Combining Universal Adversarial Perturbations

Maurus Kühne[1][0000−0002−4205−3552] and Beat Tödtli[2][0000−0003−3674−2340]

[1] Fernfachhochschule Schweiz
maurus.kuehne@students.ffhs.ch
[2] Institut für Informations- und Prozessmanagement, FHS St. Gallen
beat.toedtli@ost.ch

Abstract Universal adversarial perturbations (UAPs) are small perturbations imposed on images that are able to fool a single convolutional neural network image classifier. They have been shown to generalise well to other neural networks. Here, we report on our reproduction effort of the results given in a work by Moosavi-Dezfooli et al. on UAPs and study two methods to construct UAPs for several neural networks. While the results are not strong enough to make general conclusions, they suggest that UAPs indeed profit from being constructed on several neural networks. Also, we show that a linear interpolation between two UAPs does not produce a viable UAP on both networks.

Keywords: Adversarial Training· Universal Adversarial Perturbation

1 Introduction

The discovery of Szegedy et al. [8] that several machine learning models including deep neural networks are vulnerable to *adversarial attacks* was seminal for a new subfield of studying deep learning. Probably the most intriguing, but also unsettling result was that adversarial examples can be made quite imperceptible to the human eye, while still fooling a convolutional neural network to misclassify the image [2]. Subsequent work has developed various algorithms in a variety of white-box, gray-box and black-box attack scenarios as well as defensive strategies such as adversarial training [7]. Moosavi-Dezfooli et al. have demonstrated that *universal* perturbations exist, i.e. that a single set of of pixel modifications can be found that fools a network on a large fraction of the training data set [4]. Moreover, universal adversarial perturbations (UAPs) also fool other convolutional networks, albeit to a lesser but still very significant degree [4].

These results suggest that neural networks partly share a common structure that can be exploited by universal adversarial perturbations (UAPs), while yet other aspects are different. Much remains to be understood about the shape of the decision surfaces in neural networks and how they depend on the architecture of the neural network. In this context, we ask how the transferability of UAPs to new convolutional neural networks can be improved by combining two neural networks in the adversarial perturbation generation algorithm. By combining two neural networks one might expect to see an increase in the transferability of an adversarial perturbation to other networks. Ultimately, we hope this approach will lead to identifying a general property resulting in convolutional neural networks being susceptible to adversarial attacks.

Given the practical potential and relevance of Deep Learning and the potential security threats of adversarial perturbations, finding a robust resolution is important and urgent. However, research into the topic is hampered by the *reproducibility crisis* in machine learning [6]. A lot of research results, including on adversarial attacks, remain poorly documented in publications. While the methods described may be general enough to be useful for many similar applications, the published results are often difficult to reproduce due to undocumented values for hyperparameters, software library versions etc. [3]. Also, only in rare cases is it clear how well the published results generalise beyond the very specific data set that was used but not provided to produce the published results. While not being able to reproduce the generation of adversarial examples may seem beneficial at first sight, it is clear that a thorough resolution of such security issues requires a well-stated problem, including readily available state-of-the-art algorithms for the production of adversarial examples. They form the basis of being able to test deep learning systems against such attacks.

The authors of [4,5] show good generalization results for UAPs generated with their procedure *DeepFool* [5] across different deep learning architectures. In this paper, we report on our effort to reproduce their results, provide our code and investigate whether modifications of their UAP algorithm are able to improve the generalization capability of UAPs by using modifications of the UAP generation algorithm that take into account several networks at the same time. Specifically, we ask whether incorporating information from a second neural network architecture improves the fooling rate of UAPs on a third neural network. We investigate three combination procedures and compare them with the original adversarial attack procedure.

This paper is organized as follows. In Sec. 2 we review the basic methods to generate adversarial and universal adversarial perturbations as introduced by Moozavi-Dezfooli et al. [4]. We then present three modifications of their UAP generation algorithm to combine UAPs for several networks. In Sec. 3 we first describe our reproduction effort of the original work of Moozavi-Dezfooli et al. [4] to produce UAPs on a set of networks and provide our fooling rates. We then present our results on the three methods to produce UAPs for several networks and show that the transferability to a third network is improved. In Sec. 4 we discuss these results and conclude with Sec. 5.

