Skoltech

Skolkovo Institute of Science and Technology

MASTER'S THESIS

Black-box adversarial attacks

Master's Educational Program: Data Science

Student	Maria Sindeeva
	signature, name
Research Advisor:	Prof. Ivan Oseledets
	signature, name, title
Co-Advisor *(if applicable)*:	
	signature, name, title

Moscow 2020

Skoltech

Skolkovo Institute of Science and Technology

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

Состязательные атаки на нейросеть как чёрный ящик

Магистерская образовательная программа: Наука о данных

Студент: Синдеева Мария Андреевна

подпись, ФИО

Научный руководитель: Проф. Оселедец Иван Валерьевич

подпись, ФИО, должность

Со-руководитель:
(при наличии)

подпись, ФИО, должность

Москва, 2020

Skoltech

Skolkovo Institute of Science and Technology

Black-box adversarial attacks

Maria Sindeeva

Submitted to the Skolkovo Institute of Science and Technology
on 29 May 2020

ABSTRACT

Modern Deep Neural Networks are being widely adopted to solve all kinds of real-world problems. However, for several years now they have been known to be susceptible to adversarial attacks, i.e. patterns crafted with the sole purpose of fooling DNNs to produce incorrect predictions, e.g. misclassification of the input, impersonation in a face recognition system, disabling a detection system, etc. This work mainly focuses on studying the potential of Turing patterns as universal black-box decision-based adversarial attacks. The overall purpose of it is to explore if and how Turing patterns can be used and modified to create new types of black-box adversarial attacks. This research performs experiments to prove that Turing patterns can indeed be tailored to attack target networks and modified to produce even stronger attacks. This work also explores the similarity and connection of modified Turing patterns to Single Fourier Attacks.

Research Advisor:
Name: Ivan Oseledets
Degree: PhD
Title: Professor

Contents

1 INTRODUCTION

Modern Deep Neural Networks are being widely adopted to solve all kinds of real-world problems. However, for several years now they have been known to be susceptible to adversarial attacks, i.e. patterns crafted with the sole purpose of fooling DNNs to produce incorrect predictions, e.g. misclassification of the input, impersonation in a face recognition system, disabling a detection system, etc.

Adversarial attacks can take place in both digital and physical domains. Digital adversarial attacks take place in a fully controlled digital environment, which means that the DNN input can be manipulated arbitrarily to perform the attack. Such attacks usually present themselves as a perturbation of the original input imperceptible by humans. To obtain such perturbation, one must solve an optimization problem of finding the smallest possible noise to be injected into the input, such that the perturbed input fools a DNN. Physical domain attacks have to be realized in the physical world with unstable conditions, which may be out of the attacker's control (e.g. lighting, angle of the camera, arbitrary noise, etc.). Thus, they need to present themselves as real-world objects: stickers, glasses with specific texture, patches, makeup, etc. Generally, inconspicuous physical domain attacks are considered more desirable, so as not to alarm an adversarial attack detector or a supervising expert in a real-world setting.

In general, two attack scenarios are considered: white-box and black-box. In the white-box setting it is assumed that the attacker has full knowledge about the target system: its architecture, weights and defense system are available. Typically, attacks in such scenario are constructed by optimization of the input using the target DNN's gradient with respect to the input, obtained via back-propagation through the target DNN. In the black-box scenario there is no prior knowledge about the target system, and the attacker only has access to the target system's outputs: either output score values (e.g. class probabilities) or model's final decision (e.g. authentication passed or failed). Black-box attacks do not have direct access to the target DNN's gradients and have to rely on black-box optimization methods to construct an attack. White-box attacks are generally easier to construct and are more successful in fooling DNNs, however, the black-box scenario models a more realistic setting and is considered to pose a more realistic threat.

In both attack scenarios there exist input-agnostic and target-agnostic attacks. If an attack possesses either of these two qualities, it is called a universal adversarial attack. Although universal attacks typically perform worse than input- and target-specific attacks, they arguably present a bigger threat for a DNN due to the fact that a universal attack only needs to be constructed once and can fool either any DNN or one DNN on any input. Most recently in a work concurrent with this one, Turing patterns(patterns often found in nature that arise naturally out of a uniform state) have shown potential to be used as universal black-box adversarial attacks.

The number of real-world DNN applications keeps growing, as DNN-based models are being widely adopted to replace humans in complex tasks, such as driving, authentication, identification, etc., Thus, it is crucial to explore the robustness of these models to all kinds of possible attacks, such as universal black-box attacks, and highlight their weaknesses. In order to achieve that, black-box adversarial attacks which model realistic threats need to be studied and improved.

1.1 Aim

This work mainly focuses on studying the potential of Turing patterns as universal black-box decision-based adversarial attacks. The overall purpose of it is to explore if and how Turing patterns can be used and modified to create new types of black-box adversarial attacks that would be better in terms of DNN fooling rates, to explore the transferability of Turing patterns and their modifications, and to compare them with existing universal black-box attacks.

1.2 Objectives

In order to reach the aim of this research, there are 3 main objectives:

1. Test if evolutionary strategies as a black-box optimization method can increase the Turing patterns' effectiveness as adversarial attacks in terms of fooling rate

2. Explore the effectiveness of different modifications of Turing pattern generation algorithm to see if better performance can be achieved and how it affects attack transferability between the target models

3. Explore the similarity of modified Turing patterns to Single Fourier Attacks

2 PROBLEM STATE

There are several approaches to / scenarios for black-box adversarial attack generation at the moment.

2.1 Transfer-based attacks

Transfer-based attacks rely on transferring a white-box attack for some known DNN to a given black-box setting. Since adversarial examples have been shown to be transferable between the models and the datasets they are trained on (Szegedy et al. 2013; Goodfellow, Shlens, and Szegedy 2014; Petrov and Hospedales 2019), one can hope that some known model's white-box attack will work as an adversarial attack for a black-box DNN. In practice, this is achieved by either choosing a local (white-box) model arbitrarily (Petrov and Hospedales 2019), constructing one by training a model on the target DNN's outputs (Papernot et al. 2016), or even relying on the connection of a target model to some well-known pre-trained model (Huan et al. 2020).

2.2 Score-based attacks

Score-based attacks assume a setting where the target DNN's output scores are available to the attacker, who make queries to the black-box system to obtain them and use in attack construction. In the score-based setting an attacker can try to use the model's outputs to approximate the target DNN's properties for later use in optimization of some objective to obtain the adversarial example. For example, a zero-th order optimization based on the target's gradient and/or Hessian approximation can be applied to obtain an adversarial example (P.-Y. Chen et al. 2017; Tu et al. 2018). The target gradient has also been approximated using Natural Evolutionary Strategies (Ilyas et al. 2018) and further applied to obtain a successful adversarial attack. Gradient-free optimization has also been applied in this setting, using a genetic algorithm to obtain a population of adversarial examples (Alzantot et al. 2018). Adversarial attacks made using evolutionary strategies have also improved in the recent years by becoming more query-efficient (Meunier, Atif, and Olivier Teytaud 2019), which is important in a black-box setting. In this work Meunier, Atif, and Olivier Teytaud apply Covariance Matrix Adaptation Evolution Strategy (CMA-ES) and achieve near-perfect fooling rate.

2.3 Decision-based attacks

In the most realistic setting - the decision-based one - the attacker does not have access to the target model's output scores, but only to the model's decision, e.g. the class an input belongs to, or an indicator of whether the authentication has been successful. In this case the attack generation algorithms generally rely on stochastic processes (random walks, as in (Brendel, Rauber, and Bethge 2017)) constructed in the space of adversarial examples to obtain the attack. Evolutionary strategies, e.g. (1+1) evolutionary strategy and CMA-ES, are also being applied to construct such processes and obtain successful adversarial attacks (Dong et al. 2019).

2.4 Universal adversarial attacks

When it comes to universal adversarial attacks, the approaches to construct an attack can rely on attacking hidden layers (Khrulkov and Oseledets 2017) (exploiting (p, q)-singular vectors of the Jacobian matrices of hidden layers of a network in a white-box setting), or structural sensitivity of convolutional layers to Fourier basis functions (Tsuzuku and Sato 2018) (universal black-box decision-based attacks), or constructing and exploiting attention heatmaps of the target network (S. Chen et al. 2020) (white-box attack).

2.5 Turing patterns as universal adversarial attacks

In a concurrent work (Tursynbek et al. n.d.) explored the connection between Turing patterns and UAPs (Khrulkov and Oseledets 2017) and proved Turing patterns to be a special case of UAPs. This connection implies that Turing patterns may have the potential to be used as universal adversarial patterns (both image- and target-free).

