[Re] Subspace Attack: Exploiting Promising Subspaces for Query-Efficient Black-box Attacks

Anonymous Author(s)
Affiliation
Address
`email`

Abstract

As part of the NeurIPS 2019 Reproducibility Challenge, we chose to attempt reproduce the attack algorithm proposed in "Subspace Attack: Exploiting Promising Subspaces for Query-Efficient Black-box Attacks". Our reported results are better than the original paper in terms of the median number of queries per attack, but worse in terms of failure rate. A concise assessment of our implementation is also included.

1 Introduction

As the use of Machine Learning in services and applications has increased, it becomes important to assess its reliability and security. In the last few years, several flaws have been found in state-of-the-art Machine Learning methods and algorithms. One of the most studied flaws is the fact that many models can be attacked using the so-called adversarial examples, inputs that are minimally perturbed in order to be misclassified by the victim model.

In this paper we attempt to reproduce the results obtained in the paper "Subspace Attack: Exploiting Promising Subspaces for Query-Efficient Black-box Attacks" by Yan and Guo et al. [1], published among the proceedings of NeurIPS 2019. In their paper, the authors present a new kind of black-box attack, that, for the first time, ensembles attacks' transferability and gradient estimation for Projected Gradient Descent.

Our report has been written for the NeurIPS 2019 Reproducibility Challenge[1]. It consists of a background about adversarial attacks (Section 2), our methodology (Section 3), and a concise reproducibility section (Section 4). These are followed by our results, discussion, and conclusions (Sections 5 to 7, respectively).

2 Background

2.1 Adversarial Examples

Adversarial examples are inputs fed into a machine learning model and mislead the model to make an incorrect prediction. Several machine learning architectures have been proven to be vulnerable in adversarial settings, where the adversarial examples can be generated with several attacking techniques. Thus, the aim of an attack is to perturb an input $\mathbf{x} \in \mathbb{R}^n$ and to trick a victim model $f : \mathbb{R}^n \to \mathbb{R}^k$, such that it gives a wrong prediction, $\arg\max_i f(\mathbf{x})_i \neq y$ (where y the true label). This general definition describes the so-called *untargeted* attacks. A more specific type of attack is a *targeted* one, which aims for a wrong classification of a specific label $y' \neq y$, $\arg\max_i f(\mathbf{x})_i = y'$.

[1] https://reproducibility-challenge.github.io/neurips2019/

Submitted to 33rd Conference on Neural Information Processing Systems (NeurIPS 2019). Do not distribute.

In general, adversarial attacks can be classified into two main types: white-box and black-box. While only the confidence score from the victim model is accessible to the latter, the former has full access to the network parameters of the victim. This allows for an efficient use of various gradient-based methods [2] to generate adversarial examples. This emphasises the attacker's preference for the white-box models. However, it is not a realistic setting.

Let us now consider the more realistic setting in which the attacker has no access to the trained victim model parameters. In this case, it is not possible to compute the gradient of the model with respect to the perturbed input. Nonetheless, several methods have been proposed to overcome this issue [3, 4, 5, 6, 7, 8, 9, 10, 11]. Some of these methods rely on the so-called adversarial example transferability [12], in which a surrogate model is trained with data samples labeled by the victim model, and is then used to generate adversarial examples that work on the victim model too. The other kind of methods make use of zeroth-order optimization methods (e.g. finite differences [5]) by querying the victim model to estimate the gradient of the victim model to run Projected Gradient Descent (PGD). A drawback of these methods is the need for a large number of queries for a good estimation of the victim model's gradient. This motivates the attempt for reducing the required number of queries for a successful attack. The next subsection describes the solution proposed in [1], which harnesses the features of both zeroth-order optimization and transferability.

2.2 Subspace Attack Algorithm

In what follows, we present the essence of the subspace attack proposed in [1]. This will include an introduction to the principles of the *Bandit attack* proposed in [10], on which the subspace attack heavily relies.

Given a classification loss function $\mathcal{L}(x, y)$, where x is some input and y its corresponding label. We can formulate the adversarial attack problem as follows:

$$x' = \underset{x':\|x'-x\|_p \leq \epsilon_p}{\arg\max} \; \mathcal{L}\left(x', y\right) \tag{1}$$

Let $g^* = \nabla_x \mathcal{L}(x, y)$ be the gradient of $\mathcal{L}$ at (x, y), Then the goal of the gradient estimation problem is to find a unit vector $\hat{g}$ that maximize the inner product $\mathbb{E}\left[\hat{g}^T g^*\right]^2$.

