained in Section 1.3.2 due to its ability to solve non-separable and highly conditioned problems. Thanks to its specific mutation, differential evolution is also invariant to rotations and scaling of the search space. Although CMA-ES has also this property, the operations of differential evolution are simpler and faster, especially with longer vectors. Differential evolution is additionally easier to implement.

3.1 Evolutionary Setup

One individual represents an array of size $3 \times$ `num_tiles` $\times$ `num_tiles`, where `num_tiles` is the number of tiles per image side and 3 is the number of channels in the image. To calculate the fitness value, we translate an individual into an adversarial sample, as described in Algorithm 4. We remind that the images have $3 \times 224 \times 224$ pixels and τ denotes L_∞ bound of the adversarial noise η. Previous works used *nearest-neighbor interpolation*, resulting in uniform squares of noise. This interpolation is most easily recognized by the human eye, especially at higher noise levels. We performed our experiments mostly with this type of interpolation, but after tuning the hyperparameters of evolution, we also tested *bilinear* and *bicubic* interpolations. These types of interpolation appear less disturbing, as can be seen from the results.

```
Input: An image $\boldsymbol{x}$, an individual $\boldsymbol{i}$, int: num_tiles, float: $\tau$
$\boldsymbol{\eta} \leftarrow \boldsymbol{i}$ reshaped into $3 \times$ num_tiles $\times$ num_tiles tensor;
for each value $\eta_i$ in $\boldsymbol{\eta}$ do
    if $|\eta_i| > \tau$ then
        $\eta_i \leftarrow \mathrm{sign}(\eta_i) \cdot \tau$;
    end
end
if num_tiles $< 224$ then
    interpolate $\boldsymbol{\eta}$ to the size of $3 \times 224 \times 224$ ;
end
Output: $\boldsymbol{x} + \boldsymbol{\eta}$
```

Algorithm 4: Creating an adversarial sample from an individual

Regarding population size, we inspire ourselves by results of Alzantot et al.

[2019], who used very small populations for query efficiency. We also tried a small population size of 10 individuals, but it took too long to find suitable adversarial noise. Therefore, we set up a population size of 30 individuals. The number of generations is set to 500, but we use early stopping in later experiments. It means that if the fitness of the best individual did not improve for 30 or 15 generations, we stopped the evolution.

The considerable distinction in our evolutionary setting compared to previous works is the fitness function. Chapter 2 provided a basic overview of evolutionary attacks. All of them used the output vector of probabilities for fitness computation, some added the distance between original and adversarially perturbed sample. Since previous attacks did not aim to bypass any defense, their choice of fitness functions is reasonable. Our experiments have shown that, for example, the entropy $H(\boldsymbol{x})$ defined by Equation 2.1 is well optimized by evolutionary algorithms. However, our approach had to be different. We aimed to circumvent detection, which counts the number of steps toward the decision boundary for targeted (K_t) and non-targeted (K_u) attacks and checks the difference in classification of the image after adding random noise (Δ defined by Eq. 2.2). Therefore, our first fitness function was a weighted sum of these metrics. We also added L_2 distance to obtain an adversarial image close to the original.

```
if true label == adv label then
    return MAX;
end
else
    return A * $L_2$ distance + B * $\Delta$ + C * $K_t$ + D * $K_u$;
end
```

Algorithm 5: Pseudocode of the first tested fitness function.

Algorithm 5 describes our first tested fitness function. The goal of evolution is to minimize fitness values. Letters A, B, C and D represent the weights and a constant MAX was set to 10000. *Adv label* denotes the label of the adversarial image. As long as the adversarial label equals the true label of the original image, the attack is not successful and we continue in searching the right adversarial noise. Once the adversarial image succeeds, we minimize the weighted sum of the specified metrics.

The results were promising and differential evolution could deceive the network even with adversarial noise of lower level τ than 0.1 used by the authors of the defense. However, we encountered two issues. First, the change in classification Δ is random, and only one good value during the fitness computation can be a coincidence. Therefore, we decided to add random Gaussian noise to the image ten times, which gave us ten different values of Δ. We computed the minimum, the maximum and the average of these values. We denote them as Δ_{min}, Δ_{max} and Δ_{avg}. The fitness value is further computed with Δ_{max}, which should prevent a randomly good result of Δ.

Second, we have discovered that evolution is mostly unable to find images far from decision boundaries. Even with a weight setting that should force evolution to reduce Δ at the cost of increasing the number of steps, the number of steps towards the boundary was still low. This observation leads us to the conclusion that images far from decision boundary are hard to find naturally. The hypothesis of Yu et al. [2019] that adversarial images with low values of Δ have high values of K seems to be correct only for adversarial images created by gradient attacks. It is hard to get far from the decision boundary without information about the direction to go. Thus, we removed values of K_t and K_u from the fitness.

Next, we hypothesized that adding entropy $H(\boldsymbol{x})$ to the fitness function could help to achieve a stable adversarial image, meaning more robust to added random Gaussian noise. The lower entropy of the probability vector $\boldsymbol{y} = h(\boldsymbol{x})$ means that the model is more certain with its prediction. Therefore, the prediction could be more robust to added noise. To support this idea, we computed values of Δ_{avg}, K_t and K_u for the original images selected for both Imagenet-v3 and ResNet-101. For each image $\boldsymbol{x}$, we also computed the entropy of a probability vector $h(\boldsymbol{x})$, where h denotes either Imagenet-v3 or ResNet-101. We denote this value as H_{orig}. Correlation matrices of these variables for both models are in Figure 3.1.

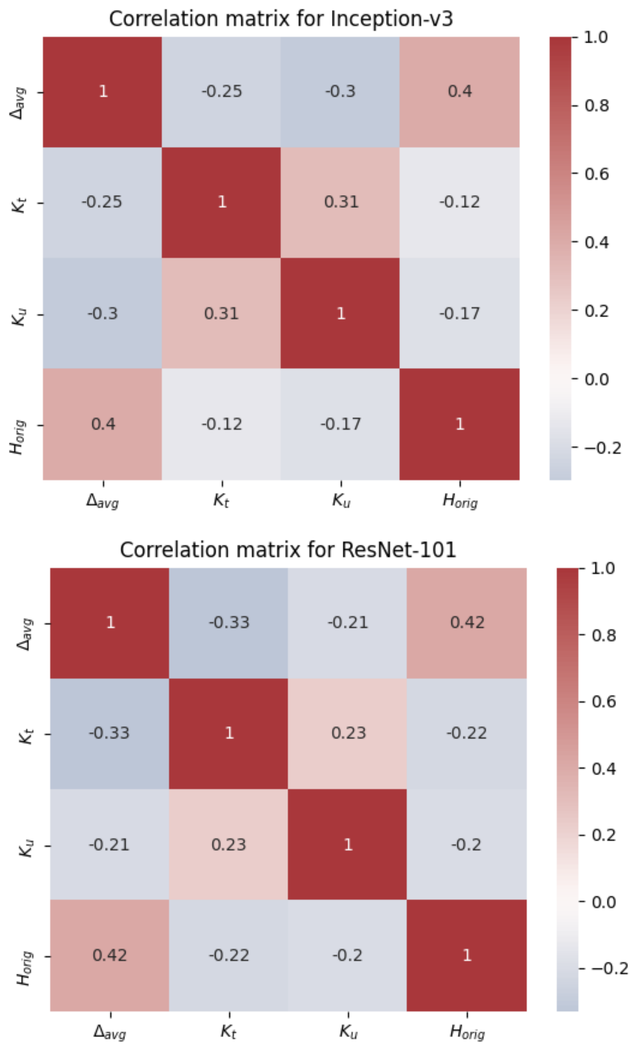


Figure 3.1: Correlation matrices of H_{orig}, K_t, K_u and Δ_{avg} of original images correctly classified by Inception-v3 and ResNet-101.

We see a positive correlation between H_{orig} and Δ_{avg}. There are also negative correlations between both K_t and K_u and Δ_{avg}. It means that the lower the classification change after adding random Gaussian noise, the higher the number of steps toward a decision boundary, and vice versa. This imply that the observations by Yu et al. [2019] about the adversarial samples also partially applies to the benign images.

```
if $true\ label == adv\ label$ then
    return MAX;
end
else
    return A * $L_2$ distance + B * $\Delta_{max}$ + C * $H_{adv}$;
end
```

Algorithm 6: Pseudocode of the second tested fitness function.

Due to the positive correlation between H_{orig} and Δ_{avg} in the original images, we decided to measure the entropy of a probability vector also for adversarial images. We denote this value as H_{adv}. We then tested second fitness, which returned either a constant 10000 or weighted sum of L_2 distance, Δ_{max} and H_{adv}.

Unfortunately, we were not able to reach values of Δ_{avg} close to zero. Therefore, we tested the fitness, which returned only Δ_{max}, but we still found it difficult for evolution to minimize it. Thus, we plotted Δ_{avg} and H_{orig} for the original images to see if it is possible to minimize classification change towards zero.

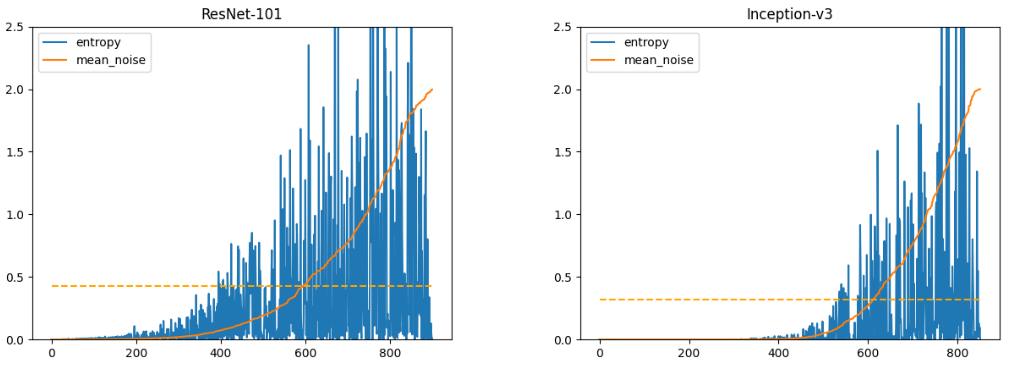


Figure 3.2: Values of Δ_{avg} and H_{orig} for the original images correctly classified by Inception-v3 and ResNet-101. The dashed orange line denotes the mean of Δ_{avg} for all images. The images are sorted in ascending order by Δ_{avg} value, thus the x-axis determines the order of the images according to this value.

Figure 3.2 shows that there are about 75 % of benign images whose value of Δ_{avg} is above average. Approximately half of the original images for both models have value of Δ_{avg} close to zero, which cohere to very low value of H_{orig}. However, the other half have higher values of both variables, and we also see a significant difference between both models. Thus, we conclude, that low values of Δ_{avg} close to zero are not necessary even for the benign images.

In the next phase of the experiments, we tested the second fitness on 50 different images chosen at random. Although the fitness function was successful in the one chosen image of a tiger shark, it was not as successful in other images. One problem was in returning a constant when the model was not deceived.

Images with lower entropy and higher confidence in prediction were harder to adversarially change and evolution did not know the right direction. Therefore, we change this part to return the probability of the true class multiplied by 10000. We also exchange entropy for the probability of the adversarial class, which we subtracted from other values. This change also minimized entropy, but at the same time better minimized the change in classification after adding random noise. The whole fitness is described by Algorithm 7. A denotes a weight which was set to 10 or 100.

```
if true label == adv label then
    return probability of true class * 10000;
end
else
    return 10 * $L_2$ distance + 100 * $\Delta_{max}$ - A * probability of adv class;
end
```

Algorithm 7: Pseudocode of the third tested fitness function.

Last fitness function is described by Algorithm 8. We used it in order to increase the transferability of adversarial examples between models. There is an extra bonus for deceiving an adversarially trained ResNet-152. We used this fitness function only with the pre-trained ResNet-152 from PyTorch.

```
if true label == adv label then
    return probability of true class * 10000;
end
else if true label == resnet adv label then
    return 10 * $L_2$ distance + 100 * $\Delta_{max}$ - 100 * probability of adv class;
end
else
    return 10 * $L_2$ distance + 100 * $\Delta_{max}$ - 100 * probability of adv class
     - 400;
end
```

Algorithm 8: Pseudocode of the last tested fitness function.

The whole pseudocode of creating an adversarial image from the benign image $\boldsymbol{x}$ describes Algorithm 9. All constants were set experimentally.

3.2 Experimental Setup

We perform our experiments against two networks used in the original paper by Yu et al. [2019], Inception-v3 and ResNet-101, both pre-trained in PyTorch. The first tests are against Inception-v3 on one image chosen at random. We selected Inception-v3 as it is harder to fool. The chosen sample is an image of a tiger shark.

First tests on one image were used to find the best fitness and population size. After that, we choose 50 images at random for both networks. We experiment with parameters of differential evolution, level of noise τ and the number of tiles per image side to find the best setting.

In this phase, we added the third model, ResNet-152, to test transferability. We chose this model because it is available pre-trained in Pytorch, but also adversarially trained by Xie et al. [2019][1].

The experiments' final part is computing adversarial examples for all correctly classified images from sample of 1000 images against all three networks and analyzing results. Images for Inception-v3 and ResNet-101 are chosen the same way as in the original paper by Yu et al. [2019]. Images for ResNet-152 are chosen at random to have greater diversity in image classes.