2.6 Research problem

To the best of my knowledge, Single Fourier Attacks (Tsuzuku and Sato 2018) is the only well-studied approach to constructing universal black-box decision-based attacks. If and how Turing patterns as UAPs can be used as adversarial patterns, and how they can be fit for the decision-based black-box setting is the topic that is explored in this work.

3 THEORY AND ALGORITHMS

3.1 Turing patterns

The concept of Turing patterns was introduced by Alan Turing, who described how patterns in nature like stripes and spots can arise naturally out of a homogeneous uniform state. Behind these patterns there is a two-component reaction-diffusion system. It has been shown that Turing patterns can be easily generated by cellular automata (Young 1984), which can be viewed as discrete partial differential equations, and belong to the same class of equation as reaction-diffusion.

Young's algorithm for Turing pattern generation goes as follows. We consider a 2D map of cells n. Each one is either "dead" ($n(i,j) = 0$) or "alive" ($n(i,j) = 1$). This cell map is initialized randomly. After that, on each step of the algorithm the value of a cell is determined by the values of neighbouring cells, like so:

- Sum of cell values within radius R_1 is weighted with some w;
- Sum of cell values within radius R_2 is weighted with -1.

If the total sum is positive, the cell value is set to 1, otherwise 0. The process can be re-written by using convolution with specific kernel Y:

$$Y(k,l) = \begin{cases} w & \text{if } |k|^2 + |l|^2 < R_1^2 \\ -1 & \text{if } R_2^2 > |k|^2 + |l|^2 > R_1^2 \end{cases} \tag{1}$$

The coefficient w is then chosen to be such that the sum of elements of kernel Y is set to be 0. On each step of the algoritm the update rule reads like so:

$$n_{k+1}(i,j) = \frac{1}{2}\left[sign\left(Y \star n_k(i,j)\right) + 1\right] \tag{2}$$

where $\star$ represents convolution operation. Parameter R_1 can be replaced with two (L_1 and L_2), so that a circle of neighbours with radius R_1 can be turned into an ellipse with width L_1 and height L_2.

Some of the examples of such patterns are demonstrated on Figure 1.

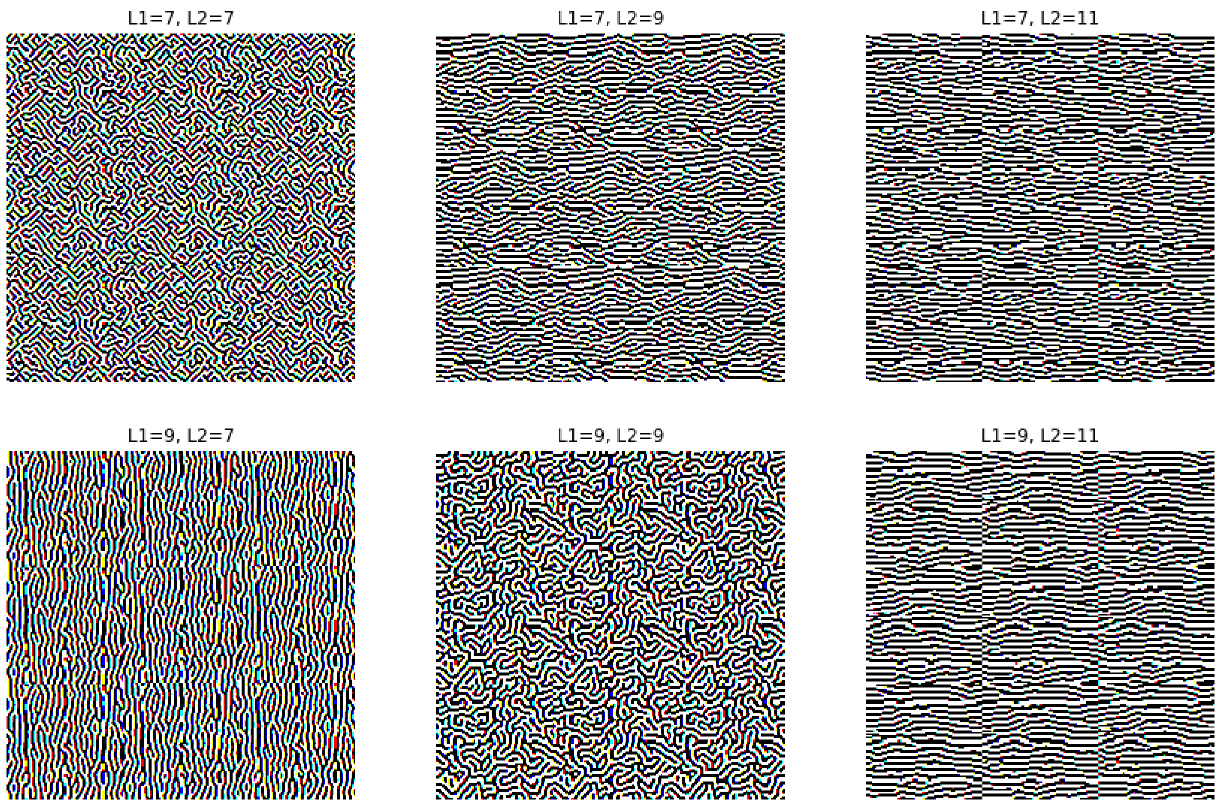


Figure 1: Examples of Turing pattern examples generated with different L_1 and L_2 values

Proposed modifications

Several modifications to this algorithm can be made to the algorithm described above to relate the patterns to the task of adversarial attacks.

1. Optimize the pattern generation parameters to fit a specific target network. That means optimizing over the following parameters: initialization map, L_1, L_2. This would give the understanding of whether Turing pattern parameters can be tailored to better attack a specific network

2. Optimize over the kernel Y of the cellular automaton to see if more efficient adversarial attacks can be achieved (not necessarily true Turing patterns). This will help gain a sense of whether new types of attacks (crafted by cellular automata) can be created

3. Optimizing with / without optimization over cellular automaton initialization maps. This will help understand whether the cellular automaton initialization plays an important role in how successful the resulting pattern is as an adversarial attack. If it is not significant, there's no need to optimize over the parameters responsible for initialization, which would simplify and speed up the optimization process

4. Explore how kernel size affects the attack success rate of the attack. This would help understand whether the size of visible lines and stripes plays a significant role in attack success rate.

5. Explore how different pattern channels should be mixed to produce a successful attack. Images that serve as the target network input have 3 channels, whereas Turing pattern generation is only described for one 2D map. This brings up the question of whether three cellular automaton generated patterns corresponding to three image channels should be completely independent or somehow mixed with each other simultaneously with cellular automaton steps. In order to understand this the following 4 options of cellular automaton modifications are tested:

   (a) Use one 2D-filter for all 3 independent pattern channels, no channel mixing

   (b) Use one 2D-filter for all pattern channels, after each convolution step the sum of each two channels becomes the remaining channel (i.e. channel3 = channel1 + channel2)

   (c) Use one 3D-filter for all 3 image channels

   (d) Use 3 separate 2D-filters (one for each channel), and a 33 matrix for channel mixing, like pointwise convolution

   One can expect for 3D-filter to be a generalization of all of the other cases, thus it would be the most optimal to optimize over. However, as will be shown further, this strategy is not definitevely better than the others.

3.2 Black-box optimization

3.2.1 Optimization task

The optimization task of finding an adversarial pattern generation is typically formulated as follows:

$$L(f(x), f(x+\tau)) \to \max_{\tau}; \quad \tau \in T \tag{3}$$

where f is the target network function, τ is the adversarial perturbation, L is some loss function, T is the search space. Typically, T is considered as a ball in l_2, l_∞ or other l_p space. Here we will consider $T = B_\infty(\varepsilon) = \{\tau : \|\tau\|_\infty \leq \varepsilon\}$ and $\varepsilon = 10$ causing each attack image pixel value to differ from the original no larger than 10. For the purpose of constructing a universal decision-based adversarial perturbation we will be using the sum of loss values for a random subset of inputs from the dataset and success rate of the attack for this small subset as loss function. Thus, the optimization task is formulated as follows:

$$\sum_{i=1}^{n} \mathbb{I}(argmax(f(x_i+\tau)) = argmax f(x_i)) \to \min_{\theta}; \quad \tau = \varepsilon \cdot Turing(\theta) \in B_\infty(\varepsilon) \tag{4}$$

where $x_i \in X = \{x_i\}_{i=1}^{n}$ is the dataset subset of size $n = 512$, θ is a set of cellular automaton parameters (e.g. L_1, L_2, initialization maps, kernel Y).

There are several options for what the set of parameters θ can look like.

