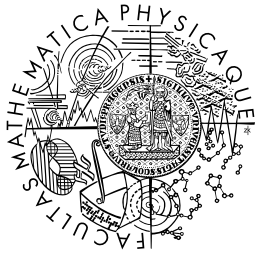


FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

BACHELOR THESIS

Jakub Hejhal

Exploring the vulnerabilities of commercial AI systems against adversarial attacks

Department of Theoretical Computer Science and Mathematical Logic

Supervisor of the bachelor thesis: Mgr. Roman Neruda, CSc.
Study programme: Computer Science
Study branch: General Computer Science

Prague 2021

I declare that I carried out this bachelor thesis independently, and only with the cited sources, literature and other professional sources. It has not been used to obtain another or the same degree.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ....

Author's signature

I would like to thank Mgr. Roman Neruda, CSc. for his patiance, helpful advice and guidance throughout my thesis work. I would also like to thank my family, my friends and my girlfriend for supporting me and for providing me with love and care that allowed me to push through occasionaly difficult times.

Title: Exploring the vulnerabilities of commercial AI systems against adversarial attacks

Author: Jakub Hejhal

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: Mgr. Roman Neruda, CSc., Department of Theoretical Computer Science and Mathematical Logic

Abstract: Abstract. TODO

Keywords: Machine learning Deep learning Adversarial attack Black-box

Contents

1. Introduction

1.1 The rise of artificial neural networks

In recent years there has been an enormous surge in applications of artificial intelligence technologies based on neural networks to various fields. One of the most significant milestones that kickstarted today's AI revolution has undoubtedly been the year 2012. ImageNet Large Scale Visual Recognition Challenge (ILSVRC), which hand-crafted specialized image-labeling systems have previously dominated, was won by AlexNet with its CNN architecture.

Artificial neural networks inspired by their biological analog had been known and researched for a long time. Even though they enjoyed much enthusiasm initially, there has been a time period (called an "AI winter" by some) when they had been neglected as an unpromising direction towards general intelligence. More classical approaches like SVM and rule-based AI systems showed better performance and computational efficiency on AI benchmarks of the time.

What changed the game has been the available computational power that came with Moore's law and the usage of GPUs, mainly their parallel nature in accelerating matrix multiplication operations, that neural networks use heavily.

Another factor that helped the rise of neural networks has been the availability of large datasets like ImageNet, which contains 1,281,167 images for training and 50,000 images for validation organized in 1,000 categories.

The availability of large amounts of labeled and unlabeled data, sometimes referred to as "Big data," is only getting better. A large part of our lives has moved to the virtual space thanks to the internet. Businesses had started to realize the value of the enormous amounts of traffic generated every day, and they are increasingly trying to figure out how to take advantage of it.

What was previously limited to academic circles had quickly become mainstream. Artificial neural networks have proven to be very versatile and have quickly been successfully applied to a wide range of problems. New neural network architectures and new training regimes allowed training deeper networks, which gave rise to a new field of machine learning called "Deep learning."

Deep neural networks have shown state-of-the-art performance in machine translation, human voice synthesis and recognition, drug composition, particle accelerator data analysis, recommender systems, algorithmic trading, reinforcement learning, and many other areas.

1.2 Concerns

Large-scale deployment of neural network systems has been criticized by many for their inherent unexplainability. It is often hard to pinpoint why neural network behaves in some way and why it makes certain decisions. One problem is the role of training data, where possible biases may be negatively and unexpectedly reflected in the behavior of the AI system. Another problem is the performance on out-of-distribution examples, where network inference occurs on different kinds of data than used in the training stage.

Those concerns lead people to study the robustness of AI systems. It turned out that image recognition CNN networks are especially susceptible to the so-called adversarial attacks, where the input is perturbed slightly, but the output of the network changes wildly. Similar kinds of attacks have since been demonstrated in other areas like speech recognition, natural language processing, or reinforcement learning.

1.3 Adversarial attacks

This vulnerability of neural networks has led to a cat-and-mouse game of various attack strategies and following defenses proposed to mitigate them.

Neural networks can be attacked at different stages:

- training,
- testing,
- deployment.

Training attacks exploit training dataset manipulation, sometimes called dataset poisoning, to change the behavior of the resulting trained network.

Testing attacks do not modify trained neural network, but often use the knowledge of the underlying architecture to craft adversarial examples which fool the system.

Deployment attacks deal with real black-box AI systems, where the exact details of the system are usually hidden from the attacker. Nevertheless, partly because similar neural network architectures are used in the same classes of problems, some vulnerabilities in testing scenarios can still be exploited in deployment, even though the exact network parameters, architecture, and output are unknown to the attacker.

The purpose of this thesis is to explore the applicability of certain classes of testing attacks on real-world deployed AI systems. For simplicity, many kinds of State-of-the-art (SoTA) adversarial attacks have only been explored in the testing regime but have not been applied to truly black-box systems.

The main aim of the thesis will be to test different types of testing attacks against Google Cloud Vision API, which provides its image labeling capabilities as Software-as-a-service (SaaS). Understandably, attacking an unknown system will be more challenging than attacking a known neural network in the testing stage. We will try to compare these two attack scenarios and come up with a reasonable attack performance metric. This information could prove helpful in selecting the most promising attack to a specific situation.

Many SoTA testing attacks were not designed to attack specific deployed systems, so some attacks will need to be slightly changed to be used. We will explore different ways to modify existing attacks and evaluate them.

If service like Google Cloud Vision is found to be vulnerable, it's very likely that other SaaS providers like Amazon Rekognition or Clarifai would be as well. This fact could have a massive impact on all downstream applications using those SaaS APIs. For instance, content moderation mechanisms, which rely mainly on automatic detection, could be circumvented.

Finally, a better understanding of the failure modes of current AI systems may help us create better ones in the future.

The structure of this work is as follows:

In Preliminaries and related work 2 we explain the most important concepts of machine and deep learning. We introduce the phenomenon of adversarial attacks and review the current progress made in the whitebox and blackbox adversarial attacks relevant to our goal.

In chapter Our approach 3 we put forward the motivations behind our solution and experiments. We suggest possible ways of combining current whitebox and blackbox attack methods with a goal of ultimately targeting the cloud blackbox classifiers.

In chapter Experiments 4 we present the detailed results of the conducted experiments, which are motivated in chapter 3. We focus mostly on transferability of adversarial images, which will be explained in chapter 2.2, as it is one of the best ways to significantly reduce queries to the attacked target and increas the chances of staying undetected in a real-world scenario.

In Future work 5 we propose a handful of potentially promising directions to explore in the future based on our results from 4.

Finally, in Conclusion 6 we sum up our contributions and evaluate the degree to which we have been able to fulfill our goals stated in this chapter 1

2. Preliminaries and related work

In this section we briefly introduce and explain the necessary theoretical background, upon which we build later on.

We first define the concepts of Machine learning (ML) and Deep neural networks (DNNs). We explain, how DNNs can learn patterns from data by using powerful gradient-based stochastic optimization algorithm called Stochastic gradient descent (SGD). Then we talk about a subset of DNNs that perform very well on image data called Convolutional neural networks (CNNs).

Finally, we describe the systemic vulnerability of DNNs and define an adversarial attack. Then we explain related terms like adversarial defenses, adversarial examples, adversarial robustness and different adversarial attack threat models and present current state of the field.

2.1 Deep learning basics

2.1.1 Machine learning

The topic of machine learning has been introduced many times over in other books and papers. Murphy [2012], Bishop [2006] and Goodfellow et al. [2016] all provide a comprehensive exposition to the field. Because of the abundance of many different kinds of resources, we don't feel the need to reinvent the wheel and come up with yet another machine learning introduction.

To put forward the most accepted definition of machine learning, the Murphy [2012] lay it as follows:

"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."

2.1.2 Deep neural networks

In our case the "computer program learning from experience" will be in most cases a deep neural network (DNN). We will introduce neural networks only very briefly at a high level as more details can be easily found in Goodfellow et al. [2016].

DNN is in its essence a parametrized mapping $y = f(x; \theta)$, where x are the network inputs, θ are the parameters and y are the outputs of the network. The "network" in its name comes from the fact that the function computation is described as an evaluation of a computational directed acyclic graph (DAG), the structure of which depends on the exact network architecture.

During evaluation, each node in the graph is asociated with a number called "activation", This activation value is computed from the values of other nodes and from the relevant parameters θ. It is then passed through a non-linear activation function $a(x)$. For a long time the $sigmoid(x)$ activation function was very popular. Nowdays $ReLU(x) = \max(0, x)$ (Rectified Linear Unit) is used more often as it is better suited for deeper networks.

The nodes in the DAG are sometimes referred to as "neurons", as there is an analogy to the functioning of biological neurons.

The function $f(x;\theta)$ can be often decomposed into several computational steps: $f(x;\theta) = f_n(f_{n-1}(...f_2(f_1(x;\theta_1);\theta_2)...;\theta_{n-1});\theta_n)$. In this decomposition the individual functions $f_i(x_i;\theta_i)$ are called "layers". When the number of layers is large, the network is said to be deep. The successive computations of $f_1(x_1,\theta_1)$, $f_2(x_2,\theta_2)$ up to $f_n(x_n,\theta_n)$ is called a "forward pass".

2.1.3 Network training

If we have a dataset D consisting of input datapoints x_i and their corresponding y_i values called "labels", we want the neural network function $f(x,\theta)$ to express the relationship between x_i and y_i. Depending on the complexity of the network, the parameters θ can be often set accordingly, such that $f(x_i,\theta) \approx y_i$. This ability to approximate the data well with the right θ is called "network's capacity". How good is the neural network's approximation is measured by a loss function $\mathcal{L}(D, f(x,\theta))$. Because the network models the relationship between x_i and y_i, it is sometimes just called a "model".