The next chapter describes our results.

```
Input: An image $\boldsymbol{x}$, int: num_tiles
$t \leftarrow 0$;
$P_t \leftarrow$ 30 individuals size $3 \times$ num_tiles$^2$ generated at random;
best_fit $\leftarrow$ 10000;
best_ind $\leftarrow$ empty array;
while $t < 500$ do
    $P_{t+1} \leftarrow \emptyset$;
    for each individual $\boldsymbol{x}$ in $P_t$ do
        $\boldsymbol{x}_a, \boldsymbol{x}_b, \boldsymbol{x}_c \leftarrow$ individuals selected from $P_t$ at random;
        $\boldsymbol{v} \leftarrow \boldsymbol{x}_a + F(\boldsymbol{x}_b - \boldsymbol{x}_c)$;
        SP $\leftarrow$ number of safe position generated at random;
        for each position $i$ in $\boldsymbol{x}$ do
            RN $\leftarrow$ real number generated at random from $\langle 0, 1 \rangle$;
            if $RN \leq p_c$ and $i \neq SP$ then
                $\boldsymbol{x}[i] \leftarrow \boldsymbol{v}[i]$;
            end
        end
        if fitness($\boldsymbol{x}$) $\leq$ fitness($\boldsymbol{v}$) then
            add $\boldsymbol{x}$ to $P_{t+1}$;
        end
        else
            add $\boldsymbol{v}$ to $P_{t+1}$;
        end
    end
    best_ind_pop $\leftarrow$ best individual in population;
    if fitness(best_ind_pop) < best_fit then
        best_ind $\leftarrow$ best_ind_pop; best_fit $\leftarrow$ fitness(best_ind_pop);
    end
    $t \leftarrow t + 1$;
    if $t \geq 15$ and best_fit did not improve last 15 generations then
        break;
    end
end
Output: adversarial image created from best_ind
```

Algorithm 9: Evolutionary Attack

[1]Adversarially trained models by Xie et al. [2019] are available on `https://github.com/facebookresearch/ImageNet-Adversarial-Training/blob/master/INSTRUCTIONS.md`.

4. Results

This chapter describes the results of our experiments. First, we try to replicate the outcomes of the original paper by Yu et al. [2019]. Then we proceed with our experiments, which include attacks that exploits tiled random noise and the evolutionary attacks described in the previous section. We also test the fitness function from the attack described by Meunier et al. [2019] and compare how the resulting adversarial images deceive the chosen defense. Finally, we test the transferability of our generated adversarial samples to other models, one of which is adversarially trained ResNet-152.

4.1 Results Replication

We attempted to replicate the authors' results of the detection mechanism. We used their implementation available online[1]. The authors tested their defense on 1000 images for each model. The images were not selected at random but one by one from the beginning which is the reason why there prevail categories of birds and fish in chosen ImageNet samples[2]. More specifically, there are images with labels 0 to 19. As we wanted to replicate the results, so we used the same strategy.

We also had to train our VGG-19 model. The authors provide a script for the training called `train_vgg19.py`. The script also downloads CIFAR-10 dataset available in PyTorch. We used this script without any changes in parameters setting. We trained the model on Google Colab[3] using GPU runtime. Table 4.1 shows our achieved model accuracy for all three architectures.

Model	Accuracy
VGG-19	90.2 %
ResNet-101	90.1 %
Inception-v3	85.1 %

Table 4.1: Our achieved accuracy for all three models from the paper.

From now on we will use two metrics to describe the quality of the attack. First one is *success rate*. This is the percentage of images from which an adversarial sample was successfully created. Success rate of white box attacks that utilize gradient is often very high. Our tests confirmed, that both PGD and CW attacks created an adversarial example from at least 99 % of correctly classified images for all models, although none had 100 % success. Second metric is *detection rate*. This indicates the percentage of adversarial images that bypassed the defense.

Our achieved detection rates for both attacks are in Table 4.2. We used learning rates corresponding to the authors' best results. Besides detection rates, we measured also *accurate false positive rates*. It is the accurate percentage

[1]https://github.com/s-huu/TurningWeaknessIntoStrength

[2]A list of ImageNet categories is available online on `https://gist.github.com/yrevar/942d3a0ac09ec9e5eb3a`.

[3]`colab.research.google.com`

of benign images rejected as adversarial on the given thresholds. We can see a significant difference in accurate false positive rates and target false positive rates for VGG-19. Furthermore, our detection rates for C&W attack against VGG-19 is distinctly lower. The reason for these discrepancies is mainly a differently trained model, as we trained our own without random seed and probably with distinct hyperparameters like the number of epochs. Also, as we have already mentioned, CIFAR-10 dataset is not suitable for the chosen defense, so we will not deal with it further.

Model	Attack	Target FPR	Accurate FPR	Detection Rate	
				Ours	Authors'
VGG-19	PGD	0.2	0.494	0.994	0.998
VGG-19	CW	0.2		0.431	0.740
VGG-19	PGD	0.1	0.285	0.960	0.966
VGG-19	CW	0.1		0.299	0.568
ResNet-101	PGD	0.2	0.152	0.667	0.612
ResNet-101	CW	0.2		0.801	0.809
ResNet-101	PGD	0.1	0.074	0.484	0.378
ResNet-101	CW	0.1		0.680	0.601
Inception-v3	PGD	0.2	0.186	0.827	0.762
Inception-v3	CW	0.2		0.799	0.788
Inception-v3	PGD	0.1	0.106	0.695	0.516
Inception-v3	CW	0.1		0.708	0.635

Table 4.2: Our detection rates and accurate false positive rates for all combinations of model, attack and target FPR, using original thresholds.

Model	Attack	FPR	T_Δ	T_{K_t}	T_{K_u}	Detection Rate
VGG-19	PGD	0.2	0.014	200	1000	0.943
VGG-19	PGD	0.1	0.078	200	1000	0.674
VGG-19	CW	0.2	1.068	149	682	0.739
VGG-19	CW	0.1	1.068	77	1000	0.383
ResNet-101	PGD	0.2	1.984	14	1000	0.781
ResNet-101	PGD	0.1	1.997	20	1000	0.683
ResNet-101	CW	0.2	1.999	14	991	0.883
ResNet-101	CW	0.1	1.997	20	1000	0.817
Inception-v3	PGD	0.2	0.445	200	1000	0.940
Inception-v3	PGD	0.1	1.651	200	1000	0.870
Inception-v3	CW	0.2	0.467	157	1000	0.895
Inception-v3	CW	0.1	1.651	200	1000	0.827

Table 4.3: New thresholds with target FPR equal to accurate FPR and corresponding detection rates.

Regarding ImageNet dataset, we see that the authors' thresholds are rather pessimistic. Attacks against ResNet-101 achieve mostly higher detection rates with lower accurate false positives rates. Attacks on Inception-v3 have also higher detection rates, but differences in accurate false positives rates are not as large. The reason for this differences between ours results and the results by Yu et al. [2019] is the randomness of gradient attacks.

4.1.1 Thresholds computation

Since we want to achieve the same values for accurate and target FPR, we computed our own thresholds for each model and each attack. We used the script provided by the authors in their implementation. Specific values are in Table 4.3. The last column of the table shows the detection rate computed with these thresholds. Of course, there are occasionally differences in thresholds between attacks, and unfortunately, it is not clear how to combine them. For ImageNet, we achieved even higher detection rates due to the authors' pessimistic thresholds. The thresholds for CIFAR-10 are again problematic as there are significant differences between attacks.

Since we are no longer interested in CIFAR-10, let us see the differences between ResNet-101 and Inception-v3 on ImageNet. For ResNet-101, the distinction between 10 % and 20 % FPR is in the value of K_t. On the other hand, the value of Δ makes the difference in detection rate at various FPR for Inception-v3. Additionally, T_Δ for 10 % FPR is significantly lower for Inception-v3 than ResNet-101 and vice versa for T_{K_t}.

These differences can be explained by the nature of the classification space formed by each architecture and the capabilities of the tested white box attacks. We have already seen the differences in H_{orig} and Δ_{avg} for both models in Chapter 3 (Figure 3.2). Figure 4.1 shows histograms of Δ_{avg} for the real images and images created by PGD attack against both models. We see significant differences in the values of adversarial images, where the images against Inception-v3 have mostly high values of Δ_{avg}, while adversarial images crafted against ResNet-101 have significantly lower and more uniformly distributed values.

Figure 4.2 shows the values of K_t. The values are only plotted for a maximum of 60 due to space limitation, but the real images classified with Inception-v3 have values of targeted steps up to 200 as well as adversarial images against ResNet-101. Again, we see a difference between the adversarial examples against Inception-v3 and ResNet-101, where the adversarial images against Inception-v3 have mostly a low values of K_t. On the other hand, the images crafted against ResNet-101 have diversified values of K_t.

According to the graphs, we can conclude that it is problematic to create an adversarial image against Inception-v3 with a high number of steps towards the decision boundary. In fact, the adversarial images have a lower values of K_t than the real images, but at the cost of higher values of Δ_{avg}. On the other hand, the adversarial images against ResNet-101 are not so sharply defined and their values in both metrics are much more uniformly distributed.

The rest of the chapter is divided into three sections according to the tested architecture. We start with attacks against Inception-v3 as it seems more intricate to deceive due to lower T_Δ thresholds.

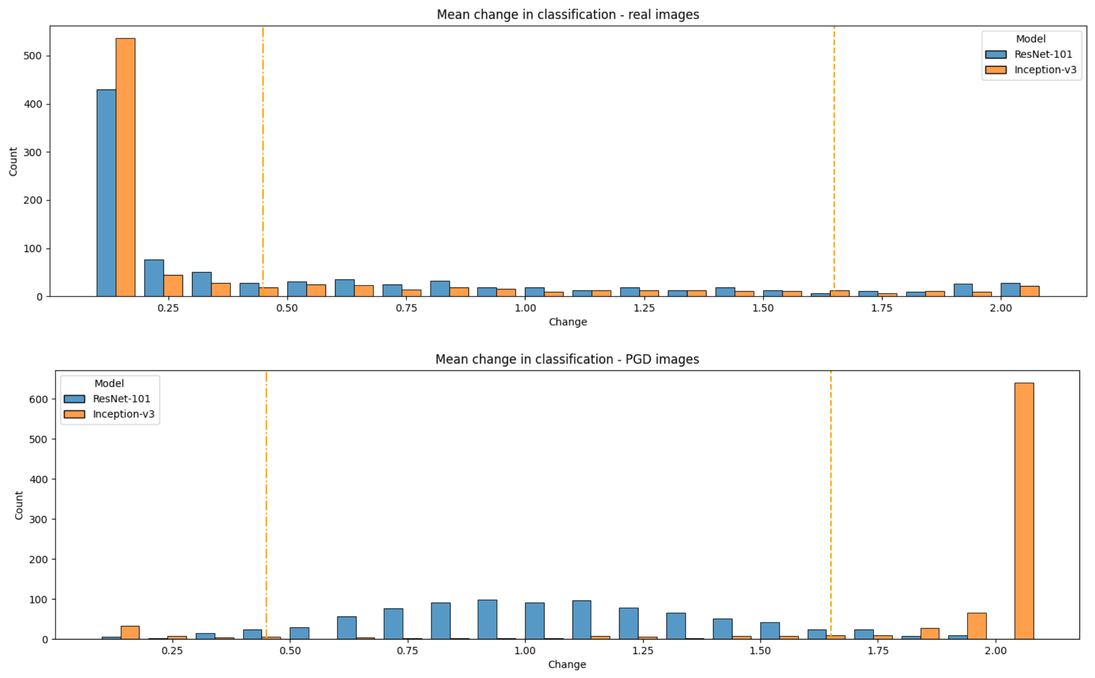


Figure 4.1: Values of a mean change in classification Δ_{avg} for real and adversarial images classified by ResNet-101 and Inception-v3. The orange lines indicate the thresholds for Inception-v3. Dashed lines specify the threshold for 10% FPR, dash-dotted lines are for 20% FPR.

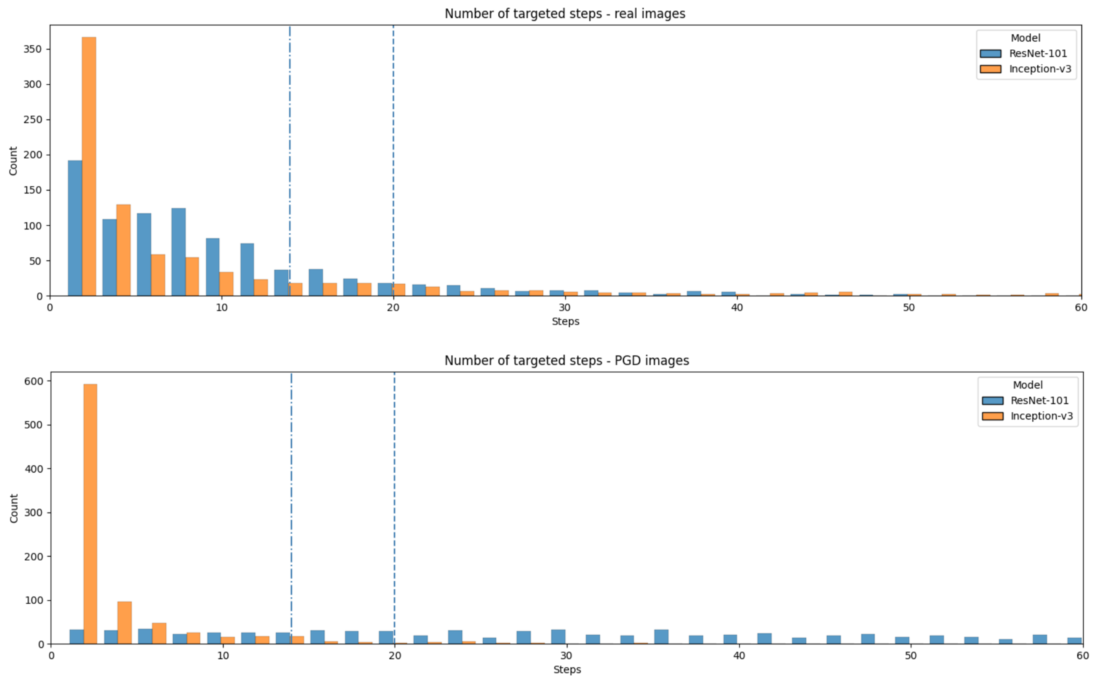


Figure 4.2: The number of targeted steps K_t for adversarial and real images classified by ResNet-101 and Inception-v3. The blue lines indicate the thresholds for ResNet-101. Dashed lines specify the threshold for 10% FPR, dash-dotted lines are for 20% FPR.