1. $\theta = (L_1, L_2,$ `init_maps`$)$: optimizing true Turing pattern generation automaton parameters
2. $\theta = (Y,$ `init_maps`$)$: optimizing over cellular automaton kernel Y and initialization maps
3. $\theta = (Y)$: optimizing over cellular automaton kernel Y

3.2.2 Evolutionary algorithms

Constructing a universal black-box decision-based adversarial attack can be strictly formulated as an optimization task that should be solved via a black-box optimization methods. The choice of the black-box optimization algorithm varies in adversarial attack construction related literature. Some employ grid search (Tsuzuku and Sato 2018), Bayesian optimization (Shukla et al. 2019), or evolutionary algorithms (Alzantot et al. 2018). This work will focus on using evolutionary algorithms as a method of choice.

Evolutionary algorithms (Schwefel and Rudolph 1970) generally follow a simple structure, described in Algorithm 1. Several candidates – possible solutions to the optimization problem, also described as the minimization of the *fitness* function – are considered as a population. The population is initialized, then the following process is repeated: the members of the population are somehow recombined to produce a set number of offsprings, the offsprings randomly mutate, the fitness of the offsprings is evaluated and then the best are selected to become the new population and repeat the process.

The simplest example of such algorithm is (1+1)-ES (evolutionary strategy), where population size is equal to 1, the number of offsprings is also 1, and mutation has standard normal distribution: $x_{mutation} = x_{offspring} + \sigma\mathcal{N}(\mathbf{0}, \mathbf{I})$, using cumulative step-size adaptation for parameter σ. See Algorithm 2 for more detailed description of the algorithm.

Algorithm 1: General outline of an evolutionary algorithm

```
Initialization
repeat
    Recombination
    Mutation
    Evaluation
    Selection
until Termination criterion fulfilled;
```

Instead of using standard normal distribution for mutation generation, an arbitrary covariance matrix can be used to ahieve more complex mutations. The Covariance Matrix Adaptation Evolution Strategy combines evolution strategies, Cumulative Step-Size Adaptation, and a specific method for adaptating the covariance matrix. An outline of this algorithm is provided in the appendix (Algorithm 3). CMA-ES is an effective and robust algorithm, but it becomes catastrophically slow in high dimension due to the expensive computation of the square root of the matrix. As a workaround the covariance matrix can be approximated by a diagonal one. This leads to a computational cost linear in the dimension, rather than the original quadratic one.

3.3 Single Fourier Attacks

The main idea of Single Fourier Attacks (Tsuzuku and Sato 2018) is to make use of the fact that sensitive directions of convolutional networks are a combination of a few Fourier basis functions. They find that networks are sensitive to the directions of Fourier basis functions of some specific frequencies and propose an algorithm of black-box attack construction which relies on finding the effective frequency for each specific target network and using the corresponding Fourier basis function as a universal adversarial perturbation. An example can be found on Figure B1.

In order to test whether the cellular automaton generated images behave as Single Fourier attacks, the following approach is used. Take discrete Fourier transform (DFT) of the pattern and remove basis functions with amplitude lower than some threshold (very high to the maximum), so that only the frequencies with the highest amplitudes are left. Then reverse the DFT and check how this new pattern compares to the original one in performance as an adversarial attack. If the change is insignificant, then the original pattern effectively performs a Single Fourier attack.

4 METHODOLOGY

To conduct the necessary experiments I'm using `Python 3.6` and `pytorch` module for target model construction. To employ evolutionary algorithms, `nevergrad` module (Rapin and O. Teytaud 2018) is used, which contains the CMA-ES algorithm implementation, with optimal parameter selection built-in (see Algorithm 3 for more details).

The networks chosen to test the pattern attack success rates include: MobileNetV2, VGG-19, InceptionV3 – all pre-trained on the ImageNet (Russakovsky et al. 2015) dataset (models available from `torchvision` module). The same dataset is used for construction and testing of generated patterns. All pattern fooling rates presented in the results are computed for $n = 10000$ random dataset images. All fooling rates for patterns constructed without initialization maps are averaged over results of $n = 50$ patterns generated with random initializations of the cellular automata.

To reduce the number of parameters (important for CMA-ES speed) and the number of queries, the "tiling trick" (Meunier, Atif, and Olivier Teytaud 2019) is used: the initialization maps are considered as 7×7 big square tiles, thus reducing the number of parameters responsible for initialization maps by a factor of 1024 from $224 \times 224 \times 3$ to $7 \times 7 \times 3$. The size of the cellular automaton is by default chosen to be 13 as empirically best in (Tursynbek et al. n.d.). The number of parameters k to optimize over varies in different settings:

- $\theta = (L_1, L_2, \texttt{init\_maps})$: $k = 1 + 1 + 7 \times 7 \times 3 = 149$
- $\theta = (Y, \texttt{init\_maps})$: $k = 13 \times 13 + 7 \times 7 \times 3 = 316$
- $\theta = (Y)$, independent channels: $k = 13 \times 13 = 169$
- $\theta = (Y)$, summation channel mixing: $k = 13 \times 13 = 169$
- $\theta = (Y)$, 3D filter channel mixing: $k = 13 \times 13 \times 3 = 507$
- $\theta = (Y)$, Pointwise filter channel mixing: $k = 13 \times 13 \times 3 + 3 \times 3 = 516$

All patterns presented and evaluated below were obtained after 500 queries to the target model, which seems to be more than enough to converge to a good solution: even for cases with more than 500 parameters further optimization does not significantly improve the results.

As mention earlier in section 3.2.1, attack perturbations in $B_\infty(\varepsilon) = \{\tau : \|\tau\|_\infty \leq \varepsilon\}$ for $\varepsilon = 10$ are considered. This means that for each cellular automaton generated pattern p,

$p(i,j) \in \{0,1\}$ the attack for image x, $x(i,j) \in \{0,..255\}$ is constructed like so: $x_{attack} = x + \varepsilon \cdot (2 \cdot p - 1) = x + 10 \cdot (2 \cdot p - 1)$, so that each pixel value $x(i,j)$ is either decreased or increased by $\varepsilon = 10$.

All code and resultin patterns are available at `https://github.com/lapsya/cellular_automata_attacks`

5 RESULTS

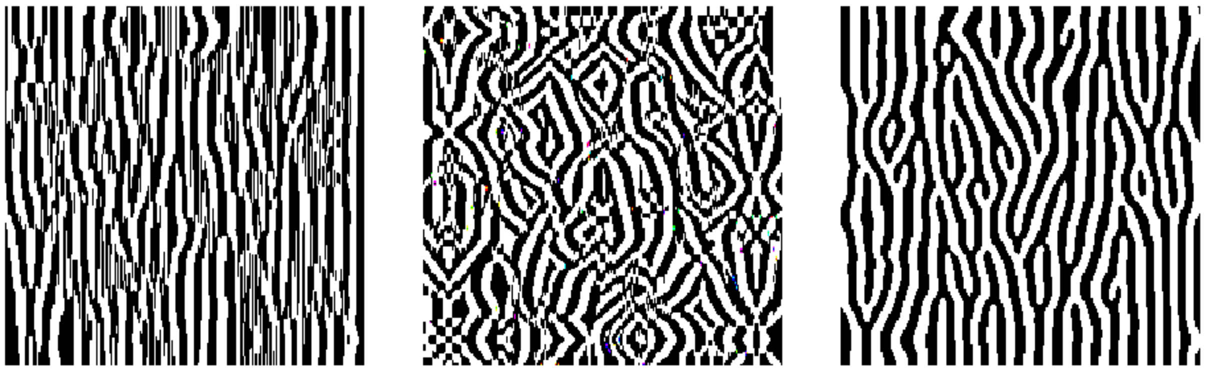
5.1 Optimizing Turing pattern parameters

In the case of optimizing over $\theta = (L_1, L_2, \texttt{init\_maps})$, the following results can be observed (see Table 1). Since Turing patterns themselves are target-free, the average fooling rate over several instances with random parameters is provided in the column "Original (average)". It can be seen that trying to fit Turing pattern parameters to a specific network definitely provides higher fooling rates, which means that cellular automaton generated patterns can indeed be modified to generate better attacks.

Table 1: Fooling rates of a random Turing pattern vs. the one optimized in parameters

Target	**Original (average)**	**Optimized in L_1, L_2 and initialization**
MobileNetV2	52.79%	56.15%
VGG-19	52.56%	55.4%
InceptionV3	45.32%	47.18%

An example of what these Turing pattern attacks look like can be found in Figure 2.



