Improved DeepFool: Efficient Adversarial Attacks via Optimisation and Refinement

Łukasz Mikołajczyk
AGH University of Krakow
Centre of Security Technologies
Krakow, Poland
Czestochowa University of Technology
Faculty of Computer Science and Artificial Intelligence
Czestochowa, Poland *lukasz.mikolajczyk@agh.edu.pl*

Piotr Duda
Robert Nowicki
Rafał Scherer
Czestochowa University of Technology
Faculty of Computer Science and Artificial Intelligence
Czestochowa, Poland
AGH University of Krakow
Faculty of Computer Science
and Center of Excellence in Artificial Intelligence
Krakow, Poland *{piotr.duda, robert.nowicki, rafal.scherer}@pcz.pl*

Abstract

This study addresses the vulnerability of AI systems to adversarial attacks by extending the DeepFool algorithm. The paper proposes four new approaches and evaluates them according to a set of criteria. The methods are inspired by various optimisation algorithms. One of the proposed improvements adds the independent refinement stage, which reduces the final perturbation without extra gradient computations. Experimental results show that an appropriately modified algorithm reaches the decision boundary in fewer steps and with fewer gradient evaluations, while the refinement stage further decreases the magnitude of the perturbation. The combined approach can improve attack efficiency and reduce detectability, suggesting the potential for a wider application of advanced optimisation techniques in adversarial example generation.

Keywords: Adversarial attack, DeepFool, Convolutional neural networks.

1. Introduction

Despite their remarkable success in various domains, neural networks remain susceptible to adversarial attacks, which are carefully crafted perturbations of input data designed to deceive models into making erroneous predictions. These attacks exploit the high-dimensional decision boundaries learned by deep learning systems, where even imperceptible modifications can lead to significant misclassifications. The existence of such vulnerabilities raises concerns for real-world applications, particularly in security-sensitive areas such as autonomous driving, medical diagnosis and biometric authentication.

Adversarial attacks are typically categorised by the attacker's knowledge (white box vs black box) and their objectives (targeted vs non-targeted). White-box attacks, such as the Fast Gradient Sign Method (FGSM) [4] and Projected Gradient Descent (PGD) [17], use full-model access to compute gradients and optimise perturbations. In contrast, black-box attacks rely on transferability or query-based strategies to infer model behaviour. Among these, the DeepFool

algorithm [13] stands out as a white-box method that iteratively pushes inputs toward the nearest decision boundary with minimal perturbation, revealing the intrinsic fragility of the model. The primary objective of this family of algorithms is to identify the smallest possible perturbation of the input data that leads to a change in the output of the classifier. In the original formulation, the authors employed an iterative method based on following the direction of the steepest descent from the input data to the decision boundary. Each update is computed using the gradient of the classifier output relative to the input. This approach can be viewed as an analogy to gradient-based learning in neural networks, where the "layer" being trained is the input data perturbation itself.

The arms race between adversarial attacks and defences underscores the need for robust models and a deeper understanding of attack mechanisms. Although adversarial training and input preprocessing offer partial mitigation, improving the efficiency and concealment of attacks (e.g. reducing perturbation magnitude or computational cost) remains an active research area.

This work extends DeepFool by integrating distinct optimisation and adaptive step size techniques to accelerate convergence while maintaining low perceptibility. In other words, we propose a modification of the decision boundary search steps by augmenting the basic gradient-based approach with scaling or second-order methods, specifically, techniques inspired by momentum-based learning and the Adam optimisation algorithm. In addition, an auxiliary perturbation refinement stage is introduced; this stage is independent of the main algorithm variant used and aims to reduce the magnitude of the perturbation, thereby decreasing the likelihood of detection. Our experiments demonstrate that these modifications are able to improve attack efficiency, reduce detectability, and offer insights into the interplay between optimisation methods and adversarial robustness.