4.2 Inception-v3

This section describes series of experiments performed against Inception-v3. Since evolutionary attacks can be computational demanding in higher dimensions, we chose the trick with tiles by Ilyas et al. [2019] for dimensionality reduction. As mentioned in Chapter 2, Meunier et al. [2019] claim that convolution networks are not robust even to tiled random noise. For this reason, we decided to first try this simple attack before evolution itself.

4.2.1 Noisy attacks

Meunier et al. [2019] tested discrete tiled noise which means they added to a pixel value either $+\epsilon$ or $-\epsilon$. They attacked three architectures, Inception-v3, VGG-16 with batch normalization and ResNet-50, and tested four levels of noise and various numbers of tiles. Their results on ImageNet are in Figure 4.3.

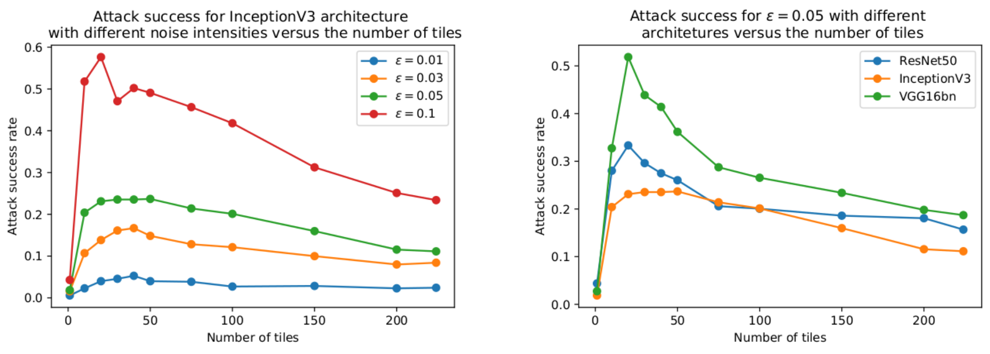


Figure 4.3: Results of the random tiled attack by Meunier et al. [2019].

We see that the higher the noise, the more effective the attack, which is not surprising. More effective attacks are also with 20 to 50 tiles per image. According to the right plot, we see that Inception-v3 is the most robust against this type of attack due to blocks of parallel convolutions with different filter sizes. We emphasize that Meunier et al. [2019] attacked clean models without any defense mechanism.

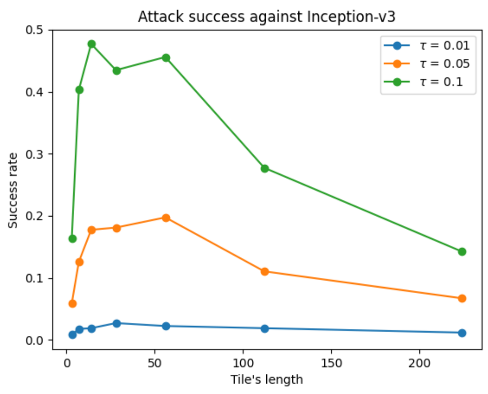


Figure 4.4: Our results of the random tiled attack against Inception-v3. Tested numbers of tiles per image side are 3, 7, 14, 28, 56, 112 and 224.

We tested our noisy attacks against with different numbers of tiles per image side. One tile spans through all three image channels. The noise is not discrete but continuous, generated from a normal distribution with a mean 0 and a variance corresponding to the value of L_∞ bound $\tau \in [0.001, 0.01, 0.05, 0.1]$. Our results are in Figure 4.4. We did not plot the results for the attack with L_∞ bound $\tau = 0.001$ due to its ineffectiveness.

We see that attacks with $14, 28$ and 56 tiles achieve the best results. Also, attacks with $\tau = 0.1$ have the best success rate, which is not surprising. More interesting are detection rates. We computed values of Δ_{avg}, K_t and K_u for each noisy image and then we computed detection rates with the original and our thresholds according to Algorithm 3. The results are in Table 4.4. The column *Tile Size* in the table denotes the number of tiles per image side. The detection rate is divided into three columns by the maximal level of adversarial noise τ.

FPR	Tile Size	Detection Rate $\tau = 0.01$ Original	Detection Rate $\tau = 0.01$ Ours	Detection Rate $\tau = 0.05$ Original	Detection Rate $\tau = 0.05$ Ours	Detection Rate $\tau = 0.1$ Original	Detection Rate $\tau = 0.1$ Ours
10 %	3	0 %	14 %	18 %	60 %	18 %	56 %
	7	13 %	20 %	21 %	51 %	21 %	59 %
	14	0 %	13 %	16 %	51 %	19 %	52 %
	28	4 %	26 %	4 %	31 %	8 %	33 %
	56	16 %	26 %	8 %	27 %	5 %	29 %
	112	19 %	44 %	4 %	22 %	3 %	20 %
	224	10 %	10 %	9 %	28 %	2 %	19 %
20 %	3	14 %	100 %	38 %	90 %	39 %	89 %
	7	20 %	100 %	40 %	92 %	41 %	93 %
	14	6 %	63 %	31 %	88 %	38 %	93 %
	28	22 %	78 %	18 %	84 %	19 %	88 %
	56	21 %	100 %	17 %	85 %	14 %	81 %
	112	25 %	94 %	10 %	81 %	9 %	81 %
	224	10 %	80 %	18 %	82 %	12 %	76 %

Table 4.4: Detection rates of noisy attack against Inception-v3 based on the original and our thresholds.

As we saw in the previous section, adversarial images created against Inception-v3 have not as uniformly distributed values of monitored metrics as ResNet-101. This property serves Inception-v3 well against noisy attacks. We see that our computed thresholds are more efficient because they utilize this integrity property more. Interestingly, there is not much difference in detection adversarial images with $\tau = 0.05$ and $\tau = 0.1$. The defense mechanism achieves the highest detection rates against attacks with larger tiles. Figure 4.4 shows that attacks with these tiles have also the highest success rate. Thus we can conclude that even though Inception-v3 is vulnerable to random tiled noise, the chosen defense is able to protect it.

4.2.2 Evolutionary attacks

We saw that our chosen defense is able to protect Inception-v3 against misclassification of adversarial images by filtering them according to their values of Δ and K_t. At the same time this capability to detect adversarial images depends on specific thresholds for particular attributes. Next, we test our evolutionary attack designed in previous section.

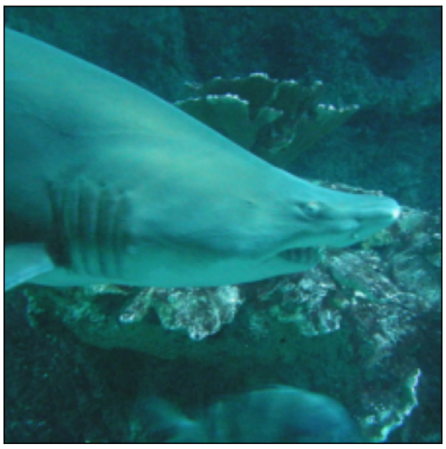
We performed our first evolutionary experiments on one image selected at random from images correctly classified by Inception-v3. Figure 4.5 (a) shows the original image with its label and monitored values. We set the parameters of differential evolution to $CR = 0.25$ and $F = 0.8$ as these values are often used. We set the L_∞ bound for the adversarial noise to $\tau = 0.05$ because we saw from the results of the random tiled attack that it is possible to deceive the network even with this level of noise. We also used early stopping, which means that we stopped the evolution after 30 generations without fitness improvement.

Tiles	Fitness	Success	Δ_{min}	Δ_{avg}	Δ_{max}	H_{adv}	Detection
28	second (Alg. 6)	100 %	0.70	1.01	1.36	1.30	0
56		88 %	0.46	0.99	1.43	1.43	0
112		94 %	0.34	0.83	1.18	1.02	0
no tiles		100 %	0.27	0.77	1.09	1.01	0
112	only Δ_{max}	100 %	0.36	0.83	1.22	1.00	0
no tiles		100 %	0.28	0.78	1.14	1.01	0

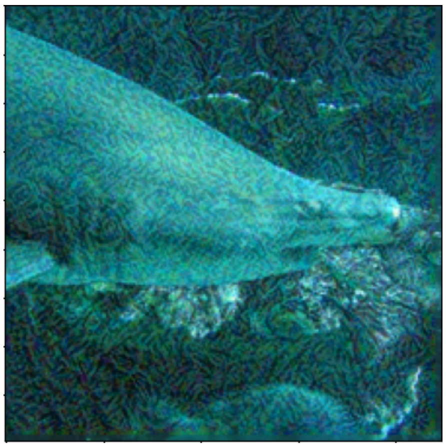
Table 4.5: Inception-v3. Results of 50 runs on one image with different numbers of tiles per image side and two fitness functions.

Table 4.5 demonstrates the results. The column *Fitness* denotes the tested fitness functions. The column *Success* indicates what percentage of the runs were successful in creating an adversarial image. We see that the attack with no tiles gives the best results as it has 100 % success and also the lowest value of all Δ. There is no significant difference between fitness that minimizes H_{orig} and Δ_{max} and fitness that minimizes only Δ_{max}. This leads to the conclusion that it is difficult for differential evolution to minimize the change in classification after adding random Gaussian noise. The last column *Detection* indicates the detection rate of successful adversarial images by the defense. We used our computed thresholds for 10 % FPR because these thresholds are even stricter than the original thresholds for 20 % FPR. It means that if the adversarial sample deceives the defense with our 10 % FPR thresholds, it also deceives the defense with the original thresholds. We see that the defense mechanism failed in detecting our attack, so we can consider it successful.

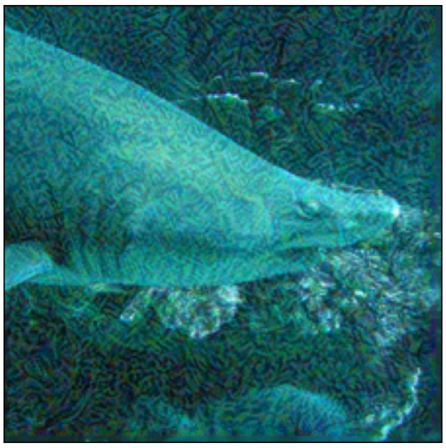
A noteworthy discovery concerns the labels of the adversarial images created by evolution. Figure 4.5 (d) shows the sample created by the evolutionary attack with adversarial noise without any tiles. We see that the label of the sample is a hammerhead shark, which is visually close to the original tiger shark. The specific label of an adversarial example created by evolution depends on the size of the tiles. Adversarial noise without tiles or with 112 tiles per image side, which is a tile of length 2 pixels, creates adversarial images with the label "hammerhead shark". Noise with the tiles of size 4×4 pixels usually produces samples with the



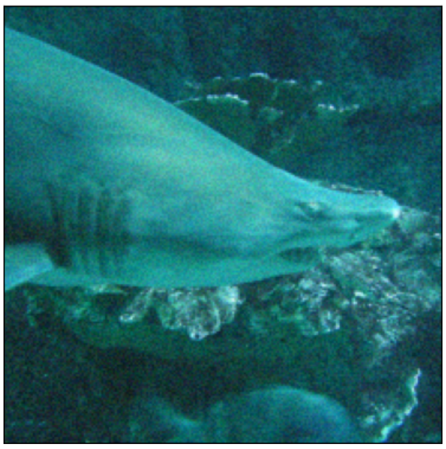
(a) Original image
"tiger shark"
0.42, 1, 1, 0.09



(b) PGD attack
"T-shirt"
1.76, 3, 1, 3.92



(c) CW attack
"fly"
2.00, 3, 1, 0.04



(d) DE attack with no tiles
"hammerhead shark"
0.77, 1, 1, 1.01

Figure 4.5: Comparison of gradient and evolutionary attacks. Labels of images are in quotation marks. The values on the third line are the change of classification Δ, the number of targeted steps K_t, the number of untargeted steps K_u, entropy H.

label "garfish" and noise with the tiles of size 8×8 pixels creates mostly samples with the label "barracouta".

In contrast, the white box attack created adversarial samples with completely different labels, such as a T-shirt or a fly. The reason behind this observation is the nature of both attacks. The white box attacks utilize only the information about gradients, not the whole classification space. On the other hand, evolutionary algorithms search the classification space to find the best results with no information about the gradients. Therefore, the easiest approach is to find an image in a proximity of the original class and then minimize the prescribed metrics. Our fitness function also changes its behaviour immediately after finding the first successful image that is probably close to the original.

Regarding the monitored metrics, there is a significant difference between

Δ values, where the evolutionary attack crafted a sample with less than half the values of the samples created by the white box attacks. The number of steps towards the decision boundary is also lower. Our adversarial sample would deceive the defense with the original thresholds and the defense with our thresholds at 10 % FPR. The deception of the defense with our thresholds at 20 % FPR would depend on the generated random Gaussian noise that is added to the image. Table 4.5 shows that the minimal change in classification Δ_{min} out of ten cases was 0.28, which is below the threshold 0.45. Actually, when we add random noise to the original image ten times, the mean value of the change in classification is 0.42, which is very close to the threshold, and the maximum value Δ_{max} is 1.71. We can therefore conclude that our adversarial image was successful.

Tiles per Side	Success	Δ_{min}	Δ_{avg}	Δ_{max}	H_{adv}
28	64 %	0.50	0.78	1.06	0.94
56	64 %	0.46	0.84	1.22	1.04
112	58 %	0.52	0.89	1.21	0.97
no tiles	48 %	0.57	0.92	1.25	1.14

Table 4.6: Inception-v3. Results of the second fitness function on 50 different images from ImageNet.