(a) MobileNetV2 (b) VGG-19 (c) InceptionV3

Figure 2: Turing patterns generated using optimized parameters L_1, L_2 and initialization maps for specific targets

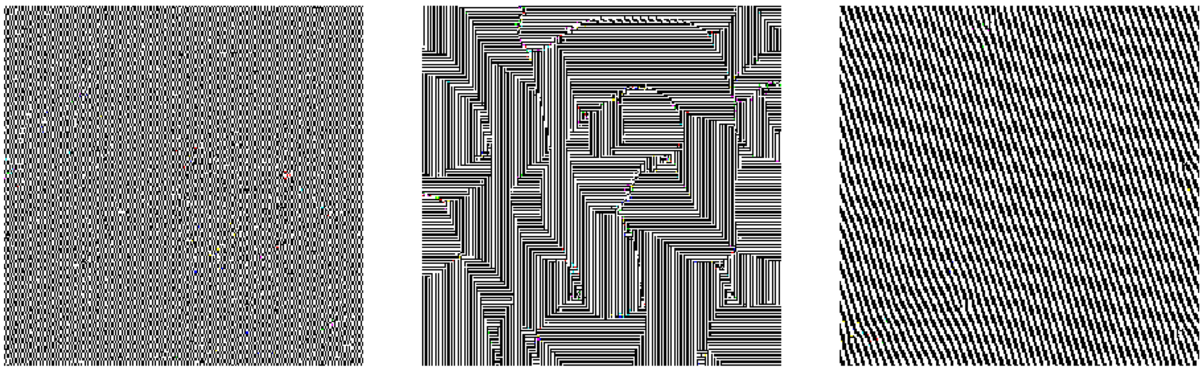
5.2 Optimizing cellular automaton parameters

In this section I will compare several approaches simultaneously: the case of $\theta = (Y, \texttt{init\_maps})$, $\theta = (Y)$, and possible channel mixing approaches for the latter case. The resulting fooling rates can be found in Table 2.

In the case of $\theta = (Y, \texttt{init\_maps})$, a significant increase in fooling rates compared to previous approaches can be seen (compare to 1), which proves that cellular automata-generated patterns can be used Examples of such attacks can be seen in Figure 3.

Table 2: Fooling rates in various optimization cases: with and without initialization and using different channel mixing strategies

	$\boldsymbol{\theta} = (\boldsymbol{Y},$	$\boldsymbol{\theta} = (\boldsymbol{Y})$			
Target	**`init_maps`)**	**Independent**	**Summation**	**3D filter**	**Pointwise**
MobileNetV2	90.59%	60.87%	**94.78%**	81.97%	94.26%
VGG-19	75.64%	60.74%	77.5%	**78.87%**	78.0%
InceptionV3	53.9%	47.45%	53.1%	51.31%	**53.33%**



(a) MobileNetV2 (b) VGG-19 (c) InceptionV3

Figure 3: Turing patterns generated using optimized filter Y and initialization maps for specific targets

When it comes to the case of $\theta = (Y)$, there are four channel mixing setups to experiment with (see full description of each one in section 3.1). While all of these approaches perform better than the average Turing pattern or even optimized Turing pattern (compare with Table 1), there is no single best approach. As mentioned earlier, it is expected that a single 3D filter, as a generalization of all the other cases, would decidedly be the best. However, the results suggest that it is not definitevely better than the other approaches. It can be seen that summation, 3D and pointwise approaches channel mixing give rather similar results, with pointwise performing slightly better on average over the three target networks. A full set of examples of the resulting patterns can be found in Appendix B Figures B4 – B7.

5.3 Transferability

Turing patterns are image-agnostic and model-agnostic adversarial examples. However, the modification of their generation algorithm proposed here is not model-agnostic, which means that the new patterns' transferability must be studied. In this section a comparison between the original Turing patterns fooling rate and fooling rates of attack transfers is presented.

In the setting of optimizing over $\theta = (L_1, L_2, \texttt{init\_maps})$, there are some cases of transfer, e.g. MobileNetV2 to VGG-19 or VGG-19 to InceptionV3, where the fooling rate is even higher than that of a random Turing pattern (approx. +4%), however there are also cases of transfer where the fooling rate drops by 3-4%. See Table 3 for full transferability results and Table 1 for comparison. Thus, transferability doesn't change drastically after optimization, and the attack transfers rather well to other network architectures.

Table 3: Transferability of optimized Turing patterns ($\theta = (L_1, L_2, \texttt{init\_maps})$)

Trained on \ Target	MobileNetV2	VGG-19	InceptionV3
MobileNetV2	56.15%	56.96%	45.16%
VGG-19	50.49%	55.4%	49.98%
InceptionV3	49.55%	48.76%	47.18%

In the cases when θ contains kernel Y (whether including intialization maps or not), the transferability seems to depend significantly on the target network. More specifically, experiments show that patterns transfer easily to MobileNetV2 and VGG-19, as the fooling rates of attacks on them are significantly higher than those of random Turing patterns. However, patterns do not transfer anywhere near as easily to InceptionV3, as there is a significant drop in fooling rates of such attacks compared to the fooling rate of random Turing patterns for this model (2-14% drop is observed). The best-transferable attack in this scenario seems to be 3D-filter cellular automaton generated patterns (see Table 4), for which the single unsuccessful attack transfer only gives 2% drop compared to a random Turing pattern. Generally, it can't be stated that these types of patterns can easily transfer between different architectures, although for some architectures they do transfer rather well (MobileNetV2 and VGG-19). For full transferability results see Appendix A Tables A1 – A4.

Table 4: Transferability of 3D-filter cellular automaton generated patterns ($\theta = (Y)$)

Trained on \ Target	MobileNetV2	VGG-19	InceptionV3
MobileNetV2	81.97%	68.30%	48.24%
VGG-19	76.15%	78.87%	43.83%
InceptionV3	61.36%	61.62%	51.31%

5.4 Cellular automaton filter size

It should be explored how Y filter size influences the attack success rate. According to the experiment results for cases when θ both does and does not include optimization over initialization maps, there is no clear dependence of the attack fooling rate on the filter size. The connection observed is depicted in Figure 4 (results for patterns without specific initialization maps are averaged over 10 random initializations). Even when a number of experiments for each filter size is made (green shaded area), the dispersion is much smaller than the fooling rate difference for different sizes. This phenomenon definitely should be carefully studied in further works.

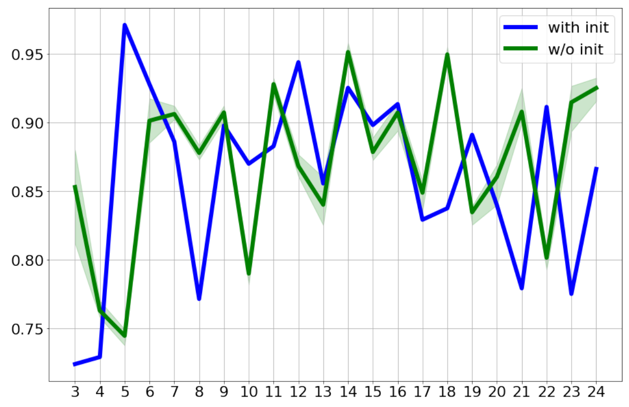


Figure 4: Dependence of attack fooling rate on Y filter size

5.5 Relation to Single Fourier Attacks

Some of the patterns resulting from cellular automata with optimized kernel Y visually look very similar to the ones generated as single Fourier harmonics, as in Tsuzuku and Sato 2018. To check if they indeed constitute the same approach to attack construction, the method described in section 3.3 was used. For the amplitude threshold two cases were considered: $a_{max} - 1$ and $0.9a_{max}$, where a_{max} is the maximum amplitude. The pattern transformation with these thresholds was applied to the two best-performing approaches: $\theta = (Y)$ with summation and pointwise channel mixing.

As a result of the pattern transformation, in most cases neither the fooling rates (e.g. Table 5) nor the visual representation of the pattern (see Figure 5) did not change drastically, which means that in these cases it is indeed a Single Fourier Attack that is taking place. However, in some cases, e.g. the case of patterns generated using summation channel mixing to attack InceptionV3, it is not true (see Figure 6, Table 6). This means that while cellular automata based generation methods can generate Single Fourier Attacks, not all the attacks generated using these methods are SFA.

For more visual examples of such transformation see Appendix B Figures B2 – B3.

