arXiv:1709.03582v2 [cs.CV] 20 Nov 2017

Art of singular vectors and universal adversarial perturbations

Valentin Khrulkov
Skolkovo Institute of Science and Technology
143025, Nobel St. 3, Skolkovo Innovation Center
Moscow, Russia
valentin.khrulkov@skolkovotech.ru

Ivan Oseledets
Skolkovo Institute of Science and Technology
Institute of Numerical Mathematics of the
Russian Academy of Sciences
119333, Gubkina St. 8
Moscow, Russia
i.oseledets@skoltech.ru

Abstract

Vulnerability of Deep Neural Networks (DNNs) to adversarial attacks has been attracting a lot of attention in recent studies. It has been shown that for many state of the art DNNs performing image classification there exist universal adversarial perturbations — image-agnostic perturbations mere addition of which to natural images with high probability leads to their misclassification. In this work we propose a new algorithm for constructing such universal perturbations. Our approach is based on computing the so-called (p, q)-singular vectors of the Jacobian matrices of hidden layers of a network. Resulting perturbations present interesting visual patterns, and by using only 64 images we were able to construct universal perturbations with more than 60 % fooling rate on the dataset consisting of 50000 images. We also investigate a correlation between the maximal singular value of the Jacobian matrix and the fooling rate of the corresponding singular vector, and show that the constructed perturbations generalize across networks.

1. Introduction

Deep Neural Networks (DNNs) with great success have been applied to many practical problems in computer vision [11, 20, 9] and in audio and text processing [7, 13, 4]. However, it was discovered that many state-of-the-art DNNs are vulnerable to adversarial attacks [6, 14, 21], based on adding a *perturbation* of a small magnitude to the image. Such perturbations are carefully constructed in order to lead to misclassification of the perturbed image and moreover may attempt to force a specific predicted class (targeted attacks), as opposed to just any class different from the ground truth (untargeted attacks). Potential undesirable usage of adversarial perturbations in practical applications such as autonomous driving systems and malware detection has been studied in [10, 8]. This also motivated the research on defenses against various kinds of attack strategies [16, 5].

In the recent work Moosavi *et al*. [14] have shown that there exist *universal* adversarial perturbations — image-agnostic perturbations that cause most natural images to be misclassified. They were constructed by iterating over a dataset and recomputing the "worst" direction in the space of images by solving an optimization problem related to geometry of the decision boundary. Universal adversarial perturbations exhibit many interesting properties such as their universality *across networks*, which means that a perturbation constructed using one DNN will perform relatively well for other DNNs.

We present a new algorithm for constructing universal perturbations based on solving simple optimization problems which correspond to finding the so-called (p, q)-singular vector of the Jacobian matrices of feature maps of a DNN. Our idea as based on the observation that since the norm of adversarial perturbations is typically very small, perturbations in the non-linear maps computed by the DNN can be reasonably well approximated by the Jacobian matrix. The (p, q)-singular vector of a matrix A is defined as the solution of the following optimization problem

$$\|Av\|_q \to \max, \quad \|v\|_p = 1, \tag{1}$$

and if we desire $\|v\|_p = L$ instead, it is sufficient to multiply the solution of (1) by L. Universal adversarial perturbations are typically generated with a bound in the ∞-norm, which motivates the usage of such general construction. To obtain the (p, q)-singular vectors we use a modification of the standard power method, which is adapted to arbitrary p-norms. The main contributions of our paper are

- We propose an algorithm for generating universal adversarial perturbation, using the generalized power

method for computing the (p, q)-singular vectors of the Jacobian matrices of the feature maps.

- Our method is able to produce relatively good universal adversarial examples from a relatively small number of images from a dataset.
- We investigate a correlation between the largest (p, q)-singular value and the fooling rate of the generated adversarial examples; this suggests that this singular value can be used as a quantitative measure of the robustness of a given neural network and can be in principle incorporated as the regularizer for the DNNs.
- We analyze various properties of the computed adversarial perturbations such as generalization across networks and dependence of the fooling rate on the number of images used for construction of the perturbation.

