Sergio Martinez
Robin Christoffer Vold

Evaluating Evolution Strategies as a Method to Combat Adversarial Attacks on Convolutional Neural Networks

Bachelor's thesis in Computer Engineering
Supervisor: Ole Christian Eidheim
May 2022

Bachelor's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Sergio Martinez
Robin Christoffer Vold

Evaluating Evolution Strategies as a Method to Combat Adversarial Attacks on Convolutional Neural Networks

Bachelor’s thesis in Computer Engineering
Supervisor: Ole Christian Eidheim
May 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Abstract

In recent years, deep neural networks have been used for image classification, and have achieved impressive results. However, neural networks are vulnerable to adversarial attacks, where small alterations are added to the input images. This paper aims to address this vulnerability by using evolution strategies. Evolution strategies are a gradient free optimization technique which has recently resurfaced as an algorithm that can be parallelized very efficiently. We have trained models with evolution strategies, and evaluated how this affects a model's robustness against adversarial attacks. We find that models trained with evolution strategies show interesting properties during training, and are more robust against certain attacks, but not against others. [1]

[1]The source code for our project can be found here: https://github.com/Evolutionary-strategies/Evolutionary-strategies and here: https://github.com/Evolutionary-strategies/ares

Sammendrag

De siste årene har dype nevrale nettverk blitt brukt til bildeklassifisering, og de har oppnådd imponerende resultater. Desverre er nevrale nettverk sårbare for angrep, der små endringer legges til bildene. Denne artikkelen forsøker å løse dette problem ved å bruke evolusjonsstrategier. Evolusjonsstrategier er en gradientfri optimaliseringsteknikk som nylig har dukket opp igjen som en algoritme som kan parallelliseres veldig effektivt. Vi har trent modeller med evolusjonsstrategier, og evaluert hvordan dette påvirker en modells robusthet mot motstandsangrep. Vi finner at modeller trent med evolusjonsstrategier viser interessante egenskaper under trening, og er mer robuste mot visse angrep, men ikke mot andre. [2]

[2]Kildekoden for prosjektet vårt ligger her: https://github.com/Evolutionary-strategies/Evolutionary-strategies og her: https://github.com/Evolutionary-strategies/ares

Acknowledgements

We would like to express our gratitude to our supervisor, Ole Christian Eidheim, who has guided us during this project. We would also like to thank the Department of Computer Science at NTNU, and Aleksander Tandberg for providing us with access to hardware with which we could perform our experiments. Finally we wish to thank Liliana Martinez for proofreading, and providing feedback, and Janne Cathrin Hetle Aspheim and Hilde Hefte Haug for providing feedback on our result representations.

Contents

List of Figures

List of Tables

Abbreviations

AGN - Additive Gaussian Noise attack
AUN - Additive Uniform Noise attack
BIM - Basic Iterative Method
CE loss - Cross Entropy Loss
CMA - Covariance Matrix Adaptation
CNN - Convolutional Neural Network
CR - Contrast Reduction Attack
CW attack - Carlini Wagner Attack
CW loss - Carlini Wagner Loss
ES - Evolution Strategies
FGSM - Fast Gradient Sign Method
GD - Gradient Decent
MIM - Momentum Iterative Fast Gradient Sign Method
NES - Natural Evolution Strategies
PGD - Projected Gradient Descent
S&P - Salt and Pepper Noise Attack
SPSA - Simultaneous Perturbation Stochastic Approximation

1 Introduction

Starting with AlexNet in 2012, Convolutional Neural Networks (CNN) trained with Gradient Descent (GD) have shown themselves to be a powerful tool for image recognition [24]. The finding that CNNs can be trained on GPUs have made it so that CNNs can also be trained quite quickly. Since then, CNNs have been able to achieve up to a 90% accuracy on ImageNet [5]. However, with the introduction of adversarial attacks it has been shown that CNNs can be fooled into misclassifying images which have very slight alterations [42, 16]. Misclassification has also been found with real images, where modifications to the surroundings of the classifiable object have been made, causing the model to fail [6].

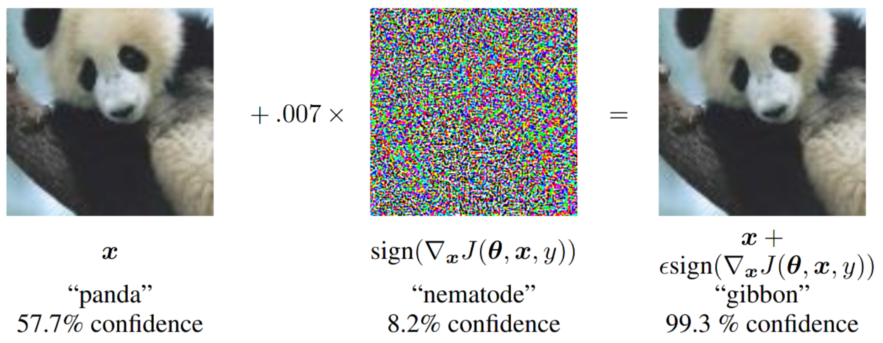
+ .007 ×
=
$\boldsymbol{x}$
"panda"
57.7% confidence
$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"nematode"
8.2% confidence
$\boldsymbol{x} + \epsilon\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"gibbon"
99.3 % confidence

Figure 1: An illustration of an adversarial attack taken from ”Explaining and harnessing adversarial examples” by Goodfellow et al. [16]. Left: An image of a panda is passed to a classifier, and classified correctly. Right: Perturbation is added to the image, causing the model to misclassify.

In Figure 1, the perturbation is imperceptible to the human eye but the model fails to classify the image correctly. The finding of attacks such as these has resulted in a developing research field, in an attempt to improve the defences of classifier models [27].

In 2017, Salimans et al. used a novel version of Evolution Strategies (ES) to train MuJoCo models to walk, and achieved good results [40]. The approach of

Salimans et al. generated the mutations for ES training by using known seeds. This allowed the algorithm to be efficiently scaled, and run by a large number of machines. Additionally, the resulting models achieved by this method of training are quite varied, and solve the problem in several different ways.

In this report, we aim to train image classification models with the algorithm proposed by Salimans et al., and see if the resulting models have increased resilience against adversarial attacks. Due to the inherent randomness in the algorithm and the results Salimans et al. achieved, we hypothesize that ES training might find a larger width of solutions than traditional GD training. Our goal is to train several models using ES, to discover whether there will be more variety in our models or not, and if any of these models will be more robust.

2 Related Work

In this chapter, we describe previous research within the fields of evolution strategies and adversarial attacks. There is a large amount of research on these topics, so we have chosen to review only what we found to be most relevant to our work. The descriptions we give in this section are brief and are only meant to give the reader a basic understanding of the significance of the research, so that the reader can better contextualize our work. We encourage interested readers to read the original papers for more detailed explanations.

2.1 Evolution Strategies

Evolution Strategies (ES) are a type of evolutionary algorithm and were first developed in Germany in the 1960s [3]. ES handles optimization problems by mutating the proposed solution several times, and then selecting the best mutations and recombining them into a new proposed solution. This process is continued in an iterative fashion until the solution is satisfactory.

2.1.1 Natural Evolution Strategies

Wierstra et al. proposed Natural Evolution Strategies (NES), where the mutations made to the solution are taken from a randomized search distribution [44]. All the candidate solutions are then evaluated using a fitness function. Using the fitness of the candidate solutions, a search gradient is estimated. Finally, gradient ascent is performed on the estimated natural gradient. As in normal ES, this process is iterated until the goal has been reached.

The algorithm used by Salimans et al. is a form of NES [40]. In their algorithm, the mutation step is expanded to rely upon random seeds to select mutations from the random search distribution. The use of random seeds enables the algorithm to be run in parallel, as it allows the worker threads to transmit the seeds they used to other workers, instead of the entire candidate solution. Each worker then reconstructs the other workers' solutions using the seeds, and performs the gradient ascent step using the fitness of these solutions. Once this is done all the workers will be synchronized, as they will have recombined the previous experiments into a common, new candidate solution. This algorithm will from now on be referred to as Parallelizable NES.

In 2021, Jacobsen and Dalheim from NTNU used an algorithm similar to Parallelizable NES to generate cellular automata [20]. They found that training with ES instead of GD produced cellular automata that were consistent over time once the desired image had been generated. On the other hand, cellular automata generated with GD would continue to make changes after the desired image was finished. This finding indicates that models trained with ES may have properties not found in models trained with GD.

2.2 Adversarial Attacks

Szegedy et al. showed that neural networks can be tricked into misclassifying images by adding perturbations to the input images [42]. By using optimization algorithms to find the smallest perturbation needed, these perturbations could be so small that they were imperceptible to the human eye. Biggio et al. similarly found that adversarial perturbation could be used to bypass a PDF malware scanner [4]. After these findings, a lot of research has gone into both the development of new more powerful attack methods, and ways to defend models against them. These attack methods differ in how much information they have about the model, and can be divided into white box and black box attacks.

2.2.1 Gradient-Based White Box Attacks

White box attacks are attacks that have complete information about the model, its parameters and its training data. The attacks used by Szegedy et al. and Biggio et al. [42, 4] belong in this category. These types of attacks use an objective function to find the loss a model has on an input image, and then use the gradient of this loss to perturb the image minimally. Several of the attacks we perform in our experiments fall within this category.

Goodfellow et al. described a white box attack using the loss gradient to linearize the loss function, with the L_∞ norm as a distance measure [16]. This attack is called the Fast Gradient Sign Method, often abbreviated as FGSM. Kurakin et al. proposed an extension of the FGSM algorithm by performing FGSM multiple times, iteratively taking small steps along the gradient, always ensuring the attack is within the chosen perturbation budget [25]. This method is called the Basic Iterative Method (BIM). Dong et al. further improved upon this idea by adding a momentum term to BIM, which helps to stabilize the update directions of each iteration [13]. This attack is known as MIM. Another similar method is the Projected Gradient Decent attack (PGD), introduced by Madry et al. [32]. It works similarly to the BIM algorithm, but it starts at a random perturbation within the perturbation budget, and iteratively takes small steps along the gradient to achieve the greatest loss within the given perturbation budget.