In [10], the authors have proven that the gradient estimation problem can be considered as a bandit optimization problem. Bandit optimization is a tool used in online convex optimization, in which there is an agent playing a game that includes a sequence of rounds. During each round t, the agent must select an action and incurs in a loss $\mathcal{L}_t$, whose expectation across all the rounds should be minimized. The main novelty introduced by [10] is the fact that the estimation is improved by accumulating prior information about gradients in a latent vector $\mathbf{g}_t$, that is updated each round with an estimation of the gradient of the victim model, performed via finite differences. In [10], each new basis vector $\mathbf{u}_t$ used for the gradient estimation is randomly sampled from a Gaussian distribution.

In the subspace attack proposal, the authors present a novel way to accumulate prior information about the gradient. Instead of using the full basis of the sample's space to estimate the true gradient, a subspace of directions is chosen by using the gradients of some reference models. These models are pre-trained on the same dataset as the victim and their parameters are fully known. Each round of the bandit optimization uses the gradient of a randomly chosen reference model, with respect to the adversarial input, as a basis for the gradient estimation. Moreover, [1] proposes a way to apply different ratios of drop-out to the enrich the set of reference models.

The method proposed by [1] is presented in Algorithm 1, which we report in our paper for the sake of clarity and ease of read.

3 Methodology

In this work, we implemented the subspace attack using PyTorch [13], following the algorithm specified in [1], without looking at the source code of the official implementation of the attack. We chose to evaluate the reproducibility of the algorithm by attempting to replicate untargeted subspace attacks on the GDAS [14] and WRN [15] models trained on the CIFAR-10 dataset [16]. We use

[2] The expectation here is taken over the randomness of the estimation algorithm

Algorithm 1: Subspace Attack, as described in [1].

Input: A benign example $\mathbf{x} \in \mathbb{R}^{\mathbf{n}}$, its label y, a set of m reference models $\{f_0, ..., f_{m-1}\}$, a chosen attack objective function $\mathcal{L}(\cdot, \cdot)$, and a victim model from which the output of f can be inferred.

Output: An adversarial example $\mathbf{x}_{adv}$ that fulfills $\|\mathbf{x}_{adv} - \mathbf{x}\|_\infty \leq \epsilon$.

Initialize the adversarial example to be crafted $\mathbf{x}_{adv} \leftarrow \mathbf{x}$.
Initialize the gradient to be estimated $\mathbf{g} \leftarrow \mathbf{0}$.
Initialize the drop-out/layer ratio p.
while not successful **do**
    Choose a reference model whose index is i, uniformly at random.
    Calculate prior gradient with drop-out/layer ratio p as $\mathbf{u} \leftarrow \frac{\partial \mathcal{L}(f_i(\mathbf{x}_{adv}, p), y)}{\partial \mathbf{x}_{adv}}$
    $\mathbf{g}_+ \leftarrow \mathbf{g} + \tau \mathbf{u}$, $\mathbf{g}_- \leftarrow \mathbf{g} - \tau \mathbf{u}$
    $\mathbf{g}'_+ \leftarrow \mathbf{g}_+ / \|\mathbf{g}_+\|_2$, $\mathbf{g}'_- \leftarrow \mathbf{g}_- / \|\mathbf{g}_-\|_2$
    $\Delta_t \leftarrow \frac{\mathcal{L}(f(\mathbf{x}_{adv} + \delta \mathbf{g}'_+), y) - \mathcal{L}(f(\mathbf{x}_{adv} + \delta \mathbf{g}'_-), y)}{\tau \delta} \mathbf{u}$
    $\mathbf{g} \leftarrow \mathbf{g} + \eta_{\mathbf{g}} \Delta_t$
    $\mathbf{x}_{adv} \leftarrow \mathbf{x}_{adv} + \eta \cdot \mathrm{sign}(\mathbf{g})$
    $\mathbf{x}_{adv} \leftarrow \mathrm{Clip}(\mathbf{x}_{adv}, \mathbf{x} - \epsilon, \mathbf{x} + \epsilon)$
    $\mathbf{x}_{adv} \leftarrow \mathrm{Clip}(\mathbf{x}_{adv}, 0, 1)$
    Update the drop-out/layer ratio p, following the original paper's policy.
end while
return $\mathbf{x}_{adv}$

as reference models VGG-11/13/16/19 [17] and AlexNet [18]. In order to verify that we use the same settings – hyper-parameters and pre-trained models – as the original subspace experiments, we checked their log files and loaded the pre-trained models from their source code repository. The latter was also required since the available models by PyTorch are pre-trained on ImageNet, while our experiment uses images from CIFAR-10.