2. Problem statement

Suppose that we have a standard feed-forward DNN which takes a vector x as the input, and outputs a vector of probabilities $p(x)$ for the class labels. Our goal given parameters $q \geq 1$ and $L > 0$ is to produce a vector ε such that

$$\arg\max p(x) \neq \arg\max p(x+\varepsilon), \quad \|\varepsilon\|_q = L, \tag{2}$$

for as many x in a dataset as possible. Efficiency of a given universal adversarial perturbation ε for the dataset X of the size N is called the *fooling rate* and is defined as

$$\frac{|\{x \in X : \arg\max p(x) \neq \arg\max p(x+\varepsilon)\}|}{N}. \tag{3}$$

Let us denote the outputs of the i-th hidden layer of the network by $f_i(x)$. Then for a small vector ε we have

$$f_i(x+\varepsilon) - f_i(x) \approx J_i(x)\varepsilon,$$ 梯度 x 扰动

where

$$J_i(x) = \left.\frac{\partial f_i}{\partial x}\right|_x,$$

is the Jacobian matrix of f_i. Thus, for any q-norm

$$\|f_i(x+\varepsilon) - f_i(x)\|_q \approx \|J_i(x)\varepsilon\|_q, \tag{4}$$

We can conclude that for perturbations which are small in magnitude in order to sufficiently perturb the output of a hidden layer, it is sufficient to maximize right-hand side of the eq. (4). It seems reasonable to suggest that while propagating further in the network it will dramatically change the predicted label of x.

Thus to construct an adversarial perturbation for an individual image x we need to solve

$$\|J_i(x)\varepsilon\|_q \to \max, \quad \|\varepsilon\|_p = L, \tag{5}$$

and due to homogeneity of the problem defined by eq. (5) it is sufficient to solve it for $\|\varepsilon\|_p = 1$. The solution of (5) is defined up to multiplication by -1 and is called the (p, q)-singular vector of $J_i(x)$. Its computation in a general case is the well-known problem [2, 3]. In several cases e.g. $p = \infty, q = \infty$ and $p = 1, q = 1$, algorithms for finding the exact solution of the problem (5) are known [22], and are based on finding the element of maximal absolute value in each row (column) of a matrix. However, this approach requires iterating over all elements of the matrix A and thus has complexity $O(nm)$ for the matrix of size $n \times m$. Typical size of such a matrix appearing in our setting, e.g. taking VGG-19 network, output of the first pooling layer, and batch size of 64 (usage of a batch of images is explained further in the text), would be 9633792×802816, which requires roughly 30 TB of memory to store and makes these algorithms completely impractical. In order to avoid these problems, we switch to *iterative* methods. Instead of evaluating and storing the full matrix A we use only the *matvec function* of A, which is the function that given an input vector v computes an ordinary product Av without forming the full matrix A, and typically has $O(n)$ complexity. In many applications that deal with extremely large matrices using matvec functions is essentially mandatory.

For computing the (p, q)-singular vectors there exists a well-known Power Method algorithm originally developed by Boyd [3], which we explain in the next section. We also present a modification of this method in order to construct *universal* adversarial perturbations.

3. Generalized power method

Suppose that for some linear map A we are given the matvec functions of A and $A^\top$. Given parameter $r \geq 1$ we also define a function

$$\psi_r(x) = \operatorname{sign} x |x|^{r-1}, \tag{6}$$

which applies to vectors element-wise. As usual for $r \geq 1$ we also define r' such that $\frac{1}{r} + \frac{1}{r'} = 1$. Then, given some initial condition x, one can apply the following algorithm 1 to obtain a solution of (5). In the case $p = q = 2$ it becomes the familiar power method for obtaining the largest eigenvalue and the corresponding eigenvector, applied to the matrix $A^\top A$.

The discussion so far applies to finding an adversarial perturbation for an instance x. To produce *universal* adversarial perturbation we would like to maximize the left-hand size of (5) uniformly across all the images in the dataset X. For this we introduce a new optimization problem

$$\sum_{x_j \in X} \|J_i(x_j)\varepsilon\|_q^q \to \max, \quad \|\varepsilon\|_p = L. \tag{7}$$

A solution of the problem defined by eq. (7) uniformly perturbs the output of the i-th layer of the DNN, and thus can

$\psi_r(x) = \mathrm{sign}\, x\, |x|^{r-1}$

$\psi_q(Ax) = \mathrm{sign}\, Ax\, |Ax|^{q-1}$

Algorithm 1 Power method for generating the (p, q)-singular vectors of a linear map A

Inputs: initial condition x, the matvec functions of A and $A^\top$
$x \leftarrow \frac{x}{\|x\|_p}$ ▷ (p, q)-singular vector
$s \leftarrow \|Ax\|_q$ ▷ (p, q)-singular value
while not converged **do**
  $Sx \leftarrow \psi_{p'}(A^\top \psi_q(Ax))$
  $x \leftarrow \frac{Sx}{\|Sx\|_p}$
  $s \leftarrow \|Ax\|_q$
return x, s

Algorithm 2 Stochastic Power Method for generating universal adversarial perturbations