The DeepFool attack, proposed by Moosavi-Dezfooli et al., is based on iterative linearization to make the model misclassify with the minimal perturbation possible [35]. The paper described using the L_2 norm as a distance measure for the attack, but variations using other L_p norms are also provided. Another gradient-based adversarial attack, quite similar to DeepFool, is the NewtonFool attack presented by Jang et al. [22]. It is an iterative attack where the step size is a function of the loss landscape. The algorithm described by Jang et al. uses the minimum norm, but a general method for all L_p norms is also given.

As a way to show that current defence methods were insufficient, Carlini and

Wagner proposed three improved gradient-based attacks, each using one of the L_0, L_2 and L_∞ norms as a distance measure [7]. Out of these, the main one is the L_2 attack, which we will hereby refer to as the CW attack. The improvements found in this attack are in part achieved by changing the objective function (the function used to determine how far an image is from being an adversarial example). The CW attack, along with PGD are considered to be the most powerful gradient-based attacks we have today [12, 7].

2.2.2 Black Box Attacks

In black box attacks, the attacker has limited information about the model, and in some cases a limited number of model queries. In this report we use attacks that have knowledge of the probability distribution of a model guess, known as score-based attacks, and attacks that only have access to a discrete model guess, known as decision-based attacks.

Uesato et al. proposed repurposing several gradient free optimization algorithms into adversarial attacks for use in score-based settings [43]. Among these were Simultaneous Perturbation Stochastic Approximation (SPSA) [41] and NES [44]. SPSA was found to reliably produce adversarial examples. Ilyas et al. proposed the use of ES in adversarial attacks, as a query efficient attack method in query limited score-based settings [19]. Ilyas et al. estimate the search gradient from model queries, using a NES algorithm similar to Parallelizable NES. Then the sign of the estimated gradient is used with PGD to generate an adversarial image. Li et al. similarly proposed using NES to attack classifiers in black box settings [28]. This attack does not estimate the gradient, and does not search for the optimal adversarial perturbation. Instead, NES is used to find a probability distribution centered around the input image, where samples drawn from the distribution have a high likelihood of being adversarial.

As da Costa et al. argue, there is a lot of uncertainty around the quality of the input images [9]. Dodge and Karam showed that CNNs are especially vulnerable to blur and noise, but also show some accuracy decrease with high amounts of contrast or compression [11]. These findings can be formalized into different attacks especially suited for a decision-based setting with only discrete predictions. Bennabhaktula et al. describe two decision-based black box attacks, Additive Uniform Noise attack (AUN) and Additive Gaussian Noise attack (AGN) [2]. Both of these work in a similar way, by adding random values drawn from either a uniform distribution or a Gaussian distribution to each pixel in the image. The formalization of noise attacks has also lead to the creation of attacks like Salt and Pepper Noise attack (S&P) and Contrast Reduction attack (CR) [39]. Nazaré et al. and da Costa et al. both use S&P Noise to show an accuracy decrease with a noisy test set [36, 9]. Dodge and Karam's findings regarding contrast-based perturbations also demonstrate the possibility of using CR as a weak adversarial attack [11].

2.2.3 Adversarial Defence

Several different approaches have been tried in the field of adversarial defence. The methods used are often divided into categories. Li et al. separate defences into four categories [27], whereas Dong et al. use five non-exclusive categories [12]. In this section, we will cover the categories known as Robust Training, Input Transformation, Randomization, Detection and Model Ensemble.

Robust Training involves making changes to the training of the CNN model to obtain greater robustness. One form of Robust Training is training the model on already adversarially modified images. This was proposed in the same paper as the FGSM attack [16]. Several methods that use this idea have been proposed [32, 47] and shown to be among the most effective against state of the art attacks such as the CW attack [1, 12].

Input Transformation uses a transformation of the input images before they are evaluated by the model. Several types of input transformation have been attempted, either where the image is changed directly [14, 46, 17] or where a generative model is used to project the input image on the training data [21, 33]. A similar method is Randomization, where randomness is added either to the input image [45, 8], or to the model [29, 31, 10]. However both Input Transformation and Randomization have been found to rely on obfuscated gradients to defend against attackers, and can be bypassed by either strong white box attacks such as the CW attack or black box attacks [1, 27].

The Detection method attempts to detect which images are adversarial examples by using an additional classifier. This can either be done preemptively [15], or by using the output of the intermediate layers of the model [34, 26]. The Model Ensemble method similarly uses several classifiers, but instead attempts to aggregate the output of the classifiers into one classification [12]. Liu et al.

combine this idea with Randomization, and shows good results against state of the art attacks [30].

2.2.4 Evaluation of Adversarial Attacks and Defence

Dong et al. propose a method for evaluating adversarial attacks [12]. This method involves evaluating the attacks by using several different perturbation budgets (epsilons). This has the unfortunate effect of increasing the computational cost, thus increasing the testing and attacking time. Nevertheless using several perturbation budgets in the evaluation is necessary for a proper analysis, as some attacks have different accuracy development with changing epsilons (i.e. attack a may have better performance than attack b at epsilon e_0, but worse performance at epsilon e_1).

3 Method

In this chapter we describe the implementation of the training algorithms and the adversarial attacks that were used in our experiments.

After reviewing the relevant litterature, we found the Parallelizable NES algorithm to be particularly interesting because of the varied and unique results achieved with it. Based on this, we formulated our hypothesis:

> Classification models trained with NES may find a larger width of solutions, and some of these models may be more robust against adversarial attacks.

In order to fully test this hypothesis, we needed several fully trained ES models which we could run attacks on. To train these models, we needed to implement a training algorithm that could train models to be sufficiently accurate in a manageable amount of time. Additionally, we considered that ES trained models might have increased resilience against some types of attacks, but not against others. To test this we needed to implement as many different types of adversarial attacks as possible.

3.1 Model

Our goal for the model architecture was to have a relatively simple model that could be trained quickly. At the same time, we also wanted the model to have as high accuracy as possible. The architecture we ended up using was implemented in PyTorch [37], and consists of two convolutional layers that are max-pooled, followed by two dense layers. We also tried modifying this architecture: We tried using three dense layers, using three convolutional layers, and changing the number of output channels on the convolutional layers. In the end, we found that the version in Figure 2 provided the best balance between short training times and good accuracy.

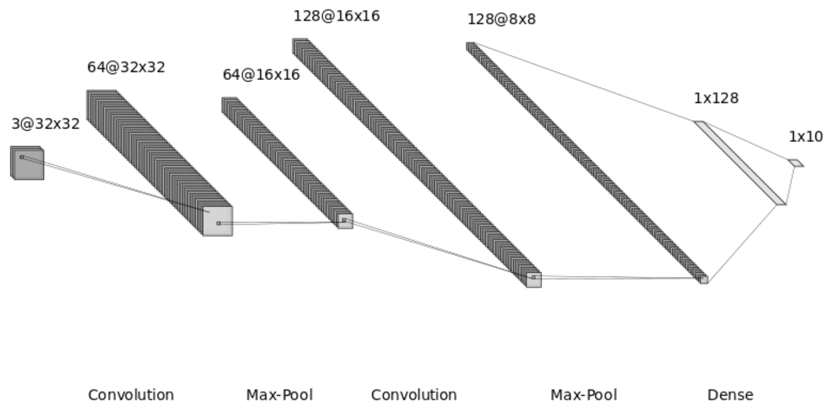

Figure 2: An illustration of our model architecture. In the first convolutional layer, the channels are increased from 3 to 64, the output is then max-pooled. In the second convolutional layer the channels are again increased to 128 and the output is once again max-pooled. Finally, the output from the second max-pool layer passes through two fully connected layers.

3.2 Training Algorithm

For our training algorithm, we considered Parallelizable NES and two CMA-ES variants. Parallelizable NES was our primary focus, and was the first algorithm we implemented. As for the CMA-ES variants, we found that these algorithms were too slow when optimizing a large number of parameters. Therefore, we made the choice to only use Parallelizable NES in our model training.

For Parallelizable NES to work, there needs to be a fitness function which is used to test the different mutations. We attempted two different approaches: One approach was to use loss as a measure of fitness, and attempt to minimize the loss like it is done in Gradient Descent (GD) training. The other approach was to use recognition accuracy as a measure of fitness directly, and attempt to maximize the accuracy. Both fitness measures had to be normalized in order to be usable as weightings for the mutations. We used the following formula where $\mathbf{w}$ are the weightings, $\mathbf{x}$ are fitnesses, $\bar{x}$ is mean fitness and σ is the standard deviation of fitnesses:

$$\mathbf{w} = \frac{\mathbf{x} - \bar{x}}{\sigma}$$

After implementing both approaches, we found that the approach using loss failed to converge. Therefore we applied the approach using accuracy in the training algorithm.

To make training times shorter, the algorithm runs in parallel. This was implemented using an architecture where one master thread communicates with a variable amount of worker threads. Each worker thread makes mutations to the parameter vector using a random seed, and tests the fitness of these mutations. The seed and the fitness are then sent to the master thread. The master thread is used for communication between workers. It collects seeds and fitness data from all the worker threads. Using the fitness that was achieved with the different seeds, it calculates the search gradient for the next iteration, i.e. how much the mutations made by each seed should be weighted. The seeds and the relative weightings are then distributed to each worker. Using this data, each worker can calculate the parameter vector for the next iteration.

For the mutations, a large table of standard normally distributed data is initialized at the start of each training run. When the workers need to fetch mutations in each iteration, they index the table using their random seed, and use the data as the mutation.