2 Methods

2.1 General Setting

Given a classifier $\hat{k}(x) = \operatorname{sgn}(f(x))$ that is based on the sign of a classification function $f : \mathbf{R}^n \to \mathbf{R}$, adversarial attacks seek a perturbation $\mathbf{v}$ such that

$$\hat{k}(\mathbf{x}+\mathbf{v}) = \operatorname{sgn}(f(\mathbf{x}+\mathbf{v})) \neq \operatorname{sgn}(f(\mathbf{x})) = \hat{k}(\mathbf{x}).$$

In the following we briefly describe the adversarial perturbation generating method DeepFool and its generalizations to UAPs in the multi-class classification setting. We then describe the two approaches analysed here to build UAPs from several networks.

2.2 DeepFool

In the UAP generation procedure DeepFool [5], the perturbation $\mathbf{v}$ for an image $\mathbf{x}$ is defined to be the shortest vector (using the L_p-norm $\|.\|_p$) such that $\mathbf{x}+\mathbf{v}$ lies on a decision boundary. If $f(\mathbf{x})$ is a linear function $f(\mathbf{x}) = \mathbf{w}\mathbf{x} + \mathbf{b}$, then $\mathbf{v}$ can be shown to be $\mathbf{v} = -\frac{f(\mathbf{x})}{\|\mathbf{w}\|_p^2}\mathbf{w}$. As f is nonlinear in general, the Taylor approximation of the function f around $\mathbf{x}$, $f(\mathbf{x}+\mathbf{v}) = f(\mathbf{x}) + (\nabla f(\mathbf{x}))^T\,\mathbf{v}$ is used to iteratively reach the decision boundary. In the multi-class setting considered here, an additional step is required that identifies the closest decision boundary.

2.3 Multi-Classifier-Universal adversarial Perturbations with DeepFool

Perturbations for each image in a dataset X (such as those generated using DeepFool) can be combined to form universal adversarial perturbations for a single network [4]. The procedure is given in Alg. 1. Essentially, DeepFool perturbations of images that are not yet misclassified are added to obtain a universal perturbation. Whenever the norm of the perturbation becomes large, a rescaling is applied. The perturbation is scaled back to satisfy a norm bound given by $\|\mathbf{v}\|_p \leq \xi$ (that potentially undoes the successful perturbation of some images). For a small value of ξ, this ensures that the perturbation remains largely invisible. For details, see [4].

Intriguingly, although the directions to the class boundaries vary for different training images, the resulting average over all image perturbations works well according to [4], even for other convolutional neural networks whose class boundaries might be expected to look rather different.

In the following subsections, we detail two approaches at generating UAPs for several classifiers.

Alternated generation of perturbations Since UAPs are constructed by adding up the perturbations generated using DeepFool, there is a natural way to combine perturbations generated by the two networks: We add up the contributions from all networks. We note that adding up the perturbations is not a commutative operation, since projections take place once the size (norm) of the perturbation becomes too large. The precise procedure used here is given in Alg. 1.

Variants of Alg. 1 exist that sample the images differently. One might for example generate perturbation by alternating the classifier for each image. We show this variant in Alg. 2. We evaluate both variants and compare their performance in Tab. 2.

Interpolation between UAPs on individual networks A simple yet instructive alternative to the above rather involved perturbation construction is given by a simple weighted average of the perturbation vectors generated on the individual networks. If we restrict our attention to the combination of two neural networks for now, then the weighted averages of the perturbations lie on a line in the high-dimensional vector space of images. In the following, we specifically investigate whether any perturbation lying on this line improves on the fooling capability of the two endpoints with respect to a third network. More formally, given UAPs $\mathbf{v}_f$ and $\mathbf{v}_g$ of two neural networks f and g, we consider

$$\mathbf{v}_{fg}\,(\lambda) = \lambda\,\mathbf{v}_f + (1-\lambda)\,\mathbf{v}_g \tag{1}$$

for values $\lambda \in [0,1]$. We seek the value of λ that maximizes the average fooling rate over f, g and a third network, given the current training data set.