Inputs: a batch of images $X_b = \{x_1, x_2, \dots x_b\}$, $f_i(x)$ - fixed hidden layer of the DNN
for $x_j \in X$ **do**
  Construct the matvec functions of $J_i(x_j)$ and $J_i^\top(x_j)$
Construct the matvec functions of $J_i(X_b)$ and $J_i^\top(X_b)$ defined in eq. (9)
Run algorithm 1 with desired p and q
return ε ▷ the universal perturbation

serve as the universal adversarial perturbation due to the reasons discussed in the introduction. Note that the problem given in eq. (7) is exactly equivalent to

$$\|J_i \varepsilon\|_q \to \max, \quad \|\varepsilon\|_p = L,$$

where J_i is the matrix obtained via stacking $J_i(x_j)$ vertically for each $x_j \in X$. To make this optimization problem tractable, we apply the same procedure to some randomly chosen subset of images (batch) $X_b \subset X$, obtaining

$$\sum_{x_j \in X_b} \|J_i(x_j)\varepsilon\|_q^q \to \max, \quad \|\varepsilon\|_p = L, \tag{8}$$

and hypothesize that the obtained solution will be a good approximate to the exact solution of (7). We present this approach in more detail in the next section.

4. Stochastic power method

Let us choose a fixed batch of images $X_b = \{x_1, x_2 \dots x_b\}$ from the dataset and fix a hidden layer of the DNN, defining the map $f_i(x)$. Denote sizes of x, $f_i(x)$ by n and m correspondingly. Then, using the notation from section 2 we can compute $J_i(x_j) \in \mathbb{R}^{m \times n}$ for each $x_j \in X_b$. Let us now stack these Jacobian matrices vertically obtaining the matrix $J_i(X_b)$ of size $bm \times n$:

$$J_i(X_b) = \begin{bmatrix} J_i(x_1) \\ J_i(x_2) \\ \dots \\ J_i(x_b) \end{bmatrix}. \tag{9}$$

Note that to compute the matvec functions of $J_i(X_b)$ and $J_i^\top(X_b)$ it suffices to be able to compute the individual matvec functions of $J_i(x)$ and $J_i^\top(x)$. We will present an algorithm for that in the next section and for now let us assume that these matvec functions are given. We can now apply algorithm 1 to the matrix $J_i(X_b)$ obtaining Stochastic Power Method (SPM). Note that in algorithm 2 we could in principle change the batch X_b between iterations of the power method to compute "more general" singular vectors. However in our experiments we discovered that it almost does not affect the fooling rate of the generated universal perturbation.

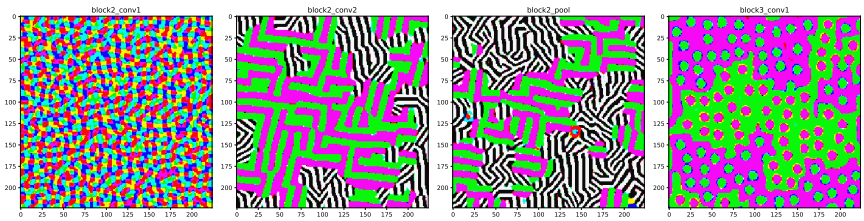
5. Efficient implementation of the matvec functions

Matrices involved in algorithm 2 for typical DNNs are too large to be formed explicitly. However, using automatic differentiation available in most deep learning packages it is possible to construct matvec functions which then are evaluated in a fraction of a second. To compute the matvecs we follow the well-known approach based on Pearlmutter's R-operator [17], which could be briefly explained as follows. Suppose that we are given an operation $\mathrm{grad}_x[f](x_0)$ which computes the gradient of a scalar function $f(x)$ with respect to the vector variable x at the point x_0. Let $f_i(x)$ be some fixed layer of the DNN, such that $x \in \mathbb{R}^n$ and $f_i(x) \in \mathbb{R}^m$, thus $J_i(x) \in \mathbb{R}^{m \times n}$ and for vectors $v_1 \in \mathbb{R}^n$, $v_2 \in \mathbb{R}^m$ we would like to compute $J_i(x)v_1$, $J_i^\top(x)v_2$ at some fixed point x. These steps are presented in algorithm 3.
For a given batch of images this algorithm is run only once.

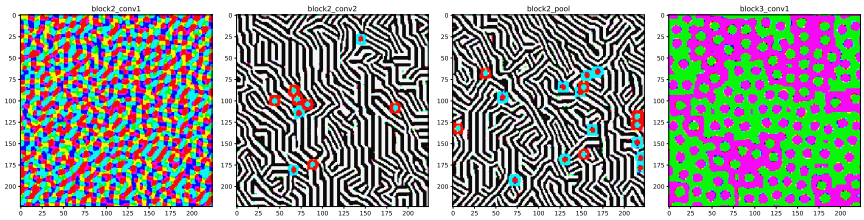
Algorithm 3 Constructing the matvec functions of the Jacobian matrix of a hidden layer of a DNN