The final training algorithm is very similar to Parallelizable NES and looks like this:

Algorithm 1 Training algorithm

```
1: Input: learning rate $\alpha$, exploration rate $\sigma$, parameter vector $x$, fitness
   function $F$, desired fitness $f^\star$
2: while $F(x) \leq f^\star$ do
3:     Create seeds $\delta$ on master, and distribute to workers
4:     for each worker i = 1,...,n do
5:         Fetch mutation $\epsilon_i \sim N(0,1)$ with $\delta_i$
6:         $\mathbf{x}_i^\star \leftarrow \mathbf{x} + \sigma\epsilon_i$
7:         Calculate fitness $f_i \leftarrow F(\mathbf{x}_i^\star)$
8:         Send $\delta_i$ and $f_i$ to master
9:     end for
10:    On master, compute weightings from received fitnesses
11:    $\mathbf{W} \leftarrow \frac{\mathbf{f}-\bar{f}}{\sigma_f}$
12:    Distribute $\mathbf{W}$ to all workers
13:    for each worker i = 1,...,n do
14:        Fetch all mutations $\epsilon$ using all seeds $\delta$ recieved from master
15:        Compute next parameter vector $\mathbf{x} \leftarrow \mathbf{x} + \frac{\alpha}{n\sigma}\sum_{j=1}^{n} W_j\epsilon_j$
16:    end for
17: end while
```

In this algorithm, there are two hyperparameters that we paid special attention to, namely, the learning and exploration rate. These hyperparameters can either be viewed as fixed, and remain constant during an entire training run, or they can change throughout a training run. In our models, we chose to view them as fixed hyperparameters. This choice was made because we had a limited amount of time for training models, and we had to choose between trying several different approaches, or training several models with the same approach. We chose to prioritize training several models with the same approach because we needed to know if our training method would produce consistent or varying results.

The initial weights can also be viewed as a hyperparameter. We decided it would be best to let these be fixed, and to let every model start from the same initial weights. This decision was made so that comparisons between models and training methods can be as fair as possible. [3]

3.3 Training

For training we were given access to part of a CPU cluster hosted by the Department of Computer Science at NTNU. This cluster consists of AMD EPYC 7742 processors in Dell PowerEdge r7525 racks. We were given access to one full processor.

CPU	AMD EPYC 7742: 64 cores, 128 threads, 256MiB l3, 32MiB l2, 4MiB l1
RAM	64GB
Disk	100GB

Table 1: Hardware used to train the ES models.

Our NES trained models were trained on the test set of CIFAR10 [23]. The GD trained model that was compared against these was also trained on the test set. A 10000 image subset of the training set, with a 1000 images of each class, was used for test accuracy evaluation and in attack experiments. The 10000 image test set was used for training, instead of the usual 50000 image training set in order to reduce training time.

All NES training was done with 127 worker threads and one master thread to fully utilize the processing power we had at our disposal.

Our GD trained model was trained on a personal machine, which is described in Table 2. It used Stochastic Gradient Descent as an optimizer. For this model, we decided to stop training prematurely, so the model accuracy would be at a comparable level to the accuracy found in the NES trained models. This decision was made to make the comparisons between the models as fair as possible.

CPU	Intel® Core™ i5-10400 Processor: 6 cores, 12 threads, 12MB intel smart cache
RAM	16GB
Disk	500GB
GPU	Nvidia GeForce RTX 3060 Ti

Table 2: Hardware used to train the GD trained models, and run ARES attacks.

[3] The source code for our training algorithm found here: https://github.com/Evolutionary-strategies/Evolutionary-strategies

In addition, a GD trained model and a NES trained model were trained for observation purposes. In these models all training was logged and converted to graphs, so that the training could be compared over time. Both models were left to train past what we assumed to be their peak, to see how both training methods would adapt to overtraining. The NES trained model was trained using the 10000 image train subset and validated with the test set. The GD trained model was trained with the full training set and validated with the test set. The full training set was chosen for GD training in order to observe what we consider to be a ”normal” GD trained model, this was not done for the NES trained model, so that it would be comparable to the NES trained models we used for attack experiments.

3.4 Adversarial Attacks

Two different libraries were used for the implementation of the adversarial attack algorithms. The decision to use two separate libraries was made because we were unable to find one library that contained all the attacks we wanted to use. Furthermore, comparing the results from two different libraries would serve as a way to validate our results.

One of the libraries we used was Foolbox. The reason for choosing this particular library was its easy integration with PyTorch [37], and its wide selection of attacks. It is also well suited for use within academic research, with its own paper dedicated to it [39]. Foolbox is also written using EagerPy [38] as a base, making it both high performing and modular, a sought after feature in case we needed to change from PyTorch to another library. Our implementation uses Python´s multiprocessor library alongside Foolbox to enable multiple attacks simultaneously. After the attacks have been carried out, the results are printed to a table, and plotted to a graph, with Matplotlib [18].

The other library we used was ARES. ARES is the library developed and used for the benchmarking done by Dong et al. [12]. It contains multiple attacks missing in the Foolbox library, in particular score-based attacks. It also offers greater customizability than Foolbox, for example allowing us to use the objective function described by Carlini and Wagner as our loss function instead of cross entropy loss in gradient-based attacks [7].

For the attack hyperparameters and configuration we attempted to stay as close as possible to what was recommended in the original papers. In some attacks we had to make small changes to the configuration in order to reduce computation time. This was necessary due to our time constraints. We arrived at these changes by running test attacks on our GD trained model, and seeing how long they took. The changes made were reducing the number of iterations allowed in the CW attack, the SPSA attack, the NES attack and NAttack. In our gradient-based attacks, we used both CW loss and CE loss if these were available, and only one of these otherwise. The attacks where both CE loss and CW loss were used are FGSM, BIM, MIM and PGD. A full list of the attacks we performed can be found below in Table 3, and in Appendix B, where the hyperparameters of each attack are also described.

Attack	L_p-Distance measure	Knowledge	Library
FGSM	L_∞ and L_2	White	ARES and Foolbox
BIM	L_∞ and L_2	White	ARES and Foolbox
PGD	L_∞ and L_2	White	ARES and Foolbox
DeepFool	L_∞ and L_2	White	ARES and Foolbox
MIM	L_∞ and L_2	White	ARES
NewtonFool	L_2	White	Foolbox
CW	L_2	White	ARES
SPSA	L_∞ and L_2	Score	ARES
NES	L_∞ and L_2	Score	ARES
NAttack	L_∞	Score	ARES
AUN	L_∞	Decision	Foolbox
AGN	L_2	Decision	Foolbox
CR	L_2	Decision	Foolbox
S&P	L_2	Decision	Foolbox

Table 3: List of all attacks used for our research, with corresponding distance measure, type and library.

To run the attack experiments we used our personal computers. Their specifications can be found in Tables 2 and 4.

CPU	AMD FX 6300 3.5GHz
RAM	8GB DDR3
Disk	300GB
GPU	AMD Radeon R9 280

Table 4: Hardware used to run Foolbox attacks.

To evaluate the results of the adversarial attacks, we used the method described by Dong et al. [12] where several perturbation budgets are used. Due to time constraints, we chose only a limited number of epsilon values when testing the models. The chosen epsilons e differ between attacks and distance measures. Additionally a different number of epsilons were chosen for each attack depending, on how computationally expensive the attack was. The epsilon values themselves were determined by running test attacks with a smaller subset of images, and finding which epsilon resulted in a significant drop in accuracy. The epsilons that were used for each attack can be found in Table 5.

Attack(s)	Perturbation Budgets
L_∞-based FGSM, BIM, MIM, PGD, DeepFool	0.005, 0.01, 0.02
L_2-based FGSM, BIM, MIM, PGD, DeepFool	0.3, 0.5, 1.0
L_∞-based NES	0.01, 0.03
L_2-based NES	0.7, 1.0, 1.3
L_∞-based NAttack, SPSA	0.01, 0.02. 0.03
L_2-based SPSA	0.5, 0.7, 1.0
L_2-based NewtonFool	0.0001, 0.001, 0.005, 0.1, 0.3, 0.5
L_∞-based AUN	0.005, 0.01, 0.02, 0.1, 0.3, 0.5, 0.8, 1.0
L_2-based CR and AGN	0.3, 0.5, 1.0, 3.0, 5.0, 8.0, 10.0, 13.0, 15.0, 18.0, 20.0
L_2-based S&P	0.001, 0.01, 0.1, 0.5, 1.0, 3.0, 5.0

Table 5: Perturbation budgets used for different attacks.

For our result graphs, we normalized the data with the following function:

$$\alpha_i = \frac{y_i}{y_0} * 100$$

where α_i is the normalized accuracy, y_i is the current data point, or accuracy value, and y_0 is the starting accuracy of the model. This allowed us to look at model accuracy as a percentage, and visualize the attack results as a relative drop in accuracy, instead of looking at the start accuracy and the absolute accuracy drop. We chose to do this because our models had different starting accuracies, which would have made the result graphs harder to read. [4]

4 Results

4.1 Models

The models we present in Table 6 were all trained on the test set and validated/attacked with a subset of the training set. In order to differentiate between the models, we have assigned them names or model numbers which we we will use to refer to them hereafter.

Model number	Accuracy	Training time	Sigma	Learning rate	Running loss
Baseline gradient descent model	54.48 %	30 minutes	-	0.001	1.269
1	54.79 %	18 days	0.1	0.01	50348.1
2	54.93 %	10 days	0.15	0.01	6445.6
3	54.50 %	5 days	0.15	0.01	4789.3
4	51.04 %	8 days	0.15	0.01	11047.8

Table 6: Our models and their achieved accuracy, training time, training parameters and running loss.

During model training, we observed that models trained with Natural Evolution Strategies (NES) do not seem to reach a peak validation accuracy in the same way Gradient Descent (GD) models do. In GD training, the model peaks at a max validation accuracy, then the validation accuracy starts to drop off. However, in NES training the validation accuracy does not necessarily seem to reach a peak, instead reaching a point where the model continues to improve at a very diminished rate. Another thing to take note of is that all our NES trained models seemed to reach this point at roughly the same accuracy. The training graphs of a GD trained model and a NES trained model can be seen in Figure 3.

[4]The source code for our attacks be found here: https://github.com/Evolutionary-strategies/Evolutionary-strategies and here: https://github.com/Evolutionary-strategies/ares

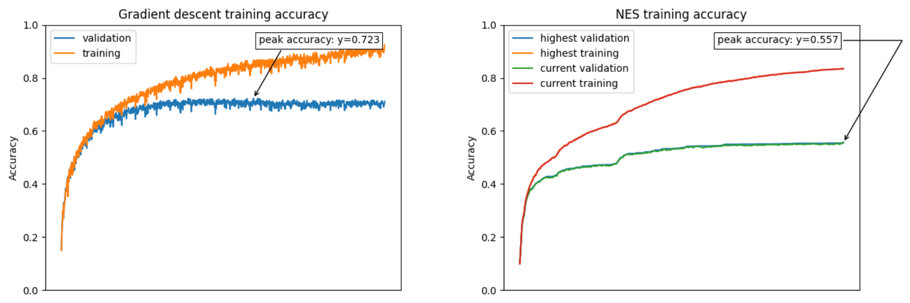

Figure 3: Graphs showing training accuracy over time with GD training and NES training. We see that the GD trained model has a more volatile accuracy graph under training, and that it reaches a peak accuracy. On the other hand, the NES trained model has a very smooth accuracy graph, and does not seem to reach a clear peak.