After first successful experiments, we randomly selected 50 other images and tested the second fitness function on them. The results are in Table 4.6. We used the same setting as before. We did not compute the detection rate as we can estimate from the values of Δ that we would deceive the defense because even the values of Δ_{max} are below the threshold $T_{\Delta} = 1.65$ for 10 % FPR.

Tiles per Side	τ	Success	Δ_{min}	Δ_{avg}	Δ_{max}	H_{adv}
7	0.05	70 %	0.73	0.92	1.21	1.28
14		78 %	0.65	0.89	1.19	1.13
28		72 %	0.46	0.79	1.18	1.14
56		68 %	0.55	0.87	1.26	1.30
112		58 %	0.48	0.86	1.26	1.14
no tiles		42 %	0.60	0.92	1.27	1.21
7	0.1	96 %	0.46	0.71	1.12	1.06
14		98 %	0.6	0.78	1.12	1.13
28		96 %	0.37	0.59	0.98	1.15
56		92 %	0.40	0.69	1.22	1.03
112		78 %	0.34	0.76	1.26	1.00
no tiles		60 %	0.52	0.88	1.27	1.12

Table 4.7: Inception-v3. Results of the third fitness function on 50 different images from ImageNet with weight A in Algorithm 7 set to 10.

We see that the second fitness function is not as successful as before. From this result we can conclude that the choice of the first image was a happy accident and not all images can be perturbed so easily. The adversarial noise without tiles also has the worst results, which is the opposite outcome than in the previous experiments. Therefore, we used the third fitness function described by Algorithm 7 and tested two levels of L_∞ bounds τ. We tested two different weights for a variable that subtracts the predicted probability of the adversarial class from the rest. Results for the lower weight are in Table 4.7. We see that larger tile sizes have better success rate, which corresponds to the results of attacks with random tiled noise, but evolution has much better efficiency. Higher weight was tested only for larger tiles, as they provide better outcomes. These results are shown in Table 4.8.

Tiles per Side	τ	Success	Δ_{min}	Δ_{avg}	Δ_{max}	H_{adv}
7		68 %	0.78	0.96	1.29	0.86
14	0.05	78 %	0.70	0.97	1.28	0.86
28		78 %	0.54	0.81	1.18	0.90
7		96 %	0.51	0.75	1.11	0.80
14	0.1	98 %	0.53	0.75	1.13	0.83
28		96 %	0.37	0.63	1.05	0.70

Table 4.8: Inception-v3. Results of the third fitness function on 50 different images from ImageNet with weight A in Algorithm 7 set to 100.

We see that a higher weight for the predicted probability of the adversarial class reduces the entropy. The values of Δ are mostly slightly higher, but these differences can only be random. The success rate of evolution in creating adversarial images is better for $\tau = 0.1$. All settings would deceive the defense with both the original threshold 1.83 at 20 % and our threshold 1.65 at 10 % FPR.

CR	F	Success	Δ_{min}	Δ_{avg}	Δ_{max}	H_{adv}
0.25	0.8	94 %	0.40	0.69	1.08	0.95
0.25	1	92 %	0.32	0.62	1.07	1.01
0.25	1.5	94 %	0.40	0.69	1.12	1.05
0.5	0.8	94 %	0.47	0.78	1.20	1.01
0.5	1	96 %	0.45	0.74	1.18	1.02
0.5	1.5	92 %	0.42	0.74	1.18	1.12
0.75	0.8	98 %	0.43	0.72	1.09	0.95
0.75	1	96 %	0.41	0.72	1.21	0.91
0.75	1.5	98 %	0.51	0.79	1.29	1.05

Table 4.9: Inception-v3. Results of different parameters of differential evolution on 50 images from ImageNet.

So far, we have only evaluated the size of tiles and L_∞ bound τ. The following experiments test the setting of differential evolution's parameters. We tried combinations of three values for CR and three values for F. The results are in Table 4.9. We set the number of tiles per image side to 28 and L_∞ bound τ to 0.1, because this setting provides high success rate at lower values of Δ. We used the third fitness function with weight A set to 100. We also decreased the number of generations for early stopping to 15 due to time consumption. This reduced the time per image from an average of eight minutes and 56 generations to three and a half minutes and 28 generations.

We see that our initial choice of parameters was not poor at all and that setting of these parameters does not have such impact as the size of tiles and L_∞ bound τ for adversarial noise. Nevertheless, we decided to use the setting $CR = 0.75$ and $F = 0.8$ for further experiments with transferability and types of interpolation.

4.2.3 Effect of Interpolation and Transferability

So far, we have performed all experiments with nearest-neighbor interpolation, which was also used in previous works. In the following experiment, we evaluate two other types of interpolation: bilinear and bicubic. Furthermore, we test the transferability of resulting adversarial images to other three models: pretrained ResNet-101 and ResNet-152 from PyTorch and adversarially trained ResNet-152 by Xie et al. [2019]. The results are in Table 4.10. The last part called *Transferability* shows the percentage of successful adversarial images that was also misclassified by another respective model.

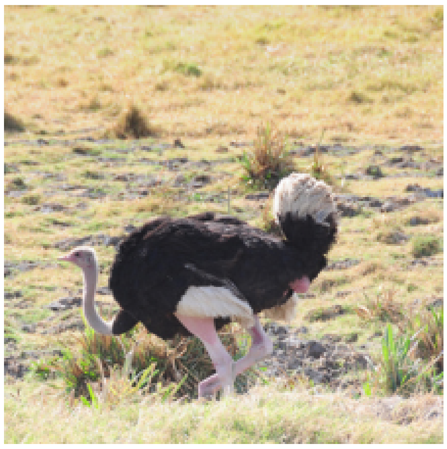
Interpolation	Success	Δ_{min}	Δ_{avg}	Δ_{max}	H_{adv}	Transferability		
						R101	R152	R152adv
nearest	96 %	0.38	0.66	1.06	0.88	56 %	54 %	35 %
bilinear	90 %	0.61	0.85	1.16	0.91	47 %	38 %	29 %
bicubic	98 %	0.54	0.82	1.15	0.79	57 %	53 %	41 %

Table 4.10: Inception-v3. Results for three different interpolations with the best setting: third fitness function with weight A in Algorithm 7 set to 100, 28 tiles per image side, $CR = 0.75$, $F = 0.8$ and $L_\infty = 0.1$. 15 generations without fitness improvement before early stopping.

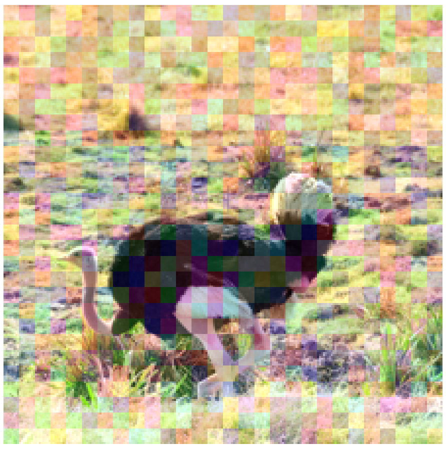
We see that bilinear interpolation has the worst performance. Adversarial images with bicubic interpolation have lower entropy and higher transferability to the adversarially trained model. On the other hand, images with nearest-neighbor interpolation are characterized by a lower values of Δ. It is worth noting that when we saved the crafted adversarial images in JPEG format, the images were then reloaded differently and the models often classified them correctly. We did not have such problem with the PNG format. An example of crafted images with their monitored values is in Figure 4.6.

Adversarial noise with bilinear interpolation is the least visible, which is probably also the reason for its worst success rate. Bicubic adversarial noise is less visible but similarly successful as nearest-neighbor. Therefore, and also because of greater transferability, we opted for bicubic interpolation. The final setting for Inception-v3 is $CR = 0.75$, $F = 0.8$, 15 generations without fitness improvement before early stopping, $\tau = 0.1$ and tiles of the size 8×8 pixels with bicubic interpolation. We performed the evolutionary attack with this setting against all 851 correctly classified images. The results are in Table 4.11.

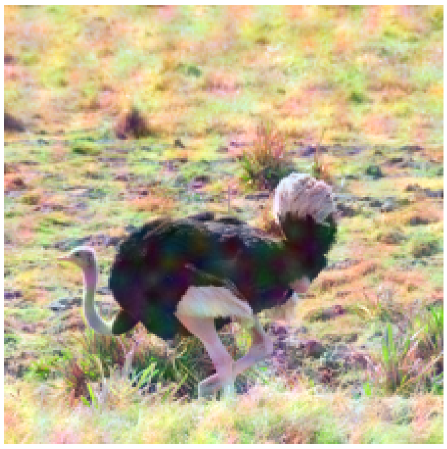
We see that evolutionary attacks are more successful compared to random tiled noise. The best success rate of a random attack with $\tau = 0.1$ was 48 % with 16×16 pixels tiles, but the detection rate with our thresholds was 52 % at 10 % FPR and 93 % at 20 % FPR. The detection rate with the original thresholds was 19 % at 10 % FPR and 38 % at 20 % FPR.



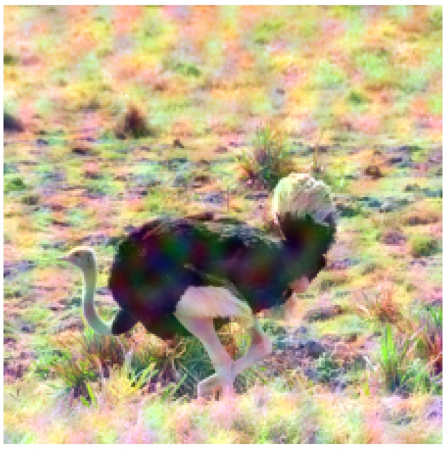
(a) Original image
“ostrich”
4×10^{-4}, 4, 174, 2×10^{-5}



(b) Nearest-neighbor
“warthog”
1.23, 1, 1, 1.2



(c) Bilinear
“gazelle hound”
1.50, 1, 1, 0.95



(d) Bicubic
“cock”
1.28, 1, 1, 2.16

Figure 4.6: Comparison of different types of interpolation. Labels of images are in quotation marks. The values on the third line are the change of classification Δ, the number of targeted steps K_t, the number of untargeted steps K_u, entropy H.

Success	Δ_{min}	Δ_{avg}	Δ_{max}	Transferability			Detection	
				R101	R152	R152-adv	Orig	Ours
93 %	0.50	0.78	1.17	44 %	39 %	30 %	3 %	9 %
				43 %	39 %	12 %	6 %	62 %

Table 4.11: Inception-v3. Results for all images with the best setting. Second row of transferability gives the values after correction. First row of detection is measured at 10 % FPR, second row is at 20 % FPR.

Compared to the white box attacks, both PGD and CW had more than 99 % success rate, and an adversarial sample was created approximately four times faster, which is not surprising. On the other hand, the detection rate with our thresholds was roughly 85 % at 10 % FPR and 90 % at 20 % FPR. The detection rate with the original thresholds was 70 % at 10 % FPR and 80 % at 20 % FPR.

In terms of transferability, we computed two values. The value in the first line tr_1 is the percentage of misclassified images out of all images correctly classified by Inception-v3. Since not all models classify everything in the same way, we adjusted the value by changing the denominator to the number of images, which were correctly classified by the chosen model in the first place (tr_2). This adjustment made the greatest change for the adversarially trained ResNet-152. It means that there are higher proportion of images, which adversarially trained ResNet-152 did not classified correctly even without the adversarial noise.

$$tr_1 = \frac{\#\text{images misclassified by the model}}{\#\text{images correctly classified by Inception-v3}} \cdot 100$$

$$tr_2 = \frac{\#\text{images misclassified by the model}}{\#\text{images correctly classified by the model}} \cdot 100$$

We also computed transferability of adversarial images created by the white box attacks. According to results in Table 4.12 we see that evolutionary images are more successful.

Attack	R101	R152	R152-adv
CW	33 %	26 %	23 %
PGD	32 %	25 %	7 %

Table 4.12: Transferability of adversarial samples created against Inception-v3 by the white box attacks.

4.2.4 Results analysis

Figure 4.7 shows a comparison for the values of Δ_{avg} for benign images, adversarial images created by evolution, and adversarial images crafted by CW attack. We do not plot a graph for the number of targeted steps K_t, as only 13 evolutionary images have a value 10 or more.

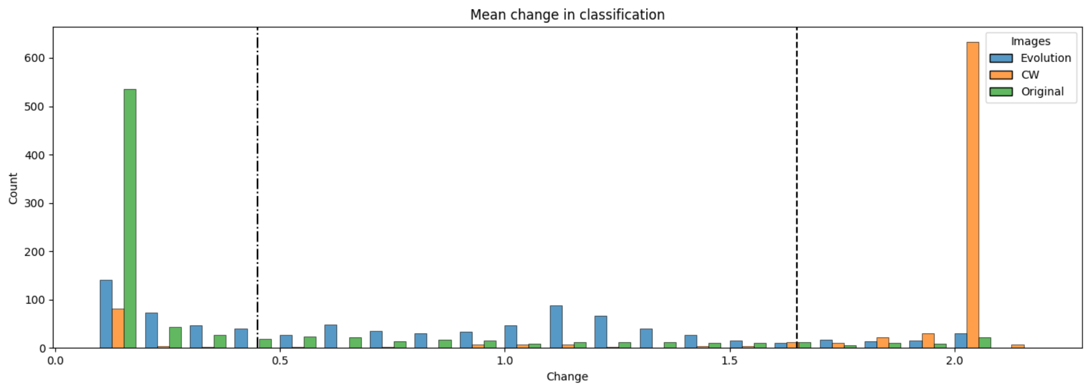


Figure 4.7: Inception-v3. Comparison of Δ_{avg} values for original images, images created by evolutionary attack, and images created by CW white box attack.