Inputs :$v_1 \in \mathbb{R}^n$, $v_2 \in \mathbb{R}^m$ - vectors to compute the matvec functions of, $f_i(x)$ - fixed hidden layer of the DNN
$J_i^\top(x)v_2 \leftarrow \mathrm{grad}_v[\langle v, f_i(x)\rangle](v_2)$
$g(v_2) \leftarrow \langle J_i^\top(x)v_2, v_1\rangle$
$J_i(x)v_1 \leftarrow \mathrm{grad}_{v_2}[g](\mathbf{0}_m)$
return $J_i(x)v_1, J_i^\top(x)v_2$

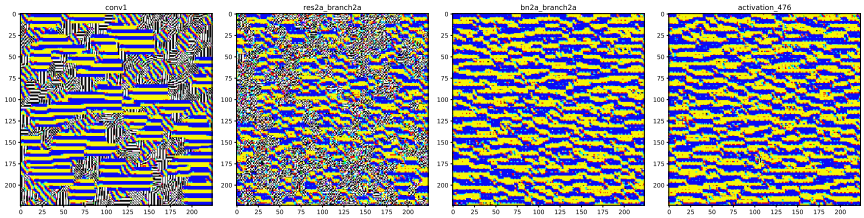
Let us summarize our approach for generating universal perturbations. Suppose that we have some dataset of natural images X and a fixed deep neural network trained to perform image classification. At first we choose a fixed random



(a) VGG-16



(b) VGG-19



(c) ResNet50

Figure 1: Universal adversarial perturbations constructed using various layers of various DNNs.

batch of images X_b from X and specify a hidden layer of the DNN. Then using algorithm 3 we construct the matvec functions of the matrix defined by eq. (9). Finally we run algorithm 2 to obtain the perturbation and then rescale it if necessary.

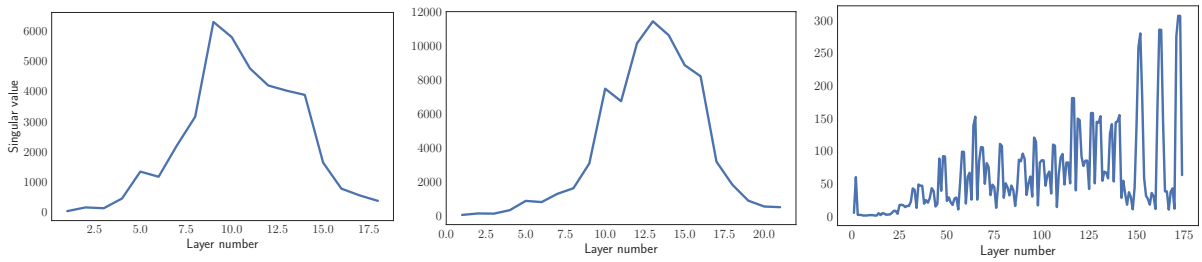
6. Experiments

In this section we analyze various adversarial perturbations constructed as discussed in section 5. For testing purposes we use the ILSVRC 2012 validation dataset [18] (50000 images).

6.1. Adversarial perturbations

In our experiments we chose $p = \infty, q = 10$ and computed the (p, q) - singular vectors for various layers of VGG-16 and VGG-19 [19] and ResNet50 [9]. $q = 10$ was chosen to smoothen optimization problem and effectively serves as the replacement for $q = \infty$, for which the highest fooling rates were reported in [14]. We also investigate other values of q in section 6.3. Batch size in algorithm 2 was chosen to be 64 and we used the same 64 images to construct all the adversarial perturbations. Some of the computed singular vectors are presented in figs. 1a to 1c. We observe that computed singular vectors look visually appealing and present interesting visual patterns. Possible interpretation of these patterns can be given if we note that extremely similar images were computed in [15] in relation to *feature visualization*. Namely, for various layers in GoogLeNet [20] the images which activate a particular neuron were computed. In particular, visualization of the layer `conv2d0`, which corresponds to *edge detection*, looks surprisingly similar to several of our adversarial perturbations. Informally speaking, this might indicate that adversarial perturbations constructed as the (p, q)-singular vectors attack a network by ruining a certain level of *image understanding*, where in particular first layers correspond to edge detection. This is partly supported by the fact that the approach used for feature visualization in [15] is based on computing the Jacobian matrix of a hidden layer and maximizing the response of a fixed neuron, which is in spirit related to our method.

To measure how strongly the (p, q)-singular vector disturbs the output of the hidden layer based on which it was constructed, we evaluate the corresponding singular value. We have computed it for all the layers of VGG-16, VGG-19 and ResNet50. Results are given in fig. 2. Note that in general singular values of the layers of ResNet50 are much smaller in magnitude than those of the VGG nets, which is further shown to roughly correspond to the obtained fooling rates. Convergence of algorithm 2 is analyzed in fig. 3. We ob-



(a) VGG-16 (b) VGG-19 (c) ResNet50

Figure 2: (p, q)-singular values for all the layers of various DNNs. Values $p = \infty, q = 10$ were used.

serve that a relatively low number of iterations is required to achieve good accuracy. In particular if each evaluation of the matvec functions takes $O(n)$ operations, the total complexity is $O(dn)$ for d iterations, which for d as small as 60 is a big improvement compared to $O(n^2)$ of the exact algorithm.