Aiming to further assess the performance of our algorithm implementation, we added options to keep track of some values encountered during the attack. In addition to the required number of queries for a successful attack of each image classification, we recorded in each iteration the cosine similarity between the estimated gradient and the true gradient (as done in [10]), and the norms of these gradients. Moreover, as the subspace attack's algorithm uses only the sign of the gradient, we also keep track of the ratio of the matching signs of the gradient elements with respect to the previous estimated gradient (denoted by *the ratio of the common signs*) and the true gradient (denoted by *the ratio of the correctly estimated signs*).

Using the evaluation metrics of [1], an attack's performance is being evaluated with respect to its failure rate and the number of queries. Since the distribution of the latter is heavy-tailed, we use the mean and the median number of queries per a successful attack.

4 Reproducibility

We first try to implement Algorithm 1 and run the experiment attacking GDAS. We used VGGNets and AlexNet as reference models and the hyper-parameters listed in [1] and [10]. We then expand our implementation to be used to attack WRN and Pyramidnet[3], and look for better hyper-parameters.

4.1 Machine Setup, experiment duration, and budget

We run our experiment both on Google Cloud Platform and on Code Ocean [19]. The Ubuntu Virtual Machine on Google Cloud Platform has an Nvidia Tesla T4 GPU, 52 GB memory and 8 vCPUs. A set of 1000 attacks against GDAS using VGGs and AlexNet as reference models took us about 7h45m to run, with a total of \$7.75 spent. On Code Ocean, the Virtual Machine we used runs Ubuntu on a 4-cores CPU with 60 GB of memory and an Nvidia Tesla K80 GPU. As Code Ocean is a sponsor of the Reproducibility Challenge, we have been provided with free compute time to run the experiments.

[3] An attack on Pyramidnet is implemented but we didn't run such experiment due to large computation time.

A set of 1000 attacks against GDAS using VGGs and AlexNet as reference models took us about 9 hours to run.

Even though our reported results are produced using GPUs, our implementation can run on a CPU as well. This was tested on a laptop with Ubuntu, an Intel Core i7-7500U and 8GB memory. The computation time took about 6 times longer than a GPU (in terms of iterations per second). In addition, the results obtained could be different, due to differences in floating point precision and implementations of low-level operations[4]. Such results are outside of the scope of this challenge, and therefore not reported.

4.2 Algorithm implementation

The source code of our implementation is available online[5]. As mentioned in section 3, our implementation is done using the PyTorch framework. We use the pre-trained reference and victim models[6] provided with the original paper's code repository, along with their corresponding Python classes.

The CIFAR-10 dataset is loaded using PyTorch's `torchvision.datasets`, and is iterated using a `DataLoader`. The dataset is shuffled, but the seed can be fixed, for comparable results. In all our experiments, both PyTorch and NumPy seeds are set to 0, and the following values are set: `torch.backends.cudnn.deterministic = True` and `torch.backends.cudnn.benchmark = False`. Regarding the implementation of the attack itself, we leveraged PyTorch autograd capabilities to compute the gradient of the reference models.

Two clipping procedures of the adversarial example are presented in lines 12 and 13 of Algorithm 1. While the former was implemented using PyTorch's functions `torch.min` and `torch.max`, in order to keep $\mathbf{x}_{adv}$ in the region included in $[\mathbf{x} - \epsilon, \mathbf{x} + \epsilon]$ with ℓ_∞-norm, the latter was implemented using PyTorch's `torch.clamp`, with 0 and 1 as arguments.

Yan and Guo et al. specify the use of "the hinge logit-diff adversarial loss from Carlini and Wagner" [1]. In their paper [20], Carlini and Wagner list a number of possible loss functions that can be used along with a hinge term, which regulates the ℓ_∞-distance of the adversarial example from the original one. However, we did not manage to understand how to implement it as part of the subspace attack algorithm. So, we use the Cross Entropy loss without the hinge term. We understood from Algorithm 1 that the constraint into the admitted perturbation bounds is applied by the clipping step in line 12.

4.3 Hyper-parameters and experiments settings

The hyper-parameters used in a subspace attack[7] are:

- τ : the bandit exploration.
- δ : the finite difference probe.
- η_g: the OCO learning rate.
- η : the image learning rate.
- p : the dropout/layer ratio of the reference models.