We see that the evolutionary attack creates images with significantly lower values of change in classification than the white box attack. The values are more distributed, but only about 40 % of images have a value below the threshold for 10 % FPR.

Figure 4.8 shows the relationship between Δ_{avg} values, entropy of the original image H_{orig} and the confidence of Inception-v3 in the prediction of true class p_{true} for the original image. We see that there is no relation between Δ_{avg} of the adversarial image and the other two variables. At the same time, we see that unsuccessful images have lower entropy H_{orig}, which in the beginning also means a higher probability of a true class. These failed images have lower values of Δ_{avg} because, despite the noise, we can consider them as benign samples and, as we saw earlier, benign images have lower values of Δ_{avg}.

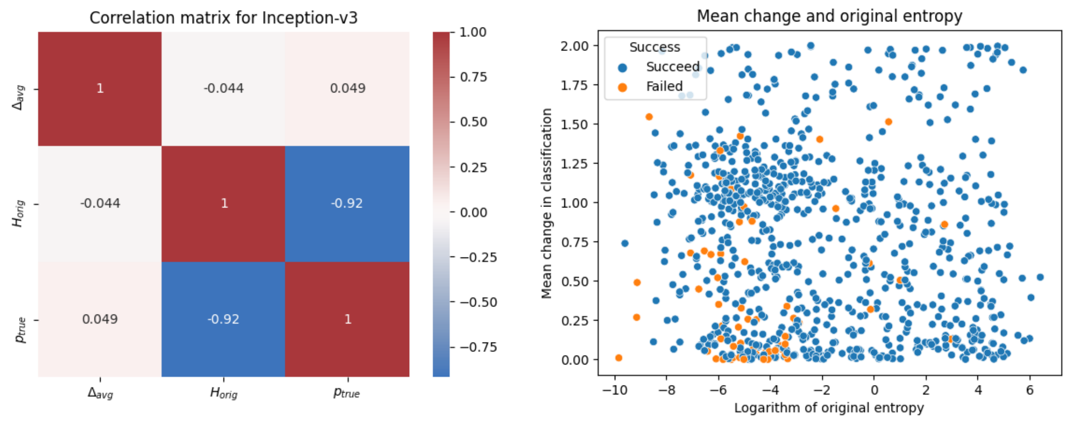


Figure 4.8: Inception-v3. Relationship between Δ_{avg} of adversarial images and H_{orig} and probability of true class for the original images.

Regarding transferability, we only analyzed pretrained ResNet-101 and ResNet-152, because adversarially trained ResNet-152 had low transferability. Figure 4.9 shows that the transferred samples have slightly lower L_2 distance between the original and the adversarial image, but the difference does not seem to be significant.

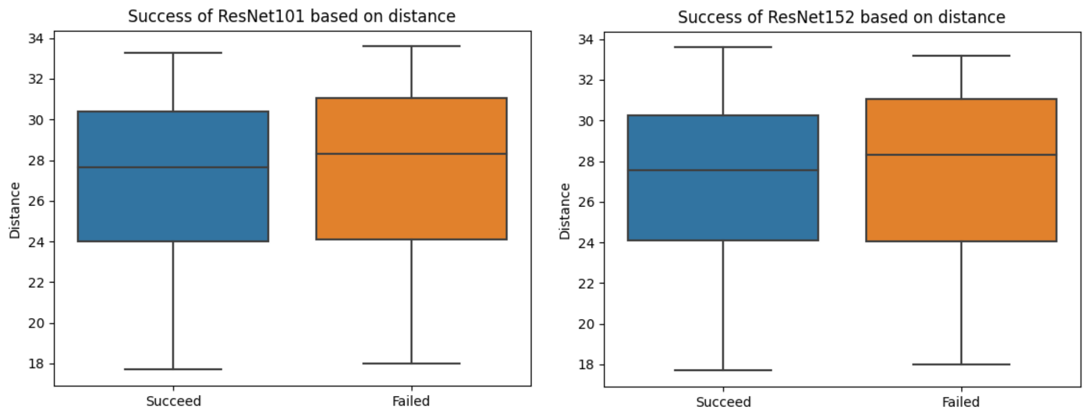
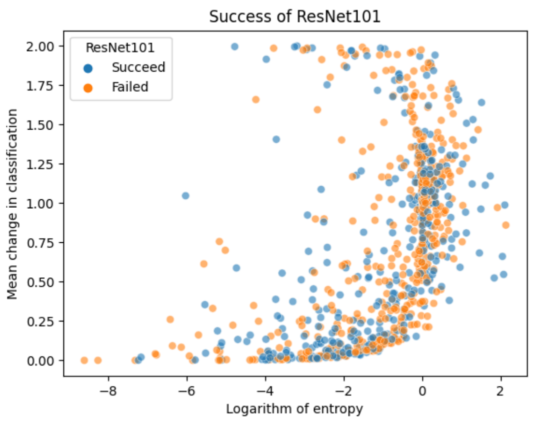


Figure 4.9: Inception-v3. Transferability between models based on the L_2 distance between the original and the adversarial image.



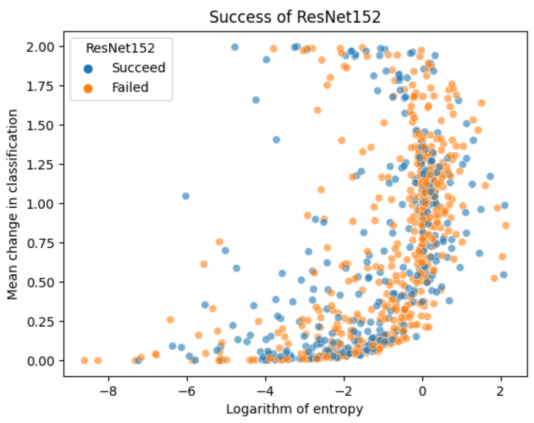


Figure 4.10: Inception-v3. Transferability between models based on values of H_{adv} and Δ_{avg} of the adversarial images.

Figure 4.10 shows transferability to the two models based on H_{adv} and Δ_{avg}. There does not seem to be any distinct pattern, but we can see a dependency between the two values, which corresponds to the positive correlation that we computed for the benign images in Chapter 3 (Fig. 3.1).

4.2.5 Comparison with the attack by Meunier et al. [2019]

Last experiments we performed against Inception-v3 compare our attack with the attack by Meunier et al. [2019]. The authors also performed evolutionary attacks together with the tiling trick for the dimensions reduction, but they used different fitness functions. We chose one of them, which minimizes the logarithm of the probability of the true class. Even though the authors utilize evolution strategies, we test their fitness function with our best setting. We only used nearest-neighbor interpolation which was also used by the authors. Also, this interpolation gives lowest Δ values. The authors did their tests with $\tau = 0.05$, but we used also $\tau = 0.1$ due to its better success rate. We performed this experiments on 50 images selected previously.

τ	Success	Δ_{min}	Δ_{avg}	Δ_{max}	H_{adv}
0.05	72 %	0.61	1.22	1.65	0.44
0.1	96 %	0.78	1.23	1.64	0.32

Table 4.13: Inception-v3. Results of the evolutionary attack with the fitness function used by Meunier et al. [2019].

Table 4.13 shows, that even though the resulting adversarial images have lower entropy, they have significantly higher values of Δ. This leads to higher detection rates, which shows Table 4.14.

FPR	Detection Rate			
	$\tau = 0.05$		$\tau = 0.1$	
	Original	Ours	Original	Ours
0.1	8 %	24 %	14 %	40 %
0.2	22 %	66 %	28 %	74 %

Table 4.14: Detection rates of of the evolutionary attack with the fitness function used by Meunier et al. [2019].

4.3 ResNet-101

This section describes the experiments we performed on ResNet-101 architecture. We start with the attacks utilizing random tiled noise and then continue with our evolutionary attacks and analysis of the results.

4.3.1 Noisy attacks

The noisy attacks were performed as before. The success of the attacks are plotted in Figure 4.11. We see that the attacks with 14 tiles per image side achieve the best results. It is also clear that attacks against ResNet-101 are more effective then attacks against Inception-v3, which is the same result mentioned by Meunier et al. [2019].

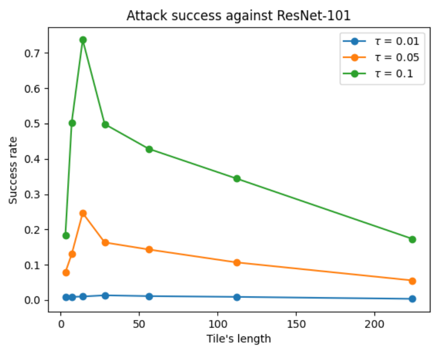


Figure 4.11: Our results of the random tiled attack against ResNet-101. Tested numbers of tiles per image side are $3, 7, 14, 28, 56, 112$ and 224.

The detection of noisy images against ResNet-101 is not very powerful, as wee see from Table 4.15. The highest detection rates are around 10 % for 10 % FPR and maximally 30 % for 20 % FPR. Detection with our thresholds gives mostly worse results, probably due to more benevolent T_Δ. Stricter T_K did not help.

4.3.2 Evolutionary attacks

We see that ResNet-101 is easier to deceive and also harder to defend due to the higher values of Δ as well as K_t of the original images. Therefore, we have not tested our evolutionary attack against a single image as with Inception-v3, but we immediately start attacking 450 different images.

FPR	Tile Size	Detection Rate					
		$\tau = 0.01$		$\tau = 0.05$		$\tau = 0.1$	
		Original	Ours	Original	Ours	Original	Ours
10 %	3	14 %	14 %	4 %	0 %	7 %	0 %
	7	0 %	0 %	4 %	1 %	9 %	2 %
	14	11 %	0 %	7 %	1 %	7 %	3 %
	28	0 %	0 %	3 %	0 %	4 %	2 %
	56	0 %	0 %	4 %	2 %	4 %	1 %
	112	0 %	0 %	3 %	5 %	5 %	8 %
	224	0 %	0 %	0 %	8 %	6 %	12 %
20 %	3	29 %	14 %	10 %	0 %	21 %	2 %
	7	25 %	0 %	16 %	3 %	24 %	3 %
	14	22 %	0 %	20 %	4 %	26 %	6 %
	28	8 %	0 %	9 %	3 %	18 %	7 %
	56	0 %	0 %	11 %	5 %	10 %	6 %
	112	0 %	0 %	8 %	10 %	8 %	13 %
	224	0 %	0 %	8 %	12 %	13 %	19 %

Table 4.15: Detection rates of noisy attack against ResNet-101 based on the original and our thresholds.

Tiles per Side	τ	Success	Δ_{min}	Δ_{avg}	Δ_{max}	H_{adv}
7	0.05	82 %	0.61	0.79	1.00	1.86
14		88 %	0.70	0.88	1.08	2.17
28		74 %	0.69	0.89	1.09	2.07
56		58 %	0.62	0.85	1.08	2.07
112		44 %	0.55	0.81	1.1	2.2
no tiles		26 %	0.52	0.78	1.18	1.7
7	0.1	98 %	0.49	0.68	0.93	2.47
14		98 %	0.63	0.77	0.97	2.86
28		96 %	0.56	0.76	1.02	2.61
56		92 %	0.47	0.71	1.00	2.36
112		76 %	0.59	0.82	1.1	2.87
no tiles		44 %	0.46	0.78	1.09	2.54

Table 4.16: ResNet-101. Results of the third fitness function with weight A in Algorithm 7 set to 10 on 50 different images from ImageNet.

We started our experiments by testing the third fitness function with two values of weight A in Algorithm 7. Values of differential evolution parameters were set to $CR = 0.25$, $F = 0.8$ and we stopped the evolution after 30 generations without fitness improvement. Results with the lower weight are in Table 4.16, results with the higher weight are in Table 4.17. We see that the higher weight for the probability of adversarial class significantly lowers the entropy of adversarial images H_{adv}. Values of Δ are way below the thresholds and the attacks with L_∞ bound τ set to 0.1 have success rate very close to 100%. As with Inception-v3, the attacks with larger tiles are more successful.

Tiles per Side	τ	Success	Δ_{min}	Δ_{avg}	Δ_{max}	H_{adv}
7		82 %	0.64	0.83	1.03	1.68
14	0.05	90 %	0.73	0.93	1.12	1.85
28		66 %	0.58	0.87	1.11	1.67
7		100 %	0.49	0.65	0.85	1.75
14	0.1	98 %	0.68	0.85	1.10	2.09
28		94 %	0.49	0.74	1.00	2.07

Table 4.17: ResNet-101. Results of the third fitness function with weight A in Algorithm 7 set to 100 on 50 different images from ImageNet.

CR	F	Success	Δ_{min}	Δ_{avg}	Δ_{max}	H_{adv}
0.25	0.8	98 %	0.58	0.78	1.03	2.06
0.25	1	98 %	0.64	0.83	1.07	2.10
0.25	1.5	98 %	0.62	0.82	1.04	2.19
0.5	0.8	98 %	0.60	0.81	1.07	2.07
0.5	1	98 %	0.69	0.86	1.10	2.32
0.5	1.5	98 %	0.69	0.91	1.15	2.37
0.75	0.8	98 %	0.59	0.78	1.01	2.24
0.75	1	98 %	0.65	0.83	1.09	2.33
0.75	1.5	98 %	0.67	0.84	1.09	2.20

Table 4.18: ResNet-101. Results of different parameters setting on 50 different images from ImageNet.

The best setting seems to be 7 tiles per image side, fitness function with the higher weight for the probability of adversarial class and $\tau = 0.1$. THe best setting for lower of $\tau = 0.05$ is the same fitness function, but 14 tiles per image side level. Table 4.16 shows that this setting reaches high success rate with low values of Δ and lower H_{adv}.

We tested different values of parameters CR and F with the vest setting for $\tau = 0.1$, but with the reduced number of generations for early stopping to 15. Results are in Table 4.18. Although all settings reach the same success rate, the original settings has the best values of all monitored metrics.