Outlook to other approaches We have also investigated other, more involved approaches that led to unsatisfactory results. In particular, we were interested in constructing a multi-class DeepFool procedure that uses the gradients of two networks towards the next class boundary to compute a perturbation of a single image. One might try to find perturbations towards class boundaries that are aligned as much as possible, but where the class boundaries correspond to different classes in different networks. As of now, these attempts have not provided efficient UAPs for multiple networks.

```
**Input**:Data set $X$, set of classifiers $K$, desired norm $\|.\|_p$ of the perturbation $\xi$
**Output**: Universal perturbation vector $\mathbf{v}$
Initialise $\mathbf{v} \leftarrow 0$
**while** Average fooling rate is too low **do**
    **foreach** image $\mathbf{x} \in X$ **do**
        **foreach** $\hat{k} \in K$ **do**
            **if** $\hat{k}(\mathbf{x}) = \hat{k}(\mathbf{x}+\mathbf{v})$ **then**
                $\Delta\mathbf{v} \leftarrow$ DeepFool$(\mathbf{x}+\mathbf{v}, \hat{k})$
                $\mathbf{v} \leftarrow \mathbf{v} + \Delta\mathbf{v}$
                $\mathbf{v} \leftarrow \sqrt{\xi}\mathbf{v}/\|\mathbf{v}\|_p$
            **end**
        **end**
    **end**
    Shuffle $X$
**end**
**return** $\mathbf{v}$
```

Algorithm 1: Computation of universal adversarial perturbations for multiple neural networks. For each image, perturbation updates are computed for all classifiers.

```
Input:Data set $X$, ordered set of classifiers $K$, desired norm $\|.\|_p$ of the perturbation $\xi$
Output: Universal perturbation vector $\mathbf{v}$
Initialise $\mathbf{v} \leftarrow 0$
while Average fooling rate is too low do
    foreach image $\mathbf{x} \in X$ do
        $\hat{k} \leftarrow$ first element of $K$
        $K \leftarrow$ cyclic rotation of $K$
        if $\hat{k}(\mathbf{x}) = \hat{k}(\mathbf{x} + \mathbf{v})$ then
            $\Delta\mathbf{v} \leftarrow \mathrm{DeepFool}(\mathbf{x} + \mathbf{v}, \hat{k})$
            $\mathbf{v} \leftarrow \mathbf{v} + \Delta\mathbf{v}$
            $\mathbf{v} \leftarrow \sqrt{\xi}\mathbf{v} / \|\mathbf{v}\|_p$
        end
    end
    Shuffle $X$
end
return $\mathbf{v}$
```

Algorithm 2: Computation of universal adversarial perturbations for multiple neural networks. For each image, perturbation updates are computed for the next classifier in the classifier sequence. Then the sequence is cyclically rotated.

3 Results

In this section we first discuss our attempt at the reproduction of the results in [4] regarding the transferability of UAPs across neural network architectures. We then provide the results of our attempts at constructing UAPs based on two neural network architectures at the same time.

3.1 Reproduction of the Original Results on Universal Adversarial Perturbations

Dezfooli et al. tested the DeepFool and Universal Adversarial Perturbations algorithms on 5 different neural networks [4]. Wo choose the same networks as the ones used in [4]. We used publicly available pretrained weights for all models, since the weights used in [4] were not specified.We compare the achieved fooling rates with those given in the original paper. Separate experiments have been performed by training individual UAPs on each of the networks, and subsequently measuring the fooling rates on all available networks. All perturbations were generated using the same random subset of 10'000 images of the ImageNet training set [1], and the fooling rates were measured on the ImageNet validation set (containing 50'000 images).