The performance of the proposed modifications to the DeepFool algorithm, including the auxiliary refinement stage, was thoroughly evaluated and compared against the original algorithm. The comparison involved metrics such as the number of steps, the magnitude of the perturbation and the runtime. The effect of the added adversarial noise has been visualised using the Grad-CAM [3] technique; see Figure 1 and 2. Grad-CAM (Gradient-weighted Class Activation Mapping) is a visualisation technique that highlights the regions of an input image most influential for a neural network's prediction. It uses gradients flowing into the final convolutional layer to produce a heat map showing where the model focuses to make its decision. This helps interpret model behaviour, debug misclassifications and validate if its attention aligns with human intuition.

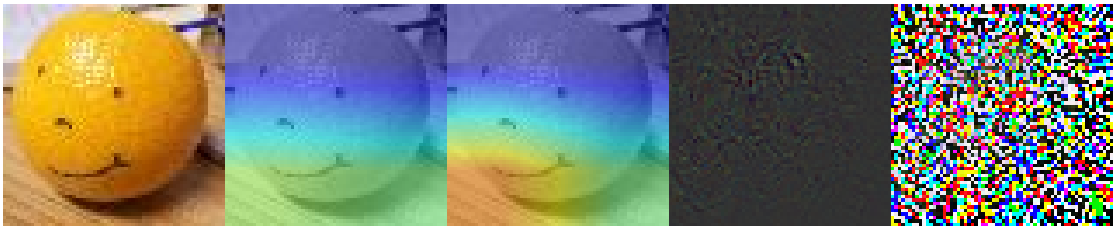

Fig. 1. Grad-CAM visualisation of a selected image from Tiny ImageNet dataset. ResNet-50. From the left: original image, heat map of original image, heat map of perturbed image, perturbation shifted above zero and $100\times$ magnified, normalised perturbation.

The rest of the paper is organised as follows. The survey on recent adaptations of attacks based on the DeepFool algorithm is presented in Section 2. A detailed description of the DeepFool algorithm is presented in Section 3. The aforementioned alterations are presented in Section 4. Section 5 contains a description of the simulations conducted and a comparison of the results obtained for various evaluation criteria. The summary and final conclusions are included in Section 6.

2. Related works

In 2016, the DeepFool algorithm [13] was proposed as the foundation for a whole line of approaches set to deceive neural networks by introducing appropriate noise that causes the classifier to change its decision regarding the most probable class.

In a study first published in 2023 [8], a strategy was considered, in which the algorithm aims not only to fool the classifier but also to ensure that the input is assigned to a user-specified target class. In addition, a condition was introduced that enforces a minimum confidence level of the classifier for the specified class.

In [1], a mathematical analysis of the considered attacks was presented, focusing on the ℓ_2 robustness metric. Based on the evaluation, a family of algorithms was proposed that improves the trade-off between computational cost and attack effectiveness.

The SparseFool algorithm is a geometry-inspired attack that takes advantage of the low mean curvature of the decision boundaries to quickly and efficiently generate sparse perturbations [12]. It outperforms the other methods compared in terms of speed, scales well with high-dimensional data and enables control over the perceptibility of perturbations without sacrificing their effectiveness.

The YOAO algorithm, proposed in [10], enhances DeepFool by analysing the gradients of all classes with respect to the true class to generate adversarial examples in just a single iteration. This results in a 68% increase in the attack success rate compared to the original version. Moreover, YOAO features lower time complexity and greater efficiency in misleading various neural networks than the existing methods.

3. DeepFool Algorithm for multi-class problem

This section presents the operating principles of DeepFool, outlined in Algorithm 1, designed for multinomial classification problems, as proposed in the original article [13] and served as a foundation for many types of attack described in Chapter 2.

Let $f(x)$ denote the output of the classifier for a data element x. The predicted class is then determined as the label corresponding to the maximum output of the classifier, i.e. $\hat{k}(x) = sign(f(x))$. The goal of the algorithm is to find a perturbation that, when added to the original image, changes the prediction of the network. To achieve this, the following steps are repeated until the desired misclassification is reached. For all labels k different from the correct one:

- The algorithm computes the difference, denoted w'_k, between the gradient of the classifier output for class k and the gradient for the class originally predicted k_0.
- The difference in activation levels between class k and k_0 is calculated, denoted as f'_k.