Table 5: Fooling rate change after pattern targeting VGG-19 modification

Mixing \ Threshold	a_{max}	$a_{max} - 1$	$0.9 * a_{max}$
Summation	77.5%	78.0%	77.9%
Pointwise	78.0%	77.7%	78.0%

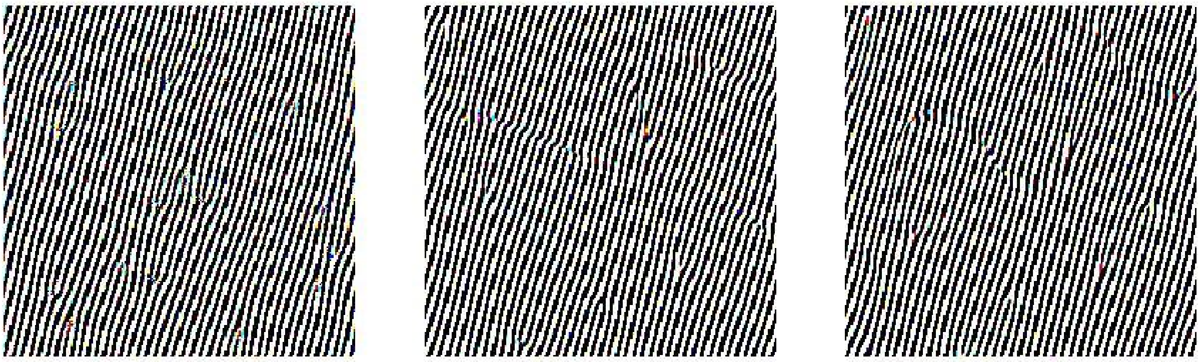


Figure 5: Pointwise channel mixing patterns attacking InceptionV3 modified with different thresholds: a_{max} (original), $a_{max} - 1$ and $0.9 * a_{max}$

Table 6: Fooling rate change after pattern targeting InceptionV3 modification

Mixing \ Threshold	a_{max}	$a_{max} - 1$	$0.9 * a_{max}$
Summation	53.1%	50.6%	51.0%
Pointwise	53.3%	53.6%	53.5%

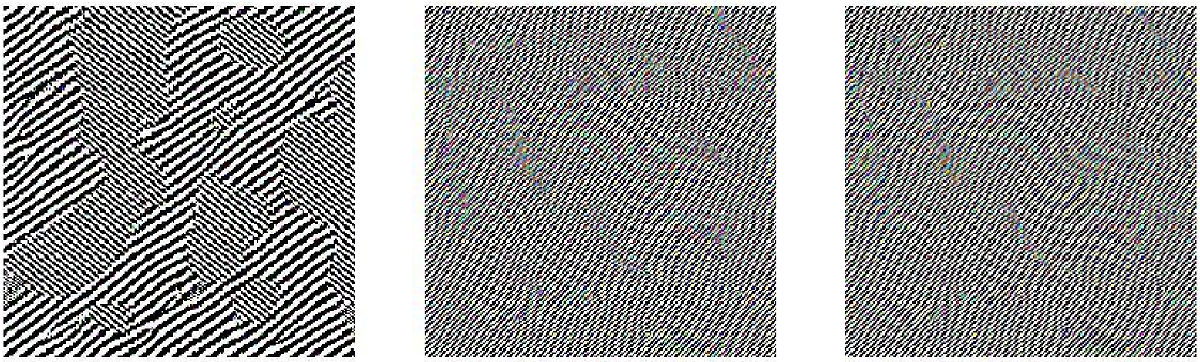


Figure 6: Summation channel mixing patterns attacking InceptionV3 modified with different thresholds: a_{max} (original), $a_{max} - 1$ and $0.9 * a_{max}$

6 DISCUSSION

6.1 Innovation

One of the innovation components of this research is the introduction of the new query-efficient methods of generating universal black-box decision-based adversarial attacks – patterns generated by cellular automata optimized for specific target network. Patterns generated by these methods have the potential to outperform existing universal (image-agnostic) black-box adversarial attack generation methods, described in section 2.4 in terms of fooling rates; see Table 7 for comparison.

Table 7: Fooling rates of universal attack methods for VGG-19, bounded to 10/255 in l_∞-norm

UAP (white-box)	**SFA (black-box)**	**Turing patterns (black-box)**	**Cellular automata generated**
64.8%	53.3%	52.5%	**78%**

The other innovation component is in the exploration of connection of Single Fourier Attacks to cellular automata generated patterns, previously not described. While no simple rule describing whether resulting attack would be a SFA or not was found, this connection deserves further studies.

6.2 Application

The methods proposed in this research can serve as adversarial attack generation algorithms for creating real-world attacks or used in adversarial training to ensure defense from these and similar attacks for DNNs already employed in real-world problem-solving.

6.3 Future research

There are several question connected to this research that remain unanswered and need to be studied further.

As was discussed in section 5.2, the evolutionary algorithm used is not well suited for search of proper channel mixing. The most simple and intuitive choice is to use "independent" mixing provides the worst result, while any other way of channel mixing gives comparable results. It may turns out that there exists some channel mixing which raise fooling rate further, and more experiments need to be conducted in that direction.

A very strange dependence of fooling rates on filter size was observed (see Figure 4). This phenomenon is completely surprising, and I don't have a reasonable explanation for this.

Finally, this work focuses on employing a specific black-box optimization method (CMA-ES), however other methods (e.g. Bayesian optimization or more complex evolutionary algorithms) may act differently and could raise the fooling rate or the transferability of the final patterns.

7 CONCLUSION

This research aimed to reach 3 main objectives. The First goal was to test if evolutionary strategies as a black-box optimization method can increase the Turing patterns' effectiveness as adversarial attacks in terms of fooling rate. It was determined that Turing patterns can indeed be tailored to attack target networks better via generation algorithm parameter optimization using evolutionary algorithms with some small loss of transferability.

The second goal was to explore the effectiveness of different modifications of Turing pattern generation algorithm to see if better performance can be achieved. The proposed modifications to Turing patterns can yield even stronger attacks in a black-box decision-based setting with some significant loss of transferability for some network architectures.

The final goal was to explore the similarity of modified Turing patterns to Single Fourier Attacks. In that regard, it has been determined that proposed methods can generate Single Fourier Attacks, however not all the attacks generated using these methods are SFA, and no specific rule to describe that was found.

REFERENCES

Szegedy, Christian et al. (2013). *Intriguing properties of neural networks.* arXiv: 1312.6199 [cs.CV] (cit. on p. 8).

Goodfellow, Ian J., Jonathon Shlens, and Christian Szegedy (2014). *Explaining and Harnessing Adversarial Examples.* arXiv: 1412.6572 [stat.ML] (cit. on p. 8).

Petrov, Deyan and Timothy M. Hospedales (2019). *Measuring the Transferability of Adversarial Examples.* arXiv: 1907.06291 [cs.LG] (cit. on p. 8).

Papernot, Nicolas et al. (2016). *Practical Black-Box Attacks against Machine Learning.* arXiv: 1602.02697 [cs.CR] (cit. on p. 8).

Huan, ZhaoXin et al. (2020). *Data-Free Adversarial Perturbations for Practical Black-Box Attack.* arXiv: 2003.01295 [cs.CV] (cit. on p. 8).

Chen, Pin-Yu et al. (2017). *ZOO: Zeroth Order Optimization based Black-box Attacks to Deep Neural Networks without Training Substitute Models.* arXiv: 1708.03999 [stat.ML] (cit. on p. 8).

Tu, Chun-Chen et al. (2018). *AutoZOOM: Autoencoder-based Zeroth Order Optimization Method for Attacking Black-box Neural Networks.* arXiv: 1805.11770 [cs.CV] (cit. on p. 8).

Ilyas, Andrew et al. (2018). *Black-box Adversarial Attacks with Limited Queries and Information.* arXiv: 1804.08598 [cs.CV] (cit. on p. 8).

Alzantot, Moustafa et al. (2018). *GenAttack: Practical Black-box Attacks with Gradient-Free Optimization.* arXiv: 1805.11090 [cs.LG] (cit. on pp. 8, 13).

Meunier, Laurent, Jamal Atif, and Olivier Teytaud (2019). *Yet another but more efficient black-box adversarial attack: tiling and evolution strategies.* arXiv: 1910.02244 [cs.LG] (cit. on pp. 8, 15).

Brendel, Wieland, Jonas Rauber, and Matthias Bethge (2017). *Decision-Based Adversarial Attacks: Reliable Attacks Against Black-Box Machine Learning Models.* arXiv: 1712.04248 [stat.ML] (cit. on p. 9).

Dong, Yinpeng et al. (2019). *Efficient Decision-based Black-box Adversarial Attacks on Face Recognition.* arXiv: 1904.04433 [cs.CV] (cit. on p. 9).

Khrulkov, Valentin and Ivan V. Oseledets (2017). “Art of singular vectors and universal adversarial perturbations”. In: *CoRR* abs/1709.03582. arXiv: 1709.03582. URL: http://arxiv.org/abs/1709.03582 (cit. on p. 9).

Tsuzuku, Yusuke and Issei Sato (2018). “On the Structural Sensitivity of Deep Convolutional Networks to the Directions of Fourier Basis Functions”. In: *CoRR* abs/1809.04098. arXiv: 1809.04098. URL: http://arxiv.org/abs/1809.04098 (cit. on pp. 9, 13, 14, 20, 28).