Figure 1 shows the perturbations generated. In their general structure and appearance, they are similar to the ones reported in [4]. In particular, the fine line-shaped structures in green and magenta are quite recognizable and are present

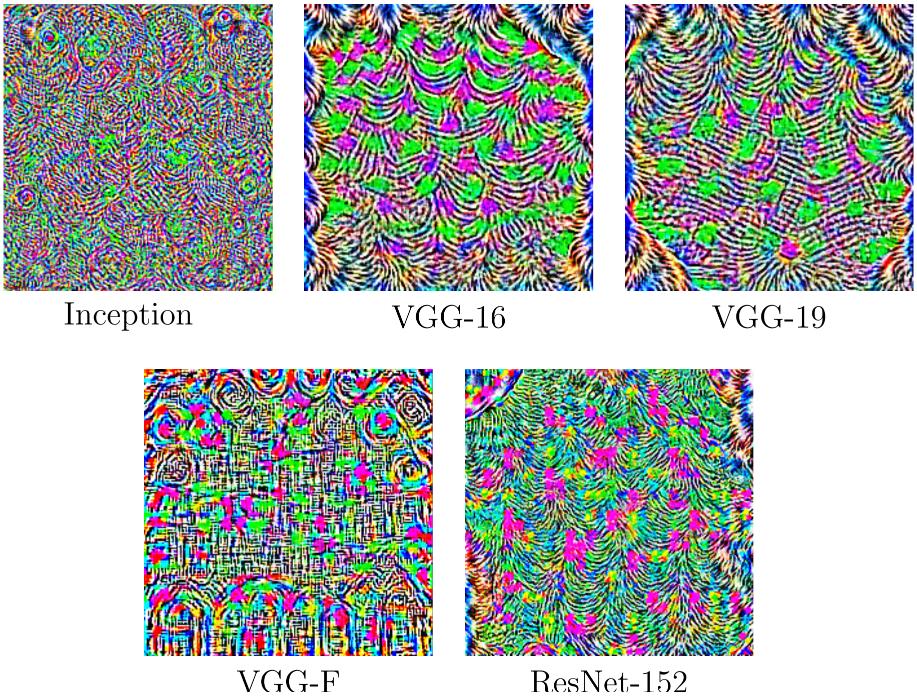

Figure 1. Visualisation of the generated perturbations. To visualise the perturbations we shifted them by $+\xi$ and extended them to the entire colour space with a scalar multiplication.

also in the original results in [4]. We therefore believe that this indicates that deviations from the original results reported below probably stem from configuration details rather than a fundamental reproduction mistake.

Tab. 1 shows the achieved fooling rates for the tested models and should be compared with Tab. 2 of Moosavi-Dezfooli et al. [4]. The first column indicates the network used to generate the UAP, while the first row gives the network on which the fooling rate is measured. The main diagonal therefore contains the self-fooling rates, i.e. the fooling rates achieved using the same model for generating the perturbation and measuring the fooling rate.

To reproduce these numbers, several insufficiently documented design choices had to be researched. For example, the original results are not stated for a given epoch, but using a stopping condition on the error rate. Values reported here are for epoch 20, at which point the mean value over the last 5 epochs typically varies by less than 0.005. Parameter values taken from and in the notation of [5] are $p = \infty$ and $\xi = 10$. We used a maximum of 10 DeepFool update iterations. The overshoot parameter $\eta = 0.02$ was again taken from [5]. An important parameter

Table 1. Fooling rates for the ImageNet validation set.

	VGG-F	Inception	VGG-16	VGG-19	ResNet-152
VGG-F	90%	56%	32%	32%	24%
Inception	42%	82%	16%	16%	16%
VGG-16	46%	60%	59%	52%	38%
VGG-19	45%	58%	51%	54%	33%
ResNet-152	42%	55%	31%	33%	73%

is the number of class boundaries, called num_classes[3]. It gives the number of class boundaries in whose direction a perturbation is searched for. Computation time is highly sensitive to this parameter, and checking all 1000 training boundaries of ImageNet was infeasible. Higher values generally improve the fooling performance, though exceptions are observed. Other minor parameters such as random initialisation values for the train-test split, etc. were taken from their provided code. We have tried to optimize these parameters using grid searches, but the available computing resources have limited these efforts.

3.2 Evaluation of Attempts to Construct Universal Adversarial Perturbations for Multiple Neural Networks

In this section we present our results for constructing UAPs that make use of two neural networks. We discuss results for the two approaches presented in Sec. 2.3, the alternated generation of perturbations and the linear interpolation between UAP on individual networks.