The algorithm selects the class $\hat{l}$ that minimises the ratio of the absolute value of f'_k and the squared ℓ_2 norm of the difference of gradients w'_k. Then a new perturbation is computed by scaling the gradient difference w'_k using this ratio. This perturbation r_i is added to the most recently modified image and the stopping criterion is checked again. Once the stopping condition is satisfied, the algorithm returns the final adversarial perturbation $\hat{r}_i$. It should be remarked that, in the Official Implementation [11], $|f'_{\hat{l}}|$ of r_i is biased with a small value of $1\mathrm{e}{-4}$ for computational stability, possibly providing too large a step in some scenarios, while the number of k labels in a search space is parametrically limited.

4. Proposed Modifications

We present several methods inspired by the DeepFool algorithm, as described in the previous section. Each variation seeks to improve the overall perturbation scale and computation time, preferably at once.

Algorithm 1 DeepFool: multi-class case

input:
Image x
Classifier f
output:
Perturbation $\hat{r}$

1: $x_0 \leftarrow x, i \leftarrow 0$
2: **while** $\hat{k}(x_i) = \hat{k}(x_0)$ **do**
3: **for** $k \neq \hat{k}(x_0)$ **do**
4: $w'_k \leftarrow \nabla f_k(x_i) - \nabla f_{\hat{k}(x_0)}(x_i)$
5: $f'_k \leftarrow f_k(x_i) - f_{\hat{k}(x_0)}(x_i)$
6: **end for**
7: $\hat{l} \leftarrow argmin_{k \neq \hat{k}(x_0)} \frac{|f'_k|}{||w'_k||_2}$
8: $r_i \leftarrow \frac{|f'_{\hat{l}}|}{||w'_{\hat{l}}||_2^2} w'_{\hat{l}}$
9: $x_{i+1} \leftarrow x_i + r_i$
10: $i \leftarrow i + 1$
11: **end while**
12: **return** $\hat{r} = \sum_i r_i$

The first modification is derived from the widely used gradient descent technique in model training. This method seeks to accelerate convergence by adjusting the step size, moving either further or closer, depending on whether the loss function's gradient changes sign. Similarly, in DeepFool, we move in the direction indicated by the gradient of the loss function, aiming only to cross the decision boundary. Therefore, this alteration of DeepFool can be formulated as an enhancement of the perturbation step, as shown in Equation (1).

$$r_i \leftarrow \frac{|f'_{\hat{l}}|}{||w'_{\hat{l}}||_2^2} w'_{\hat{l}} + \beta r_{i-1} \tag{1}$$

In recent years, *Momentum* has been largely replaced by optimisation techniques that offer faster convergence. In particular, we were inspired by the widely adopted Adam optimiser [7] and proposed its application in the task of finding minimal adversarial perturbations, as demonstrated in Algorithm 2, named here *qAdam*. It incorporates moving averages of gradients and squared gradients to adaptively modify the size of the perturbation. Steps 1 to 7 remain the same as in Algorithm 1; the difference lies in how the new perturbation is computed.

In the third method, we introduce a scaling factor γ, adjusting the magnitude of DeepFool's ith step perturbation – in Equation (2), the calculated norm is weighed by making the scalar larger, thus the result of a given loop iteration will be smaller, possibly leading to increased effectiveness. The opposite scenario is also plausible. By further modifying the weight γ_i, as seen in Equations (3) and (4), where x is a scalar base and y is a step, achieving a "best of both worlds" might be possible: firstly, the weight is decreased, thus lengthening the step, while in latter iterations, the weight increases, linearly or exponentially diminishing the added perturbation. It must be noted, however, that the non-linear increment is not going to be computationally stable in highly complex scenarios, e.g. during operations on images having high pixel counts, possibly yielding a memory overflow, therefore the selection of hyperparameters should be made with greater care; values of $x \in [0.5, 0.8]$ and $y \in [1.5, 2]$ are preferred.

$$r_i \leftarrow \frac{|f'_{\hat{l}}|}{\gamma_i ||w'_{\hat{l}}||_2^2} w'_{\hat{l}} \tag{2}$$