We also observed that models trained with NES and with accuracy as a fitness function have a very high running loss when compared to GD trained models.

4.2 Attacks

To make this section more readable we have chosen to only show attacks with one perturbation budget in the tables below. For some attacks, we have also included a graph that shows several different perturbation budgets and the corresponding model accuracies.

The perturbation budgets for the tables are shown in Table 7:

L_∞-based White box	0.02
L_2-based White box	0.5
L_∞-based Black box score	0.03
L_2-based Black box score	1.0

Table 7: Perturbation budgets used for Table 8, 9 and 11.

Attack		Model number				
Attack category	Attack name	Baseline gradient descent model	1	2	3	4
ARES White L_∞ CE loss	FGSM	20.59 %	54.75 %	54.66 %	54.15 %	50.93 %
	BIM	18.26 %	54.75 %	54.66 %	54.15 %	50.93 %
	MIM	18.87 %	54.75 %	54.66 %	54.15 %	50.93 %
	PGD	25.72 %	54.72 %	54.81 %	54.32 %	50.72 %
ARES White L_∞ CW loss	FGSM	19.64 %	0.05 %	1.66 %	1.98 %	1.60 %
	BIM	17.48 %	0.01 %	0.44 %	0.69 %	0.44 %
	MIM	17.93 %	0.01 %	0.5 %	0.79 %	0.49 %
	PGD	24.05 %	0.03 %	1.64 %	2.09 %	1.37 %
Foolbox White L_∞	FGSM	20.6 %	54.77 %	54.68 %	54.16 %	50.95 %
	BIM	17.44 %	54.77 %	54.68 %	54.16 %	50.95 %
	PGD	21.06 %	54.78 %	54.79 %	53.98 %	50.84 %
	DeepFool	17.2 %	0.01 %	0.64 %	0.86 %	0.64 %
ARES Black Score L_∞	SPSA	28.13 %	0.59 %	37.53 %	31.17 %	39.51 %
	NES	35.40 %	3.53 %	13.52 %	14.10 %	11.84 %
	NAttack	19.90 %	39.41 %	27.50 %	27.50 %	27.50 %

Table 8: Attack results for L_∞-based attacks, where each row is an attack, and each column is one of our models.

Table 8, and Figures 4 and 5 show our L_∞-based attack data. Here we found that gradient-based attacks using CE loss and L_∞ as a distance measure do not seem to work against NES trained models. This finding is consistent in both our ARES and Foolbox-based attacks. However, the same attacks using CW loss are more effective on NES trained models than on the GD trained model. We also found that NES trained models seem more resilient against NAttack than the GD trained model. On the other hand the GD trained model seems more resilient against the SPSA attack and NES attack than the NES trained models.

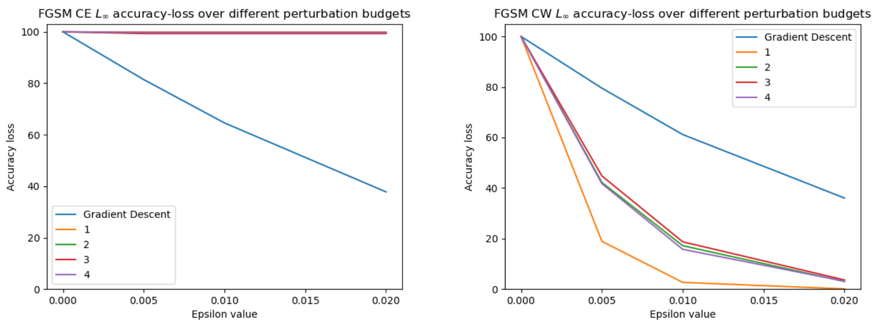

Figure 4: L_∞ ARES FGSM attack data. Left: FGSM with CE loss. Right: FGSM with CW loss. The BIM, MIM, and PGD attacks, show very similar results with both CE loss and CW loss.

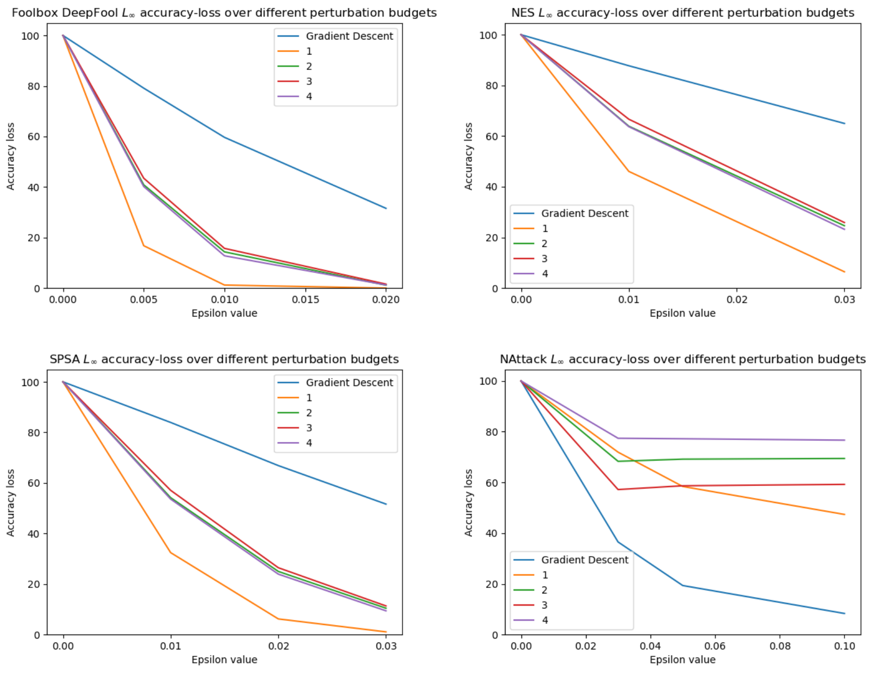

Figure 5: L_∞ Foolbox Deepfool and ARES black box score-based attack data.

Attack		Model number				
Attack category	Attack name	Baseline gradient descent model	1	2	3	4
ARES White L_2 CW loss	FGSM	30.78 %	4.05 %	8.10 %	8.89 %	6.80 %
	BIM	30.75 %	10.00 %	11.48%	11.78 %	10.96 %
	MIM	30.98 %	10.02 %	11.56 %	11.92 %	11.03 %
	PGD	31.27 %	10.02 %	11.68 %	12.08 %	11.10 %
Foolbox White L_2	FGSM	27.46 %	54.77 %	54.77 %	54.38 %	51.01 %
	BIM	25.5 %	54.77 %	54.74 %	54.36 %	51.01 %
	PGD	29.84 %	54.69 %	54.67 %	54.55 %	51.08 %
	DeepFool	24.45 %	0.07 %	2.61 %	3.2 %	2.21 %
ARES Black Score L_2	SPSA	33.01 %	1.85 %	10.55 %	11.11 %	9.03 %
	NES	45.33 %	18.14 %	29.67 %	30.90 %	27.45 %

Table 9: Attack results for L_2-based attacks, where each row is an attack, and each column is one of our models.

Table 9 shows our L_2-based attack data. We found that the CE-based Foolbox versions of FGSM, BIM and PGD failed against the NES trained models. Against the CW-based ARES FGSM, BIM, MIM and PGD, the NES trained models perform significantly worse than the GD trained model. We also found that NES trained models perform significantly worse against L_2-based NES and SPSA attacks.

Attack			Model number				
Distance measurement	Attack name	Epsilon	Baseline gradient descent model	1	2	3	4
L_∞	AUN	0.5	17.49 %	16.74 %	22.81 %	24.88 %	19.92 %
L_2	AGN	10.0	34.17 %	25.87 %	37.72 %	37.84 %	33.71 %
	CR	10.0	20.37 %	25.06 %	21.81 %	23.03 %	22.51 %
	S&P	3.0	33.28 %	5.57 %	14.4 %	17.02 %	14.63 %

Table 10: Attack results for noise or contrast-based attacks, where each row is an attack, and each column is one of our models.

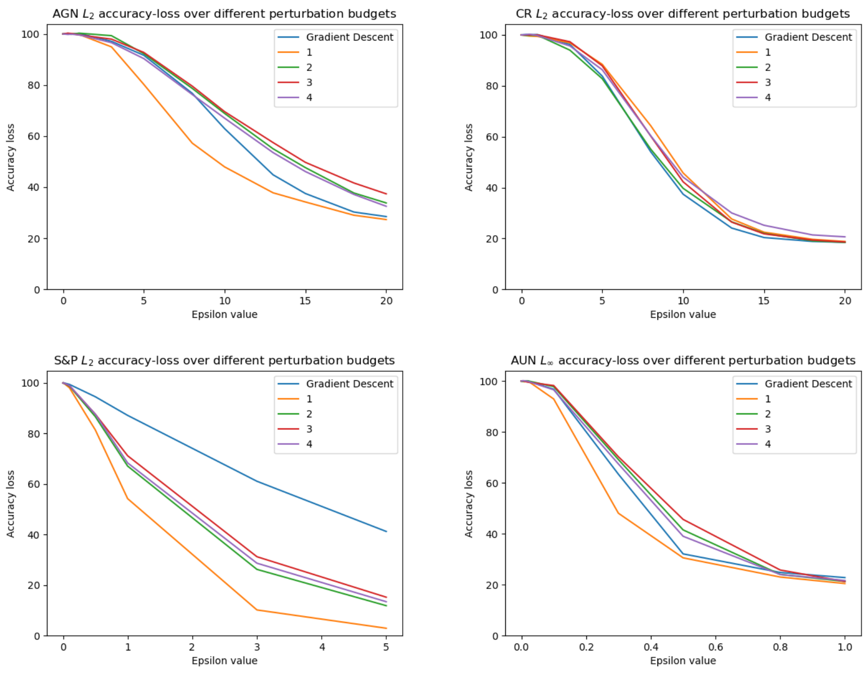

Figure 6: Attack data for noise or contrast based attacks. Upper left: L_2 Additive Gaussian Noise Attack. Upper right: L_2 Contrast Reduction Attack data. Lower left: L_2 Salt and Pepper Noise Attack. Lower right: L_∞ Additive Uniform Noise Attack.