In [1], it is claimed that the used hyper-parameters were the same as those used in [10]. However, we have found out that the notation was different and a bit confusing. We reconstructed a translation in Table 1 by cross-referencing the notation used in Algorithm 1 of [1] and in Algorithms 1, 2 and 3 of [10]. The reader should note that we have listed two letters as translations of the δ used in [1] – ϵ and η. This is due to the fact that, in Algorithm 2 of [10], an ϵ is used as finite difference probe, but in Table 3 of Appendix C of the same paper, where hyper-parameters values are listed, the finite difference probe is listed as "η (Finite difference probe)".

To choose the hyper-parameters' values, we have started from those stated in [1], which should be the same used in [10], excluding ϵ and η. However, as discussed in Section 6, we obtained results which

[4]https://pytorch.org/docs/stable/notes/randomness.html

[5]https://github.com/epfl-ml-reproducers/subspace-attack-reproduction

[6]https://go.epfl.ch/subspace-pretrained

[7]For hyper-parameters, we use the same notation as [1]

Table 1: Translation between hyper-parameters in Subspace attack [1] and the bandit attack [10].

	Subspace Attack [1]	Bandit Attack [10]	Value [1]
Bandit exploration	τ	δ	1.0
Finite difference probe	δ	ϵ/η	0.1
Image ℓ_p learning rate	η	h	1/255
OCO learning rate	η_g	η	100

were far worse than those obtained in [1]. After exploring the published logs of the experiment[8] we found out that in the original paper, a value of $\eta_g = 0.1$, instead of $\eta_g = 100$ was used. The set of hyper-parameters that yields the best results is presented in Table 2.

Table 2: The set of Hyper-parameters which yield the best results.

Hyper-parameter	Value
τ (Bandit exploration)	1.0
δ (Finite difference probe)	0.1
η (Image ℓ_p learning rate)	1/255
η_g (OCO learning rate)	0.01

The droupout was applied to the reference models, in the following procedure: each experiment starts with a 0.05 dropout ratio which is increased every iteration by 0.01, until a maximum ratio of 0.5 is reached.

Finally, using the above settings, each experiment included untargeted attacks on the classification of 1,000 images using a limit of 10,000 queries per attack – after which the attack is considered a failure. Moreover, we set a maximum perturbation of $\epsilon = 8/255$, defined using ℓ_∞-norm.

5 Results

In this section, we present the assessment of our implementation as mentioned in Section 3, followed by a comparison of the effectiveness of our subspace attacks to those reported in the original paper. These results are discussed in Section 6.

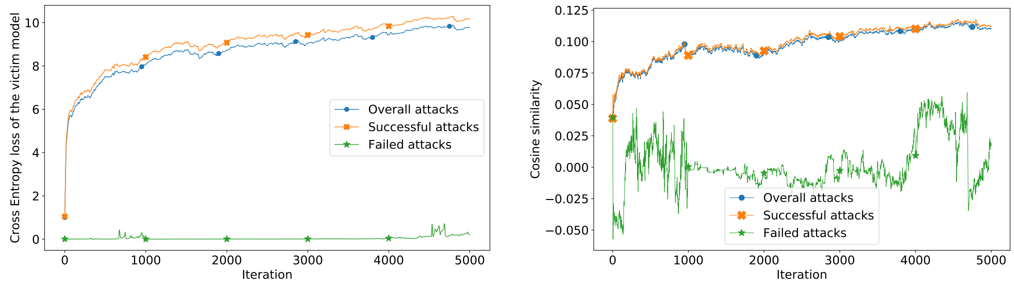
In order to evaluate the performance of our algorithm and its implementation, we saved the progression of the loss values along with the estimated and true gradients information of a set of 49 attacks on GDAS. We required each attack to complete 5,000 gradient estimation iterations, i.e. 10,000 queries. Figures 1 and 2 present the mean value along the gradient estimation process of the following parameters:

- the cross entropy loss of the victim model (Figure 1a)
- the cosine similarity between the estimated and the true gradient (Figure 1b)
- the ratio of correctly estimated gradient's sign (Figure 2a)
- the ratio of common signs between subsequent estimated gradients (Figure 2b)

Having only a single failed attack in this run, its corresponding curve presents the exact value, while the rest of the curves present value averaged on the 48 successful attacks and 49 attacks overall. The fluctuations of the curves are a feature of the stochastic nature of the algorithm.