The alternated generation of perturbations as given in Alg. 1 has been applied to the set of classifiers $K = \{\text{Inception}, \text{VGG-16}\}$. In Tab. 2 the fooling rates are reported for these two networks and for ResNet-152. The table has two sections. In the upper section, the network listed in the first column is used to generate the perturbation. The columns give the fooling rates on the network given in the column title as well as their average as our measure for the generalizability of the UAPs. We include the original network in this average. The second section reports results by applying the alternated generation of UAPs and the interpolation method given in Eq. (1) using the value $\lambda = 0.05$.

Alg. 1 achieved results similar to the ones of a perturbation generated on VGG-16 only. Alg. 2 achieved slightly higher fooling rates on VGG-16 and ResNet-152 than a perturbation generated directly on VGG-16 (with an absolute increase of 2 and 3 percentage points, respectively). For Inception, the perturbation achieved a fooling rate of 67%. This is 15% below the measured self-fooling rate of Inception (82%), but 7% higher than the fooling rate achieved with a VGG-16 model.

For linear interpolation between the UAPs of VGG-16 and Inception V1, best results were achieved for $\lambda = 0.05$ (see Eq. (1)). Using this configuration, the fooling rates remained essentially unchanged compared to the UAP generated on

[3] As given in the code at https://github.com/LTS4/

VGG-16 only. The small value $\lambda = 0.05$ results in a perturbation that is similar to the VGG-16 perturbation as the VGG-16 perturbation is weighted with $1 - \lambda = 0.95$, while the Inception perturbation contributes only with a weight of 5%. Choosing $\lambda \in [0.1, 0.15, \ldots, 0.4]$ resulted in perturbations with lower fooling rates for both models with respect to the rates achieved by separately training UAPs on the two networks. For $\lambda \in [0.4, 1.0]$, the fooling rates for Inception improved again, but did not exceed the fooling rates of a UAP generated on Inception itself. The fooling rates on VGG-16 continued to deteriorate, stabilizing at a low fooling rate of $\sim 15\%$ after $\lambda \geq 0.65$.

	Inception V1	VGG-16	ResNet-152	Average
Inception V1 UAP	82%	16%	16%	38%
VGG-16 UAP	60%	59%	38%	52%
ResNet-152 UAP	55%	31%	73%	53%
linear interpolation with $\lambda = 0.05$	60%	59%	38%	52%
alternated generation of UAPs, Alg. 1	63%	55%	36%	51%
alternated generation of UAPs, Alg. 2	**67**%	**61**%	**41**%	**56**%

Table 2. Comparison of the fooling rates by UAPs trained on individual networks (upper three rows) and of combination methods for several networks (lower three rows). Best values are shown in bold. Among the alternated generation variants (Alg. 1 and Alg. 2) and the linear interpolation procedure (Eq. (1)), Alg. 2 performs best with respect to the average fooling rate over all three networks. Results are reported on the validation set, using the l_∞-norm, $\xi = 10$ and 20 UAP iterations.

4 Discussion

4.1 Reproduction of the Original Results on Universal Adversarial Perturbations

For most models the fooling rates reported in the original paper could not be achieved. For VGG-F, VGG-16, VGG-19 and ResNet-152 our self-fooling rates were between 4 and 24 absolute percentage points lower. For Inception we achieved a self-fooling rate 3 absolute percentage points higher than the one reported in the original paper. Clearly, this is not satisfactory and further research is needed to state the precise conditions under which a reliable reproduction of the reported fooling rates is possible. As a step in this direction we provide our code.[4]

The non-diagonal values in Tab. 1 are large but typically significantly smaller than the diagonal values. They show a degree of transferability of UAPs generated with DeepFool to other models. Therefore, despite the reproducibility problems, these results broadly confirm that UAPs generated with DeepFool

[4] The code can be found here: https://github.com/mauruskuehne/lwda-paper

generalise to other network architectures. Nevertheless, it is clear that some aspects of UAPs are specific to a given neural network architecture. We discuss our results on finding a way to improve the non-diagonal elements (potentially at the cost of the diagonal ones) in the next section. Interestingly, we achieved lower fooling rates than Moozavi-Dezfooli et al., except for the Inception network, for which we achieve 3 to 8 absolute percentage points higher fooling rates. This may indicate that the Inception network is more susceptible to these perturbations. Another possibility are the chosen hyperparameter values for DeepFool and UAP, which may be particularly well suited or optimized for this model. This in turn would explain the lower fooling rates achieved on other models.