Chen, Sizhe et al. (2020). *Universal Adversarial Attack on Attention and the Resulting Dataset DAmageNet*. arXiv: 2001.06325 [cs.LG] (cit. on p. 9).

Tursynbek, Nurislam et al. (n.d.). “Adversarial Turing Patterns from Cellular Automata”. submitted to NeurIPS 2020 (cit. on pp. 9, 15).

Young, David A (1984). “A local activator-inhibitor model of vertebrate skin patterns”. In: *Mathematical Biosciences* 72.1, pp. 51–58 (cit. on p. 10).

Shukla, Satya Narayan et al. (2019). *Black-box Adversarial Attacks with Bayesian Optimization*. arXiv: 1909.13857 [cs.LG] (cit. on p. 13).

Schwefel, Hans-Paul and Günter Rudolph (Feb. 1970). “Contemporary Evolution Strategies”. In: *Advances in Artificial Life*. DOI: 10.1007/3-540-59496-5_351 (cit. on pp. 13, 33).

Rapin, J. and O. Teytaud (2018). *Nevergrad - A gradient-free optimization platform*. https://GitHub.com/FacebookResearch/Nevergrad (cit. on p. 15).

Russakovsky, Olga et al. (2015). “ImageNet Large Scale Visual Recognition Challenge”. In: *International Journal of Computer Vision (IJCV)* 115.3, pp. 211–252. DOI: 10.1007/s11263-015-0816-y (cit. on p. 15).

A TABLES

Table A1: Transferability of cellular automaton generated patterns ($\theta = (Y, \texttt{init\_maps})$)

Trained on \ Target	MobileNetV2	VGG-19	InceptionV3
MobileNetV2	90.59	61.27	35.15
VGG-19	62.15	75.64	31.91
InceptionV3	65.69	76.40	53.93

Table A2: Transferability of independent channels cellular automaton generated patterns ($\theta = (Y)$)

Trained on \ Target	MobileNetV2	VGG-19	InceptionV3
MobileNetV2	60.87	45.00	37.49
VGG-19	61.99	60.74	47.76
InceptionV3	53.22	54.54	47.45

Table A3: Transferability of summation channel mixing cellular automaton generated patterns ($\theta = (Y)$)

Trained on \ Target	MobileNetV2	VGG-19	InceptionV3
MobileNetV2	94.78	57.90	35.04
VGG-19	73.15	77.50	43.06
InceptionV3	63.90	64.46	53.10

Table A4: Transferability of pointwise channel mixing cellular automaton generated patterns ($\theta = (Y)$)

Trained on \ Target	MobileNetV2	VGG-19	InceptionV3
MobileNetV2	94.26	57.80	35.05
VGG-19	74.79	78.00	43.38
InceptionV3	59.29	67.70	53.33

B FIGURES

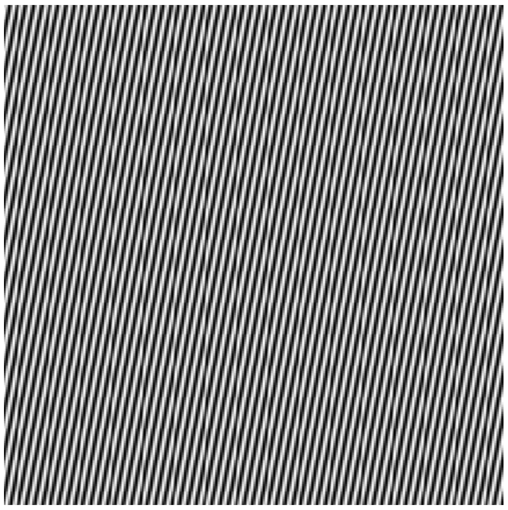


Figure B1: An example of perturbations created by Single Fourier attack, from (Tsuzuku and Sato 2018)

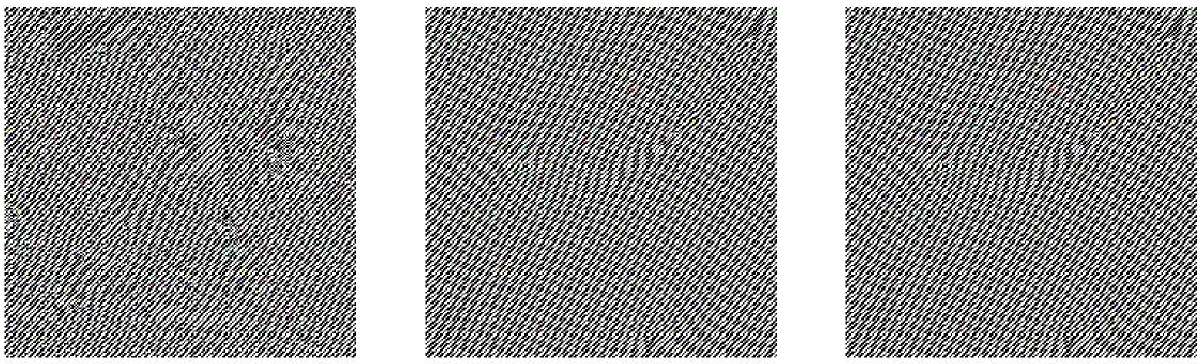


Figure B2: Summation channel mixing patterns attacking VGG-19 modified with different thresholds: a_{max} (original), $a_{max} - 1$ and $0.9 * a_{max}$

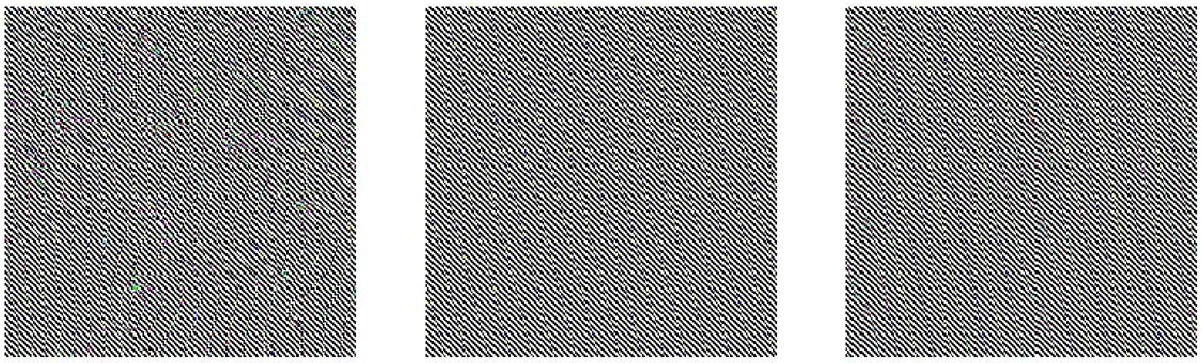


Figure B3: Pointwise channel mixing patterns attacking VGG-19 modified with different thresholds: a_{max} (original), $a_{max} - 1$ and $0.9 * a_{max}$