Algorithm 2 The *qAdam* algorithm

input parameters:
Learning rate lr
Running coefficient (decay) for $\overline{\nabla}$ β_1
Running coefficient (decay) for $\overline{\nabla}^2$ β_2
Steps 1 to 6 as in Algorithm 1
...
7: $\hat{l} \leftarrow argmin_{k \neq \hat{k}(x_0)} \frac{|f'_k|}{||w'_k||_2}$
8: $m_i \leftarrow \beta_1 \times m_{i-1} + (1 - \beta_1) \times w'_{\hat{l}}$
9: $v_i \leftarrow \beta_2 \times v_{i-1} + (1 - \beta_2) \times w'^2_{\hat{l}}$
10: $\hat{m}_i \leftarrow m \div (1 - {\beta_1}^{i+1})$
11: $\hat{v}_i \leftarrow v \div (1 - {\beta_2}^{i+1})$
12: $\alpha \leftarrow \hat{m}_i \times \frac{lr}{\sqrt{\hat{v}_i + \epsilon}}$
13: $r_i \leftarrow \frac{|f'_{\hat{l}}|}{||w'_{\hat{l}}||_2^2} w'_{\hat{l}} + \alpha$
...

$$\gamma_i \leftarrow x + yi \tag{3}$$

$$\gamma_i \leftarrow x \times y^i \tag{4}$$

Finally, we propose a last section minimisation *postMin*. As can be seen in Algorithm 3, this method relies on the last perturbation fragment found that results in a sign change (where $k_i \neq k_0$). Simple directional operation of subtraction or addition of diminishing fragment of the last gradient difference may result in a smaller norm of the perturbation, thereby improving the base algorithm, or, in boundary cases, improve the *Fooling Rate* by possibly breaching the close decision hyperplane. Since the calculation takes place outside of the main loop, the routine is independent of other modifications and can be applied to alter them further.

5. Experimental results

Testing was carried out on a machine running Ubuntu 24.04.2 LTS, AMD Ryzen 5800x, 32GiB RAM, NVIDIA RTX 3070 with 8GiB of VRAM, using Driver 550.120. The machine learning algorithms and their corresponding weights were sourced from and powered by *PyTorch* [14]:

- ResNet-50 [5] – deep residual learning image classifier, depth of 50 layers, 25M params,
- VGG-19 [16] – deep convolutional neural network, depth of 19 layers, 144M params.

The assortment was dictated by the popularity of the respective families in image classification tasks, the listed representatives being relatively close in ImageNet classification (accuracy: ResNet-50 of 75.3% vs 72.38% for VGG-19, as can be seen in the previously cited papers), yet differing significantly in the number of training parameters – providing insight in the importance of model complexity in adversarial actions.

The datasets selected for the research are modified versions of ImageNet [2], which are easily accessible on *Hugging Face*:

- Tiny ImageNet [9] – resolution limited to 64×64 px RGB images with 200 different classes, split into "Train" and "Valid" subsets, containing 100000 and 10000 records, respectively,

Algorithm 3 Minimisation of the boundary-piercing perturbation *postMin*

input parameter:
Scale limit ϵ
Steps 3 to 8 as in Algorithm 1
$x_0 \leftarrow x, i \leftarrow 0$
while $\hat{k}(x_i) = \hat{k}(x_0)$ **do**
...
$x_{i+1} \leftarrow x_i + r_i$
$i \leftarrow i + 1$
end while
$\alpha \leftarrow 0.5$
$r_{last} \leftarrow r_i$
while $\alpha > \epsilon$ **do**
if $\hat{k}(x_i) \neq \hat{k}(x_0)$ **then**
$r_i \leftarrow -\alpha \times r_{last}$
else
$r_i \leftarrow \alpha \times r_{last}$
end if
$\alpha \leftarrow 0.5 \times \alpha$
$x_{i+1} \leftarrow x_i + r_i$
$i \leftarrow i + 1$
end while
return $\hat{r} = \sum_i r_i$

- Mini-ImageNet [15] – small subset of the original, with preserved resolution and using 100 classes, having 50000, 10000 and 5000 RGB images in "Train", "Validation" and "Test" splits.