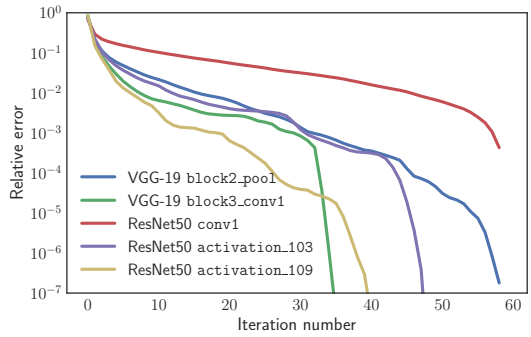


Figure 3: Convergence of the $(\infty, 10)$-singular value. Relative error of the singular value w.r.t iteration number is shown.

6.2. Fooling rate and singular values

As a next experiment we computed and compared the fooling rate of the perturbation by various computed singular vectors. We choose [1] $\|\varepsilon\|_\infty = 10$ – recall that this can be achieved just by multiplying the computed singular vector by the factor of 10. Results are given in tables 1 to 3.
We see that using only 64 images allowed us to achieve more than 40% fooling rate for all the investigated networks on the dataset containing 50000 images of 1000 different classes. This means that by analyzing less than 0.15% of the dataset it is possible to design strong universal adversarial attacks generalizing to many unseen classes and images. Similar fooling rates reported in [14, Figure 6] required roughly 3000 images to achieve (see section 6.4 for further comparison). Examples of images after addition of the adversarial perturbation with the highest fooling rate for VGG-19 are given in fig. 6, and their predicted classes for various adversarial attacks (for each network we choose the adversarial perturbation with the highest fooling rate) are reported in tables 5 to 7. We note that the top-1 class probability for images after the adversarial attack is relatively low in most cases, which might indicate that images are moved away from the decision boundary. We test this behavior by computing the top-5 probabilities for several values of the ∞-norm of the adversarial perturbation. Results are given in fig. 4. We see that top-1 probability decreases significantly and becomes roughly equal to the top-2 probability. Similar behavior was noticed in some of the cases when the adversarial example failed to fool the DNN — top-1 probability still has decreased significantly. It is also interesting to note that such adversarial attack indeed introduces many new *edges* in the image, which supports the claim made in the previous section.

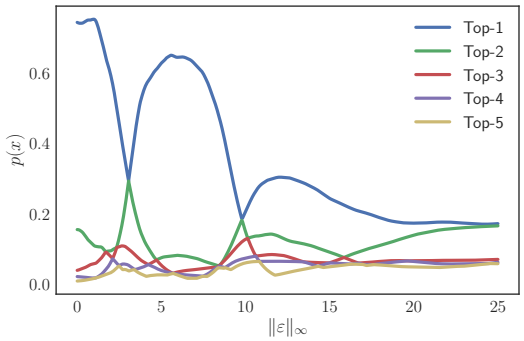


Figure 4: Top-5 probabilities predicted by the VGG-19 w.r.t ∞-norm of the universal adversarial perturbation. Tests were run for image 3 from fig. 6. Universal adversarial perturbation with highest fooling rate in table 2 was chosen.

As a next experiment we investigate the dependence of the achieved fooling rate on the batch size used in algorithm 2. Some of the results are given in fig. 5. Surprisingly, in-

[1] Pixels in the images from the dataset are normalized to be in $[0, 255]$ range, so by choosing $\|\varepsilon\|_\infty = 10$ we make the adversarial perturbations *quasi-imperceptible* to human eye

Layer name	block2_pool	block3_conv1	block3_conv2	block3_conv3
Singular value	1165.74	2200.08	3146.66	6282.64
Fooling rate	**0.52**	0.39	0.50	0.50

Table 1: Fooling rates for VGG-16

Layer name	block2_pool	block3_conv1	block3_conv2	block3_conv3
Singular value	784.82	1274.99	1600.77	3063.72
Fooling rate	**0.60**	0.33	0.50	0.52

Table 2: Fooling rates for VGG-19

Layer name	conv1	res3c_branch2a	bn5a_branch2c	activation_8
Singular value	59.69	19.21	138.81	15.55
Fooling rate	**0.44**	0.35	0.34	0.34