The process of training a neural network tries to minimize the training loss by optimizing the parameters. Formally, we search for $\theta = \underset{\theta}{arg\,min}[\mathcal{L}(D, f(x,\theta))]$.

This optimization is in practice done iteratively using an algorithm called stochastic gradient descent (SGD). It involves computing the gradient of the loss with respect to the parameters θ: $\nabla_\theta \mathcal{L}(D, f(x,\theta))$. It then updates the parameters in the opposite direction of the gradient to hopefully lower the loss.

The partial derivates $\frac{\partial \mathcal{L}(D,f(x,\theta))}{\partial \theta_i}$ can be efficiently computed by an algorithm called "backpropagation". It is very similar to a forward pass, but it starts from the loss and proceeds in the opposite direction of the DAG. Detailed explanation can be again found in Goodfellow et al. [2016]. It is also icredibly well explained in the YouTube video series about neural networks by 3Blue1Brown (Grant Sanderson)

2.1.4 Convolutional neural networks

Convolutional neural network (CNN) is a network using a special type of layers called "convolutional layers". CNNs were invented more than 20 years ago (LeCun and Bengio [1998]) and they are powerful feature extractors compressing highly dimensional spatial inputs to a smaller feature vector. They are used mainly when dealling with images, but in general, convolutions are useful for any type of spatial data with translational invarience property. When a network is asked to perform an image classification, it is often desireable for an object in the left part of the image to be detected in the same way as an object on the right side. Convolutional layer achives this by convolving each part of the input image with the same kernel, thus sharing weights and greatly reducing the parameter count. Each convolutional kernel is sometimes referred to as "filter", because the kernel convolution operation resembles a sobol operator used in classical computer vision and image processing. CNNs usually stack multiple convolutional layers on top of each other with each subsequent layer having more filters, but reducing the spatial dimension. The spatial dimension reduction is achived by an average or

max pooling layers that follow periodically after each set of convolutional layers. The final convolution outputs are either pooled together to reduce the spatial dimension to 1x1, or they are passed through a one or two densely connected layers to produce the extracted features.

2.2 Related Work

2.2.1 Definitions

As it was outlined in chapter 1, adversarial attacks are methods of producing adversarial examples. Given an original input x and a classifier model $y = f(x), y \in C$, an adversarial example $x_{adv} = x + \delta$ is a slightly pertubed version of x, such that $f(x_{adv}) \neq y$. By "slightly" we mean $\|\delta\| < \epsilon$. In this work we focus on image adversarial examples achieving misclasification on image classification models, albeit the concept of adversarial attacks is applicable across different types of neural networks and different types of input data.

One can be also interested in the minimal pertubation δ_{min} needed to change the model classification:

$$\delta_{min} = arg\min_{\delta} \|\delta\| \quad \text{such that} \quad f(x+\delta) \neq y$$

This brings us to the definition of robustness $r(x, f)$ given an input x:

$$r(x, f) = \|\delta_{min}\|$$

More informative can be the expected robustness across all of our data D, the global robustness of a model:

$$\rho(f) = \mathop{\mathbb{E}}_{x \sim \mathcal{D}} r(x, f)$$

Whether we do or do not care about the nature of misclassification distinguishes the targeted and untargeted attacks:

- targeted attacks require that $f(x_{adv}) = t, t \in C$
- untargeted attacks only aim for unspecified misclasification - $f(x_{adv}) \neq y$

The constraint on pertuation size $\|\delta\|$ makes sure, that the adversarial example x_{adv} looks almost the same as the original x. That said, there are many valid choices for the metric $\|\cdot\|$ that are commonly used:

- l_0 - the number of non-zero components of δ.
- l_1 - $\|\delta\| = \sum_i |\delta_i|$
- l_2 - $\|\delta\| = \sqrt{\sum \delta_i^2}$
- l_{inf} - $\|\delta\| = \max_i \delta_i$

2.2.2 Whitebox attacks

Whitebox adversarial attacks assume full knowledge of the target model, which allows for efficient computation of the the gradients with respect to the input. This is in contrast with blackbox attacks 2.2.4, where the attacked model is available only as a blackbox and as such the gradients can be only estimated using sampling for example.

Biggio et al. [2013] was the first to point out the inherent vulnerability of machine learning models by attacking SVMs and multi-layer perceptrons.

In the same year Szegedy et al. [2014] used an L-BFGS optimization algorithm that leverages estimates of second order partial derivates information to find minimally distorted adversarial example x by solving the following:

$$\min \|x - x_{adv}\|_2^2 \quad \text{s.t.} \quad f(x_{adv}) = t \quad \text{and} \quad x_{adv} \in [0,1]^m$$

In 2014 Goodfellow et al. [2015] introduced the Fast gradient sign method (FGSM) which we use in 4.2.4.1. It involves doing only one backpropagation, so it is very efficient and is often use for an adversarial training. Adversarial training is a method of dynamicaly extending the training dataset by adversarial examples generated on the fly. FGSM is well suited for this, because it's fast.

In 2015 Jacobian-based saliency map attack (JSMA) was proposed by Papernot et al. [2015].

Various methods trying to reduce the susceptibility of neural networks to adversarial examples are being proposed around this time. These methods are called "adversarial defences". One of those is the previously mentioned adversarial training. Few others include for example JPG compression, stochastic augmentations, feature distilation etc.

As an attempt to break a specific "distilation defense", Carlini and Wagner [2017] propose their C&W attack. FGSM and L-BFGS aren't strong-enough against the distilation defense, because the gradients are orders of magnitude smaller than in the case of an undefended target.

They reframe the optimization problem as:

$$\min \|x - x_{adv}\|_2^2 + c \cdot f(x', t) \quad \text{s.t.} \quad x' \in [0,1]^m \quad \text{where} \quad f(x', t) = (max_{i \neq t} Z(x')_i - Z(x')_t)^+$$

Their reframing allows for adaptive scaling of the objective being optimized and as such doesn't suffer as much from the small gradients. Furthemore, their objective is different from the objectives optimized previously, which also helps the C&W attack.

For a more comprehensive overview of whitebox attacks and adversarial attacks in general we will point the reader to the review paper Xu et al. [2019].

2.2.3 Transferability

It was demonstrated Tramèr et al. [2017] that adversarial examples generated by whitebox attacks have high probability of decieving another neural network. This transferability of adeversarial examples is made more severe when the two models have similar architectures and when the datasets used to train them are the same.

2.2.4 Blackbox attacks

In the blackbox setting the access to the target model is limited and only the final output is available to the attacker.

Nevertheless, the transferability proprety 2.2.3 can be used to fool the target model without any further knowledge.

A different approach of attacking a blackbox model can be to estimate the gradients by sampling to make up for the inability to backpropage through it. (Chen et al. [2017], Ilyas et al. [2018])

There is yet another class of blackbox attacks. These attacks don't rely on the gradient information at all, but instead they use random search to find the adversarial pertubation. SquareAttack Andriushchenko et al. [2020], which we utilize in this work, is one example of such attacks.

In chapter 3 we talk more about the different blackbox attacks and about possible ways of combining different approaches together.

3. Our approach

3.1 Adapting off-the-shelf attack algorithms to partial-information setting

Cloud-based image classifiers don't usually classify input images into a fixed number of classes. They instead output variable-length list of probable labels with scores. And what's worse, those scores aren't even probabilities, because they don't sum up to one.

Most of current score-based SoTA adversarial attacks assume that we have access to all output logits of the attacked target classifier. If we want to use them, we need to map somehow the cloud's variable-length score-output to fixed-length vector, which will simulate logits output of a standard CNN classifier.

3.1.1 Object-organism binary classification mapping

To simplify the experiments, we define a simple benchmarking attack scenario:

Given an image containing a living thing, fool the target into classifying it as a non-living object.

This choice makes our simulated classifier a binary one. It should assign each input image (x) an organism score $o_{organism}(x)$ and object score $o_{object}(x)$. This 2-D score vector $(o_{object}(x), o_{organism}(x))$ is further denoted by $o(x)$ for simlicity.

We chose this split, because we perform majority of our experiments on ImageNet validation dataset (Deng et al. [2009]) and ImageNet is relatively balanced between those two semantic categories.

Why ImageNet? Despite the dataset being quite old, it is still the most heavily used dataset in the research community and the majority of freely available pretrained models are pretrained on it.

Furthermore, when $\|C\| = 2$ (where C is a set of output categories), targeted and untargeted attack scenarios don't differ anymore and are neatly unified.

Adapting the attack algorithm to a different attack objective only requires swapping the label mapping layer.

3.1.1.1 Imagenet category mapping

Classic ImageNet dataset contains real-world photos, each corresponding to one and only one classification category out of 1000 possible categories. Each ImageNet category c corresponds to a unique wordnet synset $w(c)$. These synsets are rather specific, but we can take a look at their set of hypernyms $h(w(c))$. If this hypernym set $h(w(c))$ contains the *organism* synset, it should be an organism, otherwise c is probably an object.