Table 10 and Figure 6, show the results from the noise or contrast-based attacks. With AUN, AGN and CR we found that a very high epsilon value is required to make any noticeable change to the model accuracy. We also found that GD and NES trained models show similar resilience against these attacks. Against the S&P attack, NES trained models performed worse than GD trained models.

Attack		Model number				
Attack category	Attack name	Baseline gradient descent model	1	2	3	4
ARES White L_2 CE loss	FGSM	27.45 %	6.22 %	6.26 %	5.92 %	6.85 %
	BIM	25.20 %	6.22 %	6.26 %	5.92 %	6.85 %
	MIM	25.78 %	6.22 %	6.26 %	5.92 %	6.85 %
	PGD	25.79 %	6.17 %	6.25 %	5.97 %	6.83 %
ARES White L_2	Deepfool	5.71 %	11.21 %	12.65 %	15.15 %	11.98 %
	CW	0 %	0 %	0 %	0 %	0 %
Foolbox White L_2	NewtonFool	27.84 %	10.0 %	10.03 %	10.0 %	9.98 %

Table 11: Attack results for L_2-based attacks where epsilon was not used to measure attack strength, or where the attacks gave strange results. Each row is an attack, and each column is one of our models.

In Tables 8, 9, 10 and 11, we find that NES trained model 1 seems to achieve a lower accuracy compared with the other NES trained models against white box attacks using CW loss, all DeepFool variants and most black box attacks.

Table 11 shows attack data from attacks that could not be grouped together with the attacks listed in Tables 8, 10 and 9. In the case of the ARES CW attack and DeepFool attack, the attack strength was measured using a maximum number of iterations instead of an epsilon value. The results from the CW attack show that it completely bypassed the defences of both GD and NES trained models. In the case of the DeepFool attack, we found that the ARES and Foolbox implementations achieve different results. In the ARES-based Deepfool attack the NES trained models perform better than the GD trained model, in the Foolbox version the opposite happens.

Furthermore, Table 11 shows that the NewtonFool attack and the ARES L_2-based FGSM, BIM, MIM and PGD attacks seem to be very effective against NES trained models. However if we look at how these attacks behave over several epsilons, and look at the generated adversarial images we find that these results are misleading. As shown in Figure 7, the NES trained models achieve identical accuracies, no matter what value epsilon has. In Figures 8 and 9 we also see that when the NES trained models are attacked, the adversarial image becomes completely black.

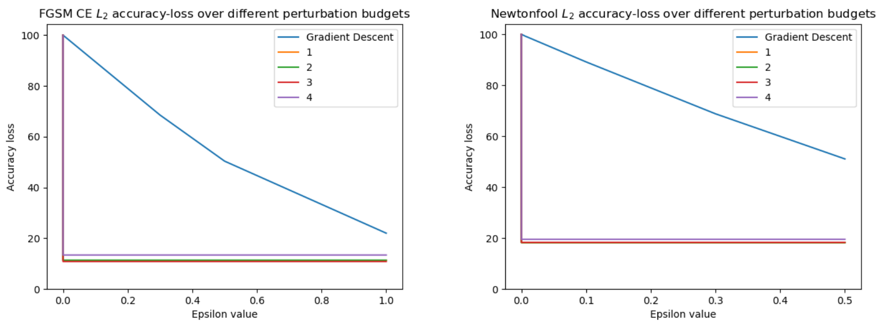

Figure 7: Results from the CE-based L_2 ARES FGSM and Foolbox NewtonFool attacks. We see that the NES trained models have the same accuracy no matter how high the epsilon value is. ARES CE-based L_2 BIM, MIM and PGD attacks show very similar results.

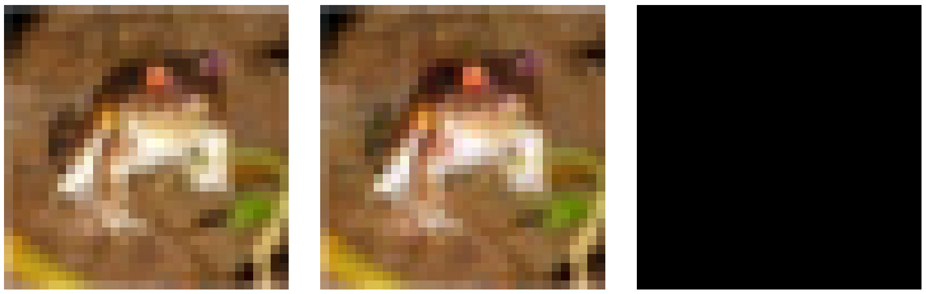

Figure 8: The resulting adversarial examples from running ARES L_2-based FGSM at 1.0 epsilon against GD and NES trained models. Left: the original image. Middle: the adversarial example from the attack on the GD trained model. Right: the adversarial example from the attack on the NES trained model. As we can see, the adversarial example for the NES trained model is completely black. This is likely because the obfuscated gradients in the NES trained models cause the attack to behave strangely.

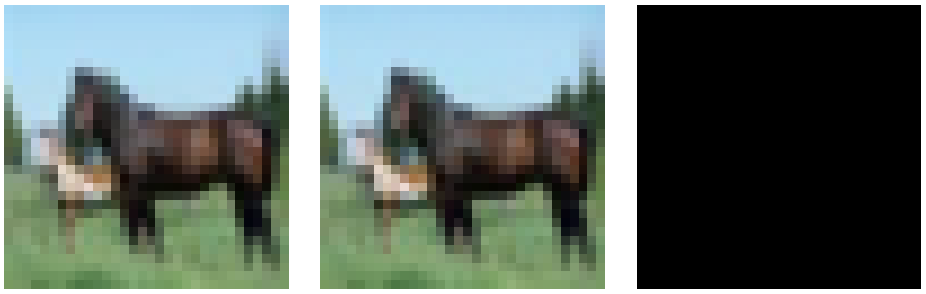

Figure 9: Adversarial examples produced by Newtonfool with an epsilon value of 0.3. Left: The original image. Middle: adversarial example generated against the GD trained model. Right: adversarial example generated against the NES trained model. Once again, we see that the adversarial example for the NES trained model is completely black.

5 Discussion

5.1 Training

One obvious drawback with using Natural evolution strategies (NES) to train CNNs is the far greater time required to train the neural network to achieve a workable accuracy, compared to Gradient Descent (GD) training. Unfortunately, the information we have gathered about training times is not exact, but our results point in the direction that the NES trained model with the most similar accuracy to the GD trained model has a training time that is greater by several orders of magnitude. This estimation does not take into consideration the different hardware used to train the different models. As GD training is done on GPU and NES training is done on CPU, it is difficult to make a direct comparison. If a comparison could be made with two equally powerful systems, we suspect that the training time difference would be even greater.

Another drawback with NES is the low accuracy. Despite the long training time, all the NES trained models seemed to reach 51-55% accuracy. It is possible that, if allowed to train long enough, the accuracy may be higher. Another approach to raising the accuracy would be adding things like weight decay or experimenting with a variable exploration rate. However, we leave this to further research, as it was not possible to test within our timeframe. The GD trained model we used in our experiments was prematurely stopped in order to have a similar accuracy to the NES trained models. If the GD trained model had not been stopped, it would likely have had an accuracy of around 72%, similar to the model displayed in Figure 3.

The finding that all NES trained models reached a similar accuracy may indicate that NES training achieves somewhat consistent results. On the other hand, our results showed that the different NES trained models had varying performance against adversarial attacks. This is similar to the results found by Salimans et al., in that it shows that NES training can find several different solutions to the same optimization problem [40].

Another interesting observation is that the validation accuracy of NES trained model did not reach a clear peak, in the way the GD trained model did. We surmise that this may be similar to the findings by Jacobsen and Dalheim during their training of cellular automata models, where their models would stop making changes to the generated image after it was completed [20]. This could imply that using NES to train CNN models achieves more stable results with prolonged exposure to the data, compared to GD training.

5.2 Attacks

5.2.1 White Box Attacks

The results from the L_∞ and Foolbox L_2-based FGSM, BIM, MIM and PGD attacks with CE loss, indicate that these attacks have little to no effect on NES trained models. One likely reason for this is that accuracy maximization was used for our training fitness function, instead of loss minimization. The resulting high loss from this type of training may cause the CE loss gradients to be obfuscated. Obfuscated gradients may cause the attacks to become confused, and add random or inadequate perturbation to the image. If this is the case, it is similar to what was found for the Input Transformation and Randomization defence methods [1].

The ARES L_2-based FGSM, BIM, MIM and PGD attacks with CE loss, as well as the Foolbox NewtonFool attack had some unexpected results. When attacking the GD trained model, the attacks worked well, but against the NES trained models they produced adversarial examples that were completely black, instead of producing images similar to the originals. This happened no matter what epsilon was used, and always happened to a similar number of images, resulting in a similar accuracy, regardless of the supposed attack strength. All of these attacks are gradient-based and use CE loss. We believe that what happened here was similar to what happened with the L_∞-based versions of the same attacks, where the attack became confused because of the obfuscated loss gradient. In this case, instead of adding inadequate perturbation to the image, the attack showed unexpected behaviour, and completely blackened the image to make sure it is misclassified. We can not say for certain that obfuscated gradients are the cause of the unexpected behaviour, and even if they are, the images being blackened regardless of the epsilon value is not something that should happen. We leave it to further work to find out what caused these attacks to behave like this.

In all white box attacks using CW loss instead of CE loss, the NES trained models performed significantly worse than the GD trained model. Attacks using CW loss performing better than attacks using CE loss is an expected result, as the CW objective functions have previously been shown to be superior to CE loss, and to work well despite the CE loss gradients being obfuscated [7].

The finding that using CW loss bypasses the defences of NES trained models, in the same way that it bypasses Input Transformation and Randomization based defences is also interesting. This finding strengthens the implication that the CE-based L_∞ attacks fail against NES trained models because the CE loss gradients are obfuscated.