The last and least numerous splits of the datasets were used. The resolution of Mini-ImageNet was limited to, at most, 1080000 px. DeepFool's class label search was limited to default top 10 and overshoot set to default 0.02. The tested method *postMin* is limited by a very small $\epsilon = 1\text{e}{-12}$, leading to an increase in the number of loop iterations equal to 39, as $\frac{1}{2}^{39} \approx 1.82\text{e}{-12}$ and will be lower for the stopping condition, as seen in Algorithm 3. Results for *qAdam* obtained with standard $\beta_1 = 0.9$ and $\beta_2 = 0.999$. For additional information on the data in tables, please refer to the footnotes. The result instability of $\pm 1\%$ should be assumed for method safety.

5.1. Prerequisites

The preprocessing of the data is a crucial step prior to the neural network training. However, the adversary cannot control the actions of a potential victim, who is very likely to apply random rotation, scaling and standardisation. For the purpose of the comparative study, a level playing field is necessary – as seen in Table 1, the process of standardisation (Z-score calculation) using ImageNet's values of $\mu = \{0.485, 0.456, 0.406\}$ and $\sigma = \{0.229, 0.224, 0.225\}$ yields a change in perturbation magnitude and diminishes effects of the modifications, without shifting their position compared to the relatives. It must be noted that applying some form of a standardisation, preferably image-based, might be crucial for the bad actor – with the aforementioned process in place, perturbations based on raw data will perish. Visual comparison is available in Figure 2. Since the study does not focus on their transferability, raw data provides a view of the best-case scenario, isolating the effects of the modification, similar to the article [6]. The presented values of perturbation robustness ρ were calculated using Equation (5) as a basis, which was proposed

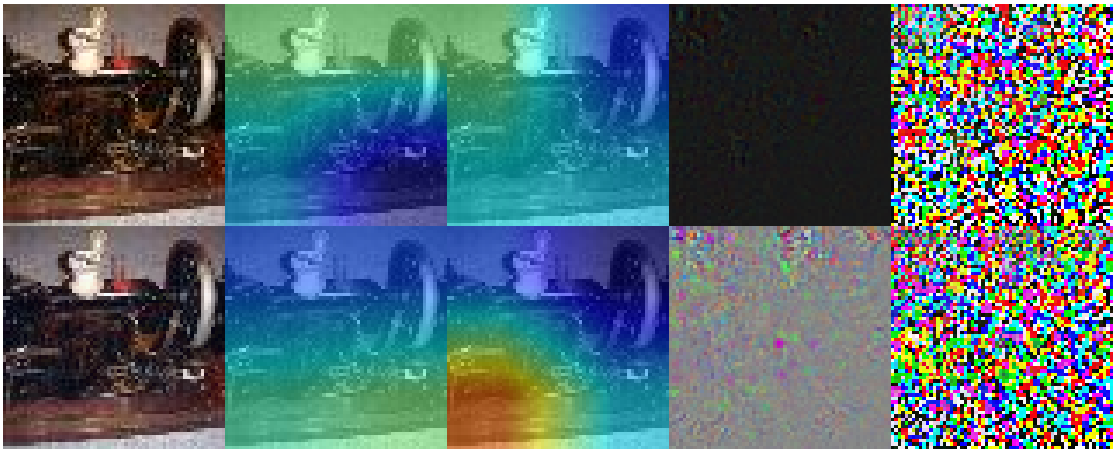

Fig. 2. Grad-CAM visualisation of a selected image from Tiny ImageNet dataset. ResNet-50. Top: raw data, bottom: standardised. From the left: original image, heat map of original image, heat map of perturbed image, perturbation shifted above zero and $100\times$ magnified, normalised perturbation.

in the original publication [13], where f is a classifier, $\bar{\rho}_{adv}(f)$ is an average perturbation robustness over the dataset $\mathscr{D}$. For simplification, this given designation is shortened to $\bar{\rho}$ in the tables. Similarly, $\widetilde{\rho}$ denotes the median value of ρ in the dataset, while σ_ρ denotes the standard deviation of the values.

$$\hat{\rho}_{adv}(f) = \frac{1}{\mathscr{D}} \sum_{x \in \mathscr{D}} \frac{||\hat{r}(x)||_2}{||x||_2} \tag{5}$$

5.2. Result Analysis

The proposed modifications were tested with a range of hyperparameters, creating a broad possible time or perturbation robustness spectra. It can be noted that *postMin* improves upon the base methods, yet resulting in a significant extension of the runtime, still proving its usefulness. The vastness of improvement depends on the tested network and, possibly, magnitude of the perturbation – higher orders thereof might mean the possible range of improvement (distance from hyperplane) is relatively very small, diminishing the return (possible explanation for results in Table 2). A good medium between the mentioned method and the original version is γ weighing, resulting, in most cases, in a not very significant increase in runtime, while leading to a smaller perturbation.