Writen more rigorously, we map each ImageNet category c into $\{organism, object\}$ using the following maping $m_{local}(c)$:

$$m_{local}(c) = \begin{cases} organism & organism \in h(w(c)) \\ object & organism \notin h(w(c)) \end{cases}$$

3.1.1.2 Cloud label mapping

This situation isn't so clear-cut in the case of general labels returned by cloud classifier. Labels might not even be single words, but whole sentences. We therefore resort to a more powerful label classification method and use a GPT-2 transformer (Radford et al. [2019]) for this matter. More specifically, we use the HuggingFace (Wolf et al. [2020]) zero-shot classification pipeline to encode text labels into embedding vector space and then compute their similarity $s(l_{cloud}, l_{gold})$ with carefully chosen set of organism labels $L_{organism} = \{animal, species\}$ and with a set of object labels $L_{object} = \{object, instrument\}$. The resulting binary cloud label mapping $m_{cloud}(l_{cloud})$ is defined as follows:

$$m_{cloud}(l_{cloud}) = \begin{cases} organism & \underset{c \in (L_{organism} \bigcup L_{object})}{arg\,max} \; s(l_{cloud}, c) \in L_{organism} \\ object & \underset{c \in (L_{organism} \bigcup L_{object})}{arg\,max} \; s(l_{cloud}, c) \in L_{object} \end{cases}$$

From now on, by $m(c)$ we mean either $m_{local}(c)$ or $m_{cloud}(l_{cloud})$ where the distinction wouldn't make any difference.

3.1.1.3 Computing the adversarial loss

There is one more step we have to do to transparently simulate a binary classifier with 2 output logits.

By passing the model outputs through the separation mapping $m(c)$ we obtain two score sets: $S_{organism}$ and S_{object}. In the local case:

$$\|S_{organism}\| + \|S_{object}\| = \|L_{\text{ImageNet}}\| = 1000$$

.

In the case of a general cloud classifier these sets have variable sizes and one or both can be even empty.

There are multiple sensible ways to produce output logits vector $o(x)$ that would roughly correspond to an organism binary classifier output and which could be attacked using standard untargeted adversarial attacks.

Just to name a few choices for $o(x)$ that could intuitively work:

1. top-1 score: $\max S$
2. sum of logits: $\sum S$
3. top-1 score for $S_{organism}$, least-likely class for S_{object}
4. sum of un-normalized probabilities: $\sum exp(S)$
5. log-sum of probabilities: $\log \sum softmax(S)$
6. ...

But in the end we want to achive a misclasification of the original organism image x_{org}.
If we go with 1) and produce output vector $o(x) = (\max S_{object}, \max S_{organism})$,

misclasification is achived when $o(x)_0 > o(x)_1$. We can define a margin loss objective $\mathcal{L}_{margin}(x, \kappa) = \max S_{organism}(x) - \max S_{object}(x) + \kappa$ with additional parameter κ, by which we can adjust our requirement for the degree of misclasification.

Another hint that this might be a solid choice comes from the Carlini and Wagner [2017], where they discuss the choice of optimization objective. They try a number of different alternatives, but in the end they conclude that optimizing the margin loss works the best, so we stick with it.

In the future 3) might also be worth a try. It is somewhat similar to the objective function of the Iterative least-likely class method introduced in Kurakin et al. [2017].

3.2 PoC black-box GVision attacks

We first explored the viability and sample-efficiency of current SoTA black-box attacks. We ran the following against Google Vision API image classifier.

- 3.2.1 TREMBA (Huang and Zhang [2020])
- 3.2.2 RayS (Chen and Gu [2020])
- 3.2.3 SquareAttack (Andriushchenko et al. [2020])
- 3.2.4 Sparse-RS (Croce et al. [2020])

TODO: describe each attack briefly and show sample images

3.2.1 TREMBA

3.2.2 RayS

3.2.3 SquareAttack

3.2.4 Sparse-RS

We go into more details in the Experiments 4

3.3 The need for query-efficient attack

In the previous section 3.2 we empirically showed, that Google Vision API isn't 100% robust to iterative blackbox atttacks. But although the previously mentioned blackbox attacks are often successful in producing adversarial image, query count to the target may be often unacceptably high. Huge query stress to the target is troublesome for several reasons:

- High cost - 1.5$ per 1000 queries
- Raising suspicion
- Often unrealistic in practical setting

The problem is that these blackbox attacks (with the exception of TREMBA) mostly rely on random search and don't make use of the gradient similarity of various CNN models. The high dimensionality of the input data doesn't make the blackbox optimization task easy. Current SoTA blackbox attacks that don't use any gradient priors are already at the query efficiency limit with their medium queries being often less than 100 (but that of course depends on the precise threat-model under which the attack is evaluated). Even though the median in the hundreds is amazing when compared to early attempts like ZOO (Chen et al. [2017]) which required queries on the order of 10^4, it is still not satisfactory for a practical use.

3.4 Leveraging transferability

After these early experiments that proved the concept, we focused our attention on transferability, which has a huge potential to save queries.

3.4.1 Transfer attacks provide better seeds for blackbox optimization

Even if the localy-produced adversarial images don't transfer directly to the target, Suya et al. [2020] showed, that the output of transfer-attack can provide better starting seeds for blackbox optimization attacks and improve their query efficiency. The option to choose from several starting points basically adds a degree of freedom to the blackbox optimization. They also discuss different prioritization strategies, as the the number of seeds produced isn't limited by target queries and we can therefore afford to produce as many candidate starting points as we like.

3.4.2 Train, validate, test

There is a loose analogy between the crafting of an adversarial example and the training of machine learning model. ML model weights are first fitted againt specified loss contraint. This constraint is (among other things) a function of training data. The weights are then validated and checked against overfitting on a slightly different constraint, which now depends on a validation dataset. When all is good, model is happily deployed to production.

With a bit of imagination, ML model weights correspond to pixel values of an adversarial image. The pixels are first trained by gradient descent on training loss provided by a surrogate model. They are validated against ensemble set of diverse indepenent classifiers, and when the foolrate is good, they are sent for test evaluation to the cloud.

3.4.3 Local training and validation is cheap

We want to offload the cloud query-stress to local simulation as much as possible. An attacker can often afford to spend orders of magnitude more queries to local surrogates and validation models than to the actual target.

3.4.4 Multiple candidates save queries

Iterative black-box attacks usually have query-distrubutions which are tail-heavy. In other words, the median queries needed to create a successful adversarial image are much lower than the average queries.

Let's image an attack scenario, where we want to submit a photo to a platform with automatic content moderation mechanism. Querying the target hundreds of times would certainly attract unwanted attention and our heavy queries can quickly trigger human evaluation. If our primary goal is to craft only one adversarial image and as much as possible evade detection, having multiple candidate images would give us another degree of freedom and it could potentially mitigate the heavy-tail problem. This approach can be in principle transparently combined with the multiple-seed candidate suggestions mentioned in 3.4 using the same prioritization candidate scoring mechanism.

3.5 The need for attack pipeline

We argued in 3.4 that combining multiple whitebox and blackbox attack approaches could create more powerful attack as well as giving us more freedom and flexibility to tailor this combination to a specific attack scenario constraints. As of now, there isn't any general whitebox/blackbox attack pipeline which would combine different algorithms and allow us attacking cloud services in a practical way.

3.5.1 Possibility of multiple blackbox workers

We can also image running multiple different attacks in paralel and having some meta-controller orchestrating individual attack algorithms such that we minimize queries to the target and efficiently make use of the additional degrees of freedom.

3.5.2 The need for unified attack and model API

There are sereval frameworks unifing whitebox/blackbox attacks. To name a few, there is FoolBox (Rauber et al. [2020]) or AutoAttack (Croce and Hein [2020]).

Although they are excellent at testing the robustness of local models, they don't give us the flexibility we need to implement all the pipeline features mentioned previously. They cannot be used without some modification to attack cloud models and their optimization attacks cannot be cooperatively scheduled step by step, which is what would be required for effective multi-attack orchestration.

3.6 AdvPipe

3.6.1 The vision

Solves all our problems. At least in theory.

3.6.2 The reality

Solves some of our problems. Like 10%.

TODO: Make some excuses why you didn't make it in time.

TODO: Make some flowchart of what is actually working.

3.7 Implementation

AdvPipe is implemented in Python 3.8 and uses primarily PyTorch 1.9.0 (Paszke et al. [2019]) and NVIDIA Cuda as the computational backend.

3.7.1 Attack regimes

3.7.1.1 Cooperative iterative regime

3.7.1.2 Transfer regime

3.7.1.3 Transfer regime with multiple targets

3.7.2 Attack algorithms

3.7.2.1 Whitebox

3.7.2.2 Blackbox

3.7.3 Wrapping whitebox and blackbox models

All models (cloud, local) are wrapped as PyTorch `torch.nn.Modules`. This way they can be used in a plug-and-play manner and passed easily to existing attack algorithms or frameworks like FoolBox (Rauber et al. [2020]). Model outputs are mapped to a 2-D score vector using the mapping discussed in 3.1.1, which allows us to optimize our custom binary objective.

Cloud models are compatible with any blackbox attacks, as long as the attacks don't require the gradients (which they shouldn't anyway).

Local models are on the other hand fully differentiable, so we can attack them with whitebox attacks on top of the standard blackbox attacks.

3.7.3.1 Preprocessing

In chapter 4 we thoroughly explore the effects of different augmentation and regularization techniques to enhance transferability. These are often implemented as stochastic preprocessing layers which are differentiable. For this purpose we mostly use Kornia – differentiable computer vision library for PyTorch (Riba et al. [2019]).

3.7.4 Configuration

AdvPipe is highly configurable by using YAML config files. These are parsed and checked by `config_datamodel.py`.

3.7.4.1 Config templating

The YAML configs files also support simple templating mechanism, which can be used to run multiple experiments with slightly different hyperparameters. This is in line with the DRY pricinple and helps to keep the number of configuration files in sane numbers.