The results from our DeepFool attack experiments show an interesting discrepancy. In the L_2-based ARES DeepFool attack our findings indicate that the NES trained models are more robust against DeepFool than the GD trained models. On the other hand, our findings from L_2-based and L_∞-based Foolbox DeepFool indicate the reverse. This discrepancy may be caused either by implementation differences within the libraries, or within our code. A potential reason may be that different hyperparameters were used to control the attack strength in the Foolbox and ARES versions of DeepFool. In the Foolbox version, an epsilon value was used as the perturbation budget, and several epsilons were tested. In the ARES version, no epsilon value was specified, instead using a max number of iterations as a way to control the attack strength. Regardless of what the cause is, we have chosen to view the results from our DeepFool experiments as inconclusive. We leave it to further research to do more thorough experimentation with DeepFool attacks on NES trained models. We believe that these results support the findings of Dong et al., showing the importance of evaluating attacks using multiple epsilons and different hyperparameters [12].

5.2.2 Black Box Attacks

In our score-based attack experiments, the NES trained models performed worse than the GD trained model against the SPSA and NES attacks. This suggests that attacks using gradient estimation perform well against NES trained models.

Against the NAttack, the NES trained models performed better than the GD trained model (As we mentioned earlier, the NAttack does not attempt to find the optimal adversarial perturbation). This finding may indicate that NES trained models perform better against sub-optimal adversarial examples, such as the ones produced by the NAttack. The cause of this is something we leave to further research, as we have been unable to determine it.

The NES trained models and the GD trained models performed quite similarly against all noise- and contrast-based attacks except S&P. Against the S&P attack, the NES trained models all performed significantly worse than the GD trained models. These findings suggests that NES trained models do not generalize better than GD trained models in noise or contrast induced environments. In addition, our results suggest that the S&P attack is a more powerful attack than CR and AGN, causing a greater accuracy loss at the same perturbation budget.

The high perturbation budget required to successfully attack models using contrast-based attacks confirms the findings by Dodge and Karam [11]. However, in our experiments, the CR attack seems to be stronger than the AGN attack, albeit only marginally, which is in conflict with the results of Dodge and Karam [11]. Further research is needed before concluding which attack is stronger.

Model 1 performed worse than model 2-4 against white box attacks using CW loss, all DeepFool variants and most black box attacks. Model 1 has certain characteristics that the other NES trained models do not. One of these characteristics is that it uses a different exploration rate, which may lead the training to a different solution, thus making the model less robust to noise attacks. Another potential reason is that it has a higher running loss, and a longer training time compared to the other models. We believe either of these could be the reason for the lower performance, but we can not be certain. For future research, we suggest training NES models with different hyperparameters and training durations.

5.3 Key Findings

- NES training is slower than GD training and produces models with lower accuracy.
- NES training produces models with a consistent accuracy, but with varying resilience against adversarial attacks.
- NES training shows increased stability under prolonged training exposure when compared to GD training.
- NES trained models show strong resilience against Gradient based attacks that use CE loss. This is likely due to obfuscated gradients.
- Some attacks showed unexpected behavior, such as blackening the input image, when run against NES trained models.
- If CW loss was used instead of CE loss for gradient based attacks, NES trained models performed worse than GD trained models.
- NES trained models perform worse than GD trained models against the NES attack and the SPSA attack.
- NES trained models perform better than GD trained models against NAttack.
- NES trained models perform similarly or worse than GD trained models against noise or contrast based attacks.

6 Conclusion

NES trained models show interesting properties during training, such as more stability over prolonged time than GD trained models. Against adversarial attacks, NES trained models are successful in defending against most gradient-based attacks. This is likely because of obfuscated CE loss gradients.

Against the NAttack, NES trained models also show increased resilience. The same robustness is not seen against attacks that can avoid using the CE loss to calculate a gradient, either by using another objective function as a loss function, or by estimating the gradient through other means. NES trained models also do not show increased resilience against noise or contrast induced images. Given that NES trained models do not defend well against all attacks, NES training can not be considered a reliable method of defending against adversarial attacks.

6.1 Further Work

During this project we were not able to test all the approaches we wanted to test during model training, and we were not able to find the cause of some of the results we had. We believe this shows the need for further research where classifier models are trained with NES and adversarial attacks are run against them.

One line of further work would be to test more approaches for model training. We suggest that using weight decay, variable learning rate or variable exploration rate might yield different results, both when it comes to the accuracy of the resulting models and their resilience against adversarial attacks. As we observed that models show stability with prolonged exposure to NES training, it would also be interesting to train a model with NES for a longer duration of time.

Another line of further work is to continue running attacks against NES trained models, to clarify or explain our results in cases where the results were unclear, or could not be explained. We did not have time to find the cause of the image blackening behaviour exhibited by the ARES FGSM, BIM, MIM, PGD and FoolBox NewtonFool attacks with CE loss and L_2 distance measure. We believe the easiest way to research this would be to attempt to reproduce the image blackening, and analyze why it happens. It would also be interesting to run the attacks that showed unexpected behaviour against models trained with Randomization and Input Transformation defences, to see if the same image blackening behaviour is found there, as these defences also make use of obfuscated gradients.

Our DeepFool experiments gave inconsistent results. We believe that running more DeepFool experiments against NES trained models would be sufficient to produce clearer results. These experiments would ideally, experiment more with hyperparameter changes, and use a larger range of perturbation budgets, both in terms of epsilon values and max iterations.

In our NAttack results, we found that NES trained models performed better than GD trained models, but we were unable to determine why this was the case. It would be useful to run more experiments with the NAttack against NES trained models, and analyze the results more thoroughly, to determine the reason for the increased resilience seen in NES trained models.

References

[1] A. Athalye, N. Carlini, and D. Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples, 2018.

[2] G. S. Bennabhaktula, J. Antonisse, and G. Azzopardi. On improving generalization of cnn-based image classification with delineation maps using the corf push-pull inhibition operator. In *International Conference on Computer Analysis of Images and Patterns*, pages 434–444. Springer, 2021.

[3] H. Beyer. Evolution strategies. *Scholarpedia*, 2(8):1965, 2007. revision #193589.

[4] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Šrndić, P. Laskov, G. Giacinto, and F. Roli. Evasion attacks against machine learning at test time. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 387–402. Springer, 2013.

[5] A. Brock, S. De, S. L. Smith, and K. Simonyan. High-performance large-scale image recognition without normalization, 2021.

[6] T. B. Brown, D. Mané, A. Roy, M. Abadi, and J. Gilmer. Adversarial patch, 2017.

[7] N. Carlini and D. Wagner. Towards evaluating the robustness of neural networks, 2016.

[8] J. M. Cohen, E. Rosenfeld, and J. Z. Kolter. Certified adversarial robustness via randomized smoothing, 2019.

[9] G. B. P. da Costa, W. A. Contato, T. S. Nazare, J. E. Neto, and M. Ponti. An empirical study on the effects of different types of noise in image classification tasks. *arXiv preprint arXiv:1609.02781*, 2016.

[10] G. S. Dhillon, K. Azizzadenesheli, Z. C. Lipton, J. Bernstein, J. Kossaifi, A. Khanna, and A. Anandkumar. Stochastic activation pruning for robust adversarial defense, 2018.

[11] S. Dodge and L. Karam. Understanding how image quality affects deep neural networks. In *2016 eighth international conference on quality of multimedia experience (QoMEX)*, pages 1–6. IEEE, 2016.

[12] Y. Dong, Q.-A. Fu, X. Yang, T. Pang, H. Su, Z. Xiao, and J. Zhu. Benchmarking adversarial robustness on image classification. In

Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 321–331, 2020.

[13] Y. Dong, F. Liao, T. Pang, H. Su, J. Zhu, X. Hu, and J. Li. Boosting adversarial attacks with momentum, 2017.

[14] G. K. Dziugaite, Z. Ghahramani, and D. M. Roy. A study of the effect of jpg compression on adversarial images, 2016.

[15] Z. Gong, W. Wang, and W.-S. Ku. Adversarial and clean data are not twins, 2017.

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

[17] C. Guo, M. Rana, M. Cisse, and L. van der Maaten. Countering adversarial images using input transformations, 2017.

[18] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[19] A. Ilyas, L. Engstrom, A. Athalye, and J. Lin. Black-box adversarial attacks with limited queries and information, 2018.

[20] J. B. Jacobsen and W. Dalheim. Regeneration and generalization of cellular automata through evolution strategies, 2021.

[21] A. Jalal, A. Ilyas, C. Daskalakis, and A. G. Dimakis. The robust manifold defense: Adversarial training using generative models, 2017.

[22] U. Jang, X. Wu, and S. Jha. Objective metrics and gradient descent algorithms for adversarial examples in machine learning. In *Proceedings of the 33rd Annual Computer Security Applications Conference*, pages 262–277, 2017.

[23] A. Krizhevsky, V. Nair, and G. Hinton. Cifar-10 (canadian institute for advanced research).

[24] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[25] A. Kurakin, I. Goodfellow, S. Bengio, et al. Adversarial examples in the physical world, 2016.

[26] X. Li and F. Li. Adversarial examples detection in deep networks with convolutional filter statistics, 2016.

[27] Y. Li, M. Cheng, C.-J. Hsieh, and T. C. M. Lee. A review of adversarial attack and defense for classification methods. *The American Statistician*, pages 1–17, jan 2022.

[28] Y. Li, L. Li, L. Wang, T. Zhang, and B. Gong. Nattack: Learning the distributions of adversarial examples for an improved black-box attack on deep neural networks, 2019.

[29] X. Liu, M. Cheng, H. Zhang, and C.-J. Hsieh. Towards robust neural networks via random self-ensemble, 2017.

[30] X. Liu, M. Cheng, H. Zhang, and C.-J. Hsieh. Towards robust neural networks via random self-ensemble, 2017.

[31] X. Liu, Y. Li, C. Wu, and C.-J. Hsieh. Adv-bnn: Improved adversarial defense through robust bayesian neural network, 2018.

[32] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. Towards deep learning models resistant to adversarial attacks, 2017.

[33] D. Meng and H. Chen. Magnet: a two-pronged defense against adversarial examples, 2017.