4.3.3 Effect of Interpolation and Transferability

The last interpolation tests were performed with $CR = 0.25$, $F = 0.8$, $\tau = 0.1$, 7 tiles per image side and 15 generations without fitness improvement before early stopping. The results are in Table 4.19.

We see that although bilinear and bicubic interpolation give images with lower H_{adv} values, nearest-neighbor interpolation provides the best transferability. Δ_{avg} values are similar for all three types, but the success rate is clearly better for nearest-neighbor. Therefore, we chose this type of interpolation to test the attack against all images from the sample of 1000, which were correctly classified by ResNet-101.

First line in Table 4.20 shows the results. We see that the evolutionary attack achieved high success rate with low detection for all thresholds. The adversarial

Interpolation	Success	Δ_{min}	Δ_{avg}	Δ_{max}	H_{adv}	Transferability		
						I-v3	R152	R152adv
nearest	100 %	0.59	0.76	0.98	2.18	28 %	44 %	28 %
bilinear	80 %	0.58	0.78	0.99	1.57	8 %	20 %	28 %
bicubic	88 %	0.60	0.75	0.97	1.59	9 %	20 %	30 %

Table 4.19: ResNet-101. Results for three different interpolations with the best setting: third fitness function with weight A in Algorithm 7 set to 100, 7 tiles per image side, $CR = 0.25$, $F = 0.8$ and $\tau = 0.1$. 15 generations without fitness improvement before early stopping.

τ	Success	Δ_{min}	Δ_{avg}	Δ_{max}	Transferability			Detection	
					I-v3	R152	R152adv	Orig	Ours
0.1	97 %	0.63	0.81	1.07	41 %	55 %	36 %	4 %	3 %
					36 %	54 %	19 %	8 %	6 %
0.05	77 %	0.78	0.96	1.18	23 %	33 %	32 %	3 %	4 %
					16 %	31 %	6 %	7 %	8 %

Table 4.20: ResNet-101. Results for all images with the best setting. Second row of transferability gives the values after correction. First row of detection is measured at 10% FPR, second row is at 20% FPR.

images are best transferred to ResNet-152, which is not adversarially trained. When we compare the attack with random tiled noise, we find that evolution is better again. The best success rate for $L_\infty = 0.1$ was 74 % with twice as small tiles. The detection rate was 7 % at 10 % FPR and 26 % at 20 % FPR. These detection rates are better then white box attacks, but evolution achieves even better results.

On the other hand, adversarial noise with larger tiles and nearest-neighbor interpolation is very visible, as we see in Figure 4.12. Based on previous experiments, it is also easier to deceive ResNet architectures. Therefore, we tested one more attack with a lower level of adversarial noise τ. We chose 14 tiles per image side based on the results in Table 4.17. The other settings remained the same as before.

Second line in Table 4.20 shows, that the detection rate is still very low because Δ_{avg} is below the thresholds. However, images with lower level of noise are not as successful in terms of transferability. This corresponds to an overall lower success rate. Nonetheless, compared to a random tiled attack, evolution is significantly more successful as the best result of random attack with this level of noise was 25 % success rate with the same tile size.

When we compare these two attacks in terms of computational efficiency, then the evolutionary attack with the lower noise level τ is more efficient. Although it took 36 generations on average to create an image compared to 30 generations at $\tau = 0.1$, the time required for one adversarial image at $\tau = 0.05$ was about a quarter less on average.

We can notice another curious thing in Figure 4.12. It concerns the labels of the adversarial images. We see that the images with a lower level of noise

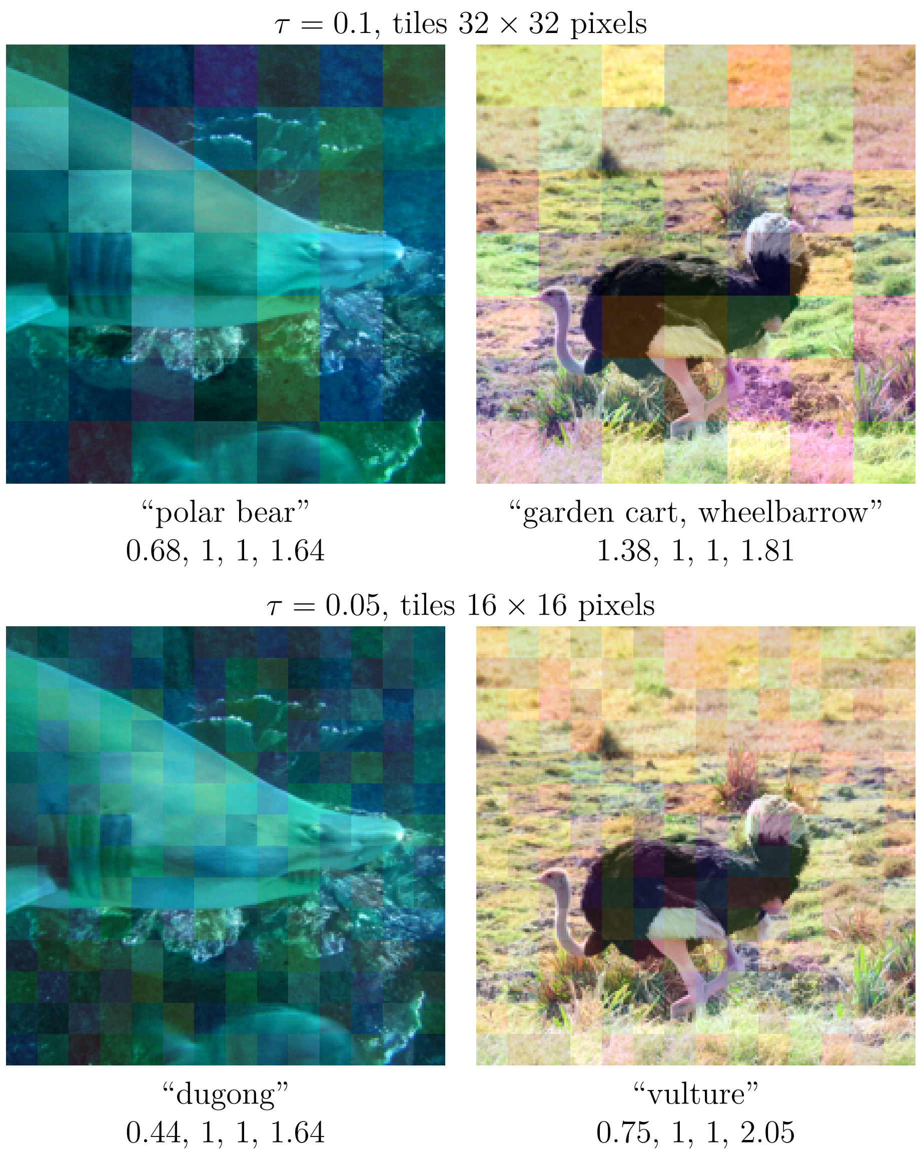

$\tau = 0.1$, tiles 32×32 pixels
“polar bear”
0.68, 1, 1, 1.64
“garden cart, wheelbarrow”
1.38, 1, 1, 1.81
$\tau = 0.05$, tiles 16×16 pixels
“dugong”
0.44, 1, 1, 1.64
“vulture”
0.75, 1, 1, 2.05

Figure 4.12: ResNet-101. Comparison of different levels of adversarial noise. Labels of images are in quotation marks. The values on the second line are the change of classification Δ, the number of targeted steps K_t, the number of untargeted steps K_u, entropy H.

still have labels similar to the original ones. On the other hand, the images with a higher level of adversarial noise have labels that are visually farther from the original. However, we can still say that there is a visual similarity between them.

4.3.4 Results analysis

Figure 4.13 compares the values of Δ_{avg} and K_t for the original images, adversarial images crafted by CW attack and evolutionary images with the two levels of adversarial noise. We see that, unlike Inception-v3, the adversarial images created by the gradient attack have lower values of Δ_{avg} than evolutionary attacks. On the other hand, the evolutionary images have lower values of targeted steps K_t, even lower than the original images.

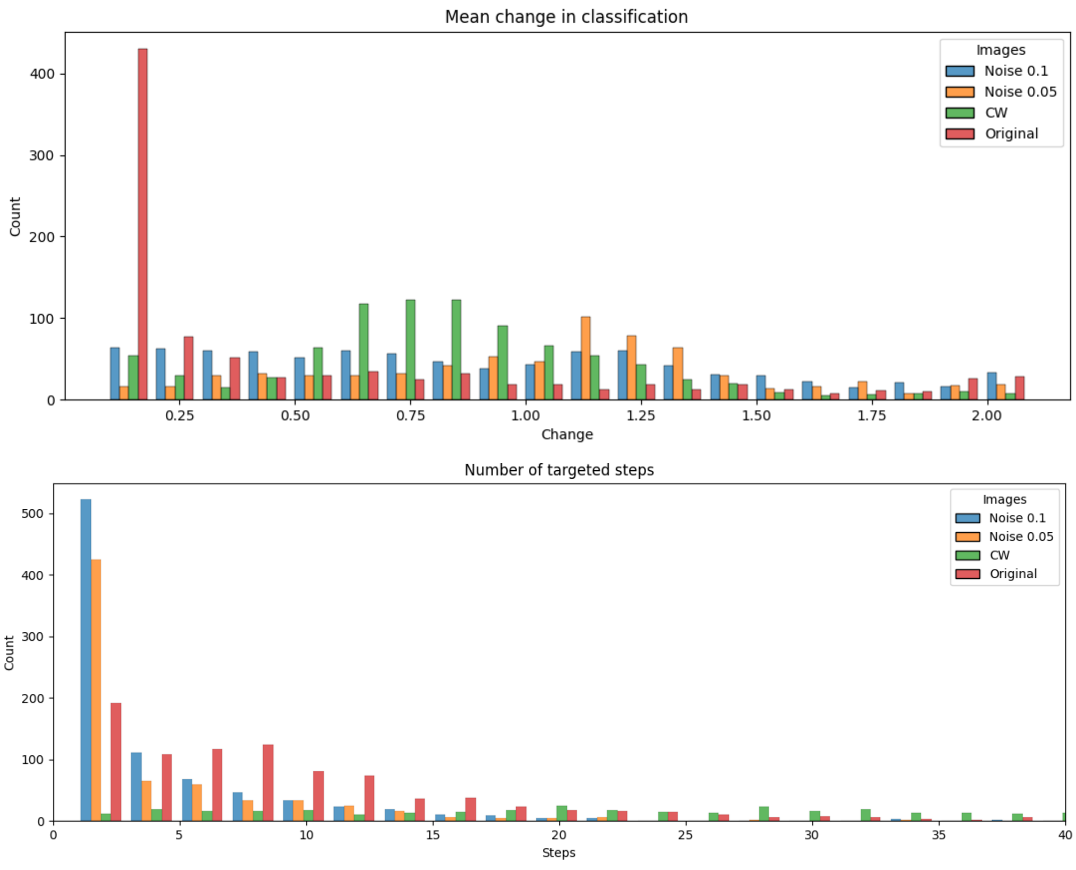


Figure 4.13: ResNet-101. Comparison of values Δ_{avg} and K_t for the original images, the images created by evolutionary attack, and the images created by CW white box attack.

The graphs in Figure 4.14 are plotted only for evolutionary images with the lower noise level τ because they have greater ratio of unsuccessful images. Transferability is plotted for the pretrained ResNet-152, which has the highest ratio of transferred images.

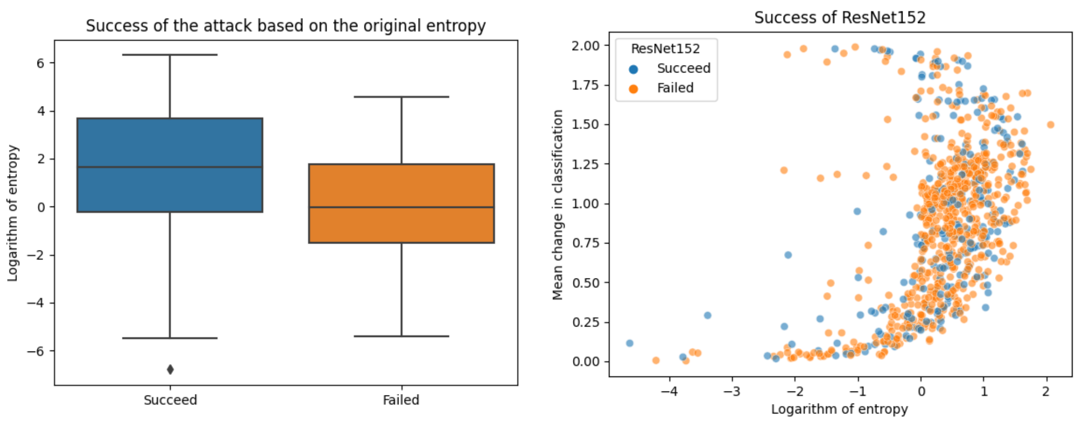


Figure 4.14: ResNet-101. Success of the attack based on entropy of the original images H_{orig} (left) and transferability to ResNet-152 based on entropy H_{adv} and change in classification Δ_{avg} of the adversarial images (right).

We see that successful images usually have higher H_{orig}. On the other hand, transferability of adversarial images does not seem to depend ether on H_{adv} or Δ_{avg}. These results are the same as for the images created against Inception-v3.

4.4 ResNet-152

We decided to test one more model that is not in the paper by Yu et al. [2019]. It is ResNet-152, which differs from ResNet-101 in the number of layers. We chose this model because it is pretrained in PyTorch and its adversarially trained version is available online thanks to Xie et al. [2019]. We did not perform attacks with random tiled noise as we hypothesized that the results would be the same as for ResNet-101.

Since we have no comparison with the original paper about the defense we aim to defeat, we used different approach to select 1000 images from ImageNet. Instead of choosing images one by one, we selected them at random. In this way, we have ensured much greater variability in image labels.