3.7.5 Dependency management

The motivation to run our framework on a number of differrent machines with different environments, easily installing AdvPipe reliably and escaping dependency hell became our priority number one. We needed to accomodate Tensorflow (required by the HuggingFace pipeline) and PyTorch and a number of other dependencies in the same Python environment. We decided to do away with pip, that can often leave the Python virtual environment in an inconsistent state and also with conda environment manager (ana [2020]), which ensures package compatibily, but doesn't always contain all the latest Python package versions. We instead moved to Poetry - an excellent pip alternative - to manage our dependencies.

4. Experiments

4.1 Blackbox PoC on Google Cloud Vision API

Here we go into more technical details about previously mentioned blackbox attacks we initially tried.

- TREMBA
- RayS
- SquareAttack
- Sparse-RS

4.1.1 Baseline

We picked two sample images (shark and cat), on which we tested these blackbox attacks.

Figure 4.1: Cat and Shark, GVision baseline

Here's how Google Cloud Vision API classifies the original samples.

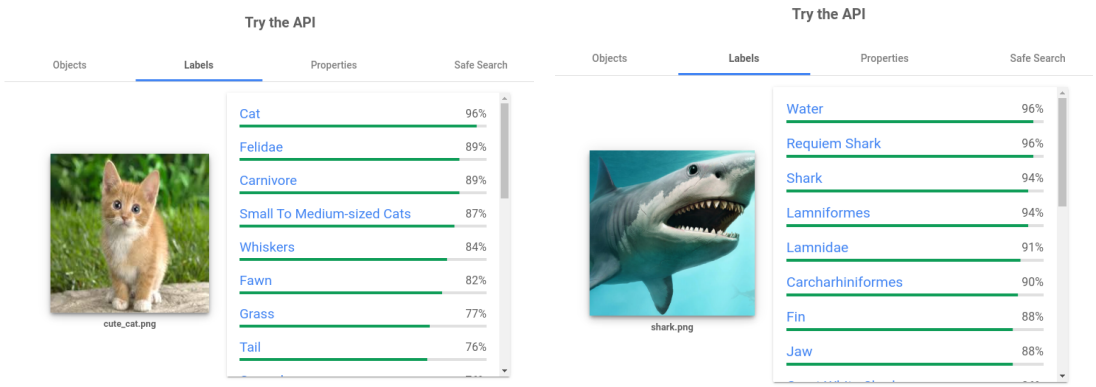


Figure 4.2: Cat and Shark, GVision baseline

4.1.2 TREMBA

4.1.3 RayS

This one is hard label attack and doesn't use the continuos loss from GVision.

4.1.4 SquareAttack

Because of high query intensity, we have only tested the "cat" sample image.

4.1.4.1 SquareAttack L2

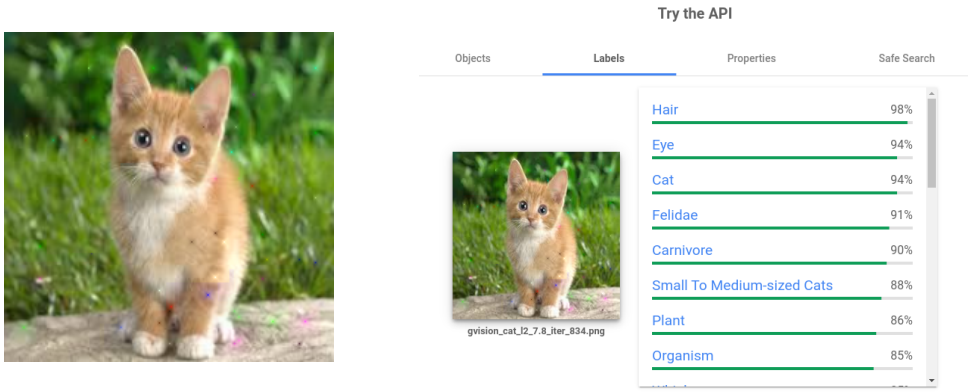


Figure 4.3: SquareAttack, 834 queries, pertubation norm $l_2 = 7.84$

4.1.4.2 SquareAttack Linf

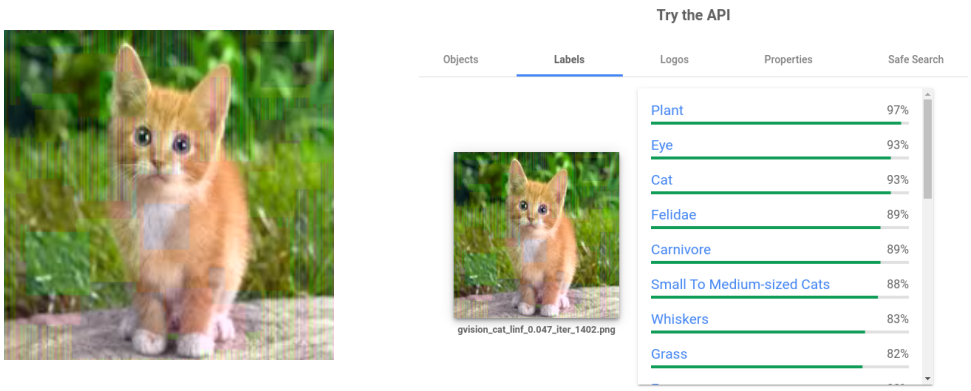


Figure 4.4: SquareAttack, 1402 queries, pertubation norm $l_{inf} = 0.047$

4.1.4.3 Evaluation of the GVision SquareAttack results

In both L2 and Linf modes we have achived out top-1 misclassification objective and the cat label was taken in both cases to the top-3 place.

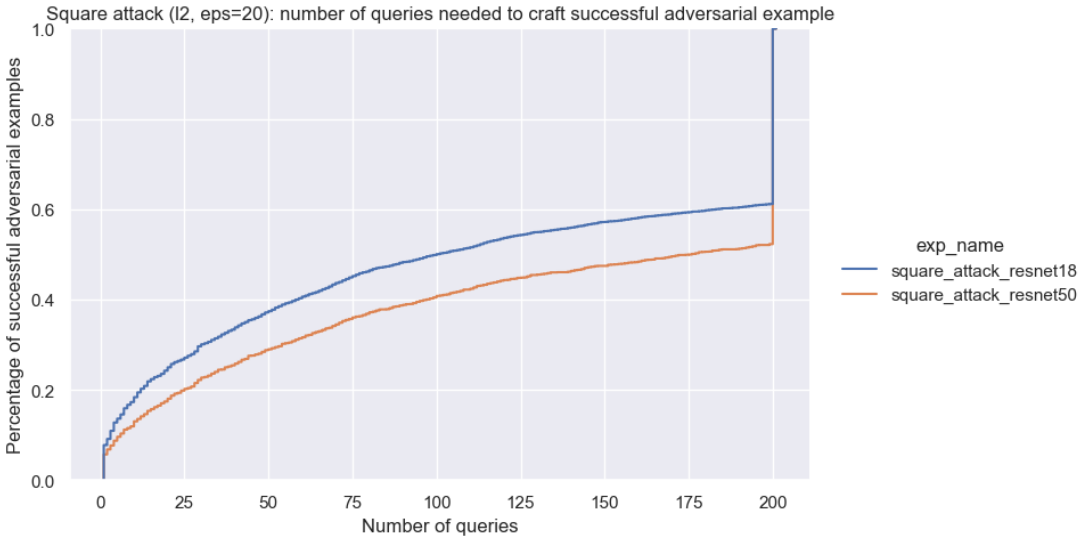
In this experiment of sample size = 1 the L2 version of SquareAttack seems to produce much less visually perceptible pertubation and is able to achive the top-1 misclassification with l_2 norm of only 7.84.

The Linf version on the other hand had to be given two times larger query budget, but we still had to turn up the l_{inf} pertubation norm to 0.047 to achieve our misclassification objective.

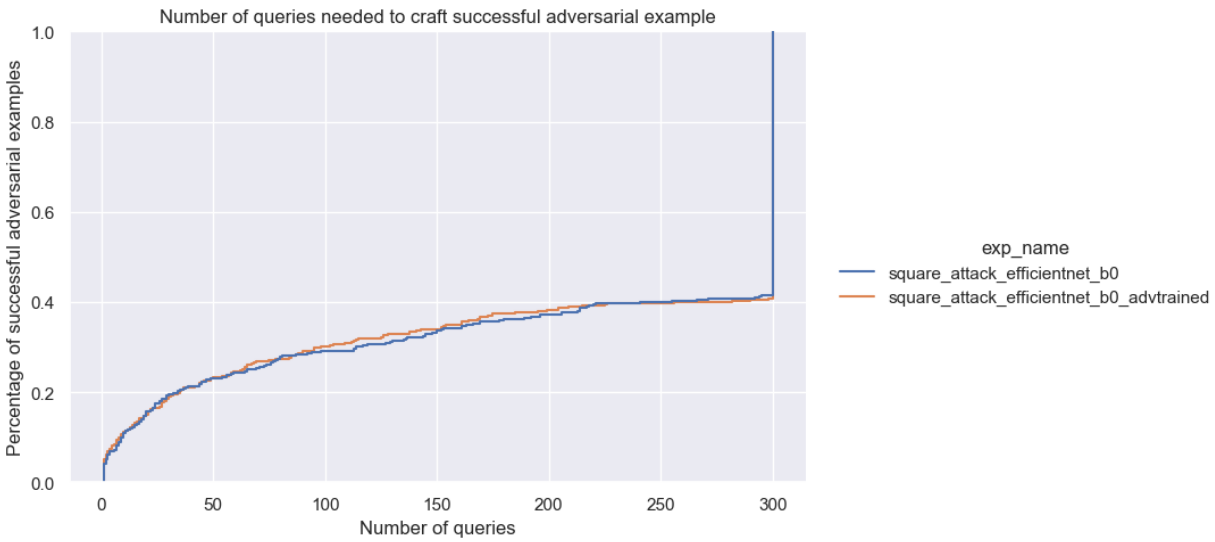
This observation made us focus more on the l_2 bounded attacks in later local experiments (4.2).