[34] J. H. Metzen, T. Genewein, V. Fischer, and B. Bischoff. On detecting adversarial perturbations, 2017.

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

[36] T. S. Nazaré, G. B. Costa, W. A. Contato, and M. Ponti. Deep convolutional neural networks and noisy images. In *Iberoamerican Congress on Pattern Recognition*, pages 416–424. Springer, 2017.

[37] 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 H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[38] J. Rauber, M. Bethge, and W. Brendel. EagerPy: Writing code that works natively with PyTorch, TensorFlow, JAX, and NumPy. *arXiv preprint arXiv:2008.04175*, 2020.

[39] J. Rauber, W. Brendel, and M. Bethge. Foolbox: A python toolbox to benchmark the robustness of machine learning models. *arXiv preprint arXiv:1707.04131*, 2017.

[40] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever. Evolution strategies as a scalable alternative to reinforcement learning, 2017.

[41] J. Spall. Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. *IEEE Transactions on Automatic Control*, 37(3):332–341, 1992.

[42] 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.

[43] J. Uesato, B. O'Donoghue, A. v. d. Oord, and P. Kohli. Adversarial risk and the dangers of evaluating against weak attacks, 2018.

[44] D. Wierstra, T. Schaul, J. Peters, and J. Schmidhuber. Natural evolution strategies. pages 3381–3387, 06 2008.

[45] C. Xie, J. Wang, Z. Zhang, Z. Ren, and A. Yuille. Mitigating adversarial effects through randomization, 2017.

[46] W. Xu, D. Evans, and Y. Qi. Feature squeezing: Detecting adversarial examples in deep neural networks. In *Proceedings 2018 Network and Distributed System Security Symposium*. Internet Society, 2018.

[47] H. Zhang, Y. Yu, J. Jiao, E. P. Xing, L. E. Ghaoui, and M. I. Jordan. Theoretically principled trade-off between robustness and accuracy, 2019.

7 Appendix A: Attack hyperparameter configurations

This appendix shows the hyperparameter configurations we have used for the different attacks. For the attacks where distance measure is not mentioned, the same configuration was used for both versions of the attack. If an attack is not mentioned, the only hyperparameters used for it are: epsilon, distance norm and/or loss function. Loss function and epsilon values are not included in this appendix, as they are discussed in the main report.

7.1 Foolbox

Foolbox hyperparameter configurations		
Attack	Hyperparameter	
FGSM	relative stepsize	1.0
	absolute stepsize	None
	steps	1
	random start	False
BIM	relative stepsize	0.2
	absolute stepsize	None
	steps	10
	random start	False
DeepFool	steps	50
	candidates	10
	overshoot	0.02
	loss	logits
L_∞ PGD	relative stepsize	0.03333333333333333
	absolute stepsize	None
	steps	40
	random start	True
L_2 PGD	relative stepsize	0.025
	absolute stepsize	None
	steps	50
	random start	True
Newtonfool	steps	100
	stepsize	0.01
Contrast Reduction Attack	target	0.5
Salt and Pepper Noise Attack	steps	1000
	across channels	True
	channel axis	None

7.2 ARES

ARES hyperparameter configurations		
Attack name	Hyperparameters	Values
L_∞ BIM	steps	100
	stepsize	epsilon/steps
L_∞ MIM	steps	100
	stepsize	epsilon/steps
	decay factor	1.0
L_∞ PGD	steps	100
	stepsize	epsilon/steps
L_2 BIM	steps	100
	stepsize	2.5*epsilon/steps
L_2 MIM	steps	100
	stepsize	2.5*epsilon/steps
	decay factor	1.0
L_2 PGD	steps	100
	stepsize	2.5*epsilon/steps
DeepFool	overshoot	0.2
	max iterations	50
CW	learning rate	0.2
	kappa	0
	c	0.01
	binary search steps	4
	max iterations	50
NES	samples	10
	samples per draw	20
	max queries	2000
	stepsize	epsilon/100
SPSA	learning rate	0.01
	delta	0.01
	samples	10
	samples per draw	20
	max queries	2000
NAttack	learning rate	0.02
	sigma	0.1
	sample size	100
	max queries	1000

8 Appendix B: Full experiment data

This appendix shows the raw data from our experiments.

8.1 ARES

8.1.1 White Box

FGSM with cw loss and L_∞ distance measure			
Model number	epsilon 0.005	epsilon 0.01	epsilon 0.02
Baseline GD trained model	43.33 %	33.33 %	19.64 %
1	10.36 %	1.46 %	0.05 %
2	23.19 %	9.44 %	1.66 %
3	24.42 %	10.20 %	1.98 %
4	21.31 %	8.04 %	1.60 %

FGSM with ce loss and L_∞ distance measure			
Model number	epsilon 0.005	epsilon 0.01	epsilon 0.02
Baseline GD trained model	44.37 %	35.19 %	20.59 %
1	54.75 %	54.75 %	54.75 %
2	54.66 %	54.66 %	54.66 %
3	54.15 %	54.15 %	54.15 %
4	50.93 %	50.93 %	50.93 %

FGSM with cw loss and L_2 distance measure		
Model number	epsilon 0.3	epsilon 0.5
Baseline GD trained model	40.17 %	30.78 %
1	6.43 %	4.05 %
2	16.68 %	8.10 %
3	17.59 %	8.89 %
4	14.59 %	6.80 %

BIM with cw loss and L_∞ distance measure			
Model number	epsilon 0.005	epsilon 0.01	epsilon 0.02
Baseline GD trained model	43.09 %	32.46 %	17.48 %
1	7.95 %	0.44 %	0.01 %
2	21.55 %	6.82 %	0.44 %
3	22.65 %	7.30 %	0.69 %
4	19.64 %	5.57 %	0.44 %

BIM with ce loss and L_∞ distance measure			
Model number	epsilon 0.005	epsilon 0.01	epsilon 0.02
Baseline GD trained model	44.17 %	34.31 %	18.26 %
1	54.75 %	54.75 %	54.75 %
2	54.66 %	54.66 %	54.66 %
3	54.15 %	54.15 %	54.15 %
4	50.93 %	50.93 %	50.93 %

BIM with cw loss and L_2 distance measure		
Model number	epsilon 0.3	epsilon 0.5
Baseline GD trained model	40.71 %	30.75 %
1	10.63 %	10.00 %
2	18.16 %	11.48 %
3	18.73 %	11.78 %
4	16.25 %	10.96 %

MIM with cw loss and L_∞ distance measure			
Model number	epsilon 0.005	epsilon 0.01	epsilon 0.02
Baseline GD trained model	43.14 %	32.66 %	17.93 %
1	8.31 %	0.55 %	0.01 %
2	21.84 %	7.14 %	0.50 %
3	23.01 %	7.73 %	0.79 %
4	20.02 %	5.88 %	0.49 %

MIM with ce loss and L_∞ distance measure			
Model number	epsilon 0.005	epsilon 0.01	epsilon 0.02
Baseline GD trained model	44.20 %	34.58 %	18.87 %
1	54.75 %	54.75 %	54.75 %
2	54.66 %	54.66 %	54.66 %
3	54.15 %	54.15 %	54.15 %
4	50.93 %	50.93 %	50.93 %

MIM with cw loss and L_2 distance measure		
Model number	epsilon 0.3	epsilon 0.5
Baseline GD trained model	40.84 %	30.98 %
1	10.73 %	10.02 %
2	18.41 %	11.56 %
3	19.00 %	11.92 %
4	16.45 %	11.03 %

PGD with cw loss and L_∞ distance measure			
Model number	epsilon 0.005	epsilon 0.01	epsilon 0.02
Baseline GD trained model	45.79 %	37.63 %	24.05 %
1	14.31 %	1.88 %	0.03 %
2	27.67 %	12.33 %	1.64 %
3	28.93 %	12.96 %	2.09 %
4	25.44 %	10.55 %	1.37 %

PGD with ce loss and L_∞ distance measure			
Model number	epsilon 0.005	epsilon 0.01	epsilon 0.02
Baseline GD trained model	46.82 %	39.22 %	25.72 %
1	54.50 %	54.49 %	54.72 %
2	54.83 %	54.59 %	54.81 %
3	54.19 %	54.20 %	54.32 %
4	50.92 %	50.98 %	50.72 %

PGD with cw loss and L_2 distance measure		
Model number	epsilon 0.3	epsilon 0.5
Baseline GD trained model	41.02 %	31.27 %
1	10.81 %	10.02 %
2	18.69 %	11.68 %
3	19.34 %	12.08 %
4	16.60 %	11.10 %

FGSM with ce loss and L_2 distance measure				
Model number	epsilon 0.0001	epsilon 0.3	epsilon 0.5	epsilon 1.0
Baseline GD trained model	54.48 %	37.35 %	27.45 %	12.00 %
1	6.22 %	6.22 %	6.22 %	6.22 %
2	6.26 %	6.26 %	6.26 %	6.26 %
3	5.92 %	5.92 %	5.92 %	5.92 %
4	6.85 %	6.85 %	6.85 %	6.85 %

BIM with ce loss and L_2 distance measure				
Model number	epsilon 0.0001	epsilon 0.3	epsilon 0.5	epsilon 1.0
Baseline GD trained model	54.48 %	36.29 %	25.20 %	8.13 %
1	6.22 %	6.22 %	6.22 %	6.22 %
2	6.26 %	6.26 %	6.26 %	6.26 %
3	5.92 %	5.92 %	5.92 %	5.92 %
4	6.85 %	6.85 %	6.85 %	6.85 %

MIM with ce loss and L_2 distance measure				
Model number	epsilon 0.0001	epsilon 0.3	epsilon 0.5	epsilon 1.0
Baseline GD trained model	54.48 %	36.49 %	25.78 %	9.23%
1	6.22 %	6.22 %	6.22 %	6.22 %
2	6.26 %	6.26 %	6.26 %	6.26 %
3	5.92 %	5.92 %	5.92 %	5.92 %
4	6.85 %	6.85 %	6.85 %	6.84 %