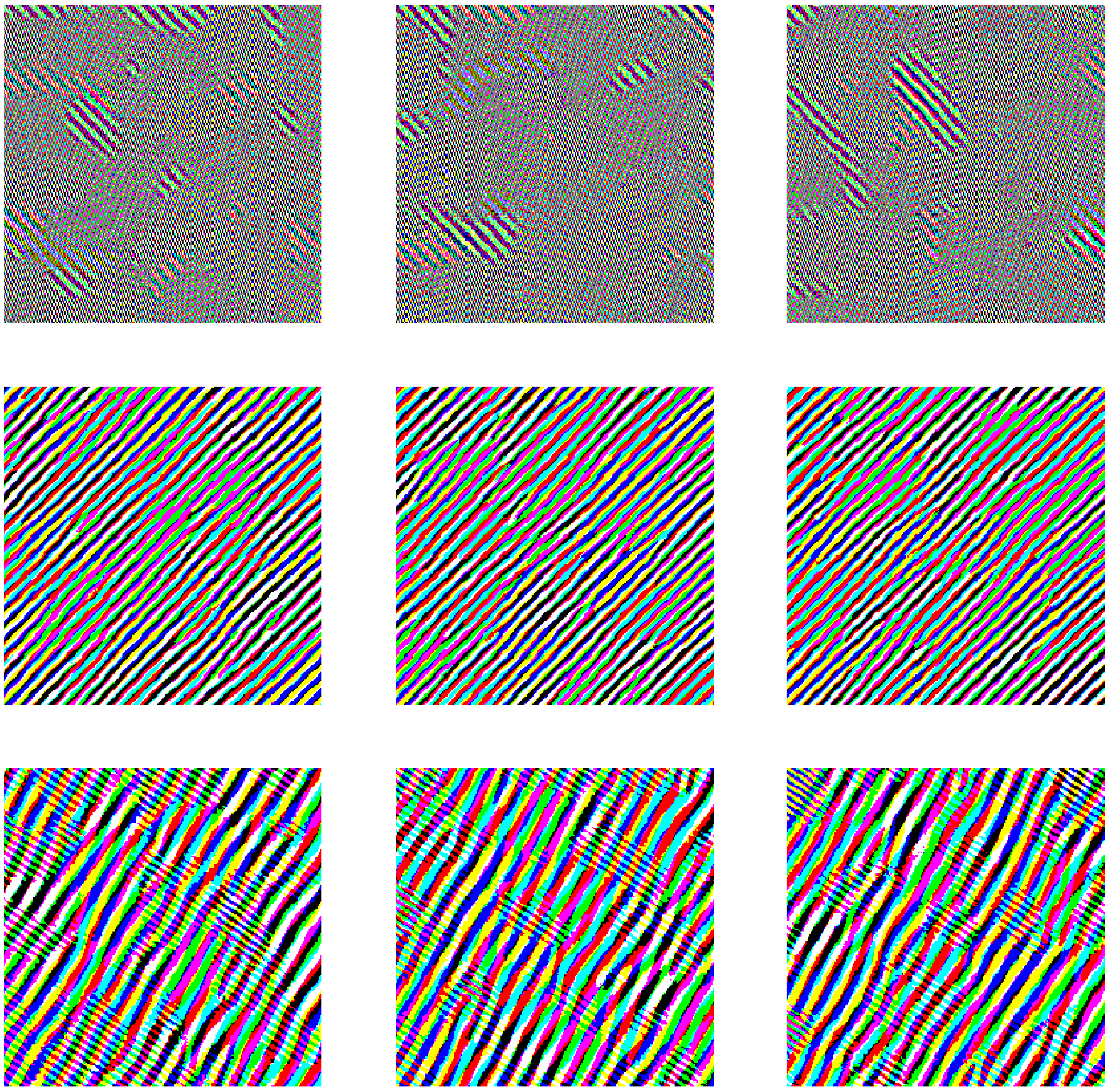


Figure B4: Examples of independent channels cellular automaton generated patterns ($\theta = (Y)$ for 3 different initializations. Top row: targeting MobileNetV2, middle row: targeting VGG-19, bottom row: targeting InceptionV3

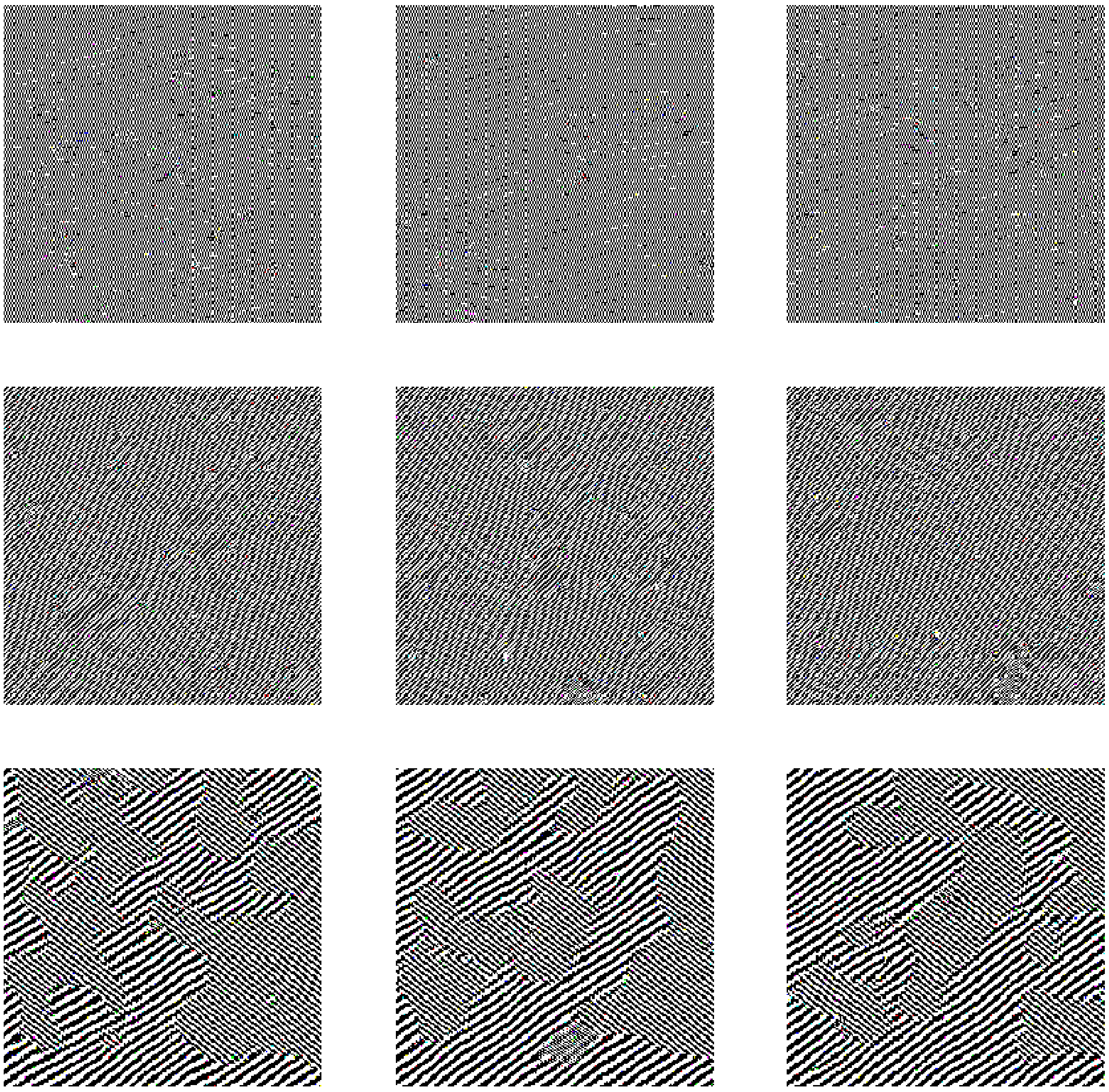


Figure B5: Examples of summation channel mixing cellular automaton generated patterns ($\theta = (Y)$ for 3 different initializations. Top row: targeting MobileNetV2, middle row: targeting VGG-19, bottom row: targeting InceptionV3