4.1.4.4 Local query distribution

To get an idea how SquareAttack would fare against some of our local models, we ran it in low query mode with 200 queries against ResNet-18 and ResNet-50, and with 300 queries against EfficientNet-b0 and it's adversarially trained counterpart. To make the job a bit easier for the SquareAttack, we relaxed the l_2 pertubation to 20.



(a) SquareAttack L2 on ResNets



(b) SquareAttack L2 on EfficientNets

Figure 4.5: SquareAttack L2 on local models

What is maybe a little bit surprising is that the adversarial training of EfficientNet-

b0 makes no difference when attacked by blackbox SquareAttack. This may be in part attributed to the fact, that the adversarial training defence primarily flattens out the gradients around training input data (Yu et al. [2018]), but doesn't provide any robustness guarantees (Kolter and Wong [2018]). As the blackbox SquareAttack doesn't use the gradient information explicitly, it may not be as affected by the adversarial training defense as the whitebox gradient attacks.

4.2 Local transferability experiments

Motivated by the not-ideal query-efficiency of pure-blackbox attacks (3.3), we moved to transfer attacks to see how far we can push the pure transfer threat model.

4.2.1 Choice of local models

We performed all our experiments on the following pretrained PyTorch ImageNet models.

- ResNet-18, ResNet-50 (He et al. [2015])
- ResNeXt-50 (32x4d) (Xie et al. [2017])
- Wide-ResNet-50-2 (Zagoruyko and Komodakis [2017])
- Squeezenet (Iandola et al. [2016])
- DenseNet-121 (Huang et al. [2018])
- EfficientNet (Tan and Le [2020])
- EfficientNet adversarially trained (Tramèr et al. [2020])

Apart from EfficientNets, all models were taken from the torchvision.models Python package. For the EfficientNets we used github.com/lukemelas/EfficientNet-PyTorch reimplementation, because the original implementation uses Tensorflow, and PyTorch is just so much better than Tensorflow.

To address the particular choice of our model set, we aimed at low model size, such that forward and backward pass fits comfortably in the 2GB of our MX-150 NVIDIA laptop GPU. This requirement for instance ruled out the VGG-style networks (Simonyan and Zisserman [2015]), the pre-ResNet 2nd-best submission to the ILSVRC 2014 (Russakovsky et al. [2015]). The Wide-ResNet-50-2 (WRN-50-2) is already at the limit with its 68,951,464 trainable parameters. In a standard FP-32 mode the model parameters, forward pass activations and backward pass partial derivatives take up almost 1GB of GPU memory. Adding to this the 620MB of GPU memory consumed by PyTorch at idle meant we could use only batch size of one for this particular Wide ResNet. But looking at the RobustBench leaderboard (Croce et al. [2021]), which uses AutoAttack (Croce and Hein [2020]) suite of parameter-free attacks to benchmark the robustness of various adversarially robust models, one can see the top positions are dominated by WideResNets, so we kept it in our model set despite its large size. We also

chose models with the same input size, such that the same dataset preprocessing (4.2.2.1) could be used for all of them and wouldn't complicate things any further. This decision meant we didn't use the popular Inception-v3 (Szegedy et al. [2015]), which takes inputs of size 299x299 instead of 224x224 of all the other models.

4.2.1.1 Inference time

In the beginning, AdvPipe had supported only batch sizes of one. In 4.6 it can be seen what kind of difference does batch size make on the inference time in such a memory-limited hardware setting. The small batch sizes didn't slow down the running times more than twice. We can also see in the figure that having batches much larger than you hardware can handle (ResNet-101, ResNet-152) can result in a sub-par performance when compared to $bs = 1$. We think that this slow-down is caused by the frequent re-allocations the Cuda-backend has to make during the forward pass (and also during the backward pass).

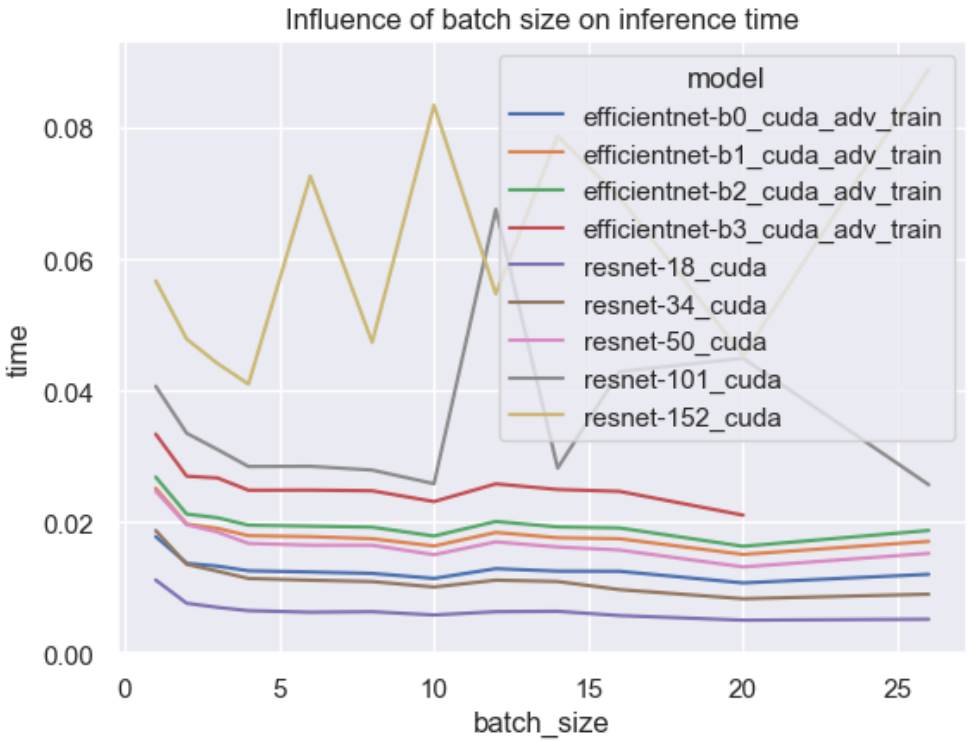


Figure 4.6: ResNet vs EfficientNet inference

4.2.2 Choice of dataset

We carry out all our experiments with ImageNet validation dataset. We pick out only the images containing organism using the mapping $m(c)$ mentioned in 3.1.1 and deem the transfer attack successful if the target loss $\mathcal{L}_{margin}(x_{adv}, 0) < 0$, or in other words the top-1 label is an object label.

For the computation limitations reasons, each experiment was conducted with only the first 500 organism ImageNet validation images.

4.2.2.1 Dataset preprocessing

All the models we use accept input tensors of size $(\text{batch\_size}, 3, 224, 224)$ and they also share the same preprocessing procedure:

```
from torchvision import transforms
transform = transforms.Compose([
 transforms.Resize(256),
 transforms.CenterCrop(224),
 transforms.ToTensor(),
 transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
 ])
```

Usually, this preprocessing is handled dynamically either by the dataloader or by the model itself. The user doesn't have to Resize, Crop or Normalize the inputs manually, but can pass in images of any size for inference or training.

However to speed up the experiments a bit we manually resized and center croped all the ImageNet val images, such that no dynamic resizing and cropping is needed. This saves up a surprising amount computation on our limited hardware.

4.2.3 Baseline

Performance on the first 500 ImageNet validation organism images	
Model	**Foolrate**
Densenet-121	1.8%
EfficientNet-b0	1.0%
EfficientNet-b0-advtrain	1.2%
EfficientNet-b4	0.8%
EfficientNet-b4-advtrain	2.4%
ResNet-18	1.8%
ResNet-50	1.2%
ResNeXt-50 (32x4d)	0.6%
Squeezenet	4.8%
Wide-ResNet-50-2	1.6%

We can see that the models used are pretty good at distinguishing between animate and inanimate things. Any differances in the accuracy (maybe with the exception of Squeezenet) are probably due to the small test set size. When we evaluate 500 test images and get 1% foolrate the 99% binomial confidence interval is (0.216%, 2.804%)

4.2.4 Whitebox attack algorithms

In the whitebox setting we tried the following whitebox optimization algorithms:

- FGSM (Goodfellow et al. [2015])
- Auto-PGD (APGD) L2 (Croce and Hein [2020])
- AdamPDG L2

4.2.4.1 Fast gradient sign method (FGSM)

Fast gradient sign method is basically one-step gradient ascent on the sign of the cross-entropy loss $J(x)$, which would be normally used to train the classifier.

$$x_{adv} = clip(x + \epsilon \cdot sign(\nabla J(x)))$$

Careful reader might wonder: How do we compute a cross-entropy, when our local model doesn't output logits, but uses the max-logits mapping from 3.1.1.3 instead? Well, we don't. We just pretend our surrogate is outputing excatly what it should and everything is fine. Figure 4.7 shows the transferability performance across different local models with varying amount of pertubation size ϵ.