PGD with ce loss and L_2 distance measure				
Model number	epsilon 0.0001	epsilon 0.3	epsilon 0.5	epsilon 1.0
Baseline GD trained model	54.46 %	36.61 %	25.79 %	12.28 %
1	6.22 %	6.22 %	6.17 %	6.16 %
2	6.25 %	6.25 %	6.25 %	6.24 %
3	5.95 %	5.95 %	5.97 %	5.96 %
4	6.87 %	6.87 %	6.83 %	6.84 %

DeepFool				
Model number	overshoot 0.005	overshoot 0.01	overshoot 0.02	overshoot 0.03
Baseline GD trained model	20.74 %	9.96 %	5.71 %	3.59 %
1	26.83 %	18.98 %	11.21 %	6.68 %
2	25.95 %	19.34 %	12.65 %	8.09 %
3	28.80 %	22.47 %	15.15 %	10.03%
4	24.43 %	17.71 %	11.98 %	8.02 %

CW		
Model number	kappa 0	kappa 0.1
Baseline GD trained model	0 %	34.98 %
1	0 %	0 %
2	0 %	0 %
3	0 %	0 %
4	0 %	0 %

8.1.2 Black Box

NES L_∞		
Model number	epsilon 0.01	epsilon 0.03
Baseline GD trained model	47.80 %	35.40 %
1	25.23 %	3.53 %
2	35.09 %	13.52 %
3	36.32 %	14.10 %
4	32.51 %	11.84 %

NES L_2			
Model number	epsilon 0.7	epsilon 1.0	epsilon 1.3
Baseline GD trained model	46.59 %	45.33 %	44.56 %
1	21.63 %	18.14 %	16.57 %
2	32.46 %	29.67 %	28.27 %
3	33.81 %	30.90 %	29.42 %
4	29.78 %	27.45 %	26.16 %

SPSA L_∞			
Model number	epsilon 0.01	epsilon 0.02	epsilon 0.03
Baseline GD trained model	45.73 %	36.43 %	28.13 %
1	17.77 %	3.40 %	0.59 %
2	29.76 %	13.73 % %	5.73 %
3	31.11 %	14.42 %	6.18 %
4	27.36 %	12.21 %	4.80 %

SPSA L_2			
Model number	epsilon 0.5	epsilon 0.7	epsilon 1.0
Baseline GD trained model	44.36 %	40.06 %	33.01 %
1	15.33 %	7.06 %	1.85 %
2	27.61 %	19.26 %	10.55 %
3	28.99 %	20.46 %	11.11 %
4	25.46 %	17.58 %	9.03 %

NAttack			
Model number	epsilon 0.03	epsilon 0.05	epsilon 0.1
Baseline GD trained model	19.90 %	10.54 %	4.54%
1	39.41 %	32.00 %	25.96%
2	37.53 %	38.00 %	38.14%
3	31.17 %	31.98 %	32.28%
4	39.51 %	39.41 %	39.13%

8.2 Foolbox

8.2.1 White Box

L_∞ FGSM			
Model	epsilon 0.005	epsilon 0.01	epsilon 0.02
Baseline GD trained model	44.39 %	35.2 %	20.6 %
1	54.77 %	54.77 %	54.77 %
2	54.68 %	54.68 %	54.68 %
3	54.16 %	54.16 %	54.16 %
4	50.95 %	50.95 %	50.95 %

L_∞ BIM			
Model	epsilon 0.005	epsilon 0.01	epsilon 0.02
Baseline GD trained model	44.05 %	34.07 %	17.44 %
1	54.77 %	54.77 %	54.77 %
2	54.68 %	54.68 %	54.68 %
3	54.16 %	54.16 %	54.16 %
4	50.95 %	50.95 %	50.95 %

L_∞ DeepFool			
Model	epsilon 0.005	epsilon 0.01	epsilon 0.02
Baseline GD trained model	43.12 %	32.51 %	17.2 %
1	9.2 %	0.66 %	0.01 %
2	22.46 %	7.88 %	0.64 %
3	23.71 %	8.57 %	0.86 %
4	20.52 %	6.53 %	0.64 %

L_∞ PGD			
Model	epsilon 0.005	epsilon 0.01	epsilon 0.02
Baseline GD trained model	45.31 %	36.38 %	21.06 %
1	54.73 %	54.59 %	54.78 %
2	54.61 %	54.72 %	54.79 %
3	54.15 %	54.08 %	53.98 %
4	50.93 %	50.92 %	50.84 %

L_2 FGSM			
Model	epsilon 0.3	epsilon 0.5	epsilon 1.0
Baseline GD trained model	37.36 %	27.46 %	12.0 %
1	54.77 %	54.77 %	54.77 %
2	54.77 %	54.77 %	54.77 %
3	54.38 %	54.38 %	54.38 %
4	51.01 %	51.01 %	51.01 %

L_2 BIM			
Model	epsilon 0.3	epsilon 0.5	epsilon 1.0
Baseline GD trained model	36.37 %	25.5 %	7.81 %
1	54.77 %	54.77 %	54.77 %
2	54.74 %	54.74 %	54.74 %
3	54.36 %	54.36 %	54.36 %
4	51.01 %	51.01 %	51.01 %

L_2 DeepFool			
Model	epsilon 0.3	epsilon 0.5	epsilon 1.0
Baseline GD trained model	34.87 %	24.45 %	8.26 %
1	1.45 %	0.07 %	0.0 %
2	11.08 %	2.61 %	0.03 %
3	11.98 %	3.2 %	0.05 %
4	9.63 %	2.21 %	0.01 %

L_2 PGD			
Model	epsilon 0.3	epsilon 0.5	epsilon 1.0
Baseline GD trained model	39.32 %	29.84 %	12.25 %
1	54.77 %	54.69 %	54.47 %
2	54.86 %	54.67 %	54.87 %
3	54.33 %	54.55 %	54.46 %
4	51.13 %	51.08 %	50.71 %

L_2 Newtonfool						
Model	epsilon 0.0001	epsilon 0.001	epsilon 0.005	epsilon 0.1	epsilon 0.3	epsilon 0.5
Baseline GD trained model	54.5 %	54.44 %	54.16 %	48.59 %	37.49 %	27.84 %
1	10.0 %	10.0 %	10.0 %	10.0 %	10.0 %	10.0 %
2	10.03 %	10.03 %	10.03 %	10.03 %	10.03 %	10.03 %
3	10.0 %	10.0 %	10.0 %	10.0 %	10.0 %	10.0 %
4	9.98 %	9.98 %	9.98 %	9.98 %	9.98 %	9.98 %

8.2.2 Black Box

L_∞ AUN								
Model	epsilon 0.005	epsilon 0.01	epsilon 0.02	epsilon 0.1	epsilon 0.3	epsilon 0.5	epsilon 0.8	epsilon 1.0
Baseline GD trained model	54.48 %	54.47 %	54.44 %	52.74 %	34.54 %	17.49 %	13.52 %	12.42 %
1	54.75 %	54.63 %	54.81 %	50.93 %	26.34 %	16.74 %	12.61 %	11.21 %
2	54.86 %	54.97 %	54.97 %	53.73 %	37.99 %	22.81 %	13.21 %	11.69 %
3	54.54 %	54.44 %	54.26 %	53.54 %	38.29 %	24.88 %	14.05 %	11.59 %
4	51.11 %	51.07 %	50.99 %	49.28 %	34.42 %	19.92 %	12.25 %	11.06 %

L_2 CR											
Model	epsilon 0.3	epsilon 0.5	epsilon 1.0	epsilon 3.0	epsilon 5.0	epsilon 8.0	epsilon 10.0	epsilon 13.0	epsilon 15.0	epsilon 18.0	epsilon 20.0
Baseline gradient descent model	54.56 %	54.53 %	54.55 %	52.44 %	45.66 %	29.44 %	20.37 %	13.12 %	11.09 %	10.25 %	10.06 %
1	54.63 %	54.55 %	54.47 %	53.08 %	48.46 %	35.22 %	25.06 %	15.15 %	12.31 %	10.77 %	10.31 %
2	54.81 %	54.83 %	54.86 %	51.64 %	45.48 %	30.25 %	21.81 %	14.57 %	12.09 %	10.47 %	10.14 %
3	54.49 %	54.5 %	54.51 %	53.04 %	47.88 %	32.82 %	23.03 %	14.4 %	11.88 %	10.55 %	10.14 %
4	51.09 %	51.17 %	50.9 %	48.83 %	44.05 %	30.8 %	22.51 %	15.32 %	12.85 %	10.91 %	10.53 %

L_2 AGN											
Model	epsilon 0.3	epsilon 0.5	epsilon 1.0	epsilon 3.0	epsilon 5.0	epsilon 8.0	epsilon 10.0	epsilon 13.0	epsilon 15.0	epsilon 18.0	epsilon 20.0
Baseline gradient descent model	54.61 %	54.53 %	54.59 %	52.91 %	49.93 %	41.81 %	34.28 %	24.43 %	20.44 %	16.49 %	15.52 %
1	54.75 %	54.88 %	54.68 %	52.02 %	43.97 %	31.36 %	26.27 %	20.71 %	18.74 %	15.9 %	14.97 %
2	54.99 %	54.85 %	55.1 %	54.56 %	50.7 %	43.22 %	37.84 %	30.23 %	26.18 %	20.72 %	18.58 %
3	54.66 %	54.6 %	54.29 %	53.4 %	50.56 %	43.36 %	37.92 %	31.32 %	27.1 %	22.7 %	20.38 %
4	50.95 %	51.0 %	50.87 %	49.3 %	46.12 %	38.94 %	34.21 %	27.33 %	23.54 %	18.97 %	16.6 %

L_2 S&P							
Model	epsilon 0.001	epsilon 0.01	epsilon 0.1	epsilon 0.5	epsilon 1.0	epsilon 3.0	epsilon 5.0
Baseline GD trained model	54.5 %	54.5 %	54.16 %	51.5 %	47.48 %	33.28 %	22.46 %
1	54.81 %	54.74 %	53.73 %	44.6 %	29.7 %	5.57 %	1.6 %
2	54.95 %	54.89 %	54.23 %	47.6 %	36.85 %	14.4 %	6.51 %
3	54.52 %	54.48 %	53.89 %	47.76 %	38.79 %	17.02 %	8.29 %
4	51.06 %	51.02 %	50.52 %	44.72 %	34.88 %	14.63 %	6.85 %

NTNU

Kunnskap for en bedre verden