We computed adversarial images by utilizing both PGD and CW attacks. Then, we computed their values of K_t, K_u and Δ. Later we will compare these values with evolutionary attacks. We also computed new thresholds for the defense based on the values of white box attacks. Table 4.21 shows a comparison of the original thresholds for ResNet-101, our thresholds for ResNet-101 and our thresholds for ResNet-152. We see that the threshold T_Δ computed specifically for ResNet-152 is stricter.

FPR	T_Δ			T_{K_t}		
	Orig	R-101	R-152	Orig	R-101	R-152
0.1	1.90	2.00	1.73	35	20	85
0.2	1.77	1.98	1.31	22	14	85

Table 4.21: The comparison of thresholds at different levels of FPR.

4.4.1 Evolutionary attacks, Interpolation, Transferability

We only tested attacks with larger tiles. We also did not test the parameters of differential evolution and used the best setting of CR and F from the experiments with ResNet-101. Table 4.22 shows results with different levels of adversarial noise τ. We see that the best tile size is the same as for ResNet-101. We then experimented with interpolation types. Table 4.23 shows that nearest-neighbor interpolation is again the most successful.

Based on the results, we can see that the evolutionary attack successfully deceives the defense because Δ_{max} values are below the lowest $T_\Delta = 1.31$. Transferability to other models seems reasonable, but based on previous experiments, we can assume that a great proportion of adversarial images that deceive adversarially trained ResNet-152 have already been misclassified at the outset. Therefore, our last experiments test the fitness function described by Algorithm 8. This fitness function adds a bonus to individuals who confuse both pretrained and adversarially trained ResNet-152. According to the result from Table 4.24,

Tiles per Side	τ	Weight	Success	Δ_{min}	Δ_{avg}	Δ_{max}	H_{adv}
7			82 %	0.64	0.83	1.03	1.68
14	0.05	100	90 %	0.73	0.93	1.12	1.85
28			66 %	0.58	0.87	1.11	1.67
7			100 %	0.49	0.65	0.85	1.75
14	0.1	100	98 %	0.68	0.85	1.10	2.09
28			94 %	0.49	0.74	1.00	2.07
7			98 %	0.59	0.73	0.96	2.05
14	0.1	10	100 %	0.61	0.74	0.97	2.03
28			90 %	0.58	0.77	0.99	2.76

Table 4.22: ResNet-152. Results of third fitness function on 50 different images from ImageNet. The column *Weight* denotes the value of weight A in Algorithm 7.

Interpolation	Success	Δ_{min}	Δ_{avg}	Δ_{max}	H_{adv}	Transferability		
						I-v3	R101	R152adv
nearest	92 %	0.69	0.82	1.01	1.84	43 %	50 %	41 %
bilinear	68 %	0.46	0.65	0.96	1.65	29 %	41 %	50 %
bicubic	76 %	0.41	0.64	0.92	1.45	34 %	39 %	50 %

Table 4.23: ResNet-152. Results for three different interpolations with the best setting: third fitness function with weight A in Algorithm 7 set to 100, 7 tiles per image side, $CR = 0.25$, $F = 0.8$ and $\tau = 0.1$. 15 generations without fitness improvement before early stopping.

Interpolation	Success	Δ_{min}	Δ_{avg}	Δ_{max}	H_{adv}	Transferability		
						I-v3	R101	R152adv
nearest	94 %	0.66	0.85	1.08	1.74	49 %	43 %	68 %
bilinear	68 %	0.42	0.62	0.92	1.58	35 %	35 %	71 %
bicubic	76 %	0.41	0.61	0.87	1.53	29 %	47 %	68 %

Table 4.24: ResNet-152. Results for three different interpolations with the last fitness function described by Algorithm 8.

we see that it has improved the success rate against the adversarially trained ResNet-152, but the transferability to other models is mostly worse.

Finally, we test the evolutionary attack with the last fitness function and the best setting against all images from the sample of 1000, which were correctly classified by ResNet-152. As before, we tested evolutionary attacks with two levels of adversarial noise τ. The setting for the attacks was the same as for ResNet-101. The results are in Table 4.25.

We see that the adversarial images created against ResNet-152 and the images created against ResNet-101 have very similar values of Δ. The transferability of the images with the higher level of adversarial noise τ to Inception-v3 and ResNet-101 is also similar, but the success rate against adversarially trained ResNet-152 is significantly higher due to modified fitness function. The transferability of the

τ	Success	Δ_{min}	Δ_{avg}	Δ_{max}	Transferability			Detection
					I-v3	R101	R152-adv	
0.1	93 %	0.68	0.85	1.07	48 %	56 %	72 %	7 %
					41 %	53 %	56 %	22 %
0.05	74 %	0.77	0.95	1.15	39 %	41 %	53 %	7 %
					28 %	36 %	17 %	18%

Table 4.25: ResNet-152. Results for all images with the best setting. Second row of transferability gives the values after correction. First row of detection is measured at 10% FPR, second row is at 20% FPR.

images with the lower τ is higher even for Inception-v3. When we compare the transferability with the images created by the white box attacks (Table 4.26), we see that the evolutionary attack has higher success rate against adversarially trained ResNet-152 at both levels of τ. On the other hand, the images created by both white box attacks are more successful with other two models.

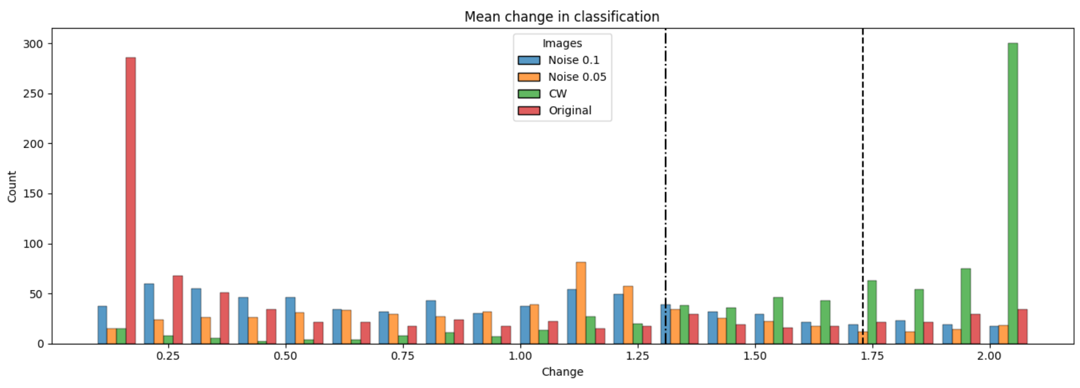
Attack	I-v3	R101	R152-adv
CW	53 %	69 %	39 %
PGD	50 %	69 %	39 %

Table 4.26: Transferability of adversarial samples created against ResNet-152 by the white box attacks, without correction.

4.4.2 Results analysis

Figure 4.15 shows the distribution of Δ and K_t values for the evolutionary and the white box adversarial images and for the real images. We see that evolution creates images with lower values of Δ then the white box attack and also lower values of K_t then both the white box and the original images. We see that proportion of the evolutionary images with $\Delta_{avg} \leq T_\Delta$ is similar to the proportion of the real images with the same property. Therefore, to detect 10 % or 20 % of adversarial images created by evolution, we have to sacrifice the same proportion of the real images. We would need to lower T_Δ to the value around 1 to increase TPR at significantly lower FPR.

Figure 4.16 shows an example of the differently perturbed image. ResNet-152 predicts true label of the real image with the probability of 56 % and the entropy of the original vector of probabilities is $H_{orig} = 1.03$. We can see that although the probability of the true class is relatively low and the entropy is distinctly higher for the real image, the value K_t is contrariwise quite high. Nevertheless, we can observe interesting behavior of the adversarial images. Three of four attacks created the image with the adversary class very similar to the original one. But, the difference between the white box and the evolutionary attacks is in the values of Δ and K_t. Evolution created images with significantly lower values of Δ, even lower then the original image has. The same is true for the entropy H. We can see that CW attacks created the image with similarly low values of K_t as evolution, with even lower H_{adv}, but the value of Δ is very large. Both



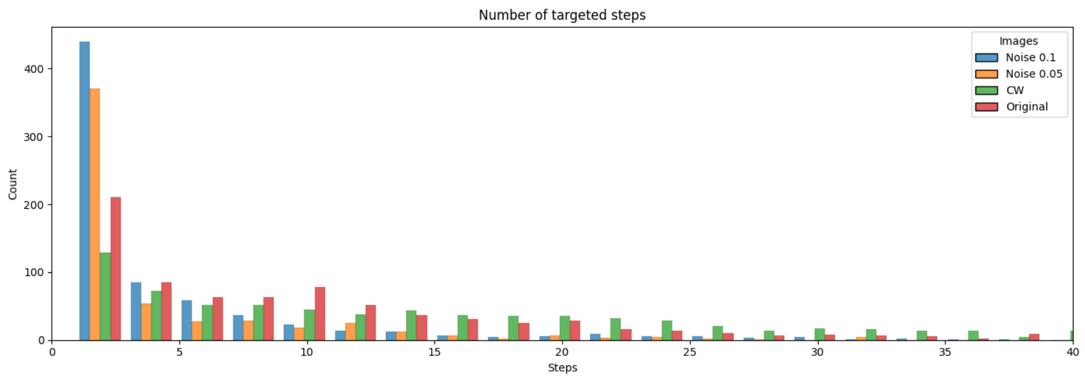
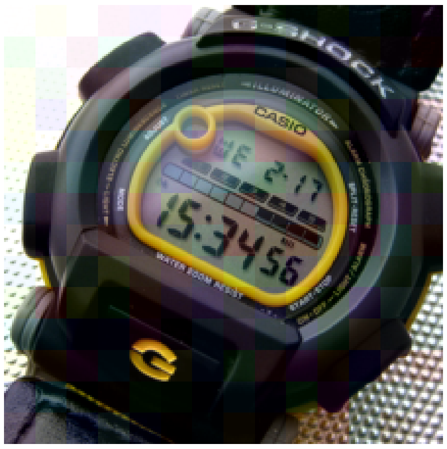
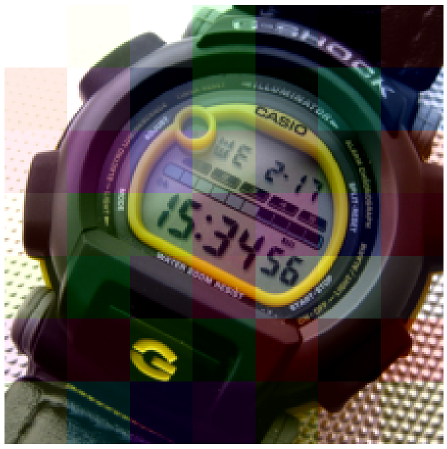


Figure 4.15: ResNet-152. Comparison of values of the change in classification Δ and the number of targeted steps K_t for original images, images created by evolutionary attack, and images created by CW white box attack.

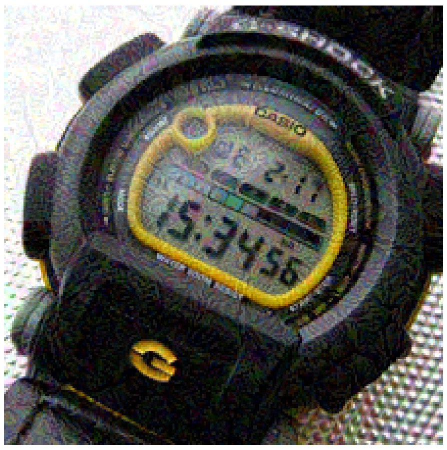
adversarial images created by the white box attacks would be therefore detected by the defense. The evolutionary images would deceive the defense because their values are "more real" in sense of the hypotheses by Yu et al. [2019] that the original image has.



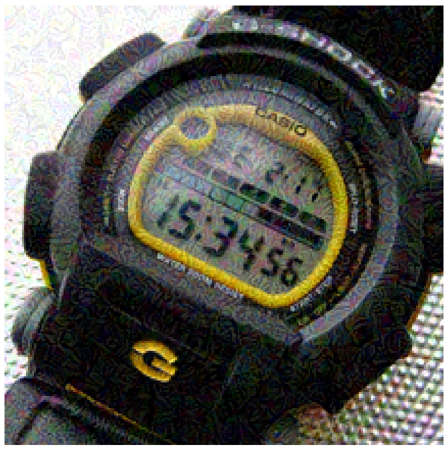
(a) $\tau = 0.05$
“digital watch”
0.12, 11, 1, 0.73



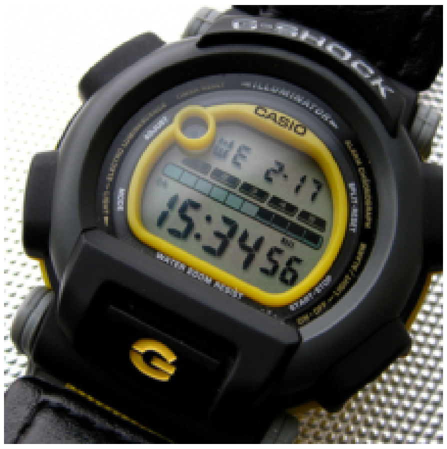
(b) $\tau = 0.1$
“digital watch”
0.17, 7, 1, 0.38



(c) CW attack
“steel drum”
1.85, 8, 1, 0.02



(d) PGD attack
“digital watch”
1.77, 19, 1, 5.03



(e) Original image
“stopwatch”
0.26, 20, 1, 1.03

Figure 4.16: ResNet-152. Comparison of different attacks. Labels of images are in quotation marks. The values on the second line are the change of classification Δ, the number of targeted steps K_t, the number of untargeted steps K_u, entropy H.