On the other end of the spectrum, methods with "sliding" γ value or *qAdam*, seem to speed up the search process – burdened with a vastly more perceptible perturbation. With certainty, those would present themselves "in a better light" if only the dataset contained large-resolution images – a foretaste of what the results for Mini ImageNet give (see Table 3). Furthermore, these necessitate additional hyperparameter tuning, possibly bringing the faster modifications

[1] All of the ρ values in scientific notation $\mathrm{e}{-6}$.
[2] Values of ρ in scientific notation $\mathrm{e}{-6}$, $\mathrm{e}{-5}$ for σ_ρ.
[3] Maximum of 50 iterations of the algorithm's inner loop used instead of 1000 in other cases.
[4] Values of $\bar{\rho}$ and σ_ρ in scientific notation $\mathrm{e}{-5}$, $\widetilde{\rho}$ in $\mathrm{e}{-6}$.
[5] Learning rate of $1\mathrm{e}{-5}$.
[6] As in 7, but with σ_ρ in $\mathrm{e}{-3}$.
[7] Values of ρ in scientific notation $\mathrm{e}{-4}$.
[8] Learning rate of $6\mathrm{e}{-5}$.

Table 1. Effect of data standardisation on selected methods: Tiny ImageNet & ResNet-50

	Variant	Raw data[1]					Standardised[2]				
		T[ms]	$\bar{i}$	$\bar{\rho}$	$\widetilde{\rho}$	σ_ρ	T[ms]	$\bar{i}$	$\bar{\rho}$	$\widetilde{\rho}$	σ_ρ
postMin	$\gamma_i = 1.3$	360	41.22	4.25	2.11	6.64	425	42.22	6.86	3.17	1.10
postMin	Original	339	40.87	4.62	2.27	7.14	376	41.56	7.33	3.38	1.18
postMin	$\beta = 0.3$	325	40.76	5.08	2.30	8.39	359	41.29	8.24	3.59	1.41
postMin	$\gamma_i = \frac{3}{4} + \frac{1}{8}i$	347	40.93	5.04	2.46	7.78	408	41.94	7.83	3.66	1.28
	$\gamma_i = 1.3$	151	2.22	5.22	2.99	6.52	209	3.19	7.44	3.74	1.10
	Original	129	1.87	6.01	3.63	7.02	172	2.58	8.22	4.27	1.19
	Original 50[3]	130	1.87	6.01	3.63	7.00	172	2.58	8.21	4.27	1.19
	$\beta = 0.3$	121	1.76	6.79	3.77	8.60	158	2.29	9.59	4.76	1.46
	$\gamma_i = \frac{3}{4} + \frac{1}{8}i$	136	1.94	7.05	4.53	7.57	192	2.93	9.12	4.92	1.28