Table 3: Fooling rates for ResNet50

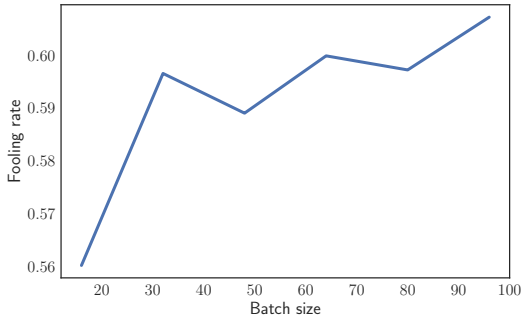


Figure 5: Dependence of the fooling rate on the batch size. `block2_pool` layer in VGG-19 was used for the experiment.

creasing the batch size does not significantly affect the fooling rate and by using as few as 16 images it is possible to construct the adversarial perturbations with 56% fooling rate. This suggests that the singular vector constructed using Stochastic Power Method reasonably well approximates solution of the general optimization problem (7).

It appears that higher singular value of the layer does not necessarily indicate higher fooling rate of the corresponding singular vector. However, as shown on fig. 2 the singular values of various layers VGG-19 are in general larger than those of VGG-16, and of VGG-16 are in general larger than the singular values of ResNet50, which is roughly in correspondence between the maximal fooling rates we obtained for these networks. Moreover, layers closer to the input of the DNN seem to produce better adversarial perturbations, than those closer to the end.

Based on this observation we hypothesize that to defend the DNN against this kind of adversarial attack one can choose some subset I of the layers (preferably closer to the input) of the DNN and include the term

$$\sum_{i \in I} \|J_i(X)\|_q^q,$$

in the regularizer, where X indicates the current learning batch. We plan to analyze this approach in future work.

Finally, we investigate if our adversarial perturbations generalize across different networks. For each DNN we have chosen the adversarial perturbation with the highest fooling rate from tables 1 to 3 and tested it against other networks. Results are given in table 4. We see that these adversarial perturbations are indeed *doubly* universal, reasonably well generalizing to other architectures. Surprisingly, in some cases the fooling rate of the adversarial perturbation constructed using other network was *higher* than that of the network's own adversarial perturbation. This universality might be explained by the fact that if Deep Neural Networks independently of specifics of their architecture indeed learn to detect low-level patterns such as edges, then adding an edge-like noise has a high chance to ruin the prediction. It is interesting to note that the adversarial perturbation obtained using `block2_pool` layer of VGG-19 is the most efficient one, in correspondence with its interesting edge-like structure.

6.3. Dependence of the perturbation on q

In the analysis so far we have chosen $q = 10$ as an approximate to $q = \infty$. However, any value of q can be used for constructing the adversarial perturbations and in this subsection we investigate how the choice of q affects the fooling rate and the generated perturbations (while keeping $p = \infty$). Perturbations computed for several different values of q are presented in fig. 7, and the corresponding

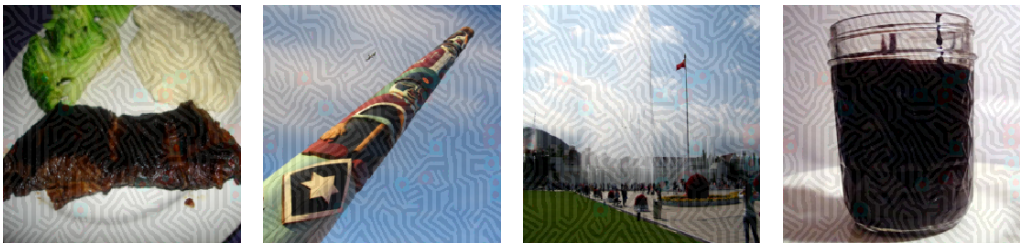


Figure 6: Examples of images misclassified after the adversarial attack (the attack based on `block2_pool` layer of VGG-19 is shown). Predicted classes are given in tables 5 to 7.

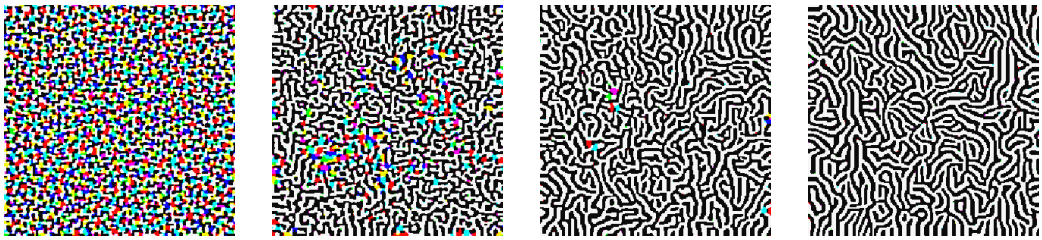


Figure 7: Adversarial perturbations constructed for various values of q. Presented images correspond to values q uniformly increasing from 1.0 to 5.0. `block2_pool` layer of VGG-19 was used.