Conclusion

The goal of this thesis was to propose a method of creating adversarial examples which would defeat the modern defenses. We decided to focus on adversarial examples in image classification due to the fact, that it is a very current and fascinating topic. The state of the art deep learning models are capable to classify images at even a better level than humans can, yet these models can be fundamentally confused by adding imperceptible noise to the images.

We summarized adversarial attacks as well as defenses to acquire knowledge about the past and current techniques used in this field. Since the discovery of adversarial examples in image processing in 2014, researchers have tried to find a way how to defend the models against these malicious attacks. One of the first suggested defenses was to adjust the technique of training deep learning models and by that decrease their propensity to misclassify the adversarially perturbed inputs. The adversarial training truly helps to defend the models, but at the same time it reduces their accuracy. In 2019 was published new interesting defense which instead of changing the training process, it verifies hypothesized properties of real images and rejects images that do not pass as adversarial. The defense was proposed against the white box attacks with the justification that other attacks are even easier to detect. We decided to deceive this defense, but with a black box attack by utilizing evolutionary algorithms.

Our attack exploits differential evolution alongside the tiling trick for the dimensionality reduction. We were successful in both creating adversarial images capable of fooling the network but also deceiving the chosen defense. We compared our evolutionary attack to the attack which utilizes only random tiled noise and discovered that the evolutionary attacks are better then the random one. This conclusion made also Su et al. [2019] with their evolutionary One Pixel attack. Furthermore, we compared our evolutionary attack with the attack by Meunier et al. [2019], which also exploited the tiling trick, but used different fitness function. We discovered that our attack has similar success rate in creating adversarial images, but significantly lower detection rate of created images by the defense mechanism.

Results of our experiments also identified an important property of adversarial images created by the black box evolutionary attack. The authors of the chosen defense claims that adversarial images are either sensitive to random Gaussian noise, which significantly changes their classification, or are far from the decision boundary in sense of gradient steps. We found out that this is not the case of adversarial images created by evolution as it is hard to get far away from the decision boundary without knowing the direction in the form of gradients. Our evolutionary attack also created adversarial samples with lower values of the change in classification after the addition of random noise than white box attacks.

Our experiment further suggests, that the properties described by Yu et al. [2019] depends not only on images themselves, but also on the type of architecture used for the image classification as we saw significant differences between results of experiments performed against different architectures. We saw, that ResNet-101 is easier to deceive and also more complicated to defend. On the other hand, Inception-v3 is more resistant to tiled noise due to its blocks of parallel convolutions with different filter sizes.

Last but not least, we tested the transferability of our created adversarial examples to other models, one of which was adversarially trained. The results again depended on the tested architecture. Our adversarial images against Inception-v3 were more transferable to other models then adversarial images created by white box attacks. For ResNet-152 it was the opposite. We further discovered that we can deceive more models at once by focusing on all of their outcomes during the attack. By this way, we were also able to increase the success rate of our attacks against adversarially trained ResNet-152. On the other hand, if we focused our fitness function more on adversarially trained network it did not help with the transferability to other models. We also identified that the transferability between models does not necessarily depend on the entropy of the output vector of primarily deceived model or the distance between the original and adversarial input, but rather on the level of adversarial noise in the image.

The goals of this work were met. As for the future research, we suggest to combine evolutionary attacks with a smart mutation, which would utilizes the gradients. This could lead to adversarial images that combine the advantages of both approaches or reveal other interesting properties of the created images.

Bibliography

Moustafa Alzantot, Yash Sharma, Supriyo Chakraborty, Huan Zhang, Cho-Jui Hsieh, and Mani B. Srivastava. Genattack: practical black-box attacks with gradient-free optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2019*, pages 1111–1119, 2019. URL `https://doi.org/10.1145/3321707.3321749`.

Wieland Brendel, Jonas Rauber, and Matthias Bethge. Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. In *6th International Conference on Learning Representations, ICLR 2018*, 2018. URL `https://openreview.net/forum?id=SyZI0GWCZ`.

Nicholas Carlini and David A. Wagner. Towards evaluating the robustness of neural networks. In *IEEE Symposium on Security and Privacy, SP 2017*, pages 39–57. IEEE Computer Society, 2017. URL `https://doi.org/10.1109/SP.2017.49`.

Pin-Yu Chen, Huan Zhang, Yash Sharma, Jinfeng Yi, and Cho-Jui Hsieh. ZOO: zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security, AISec@CCS 2017*, pages 15–26, 2017. URL `https://doi.org/10.1145/3128572.3140448`.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 248–255, 2009. URL `http://www.image-net.org/papers/imagenet_cvpr09.pdf`.

Yinpeng Dong, Fangzhou Liao, Tianyu Pang, Hang Su, Jun Zhu, Xiaolin Hu, and Jianguo Li. Boosting adversarial attacks with momentum. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018*, pages 9185–9193, 2018. URL `http://openaccess.thecvf.com/content_cvpr_2018/html/Dong_Boosting_Adversarial_Attacks_CVPR_2018_paper.html`.

A. E. Eiben and James E. Smith. *Introduction to Evolutionary Computing.* Springer Publishing Company, Incorporated, 2nd edition, 2015. ISBN 978-3-662-44873-1.

Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *3rd International Conference on Learning Representations, ICLR 2015*, 2015. URL `http://arxiv.org/abs/1412.6572`.

Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* MIT Press, 2016. `http://www.deeplearningbook.org`.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. URL `https://doi.org/10.1109/CVPR.2016.90`.

Andrew Ilyas, Logan Engstrom, Anish Athalye, and Jessy Lin. Black-box adversarial attacks with limited queries and information. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 2142–2151, 2018. URL `http://proceedings.mlr.press/v80/ilyas18a.html`.

Andrew Ilyas, Logan Engstrom, and Aleksander Madry. Prior convictions: Black-box adversarial attacks with bandits and priors. In *7th International Conference on Learning Representations, ICLR 2019*, 2019. URL `https://openreview.net/forum?id=BkMiWhR5K7`.

Malhar Jere, Briland Hitaj, Gabriela F. Ciocarlie, and Farinaz Koushanfar. Scratch that! an evolution-based adversarial attack against neural networks. *CoRR*, abs/1912.02316, 2019. URL `http://arxiv.org/abs/1912.02316`.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015*, 2015. URL `http://arxiv.org/abs/1412.6980`.

Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research), 2009. URL `http://www.cs.toronto.edu/~kriz/cifar.html`. [Online; accessed December 10, 2020].

Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. Adversarial machine learning at scale. In *5th International Conference on Learning Representations, ICLR 2017*, 2017. URL `https://openreview.net/forum?id=BJm4T4Kgx`.

Y. LeCun and C. Cortes. MNIST handwritten digit database, 1999. URL `http://yann.lecun.com/exdb/mnist/`. [Online; accessed December 10, 2020].

Erin LeDell. Image of a mlp network, 2016. URL `https://github.com/ledell/sldm4-h2o/blob/master/mlp_network.png`. [Online; accessed December 2, 2020].

Junyu Lin, Lei Xu, Yingqi Liu, and Xiangyu Zhang. Black-box adversarial sample generation based on differential evolution. *CoRR*, abs/2007.15310, 2020. URL `https://arxiv.org/abs/2007.15310`.

Seppo Linnainmaa. Taylor expansion of the accumulated rounding error. *BIT Numerical Mathematics*, 16:146–160, 1976. URL `https://doi.org/10.1007/BF01931367`.

Xiaolei Liu, Teng Hu, Kangyi Ding, Yang Bai, Weina Niu, and Jiazhong Lu. A black-box attack on neural networks based on swarm evolutionary algorithm. In *Information Security and Privacy - 25th Australasian Conference, ACISP 2020*, volume 12248 of *Lecture Notes in Computer Science*, pages 268–284, 2020. URL `https://doi.org/10.1007/978-3-030-55304-3_14`.

YiGui Luo, RuiJia Yang, Wei Sha, WeiYi Ding, YouTeng Sun, and YiSi Wang. Evolution attack on neural networks. *CoRR*, abs/1906.09072, 2019. URL `http://arxiv.org/abs/1906.09072`.

Laurent Meunier, Jamal Atif, and Olivier Teytaud. Yet another but more efficient black-box adversarial attack: tiling and evolution strategies. *CoRR*, abs/1910.02244, 2019. URL `http://arxiv.org/abs/1910.02244`.

Seungyong Moon, Gaon An, and Hyun Oh Song. Parsimonious black-box adversarial attacks via efficient combinatorial optimization. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019*, volume 97 of *Proceedings of Machine Learning Research*, pages 4636–4645, 2019. URL `http://proceedings.mlr.press/v97/moon19a.html`.

Nina Narodytska and Shiva Prasad Kasiviswanathan. Simple black-box adversarial attacks on deep neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops, CVPR 2017*, pages 1310–1318, 2017. URL `https://doi.org/10.1109/CVPRW.2017.172`.

Anh Mai Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015*, pages 427–436, 2015. URL `https://doi.org/10.1109/CVPR.2015.7298640`.

Nicolas Papernot, Patrick D. McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *IEEE European Symposium on Security and Privacy, EuroS&P 2016*, pages 372–387, 2016a. URL `https://doi.org/10.1109/EuroSP.2016.36`.

Nicolas Papernot, Patrick D. McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *IEEE Symposium on Security and Privacy, SP 2016*, pages 582–597. IEEE Computer Society, 2016b. URL `https://doi.org/10.1109/SP.2016.41`.

Nicolas Papernot, Patrick D. McDaniel, Ian J. Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, AsiaCCS 2017*, pages 506–519, 2017. URL `https://doi.org/10.1145/3052973.3053009`.

Martin Pilát. Evoluční algoritmy - úvod, 2020. URL `https://martinpilat.com/cs/prirodou-inspirovane-algoritmy/evolucni-algoritmy-uvod`. [Online; accessed December 12, 2020].

J. Rapin and O. Teytaud. Nevergrad - A gradient-free optimization platform. `https://GitHub.com/FacebookResearch/Nevergrad`, 2018.

Raúl Rojas. *Neural Networks - A Systematic Introduction.* Springer, 1996. `https://page.mi.fu-berlin.de/rojas/neural/neuron.pdf`.

Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958. URL `https://doi.org/10.1037/h0042519`.

Andras Rozsa, Ethan M. Rudd, and Terrance E. Boult. Adversarial diversity and hard positive generation. In *2016 IEEE Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops 2016*, pages 410–417, 2016. URL `https://doi.org/10.1109/CVPRW.2016.58`.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning internal representations by error propagation. In David E. Rumelhart and James L. Mcclelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations*, pages 318–362. MIT Press, 1986.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *3rd International Conference on Learning Representations, ICLR 2015*, 2015. URL `http://arxiv.org/abs/1409.1556`.

Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. In Yoshua Bengio and Yann LeCun, editors, *2nd International Conference on Learning Representations, ICLR 2014*, 2014. URL `http://arxiv.org/abs/1312.6034`.

Milan Straka. Deep learning, lecture 4 - convolutional networks, 2018. URL `https://slideslive.com/38906635/deep-learning-lecture-4-convolutional-networks`. [Online; accessed December 7, 2020].

Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai. One pixel attack for fooling deep neural networks. *IEEE Trans. Evol. Comput.*, 23(5):828–841, 2019. URL `https://doi.org/10.1109/TEVC.2019.2890858`.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *2nd International Conference on Learning Representations, ICLR 2014*, 2014. URL `http://arxiv.org/abs/1312.6199`.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2826, 2016. URL `http://arxiv.org/abs/1512.00567`.

Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian J. Goodfellow, Dan Boneh, and Patrick D. McDaniel. Ensemble adversarial training: Attacks and defenses. In *6th International Conference on Learning Representations, ICLR 2018*, 2018.

Cihang Xie, Yuxin Wu, Laurens van der Maaten, Alan Yuille, and Kaiming He. Feature denoising for improving adversarial robustness. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. URL `https://arxiv.org/pdf/1812.03411.pdf`.

Tao Yu, Shengyuan Hu, Chuan Guo, Weilun Chao, and Kilian Weinberger. A new defense against adversarial images: Turning a weakness into a strength. In *Proceedings of the 33rd Conference on Neural Information Processing Systems (NeurIPS 2019)*, 2019. URL `https://proceedings.neurips.cc/paper/2019/file/cbb6a3b884f4f88b3a8e3d44c636cbd8-Paper.pdf`.

Xiaoyong Yuan, Pan He, Qile Zhu, and Xiaolin Li. Adversarial examples: Attacks and defenses for deep learning. *IEEE Trans. Neural Networks Learn. Syst.*, 30 (9):2805–2824, 2019. URL `https://doi.org/10.1109/TNNLS.2018.2886017`.

Jiří. Šíma and Roman Neruda. *Teoretické otázky neuronových sítí*. Vyd. 1. Matfyzpress, Praha, 1996. ISBN 80-85863-18-9.

List of Figures

List of Tables

A. Attachments

A.1 Digital Content

The attachment contains the created images in `.png` files and the results of our experiments in `.csv`. We also attached our source code for reproducing the results. Individual scripts in `.py` files are described in README.

We did not provide original scripts of Yu et al. [2019] as they are available online in the authors' Git repository[1]. We used their three essential scripts, but we had to make some changes:

`attack.py` implements $\mathcal{L}_3$ loss, $\mathcal{L}_4$ loss and creates adversarial examples. We did some adjustments according to the paper to obtain the setting for the best authors' results. We set learning rates for each attack and added λ term in Equation 2.3, $\lambda = 2$ for ImageNet and $\lambda = 3$ for CIFAR-10.

`detect.py` implements functions for adversarial examples detection, which computes the change in classification after adding random Gaussian noise, denoted as Δ, the number of steps towards the boundary for the targeted attack, denoted as K_t and the number of steps towards the boundary for the untargeted attack, denoted as K_u.

`evaluate.py` implements functions for evaluation of the images, either for one criterion or for all of them combined. There is also a function for tuning the thresholds.

[1]https://github.com/s-huu/TurningWeaknessIntoStrength