Table 2. Modification effectiveness – ResNet-50

	Variant	Tiny ImageNet[1]					Mini ImageNet[4]				
		T[ms]	$\bar{i}$	$\bar{\rho}$	$\widetilde{\rho}$	σ_ρ	T[ms]	$\bar{i}$	$\bar{\rho}$	$\widetilde{\rho}$	σ_ρ
postMin	$\gamma_i = 1.3$	360	41.22	4.25	2.11	6.64	764	41.87	1.03	4.02	1.73
postMin	Original	339	40.87	4.62	2.27	7.14	597	40.85	1.10	4.16	1.91
postMin	$\beta = 0.1$	337	40.83	4.75	2.29	7.45	583	40.73	1.13	4.17	2.01
postMin	$\beta = 0.3$	325	40.76	5.08	2.30	8.39	571	40.66	1.20	4.22	2.19
postMin	$\gamma_i = \frac{3}{4} + \frac{1}{8}i$	347	40.93	5.04	2.46	7.78	537	40.45	1.19	4.25	2.10
postMin	$\gamma_i = \frac{1}{2} + \frac{1}{8}i$	333	40.57	6.21	2.79	10.0	496	40.18	1.38	4.32	2.63
postMin	qAdam[5]	308	40.35	12.0	4.65	16.4	505	40.14	1.43	6.25	2.23
	$\gamma_i = 1.3$	151	2.22	5.22	2.99	6.52	476	2.87	1.04	4.14	1.75
	Original 50[3]	130	1.87	6.01	3.63	7.00	319	1.85	1.12	4.39	1.91
	Original	129	1.87	6.01	3.63	7.02	314	1.86	1.12	4.41	1.89
	$\beta = 0.1$	130	1.84	6.22	3.68	7.39	295	1.73	1.19	4.60	2.04
	$\beta = 0.3$	121	1.76	6.79	3.77	8.60	284	1.66	1.36	5.03	2.34
	$\gamma_i = \frac{3}{4} + \frac{1}{8}i$	136	1.94	7.05	4.53	7.57	250	1.45	1.30	5.47	2.10
	$\gamma_i = \frac{1}{2} + \frac{1}{8}i$	121	1.63	9.93	6.67	10.0	209	1.18	1.84	8.10	2.83
	qAdam[5]	101	1.35	28.3	24.2	13.7	207	1.14	3.12	24.7	2.15

Table 3. Modification effectiveness – VGG-19

	Variant	Tiny ImageNet[6]					Mini ImageNet[7]				
		T[ms]	$\bar{i}$	$\bar{\rho}$	$\widetilde{\rho}$	σ_ρ	T[ms]	$\bar{i}$	$\bar{\rho}$	$\widetilde{\rho}$	σ_ρ
postMin	$\gamma_i = 1.3$	262	40.74	8.02	2.59	1.29	1331	41.08	4.89	1.82	0.95
	Original	260	40.59	8.78	3.09	1.39	1290	40.83	5.25	1.90	1.01
	$\beta = 0.1$	246	40.56	8.85	3.05	1.40	1234	40.81	5.43	1.95	1.02
	$\beta = 0.3$	249	40.54	9.34	3.31	1.52	1210	40.75	5.87	2.01	1.24
	$\gamma_i = \frac{3}{4} + \frac{1}{8}i$	254	40.46	9.27	3.35	1.48	1179	40.65	5.76	2.16	1.11
	qAdam[8]	253	40.54	9.28	3.52	1.41	1197	40.70	5.82	2.52	1.07
	$\gamma_i = \frac{1}{2} + \frac{1}{8}i$	231	40.32	10.6	4.10	1.74	1085	40.39	6.63	2.46	1.34
	$\gamma_i = 1.3$	160	1.74	10.8	5.94	1.39	816	2.10	5.76	2.77	0.97
	$\beta = 0.1$	145	1.57	13.0	7.45	1.30	705	1.81	6.73	3.34	1.11
	Original	148	1.60	12.8	7.51	1.56	712	1.85	6.51	3.38	1.06
	Original 50[3]	149	1.58	12.8	7.52	1.58	698	1.86	6.50	3.34	1.06
	$\beta = 0.3$	142	1.54	14.0	7.84	1.79	679	1.73	7.46	3.61	1.30
	$\gamma_i = \frac{3}{4} + \frac{1}{8}i$	157	1.46	14.7	9.03	1.73	627	1.65	7.29	3.87	1.18
	qAdam[8]	154	1.54	13.6	8.48	1.53	680	1.69	7.49	4.30	1.10
	$\gamma_i = \frac{1}{2} + \frac{1}{8}i$	136	1.30	19.9	12.6	2.32	536	1.4	9.74	5.30	1.52

closer to the original.