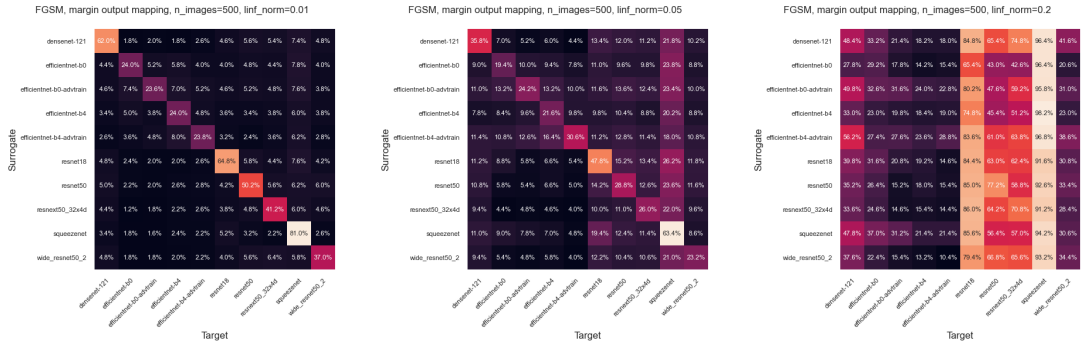


Figure 4.7: FGSM, margin output mapping

To test the impact of different logits mapping function, we also try to pass the logits from the pretrained ImageNet classifier through the $softmax$, sum the two sets of probabilities, take the logarithm and pass that to the FGSM as an attempt to better mimick some binary animal-object classifier. The motivation here was to allow the gradient to pass through more than 2 output activations of the 1000-D ImageNet output vector, hopefullly to provide more fine-grained optimization information to the one step of FGSM.

But the figure 4.8 shows that this approach is approximately equal in transferability to the margin mapping, and for some unknown reason fails to produce whitebox adversarial images for the EfficientNets.

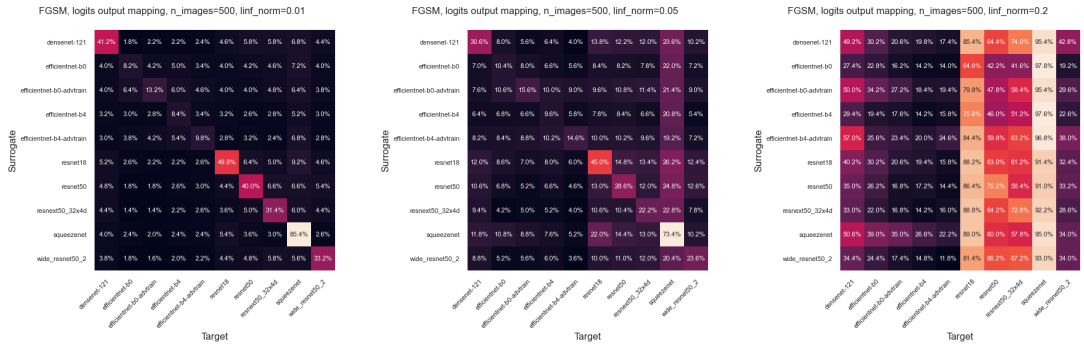


Figure 4.8: FGSM, probability sum mapping

4.2.4.2 Underfitting vs. overfitting

FGSM experiment showed that one gradient optimization step isn't quite enough, because ideally we would like to see the whitebox foolrate (the diagonal in the heatmaps) being close to 100%. If we were to use the analogy from 3.4.2, the diagonal would correspond to the training performance. Anything off the diagonal can be though of as validation performance. The training accuracy of FGSM is already quite low, so we cannot expect the validation accuracy to be much higher. In other words, FGSM massively underfits. It is the extreme case of early-stopping, which in the classical machine learning is sometimes used to prevent overfitting (Caruana et al. [2000]).

4.2.4.3 The need for a better optimizer

We argued in 4.2.4.2 that FGSM underfits badly even on adversarially undefended whitebox networks with large pertubation budget. What we need here is a stronger optimizer.

There have been proposed numerous iterative whitebox attack algorithms that are doing in one form or another a gradient descent (or ascent) on the adversarial loss. Just to name a few:

- Basic iterative method - BIM (Kurakin et al. [2017])
- Projected gradient descent - PGD (Madry et al. [2019])
- Momentum iterative fast gradient sign method - MI-FGSM (Dong et al. [2018])
- Nesterov Iterative Fast Gradient Sign Method - NI-FGSM (Lin et al. [2020])
- Auto-PGD - APGD (Croce and Hein [2020])
- Adam Iterative Fast Gradient Method - AI-FGM (Yin et al. [2021])

Out of those optimization methods we really liked the APGD, because unlike the others, it is hyperparmeter free.

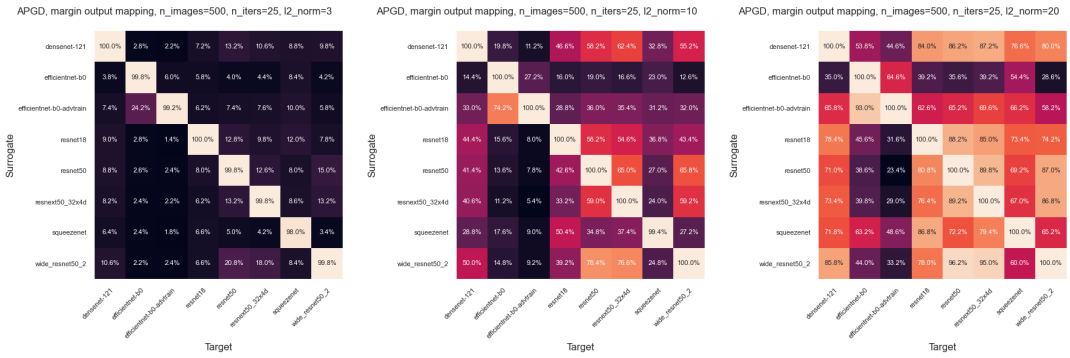
The only knobs to we can tweak are:

- pertubation size
- number of gradient step iterations
- number of gradient samples (when dealing with non-deterministic models)

Actually it is so good at optimizing the adversarial loss, that in our experiments the output probability of the organism class would often go exactly to zero. After computing the cross-entropy loss, which involves taking the logarithm of this probability, we would be getting infinities in the objective and the subsequent gradients would become NaN.

4.2.4.4 APGD baseline

To establish the baseline performance of APGD and its ability to optimize our custom binary classification loss, we tried a number of different l_2 norm budgets and executed it with 25 iterations, which is the lowest number of gradient steps the authors use in their comparisions APGD to classical PGD (Croce and Hein [2020]). We used this low-end of the N_{iters} APGD boundary to keep the experiment running times under control.



(a) $\epsilon = 3$ (b) $\epsilon = 10$ (c) $\epsilon = 20$

Figure 4.9: APGD L2 baseline

What is obvious is that the 25 iterations did its work and took the training loss to zero. The whitebox foolrate is now 100% in almost all cases, even when the pertubations are small ($l_2 = 3$). While succeeding on the diagonal, the biggest problem now is the overfitting to one particular surrogate.

Figures 4.10, 4.11 and 4.12 show some examples of the adversarial images produced:

(a) $\epsilon = 3$ (b) $\epsilon = 10$ (c) $\epsilon = 20$

Figure 4.10: APGD L2 baseline ResNet-18

It is interesting to see how the adversarially trained EfficientNet-b0 is more sensitive to the dog's nose and mouth area in the dog's image, while the ResNets produce pertubations that are much more uniform.

(a) $\epsilon = 3$ (b) $\epsilon = 10$ (c) $\epsilon = 20$

Figure 4.11: APGD L2 baseline ResNet-50

(a) $\epsilon = 3$ (b) $\epsilon = 10$ (c) $\epsilon = 20$

Figure 4.12: APGD L2 baseline EffecientNet-b0-advtrain

4.2.5 Augmentation is all you need!

Seeing how little transferability there is between the models from the ResNet family and the EfficientNets, we have started experimenting with some image augmentations techniques, which are commonly used to to prevent overfitting and enhance generalization when training normal CNN classifiers. We have tried augmenting images with:

- Guassian-noise
- Blur
- Elastic transformation
- Affine transformation

4.2.5.1 Guassian-noise augmentation

Wu and Zhu [2020] show how convolving the loss surface with a Gaussian filter has a smoothing effect on the gradient.

$$J_\sigma(x) = \mathbb{E}_{\xi \sim \mathcal{N}(0,I)}[J(x + \sigma\xi)]$$
$$\nabla J_\sigma(x) = \mathbb{E}_{\xi \sim \mathcal{N}(0,I)}[\nabla J(x + \sigma\xi)]$$

In their experiments they visualize the saliency map of the gradient $\nabla J_\sigma(x)$ with increasing values of σ. They go on to show how increasing the values of σ filters out the noise in the gradient significantly while still capturing the most significant semantic information.

We confirm this observation in our experiments. In figure (4.13) we demonstrate how increasing the σ compels the optimization to focus only on a smaller part of the image. It cannot afford to spread the pertubation around the image uniformly, because low-intensity signal is destroyed by the Guassian noise. It must produce a pertubation in a smaller area but with higher intensity, to keep the pertubation's signal-to-noise ratio high.

Because the surrogate is stochastic and during each forward pass applies a random transformation $t \sim T$, we actually optimize the expected surrogate loss in a Expectation over Transformation (EoT - Athalye et al. [2018]) style:

$$J_T(x) = \mathbb{E}_{t\sim T}[J(t(x))]$$
$$\nabla J_T(x) = \mathbb{E}_{t\sim T}[\nabla J(t(x))]$$

We estimate the true $\nabla J_T(x)$ using the sample mean estimator.

$$\widehat{\nabla} J_T(x) = \frac{1}{N_{EoT}} \sum_{i=1}^{N_{EoT}} \nabla J(t(x))$$

The N_{EoT} is somewhat analogous to a batch size in SGD neural net training.