	VGG-16	VGG-19	ResNet50
VGG-16	0.52	**0.60**	0.39
VGG-19	0.48	**0.60**	0.38
ResNet50	0.41	**0.47**	0.44

Table 4: Generalization of the adversarial perturbations across networks. Columns indicate the DNN for which the adversarial perturbation was computed, rows indicate on which network it was tested. Adversarial perturbations with highest fooling rates in tables 1 to 3 were chosen.

fooling rates are reported in fig. 8. We observe that bigger values of q produce more clear edge-like patterns, which is reflected in the increase of the fooling rate. However, the maximal fooling rate seems to be achieved at $q \approx 5$, probably because it is 'smoother' substitute for $q = \infty$ than $q = 10$, which might be important in such large scale problems.

6.4. Comparison of the algorithms

In this subsection we perform a comparison of the algorithm presented in Moosavi *et al.*, which we refer to as UAP, and our method. For the former we use the `Python` implementation https://github.com/LTS4/universal/. Since one of the main features of our method is an extremely low number of images used for constructing the perturbation, we decided to compare the fooling rates of universal perturbations

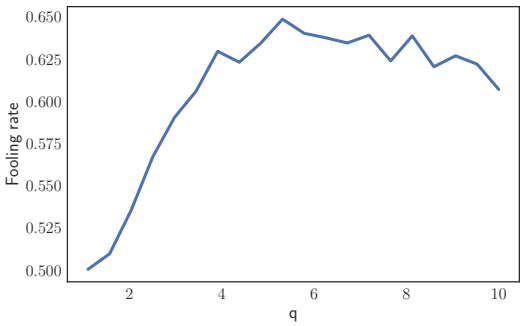


Figure 8: Dependence of the fooling rate on the value of q. As before, $p = \infty$ and norm of the adversarial perturbation was set to be 10.

constructed using these two methods for various batch sizes. Results are presented in fig. 9. Note that our method indeed captures the universal attack vector relatively fast, and the fooling rate stabilizes on roughly 63%, while the fooling rate of the perturbation constructed by the UAP method starts low and then gradually increases as more images are added. Running time of our algorithm depends on which hidden layer we use. As an example for `block2_pool` layer of VGG-19 the running time per iteration of the power method for a batch of one image was roughly 0.06 seconds (one NVIDIA Tesla K80 GPU was used and the algorithm was implemented using `Tensorflow` [1] and `numpy` libraries). Since the running time per iteration linearly de-

	image_1	image_2	image_3	image_4
$p(x)$	mashed_potato 53.9%	pole 37.6%	fountain 55.0%	goblet 12.1%
$p(x+\varepsilon)$	head_cabbage 29.4%	rubber_eraser 39.3%	carousel 61.4%	bucket 36.6%

Table 5: VGG-16

	image_1	image_2	image_3	image_4
$p(x)$	mashed_potato 68.3%	flagpole 37.4%	fountain 74.5%	coffee_mug 23.1%
$p(x+\varepsilon)$	flatworm 26.5%	letter_opener 19.9%	pillow 20.6%	candle 40.3%

Table 6: VGG-19

	image_1	image_2	image_3	image_4
$p(x)$	mashed_potato 94.2%	totem_pole 43.1%	flagpole 35.3%	chocolate_sauce 22.0%
$p(x+\varepsilon)$	stole 21.7%	fountain_pen 27.6%	monitor 9.76%	goblet 40.3%

Table 7: ResNet50

pends on the batch size b, the total running time could be estimated as $0.06bd$ seconds for d iterations. By fixing $b = 32$ and $d = 30$ we obtain that the total running time to generate the universal perturbation with approximately 60% fooling rate on the whole dataset is roughly 1 minute (we did not include the time required to compute the symbolic Jacobian matvecs since it is performed only once, and is also required in the implementation of UAP, though different layer is used). In our hardware setup the running time of the UAP algorithm with batch size 128 was approximately 10 minutes, and the fooling rate of roughly 20% was achieved. According to [14, Figure 6] approximately 3000 images will be required to obtain the fooling rates of order 60%.

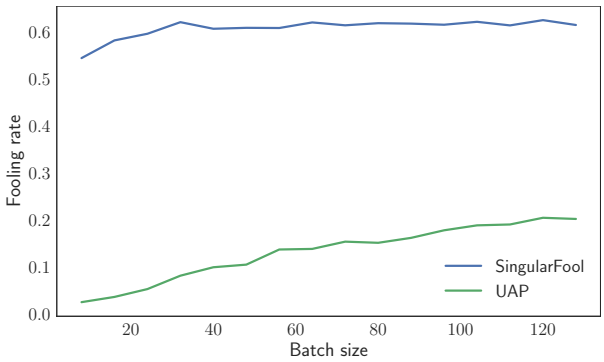


Figure 9: Dependence of the fooling rate on the number of images used for constructing the universal perturbation. SingularFool denotes the method proposed in the current paper, UAP denotes the algorithm presented in [14]. $q = 5$ and `block2_pool` layer of VGG-19 were used.