6. Summary and Conclusions

This study presents an extended concept of the DeepFool method along with experimental results that allow comparison with the original algorithm. The proposed extension consists of two components. The first involves introducing a modified iterative procedure for calculating perturbations, analogous to neural network training using momentum and the Adam algorithm, just as the original DeepFool algorithm can be closely associated with the backpropagation method. It is expanded further by scaling factor of the perturbation. Both the original and the extended algorithms are supplemented by an additional stage aimed at minimising the perturbation to the necessary minimum.

The experimental results confirmed that the modified DeepFool method can reach the decision boundary in fewer iterations compared to the original approach. In particular, in all variants of the evaluated algorithm, the distance gradient to the decision boundary is computed at each iteration. Therefore, the proposed modifications may also reduce the total number of gradient computations. However, there is a side effect of the modification – a larger final perturbation, severity of which is highly dependant on the chosen hyperparameters, which can be concluded from the result, albeit limited. In contrast, an increased number of iterations in computing gradients might prove to be more time saving, compared to the proposed additional refinement stage; it has improved the results of every single variant tested, yet it has emphatically increased execution time. Importantly, these conclusions are based on relatively low-resolution images, thus the cost of gradient calculation rise along with the resolution is highly probable.

An intriguing outcome of the conducted research is the potential application of various optimisation criteria previously used in neural network training. The scope of such criteria extends beyond the modifications proposed and evaluated in this publication. Many of these are more sophisticated and warrant further investigation to assess their applicability in this context.

Acknowledgement

We express our gratitude for the funding support provided by the program "Excellence initiative–research university" for the AGH University in Krakow, as well as the ARTIQ project (UMO-2021/01/2/ST6/00004 and ARTIQ/0004/2021), and by the funds of the Polish Ministry of Science and Higher Education assigned to the AGH University of Krakow.

References

[1] Abroshan, M., Moosavi-Dezfooli, S.M., et al.: Superdeepfool: a new fast and accurate minimal adversarial attack. Advances in Neural Information Processing Systems 37, pp. 98537–98562 (2024)

[2] Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: Imagenet: A large-scale hierarchical image database. In: 2009 IEEE Conference on Computer Vision and Pattern Recognition. pp. 248–255 (2009)

[3] Gildenblat, J., contributors: Pytorch library for cam methods. `https://github.com/jacobgil/pytorch-grad-cam` (2021)

[4] 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)

[5] He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)

[6] Jelassi, S., Li, Y.: Towards understanding how momentum improves generalization in deep learning. In: International Conference on Machine Learning. pp. 9965–10040. PMLR (2022)

[7] Kingma, D.P., Ba, J.L.: Adam: A method for stochastic optimization. In: the 3rd International Conference for Learning Representations, San Diego. vol. 5 (2015)

[8] Labib, S.F.R., Mondal, J.J., Manab, M.A., Xiao, X., Newaz, S.: Tailoring adversarial attacks on deep neural networks for targeted class manipulation using deepfool algorithm. Scientific Reports 15(1), pp. 10790 (2025)

[9] Le, Y., Yang, X.: Tiny imagenet visual recognition challenge. CS 231N 7(7), pp. 3 (2015)

[10] Li, J., Xu, Y., Hu, Y., Ma, Y., Yin, X.: You only attack once: Single-step deepfool algorithm. Applied Sciences 15(1), pp. 302 (2024)

[11] LTS4: DeepFool. `https://github.com/lts4/deepfool` (2015), accessed 2025-03-16

[12] Modas, A., Moosavi-Dezfooli, S.M., Frossard, P.: Sparsefool: a few pixels make a big difference. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 9087–9096 (2019)

[13] Moosavi-Dezfooli, S.M., Fawzi, A., Frossard, P.: Deepfool: a simple and accurate method to fool deep neural networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 2574–2582 (2016)

[14] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: Pytorch: An imperative style, high-performance deep learning library (2019), https://arxiv.org/abs/1912.01703

[15] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A.C., Fei-Fei, L.: ImageNet Large Scale Visual Recognition Challenge. International Journal of Computer Vision (IJCV) 115(3), pp. 211–252 (2015)

[16] Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: 3rd International Conference on Learning Representations (ICLR 2015). Computational and Biological Learning Society (2015)

[17] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., Fergus, R.: Intriguing properties of neural networks. In: 2nd International Conference on Learning Representations, ICLR 2014 (2014)