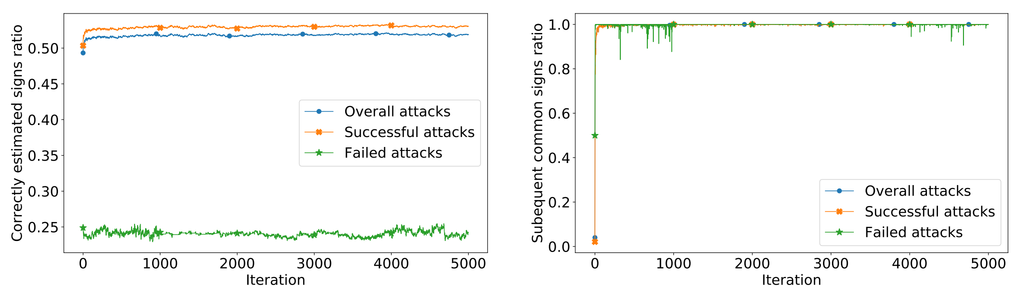
In addition, we scanned the values of the η_g hyper-parameter. However, as presented in Table 3, it seems that $\eta_g = 0.1$ yields the best results in terms of a trade-off between failure rate and median number of queries. Our best results of the different attack experiments are compared with the two baselines [1, 10] in Table 4. There, it is demonstrated that while our implementation managed to get similar mean and better median numbers of queries as the subspace attack paper, it leads to higher failure rates with respect to the baselines.

[8] https://go.epfl.ch/subspace-original-logs



(a) Cross Entropy loss *(b) Cosine similarity*

Figure 1: The mean value of the Cross Entropy loss of the true victim model (a) of cosine similarity between the estimated and the true gradient (b) as a function of the estimation iteration in our Subspace Attack implementation. The results of the successful attacks, the failed attacks, and all the attacks were averaged over 48, 1, and 49 attacks respectively.



(a) Correctly estimated signs *(b) Common signs between estimated gradients*

Figure 2: The mean value of the ratio of correctly estimated signs (a), and the ratio of the common signs between two subsequent estimated gradients (b) as a function of the estimation's iteration in our Subspace Attack implementation. The results of the successful attacks, the failed attacks, and the all the attacks were averaged over 48, 1, and 49 attacks respectively.

Table 3: Comparison between the results obtained with different η_g.

Victim	η_g	Mean Queries	Median Queries	Failure Rate
WRN	0.01	369	16	3.2%
	0.1	321	16	3.4%
	100	288	12	10.1%
GDAS	0.001	263	18	6.3%
	0.01	321	20	5.9%
	0.1	282	18	6.7%
	100	354	16	12.7%

Table 4: Comparison between the results obtained by our implementation and those of the original paper [1] and of [10]. For the results obtained with the other implementations we report the results presented in [1]. In η_g for the results obtained by the original paper we state 0.1 as this is the value we found in their logs, and not the one stated in their paper.*

Victim Model	Method	Ref. Models	η_g	Mean Queries	Median Queries	Failure Rate
WRN	Bandit-TD [10]	-	100	713	266	1.2%
	Subspace Attack [1]	AlexNet+VGGNets	0.1*	392	60	0.3%
	Subspace Attack (Ours)	AlexNet+VGGNets	0.1	330	14	3.7%
GDAS	Bandit-TD [10]	-	100	373	128	0.0%
	Subspace Attack [1]	AlexNet+VGGNets	0.1*	250	58	0.0%
	Subspace Attack (Ours)	AlexNet+VGGNets	0.1	282	18	6.7%

6 Discussion

The trends of the mean values of the loss and the cosine similarity (Figures 1a and 1b) confirms that our implementation is indeed efficiently attacking the victim model, as the loss is increased, and the directional agreement between the estimated and the true gradient improves. Furthermore, the obtained results of cosine similarity exceed the ones presented in [10], since our successful attacks demonstrate twice the cosine similarity than those reported in the reference, in the first 5,000 iterations.

Regarding the percentage of correctly estimated signs, we see in Figure 2a that the successful attacks achieved a mean ratio of correctly estimated signs of about 55%. This is only slightly above the agreement expected by a randomly chosen vector (50%). However, the success of the attacks suggests that we are indeed exploiting the reference models to estimate only the gradient components corresponding to the most relevant subspaces used for the classification, as aimed in [1, 10]. Clearly, the reason why the failed attack did not succeed is the inability to correctly estimate the signs. Furthermore, we can observe in Figure 2b that the signs of the gradients almost stop changing after the very first few iterations. This leads us to think that the correctly estimated signs are always the same, and could correspond exactly to the most relevant subspaces.