In figure 4.14 explore, whether this iterated sampling is even necessary, or whether the noise in the gradient could be handled by the APGD. We set a constant query bugdet of 250 and run the APGD in two configurations: $N_{EoT} = 1$ and $N_{EoT} = 10$. The results suggest that sampling the gradient only once is not enough, but it's hard to say what's the optimal N_{EoT} given a constant total query budget.

Intuitively, noisier gradients need more N_{EoT} samples. If we take first order Taylor approximation of the loss - $J(x + \epsilon) \approx J(x) + \epsilon \nabla J(x)$, then:

$$std(\nabla J(x + \sigma\mathcal{N}(0, I))) = std(\nabla J(x) + \sigma\mathcal{N}(0, I)\nabla^2 J(x)) = \sigma\nabla^2 J(x) = \sigma c$$

The standard deviation of the sample mean of the gradient would be:

$$std(\widehat{\nabla} J_T(x)) = std(\frac{1}{N_{EoT}} \sum_{i=1}^{N_{EoT}} \nabla J(x + \sigma\mathcal{N}(0, I))) =$$

$$= \frac{1}{N_{EoT}} \sqrt{N_{EoT}}\, std(\nabla J(x + \sigma\mathcal{N}(0, I))) = \frac{1}{\sqrt{N_{EoT}}}\sigma c$$

So to keep the gradient estimate noisiness constant, we would need to scale N_{EoT} quadratically with the gaussian noise's σ.

Figure 4.14 also shows that when the total query budget is set at 250, the optimal Guassian-noise augmentation σ is somewhere near $\sigma = 18$.

This is in agreement with Wu and Zhu [2020], where they find the distortion level $\sigma = 15$ to be performing the best.

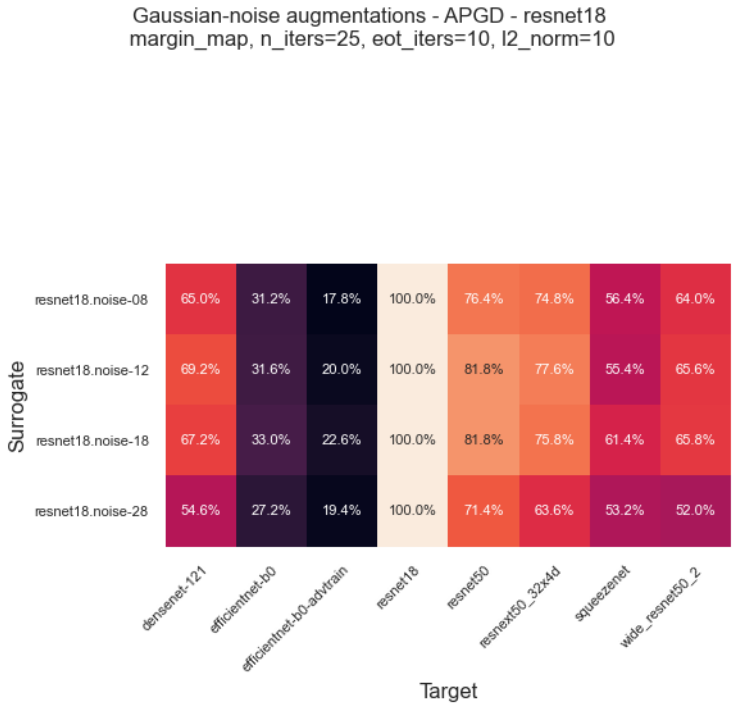
To explore how much the additional gradient sampling helps, we have set σ to an excessive $\sigma = 35$ and ran experiments with $N_{EoT} \in \{1, 3, 10, 30, 100\}$.

(a) $\sigma = 2$ (b) $\sigma = 4$ (c) $\sigma = 8$

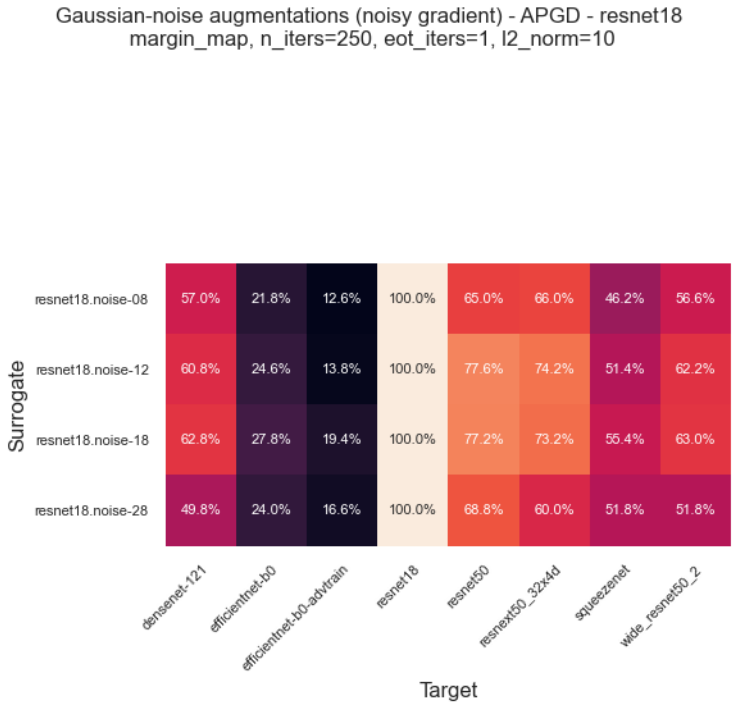
(d) $\sigma = 12$ (e) $\sigma = 16$ (f) $\sigma = 20$

(g) $\sigma = 24$ (h) $\sigma = 28$ (i) $\sigma = 32$

Figure 4.13: Guassian noise augmentation ResNet-18



(a) 25 iterations, 10 gradient estimates per step



(b) 250 iterations, 1 gradient estimate per step

Figure 4.14: Guassian noise augmentation

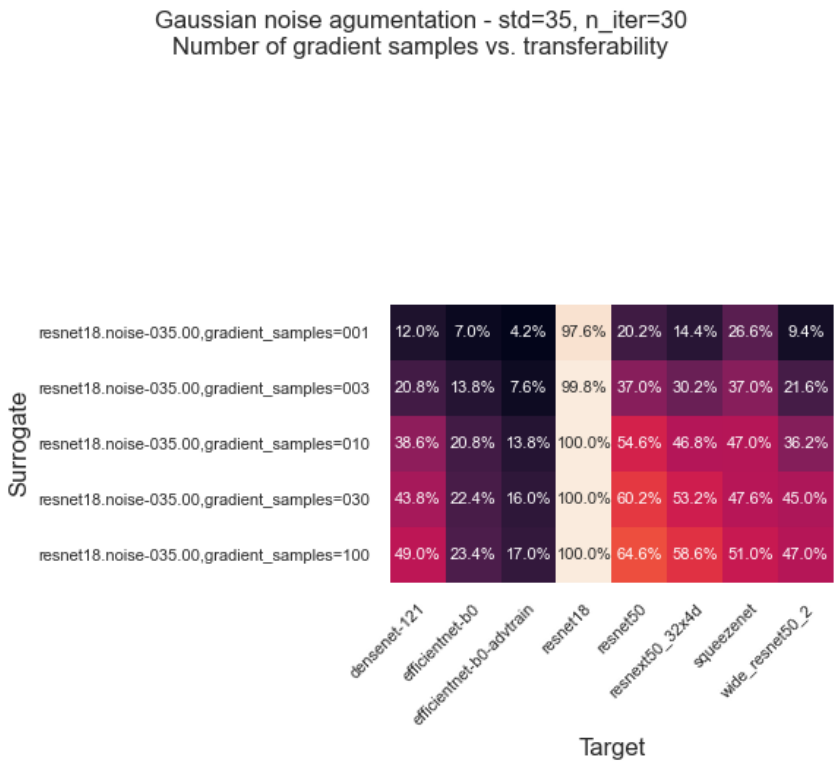


Figure 4.15: The effect of N_{EoT} in noisy gradient situation

Figure 4.15 shows the foolrate steadily increasing up to $N_{EoT} = 100$. It is a reminder that we shouldn't underestimate the EoT gradient sampling when using very noisy stochastic augmentations, or when stacking severeal stochastic augmentations in the preprocessing pipeline of our surrogates, in which case the gradient noise is multiplied at each stage.

4.2.5.2 Box-blur

In this experiment we have tested the effect of box-blur on adeversarial-image transferability. Box-blur augmentation applies uniform normalized convolution kernel of size x over the input image.

We can see in figure 4.16 that a small amout of blur is beneficial, but larger amounts are essentially strong low-pass filters that introduce large surrogate bias. Large blur makes the surrogate a much weaker classifier, which is quite easy to fool at train-time, but when the blurring is removed from the preprocessing pipeline at test-time, the model becomes stronger again and isn't fooled by the adversarial image produced on the weak surrogate.

Figure 4.17 shows the effect of increasing the kernel size of the box-blur on the frequency of the adversarial pertubation. Bigger blurring kernels lower the pertubation frequency, because the high-freqency information cannot pass through the blur averaging effectively.