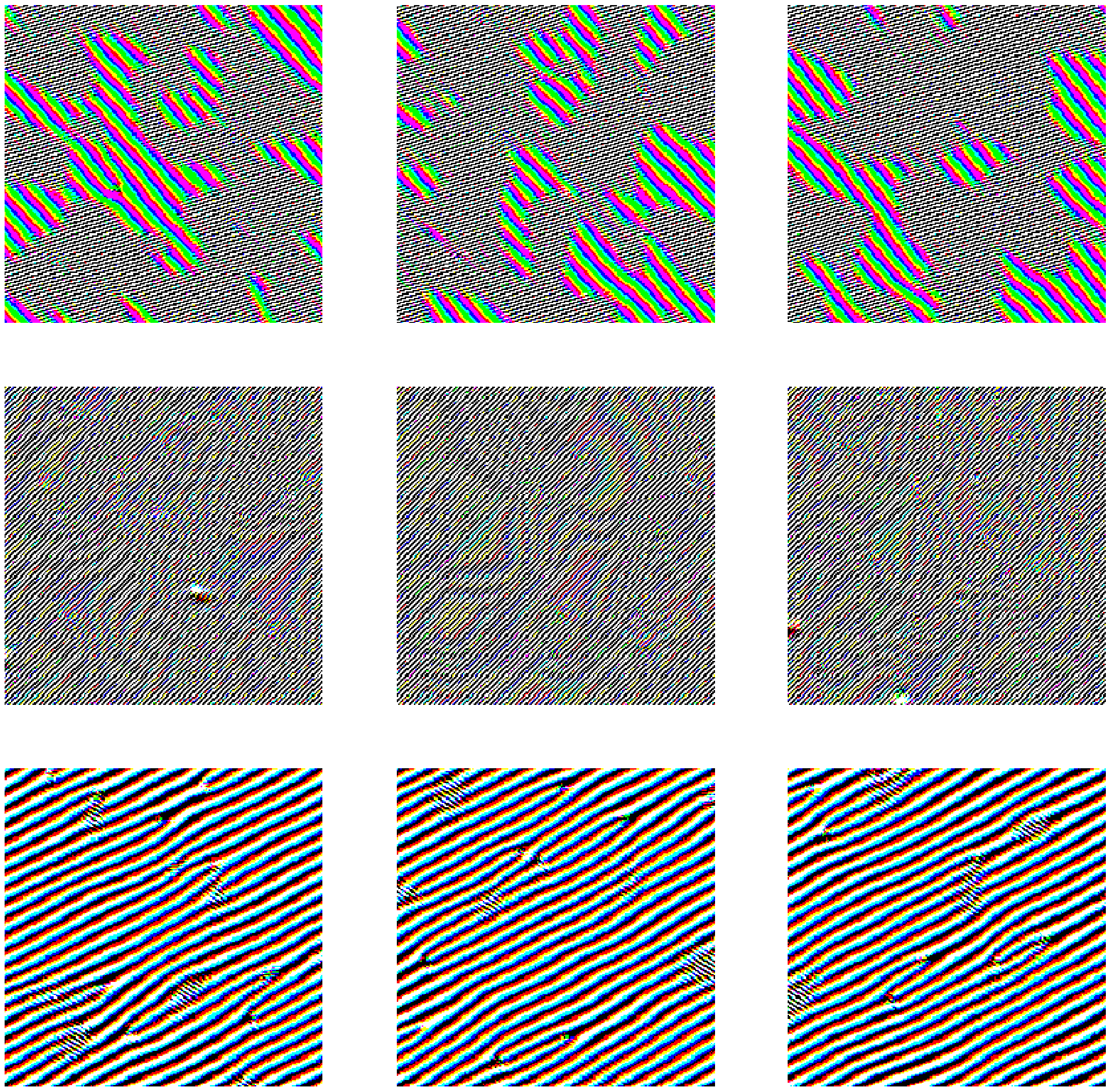


Figure B6: Examples of 3D-filter channel mixing cellular automaton generated patterns ($\theta = (Y)$ for 3 different initializations. Top row: targeting MobileNetV2, middle row: targeting VGG-19, bottom row: targeting InceptionV3

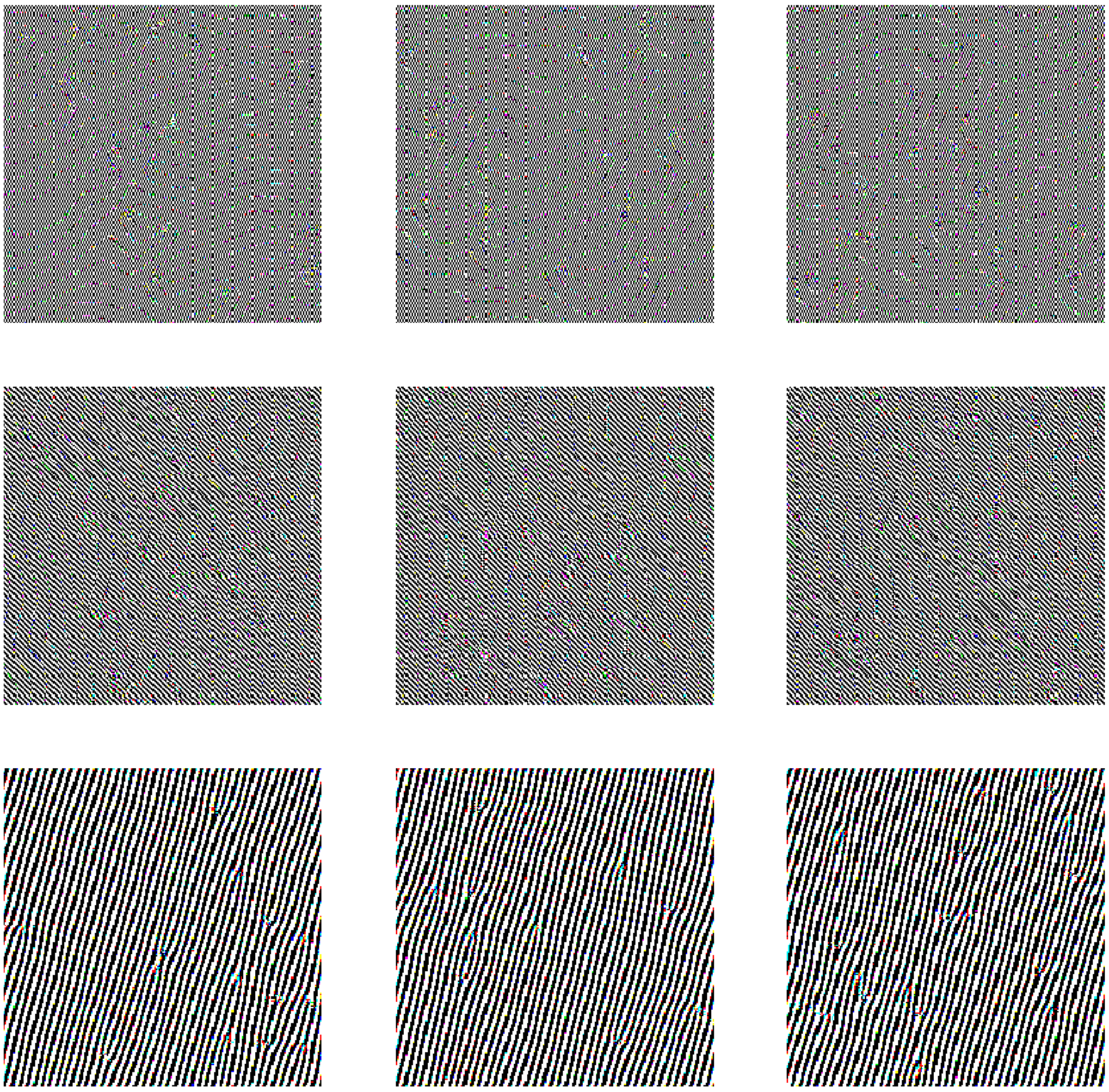


Figure B7: Examples of pointwise channel mixing cellular automaton generated patterns ($\theta = (Y)$ for 3 different initializations. Top row: targeting MobileNetV2, middle row: targeting VGG-19, bottom row: targeting InceptionV3

C ALGORITHMS

```
Algorithm 2: (1+1)-ES
$P_0 \leftarrow \{\mathbf{x}\}$
$\phi \leftarrow f(\mathbf{x})$
$t \leftarrow 0$
initialize archive $A$ for storing successful mutations
repeat
    $t \leftarrow t+1$
    $\mathbf{x}' \leftarrow \mathbf{x} + \sigma \cdot \mathcal{N}(\mathbf{0}, \mathbf{I})$
    $\phi' \leftarrow f(\mathbf{x}')$
    if $\phi' < \phi$ then
        $\mathbf{x} \leftarrow \mathbf{x}'$
        $\phi \leftarrow \phi'$
        store success in A
    else
        store failure in A
    end
    $P_t \leftarrow \{\mathbf{x}\}$
    if $t \mod n = 0$ then
        get $\#successes$ and $\#failures$ from at most $10n$ entries in $A$
        $p_S = \frac{\#successes}{\#successes + \#failures}$
        $\sigma = \begin{cases} \sigma * c & \text{if } p_S < 1/5 \\ \sigma / c & \text{if } p_S > 1/5 \\ \sigma & \text{if } p_S < 1/5 \end{cases}$
    end
until termination criterion fulfilled;
```

According to (Schwefel and Rudolph 1970), there exists an optimal set of values for (μ, w, λ) for the CMA-ES algorithm outlined below based on input dimensionality.

```
Algorithm 3: $(\mu_w, \lambda)$-CMA-ES algorithm
initialize $< \mathbf{x} >$
$\mathbf{C} \leftarrow \mathbf{I}$;
repeat
    $\mathbf{B}, \mathbf{D} \leftarrow eigendecomposition(\mathbf{C})$
    for $i = 1 \leftarrow \lambda$ do
        $y_i \leftarrow \mathbf{B} \cdot \mathbf{D} \cdot \mathcal{N}(\mathbf{0}, \mathbf{I})$
        $\mathbf{x_i} \leftarrow < \mathbf{x} > + \sigma y_i$
        $f_i \leftarrow f(\mathbf{x_i})$
    end
    $< y > \leftarrow \sum_{i=1}^{\mu} w_i y_{i:\lambda}$
    $< \mathbf{x} > \leftarrow < \mathbf{x} > + \sigma < y > = \sum_{i=1}^{\mu} w_i \mathbf{x}_{\mathbf{i}:\lambda}$
    $\sigma \leftarrow update(\sigma, \mathbf{C}, y)$
    $\mathbf{C} \leftarrow update(\mathbf{C}, w, y)$
until Termination criterion fulfilled;
```