Keeping the guiding rule of [1], we use the hyper-parameters of [10], excluding ϵ and η. As we are performing attacks that make use of ℓ_∞-norm, we should use $\eta_g = 100$. This yields worse results, as seen in Table 3. Information extracted from the logs of the subspace paper mention the use of $\eta_g = 0.1$. The fact that our use of $\eta_g = 0.1$ also result with better results indicates that our implementation is close to the original one.

Comparing our results to those of the original paper, we notice that while we managed to reduce the computational cost of the attacks, we couldn't reach the reported failure rates of either the original paper or the Bandit attack. Even though we verified that we followed every step of the algorithm, checked for differences in input models and hyper-parameters, we couldn't improve our produced failure rates. It could be that the source of the problem lies in the inconsistency of the implementation of the loss function of [20].

7 Conclusion

In this work, we re-implemented the algorithm of the subspace black-box adversarial attack presented by Yan et al. [1], and introduced methods for the evaluation of its performance. However, we didn't manage to match our effectiveness to that of the original paper. In terms of the reproducibility of [1], it is advisable for the authors to include a clearer statement regarding the used set of hyper-parameters. In addition, when relying on settings from another work, a corresponding mapping of the notation changes can be highly beneficial. Moreover, additional information about the used loss function could be helpful to replicate the obtained results.

Acknowledgments

The authors would like to thank the NeurIPS Reproducibility Challenge sponsor Code Ocean [19], for providing them with additional compute time for free.

References

[1] Y. Guo, Z. Yan, and C. Zhang, "Subspace attack: Exploiting promising subspaces for query-efficient black-box attacks," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 3820–3829. [Online]. Available: http://papers.nips.cc/paper/8638-subspace-attack-exploiting-promising-subspaces-for-query-efficient-black-box-attacks.pdf

[2] 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.

[3] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning," in *Proceedings of the 2017 ACM on Asia conference on computer and communications security*. ACM, 2017, pp. 506–519.

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

[5] P.-Y. Chen, H. Zhang, Y. Sharma, J. Yi, and C.-J. Hsieh, "Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models," in *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, ser. AISec '17. New York, NY, USA: ACM, 2017, pp. 15–26. [Online]. Available: http://doi.acm.org/10.1145/3128572.3140448

[6] A. Ilyas, L. Engstrom, A. Athalye, and J. Lin, "Black-box adversarial attacks with limited queries and information," *arXiv preprint arXiv:1804.08598*, 2018.

[7] N. Narodytska and S. Kasiviswanathan, "Simple black-box adversarial attacks on deep neural networks," in *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE, 2017, pp. 1310–1318.

[8] A. N. Bhagoji, W. He, B. Li, and D. Song, "Practical black-box attacks on deep neural networks using efficient query mechanisms," in *European Conference on Computer Vision*. Springer, 2018, pp. 158–174.

[9] C.-C. Tu, P. Ting, P.-Y. Chen, S. Liu, H. Zhang, J. Yi, C.-J. Hsieh, and S.-M. Cheng, "Autozoom: Autoencoder-based zeroth order optimization method for attacking black-box neural networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 742–749.

[10] A. Ilyas, L. Engstrom, and A. Madry, "Prior convictions: Black-box adversarial attacks with bandits and priors," *ICLR 2019*, 2018. [Online]. Available: https://arxiv.org/abs/1807.07978

[11] C. Guo, J. R. Gardner, Y. You, A. G. Wilson, and K. Q. Weinberger, "Simple black-box adversarial attacks," *arXiv preprint arXiv:1905.07121*, 2019.

[12] N. Papernot, P. D. McDaniel, and I. J. Goodfellow, "Transferability in machine learning: from phenomena to black-box attacks using adversarial samples," *CoRR*, vol. abs/1605.07277, 2016. [Online]. Available: http://arxiv.org/abs/1605.07277

[13] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[14] X. Dong and Y. Yang, "Searching for a robust neural architecture in four gpu hours," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 1761–1770.

[15] S. Zagoruyko and N. Komodakis, "Wide residual networks," *arXiv preprint arXiv:1605.07146*, 2016.

[16] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," University of Toronto, Tech. Rep., 2009.

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

[18] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[19] A. Clyburne-Sherin, X. Fei, and S. A. Green, "Computational reproducibility via containers in social psychology," *Meta-Psychology*, vol. 3, 2019.

[20] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2017, pp. 39–57.