4.2 Alternated generation of perturbations and linear Interpolation between UAPs on individual networks

As the results in Sec. 3.2 clearly show, a linear interpolation between two UAPs does not give good results. This suggests that a weighted average of the UAPs on individual networks does not produce good UAPs for both neural networks. Since neural networks are highly non-linear functions, there is no reason to assume that combining their UAPs as a weighted average would produce good perturbations on both networks. This is confirmed here and therefore this attempt can at best serve as a baseline for comparison.

The results given by the alternated generation of perturbations (Alg. 2) are much better (see Tab. 2). The fooling rate of the perturbation generated jointly on Inception and VGG-16 is better than the ones generated on any one of the two networks. A perturbation generated on Inception achieves a fooling rate of 16% on VGG-16, while a perturbation generated on VGG-16 achieves a fooling rate of 60% on Inception. Both rates are lower than the ones of a perturbation generated jointly on Inception and VGG-16, achieving 67% on Inception and 61% on VGG-16. Furthermore, the fooling rate of a jointly trained UAP on Inception and VGG-16 on a third network (Resnet-152) is better than the fooling rate of both single-network UAPs. The UAP generated jointly on both networks even worked slightly better for VGG-16 than the one trained on VGG-16 alone. The reason for this effect is not yet established. It may even be a statistical fluctuation as the effect of the particular ImageNet train-test-splitting has not been investigated due to constraints on computational resources.

5 Conclusions

The results reported here on generalizing UAPs across several networks clearly have to be interpreted cautiously given the fact that even the reproduction of previously reported results has not been satisfactory. Establishing reproducibility standards for machine learning publications remains a crucial challenge that is hampering progress.

With the above caution in mind, the results reported here suggest that finding universal adversarial perturbations that generalise across different convolutional neural networks is not a hopeless endeavour. As we found, such a UAP is likely not a linear combination of UAPs of different networks, but must be constructed in a more subtle way. Our best approach, Alg. 2, most certainly is not optimal. Nevertheless it already shows some promising results: The generalizability of the fooling rates to ResNet-152 is enhanced by combining the UAPs of two networks, with respect to the UAPs generated on either one of the Inception or VGG-16 network. This suggests that combining several or even many networks might produce UAPs that are efficient on a whole class of trained convolutional neural networks.

References

1. Deng, J., Dong, W., Socher, R., Li, L., Li, K., Li, F.: ImageNet: A large-scale hierarchical image database. In: CVPR09. pp. 248–255. IEEE Computer Society (2009). https://doi.org/10.1109/CVPR.2009.5206848
2. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. In: Bengio, Y., LeCun, Y. (eds.) 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings (2015), http://arxiv.org/abs/1412.6572
3. Gundersen, O.E., Kjensmo, S.: State of the art: Reproducibility in artificial intelligence. In: AAAI Publications (2018)
4. Moosavi-Dezfooli, S.M., Fawzi, A., Fawzi, O., Frossard, P.: Universal Adversarial Perturbations. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 86–94 (Juli 2017). https://doi.org/10.1109/CVPR.2017.17, iSSN: 1063-6919
5. Moosavi-Dezfooli, S.M., Fawzi, A., Frossard, P.: DeepFool: A Simple and Accurate Method to Fool Deep Neural Networks. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 2574–2582 (Juni 2016). https://doi.org/10.1109/CVPR.2016.282, iSSN: 1063-6919
6. Raff, E.: Quantifying independently reproducible machine learning. The Gradient (2020)
7. Ren, K., Zheng, T., Qin, Z., Liu, X.: Adversarial attacks and defenses in deep learning. Engineering **6**(3), 346 – 360 (2020). https://doi.org/10.1016/j.eng.2019.12.012
8. Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., Fergus, R.: Intriguing properties of neural networks. In: International Conference on Learning Representations. ICLR (2014), URL http://arxiv.org/abs/1312.6199