7. Related work

Many different methods [6, 14, 10, 21, 12] have been proposed to perform adversarial attacks on Deep Neural Networks in the *white box* setting where the DNN is fully available to the attacker. Two works are especially relevant for the present paper. Goodfellow *et al.* [6] propose the fast gradient sign method, which is based on computing the gradient of the loss function $l(x)$ at some image x and taking its sign as the adversarial perturbation. This approach allows one to construct rather efficient adversarial perturbations for individual images and can be seen as a particular case of our method. Indeed if we take the batch size to be equal to 1 and the loss function $l(x)$ as the hidden layer, then $\operatorname{sign} \operatorname{grad}_x l(x)$ is exactly the solution of the problem (5) with $p = 1$ and $L = 1$ (since $l(x)$ is just a number this problem does not depend on q). Second work is Moosavi *et al.* [14] where the universal adversarial perturbations have been proposed. It is based on a sequential solution of nonlinear optimization problems followed by a projection onto $p = \infty$ ($p = 2$) sphere, which iteratively computes the 'worst' possible direction towards the decision boundary. Optimization problems proposed in the current work are simpler in nature and well-studied, and due to their homogeneous property the adversarial perturbation with an arbitrary norm is obtained by simply rescaling the once computed perturbation, in contrast with the algorithm in [14].

8. Conclusion

In this work we explored a new algorithm for generating universal adversarial perturbations and analyzed their main properties, such as generalization across networks, dependence of the fooling rate on various hyperparameters and

having certain visual properties. We have showed that by using only 64 images a single perturbation fooling the network in roughly 60% cases can be constructed, while the previous known approach required several thousand of images to obtain such fooling rates. In a future work we plan to address the relation between feature visualization [15] and adversarial perturbations, as well as analyzing the defense approach discussed in section 6.2.

Acknowledgements

This study was supported by the Ministry of Education and Science of the Russian Federation (grant 14.756.31.0001), by RFBR grants 16-31-60095-mol-a-dk, 16-31-00372-mola and by Skoltech NGP program.

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org. 7

[2] A. Bhaskara and A. Vijayaraghavan. Approximating matrix p-norms. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms*, pages 497–511. SIAM, 2011. 2

[3] D. W. Boyd. The power method for lp norms. *Linear Algebra and its Applications*, 9:95–101, 1974. 2

[4] F. A. Gers, J. Schmidhuber, and F. Cummins. Learning to forget: Continual prediction with LSTM. 1999. 1

[5] I. Goodfellow, N. Papernot, and P. McDaniel. cleverhans v0. 1: an adversarial machine learning library. *arXiv preprint arXiv:1610.00768*, 2016. 1

[6] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014. 1, 8

[7] A. Graves, A.-r. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, pages 6645–6649. IEEE, 2013. 1

[8] K. Grosse, N. Papernot, P. Manoharan, M. Backes, and P. McDaniel. Adversarial perturbations against deep neural networks for malware classification. *arXiv preprint arXiv:1606.04435*, 2016. 1

[9] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 1, 4

[10] A. Kurakin, I. Goodfellow, and S. Bengio. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533*, 2016. 1, 8

[11] Y. LeCun, Y. Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995. 1

[12] Y. Liu, X. Chen, C. Liu, and D. Song. Delving into transferable adversarial examples and black-box attacks. *arXiv preprint arXiv:1611.02770*, 2016. 8

[13] T. Mikolov, S. Kombrink, L. Burget, J. Černockỳ, and S. Khudanpur. Extensions of recurrent neural network language model. In *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, pages 5528–5531. IEEE, 2011. 1

[14] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard. Universal adversarial perturbations. *arXiv preprint arXiv:1610.08401*, 2016. 1, 4, 5, 8

[15] C. Olah, A. Mordvintsev, and L. Schubert. Feature visualization. *Distill*, 2017. https://distill.pub/2017/feature-visualization. 4, 9

[16] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *Security and Privacy (SP), 2016 IEEE Symposium on*, pages 582–597. IEEE, 2016. 1

[17] B. A. Pearlmutter. Fast exact multiplication by the hessian. *Neural computation*, 6(1):147–160, 1994. 3

[18] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015. 4

[19] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 4

[20] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015. 1, 4

[21] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013. 1, 8

[22] L. N. Trefethen and D. Bau III. *Numerical linear algebra*, volume 50. Siam, 1997. 2