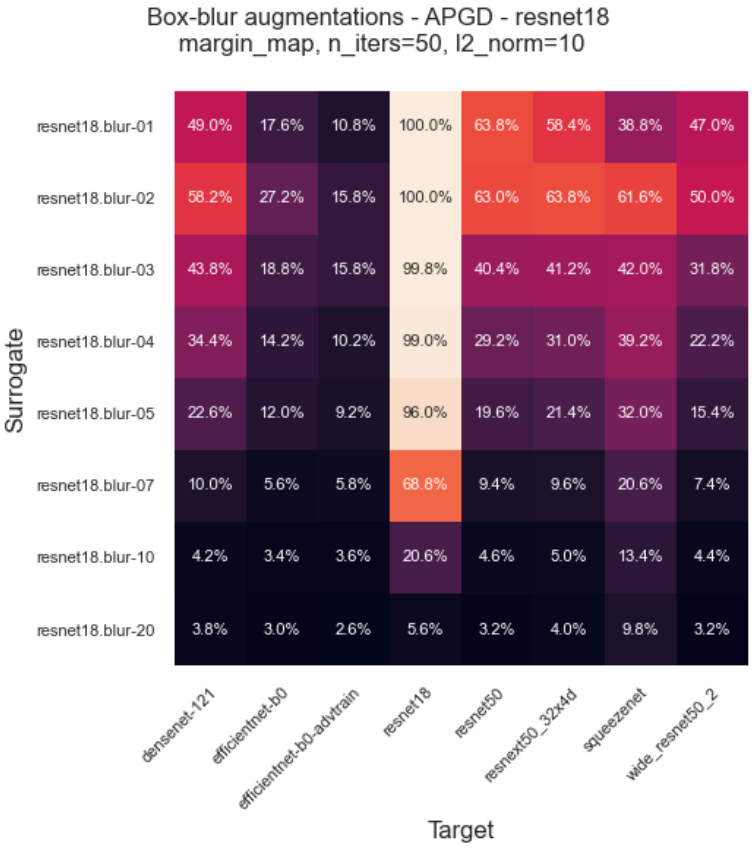


Figure 4.16: APGD box-blur augmentation

(a) kernel size = 1 (b) kernel size = 2 (c) kernel size = 3

(d) kernel size = 4 (e) kernel size = 5 (f) kernel size = 7

(g) kernel size = 10 (h) kernel size = 15 (i) kernel size = 20

Figure 4.17: Effect of box-blur kernel size on the frequency of adversarial pertubation

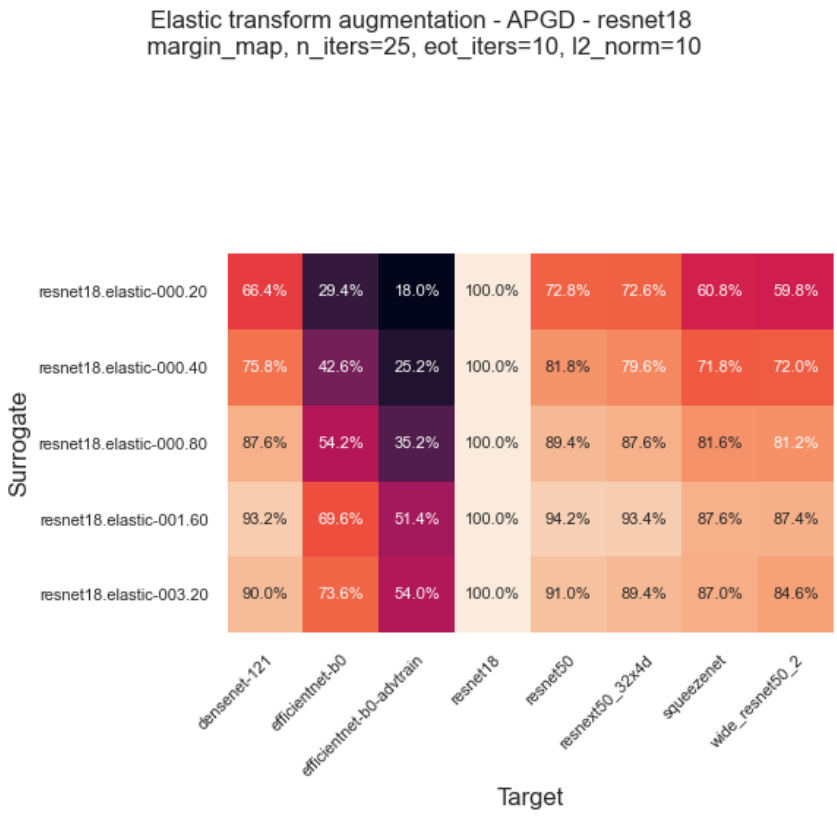


Figure 4.18: APGD elastic transformation augmentations

4.2.5.3 Elastic transformation

Qiu et al. [2020] compare various stochastic image augmentations as an adversarial attack defense. If we turn this around, by making the crafted adversarial images robust to those defensive augmentations, we might achive better transferability to models with such stochastic defenses. On top of that, our adversarial images will be robust to common image manipulations, which might easily happen if the preprocessing of the target model differs from the surrogate.

This robustness might be quite important, as we have no information about the image preprocessing that happens in cloud target.

Inspired by the defensive augmentations suggested by this the paper, we've added stochastic elastic transformarmation to our augmentation toolbox, because it was one of the best defensive augmentations when attacked by iterative FGSM. But the combination of APGD together with gradient averaging was enough to break this surrogate defense and in turn produce robust adversarial images that really transfer.

Figure 4.18 proves the potential of this augmentation, as it brought the transferability from ResNet-18 to EfficientNet-b0-advtrain to 54.0%, which is a significant improvement over the baseline 8%. It also outperformed by a significantly margin the 22% achieved by gaussian noise and the 16% achieved by box-blur.

One disadvantage of it is that due to its non-linearity, the forward and backward pass is really expensive.

On figure 4.19 we can see similar things happening as in 4.13. With stronger

elastic deformations the optimization cannot just put its decieving pertubations anywhere. Noise-like pertubations produced by the baseline on ResNet-18 are no longer an option. Their highly-tuned overfit nature makes them fragile to small image manipulations. The strong elastic regularization forces the APGD to change or somehow cover the semantic information in the image instead. In this case this means focusing on the dog instead of the wall and grass behing. The pertubations also seem to be more spatially consistent. Another thing to point out is how diverse are the types of patterns created at different levels of distortion.

Out of curiosity we also ran these sample images against Google Vision (figure 4.20). To our surprise Google Vision already makes some mistakes, even when using such a weak ResNet-18 surrogate. All successfully misclasified examples had their pertubations focused on the head of the dog, which highlights the importance of focusing on the important semantic features in the image.

4.2.5.4 Affine transformation

Another augmentation defense coming on top in Qiu et al. [2020] was a random affine transformation. Kornia library offers this type of augmentation as well so we ran a few experiments to compare it to the elastic augmentation in 4.2.5.3.

The Kornia's affine augmentation has many tunable parameters, all of which are given as ranges, that are uniformly sampled from.

- scale - the amount of isotropic scaling
- rotation - the range of degrees of rotation
- shift - how much to translate the image
- shear - the amount of shear in x and y direction

These 4 parameters make the hyperparameter search space larger than what was the case for previous augmentation. Figure 4.21 shows an incomplete grid-search over those 4 hyperparameters.

The best configuration turned out to be 30 degree shear and small random shift of 0.2. It even beat the elastic augmentation in its transferability to EfficientNets-b0-advtrain with foolrate of 56.2%.

This crude experiment shows there is difinitely a large room for improvement. The shear probably substitutes the elastic deformation and shift makes sure that only the important parts of the image are modified, because only a portion of the image is visible after each random translation.

The unreasonable effectiveness of the random shift suggests that Random Sized Padding Affine (RSPA) augmentation (Qiu et al. [2020]) might be worth trying in the future.

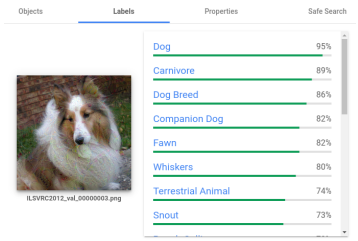
(a) $\alpha = 0$ (b) $\alpha = 0.2$ (c) $\alpha = 0.4$

(d) $\alpha = 0.8$ (e) $\alpha = 1.6$ (f) $\alpha = 2.4$

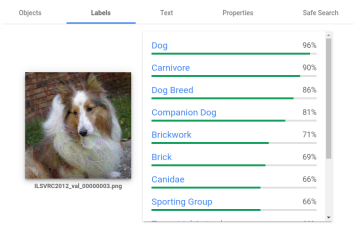
(g) $\alpha = 3.2$ (h) $\alpha = 4.0$ (i) $\alpha = 6.4$

(j) $\alpha = 9$ (k) $\alpha = 12.8$

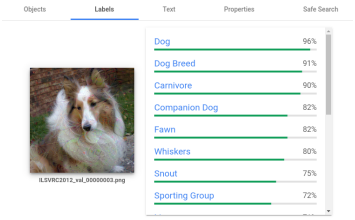
Figure 4.19: Elastic augmentation ResNet-18



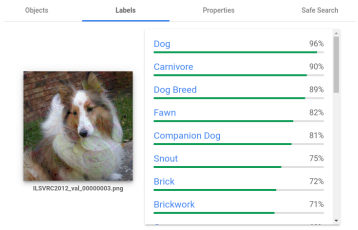
(a) $\alpha = 0$



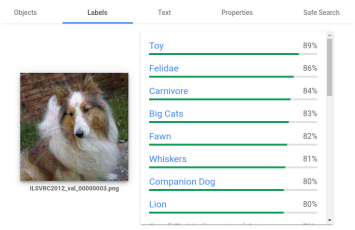
(b) $\alpha = 0.2$



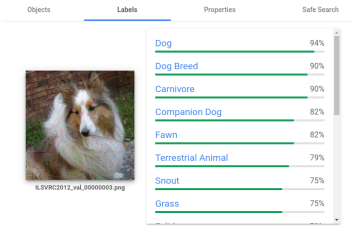
(c) $\alpha = 0.4$



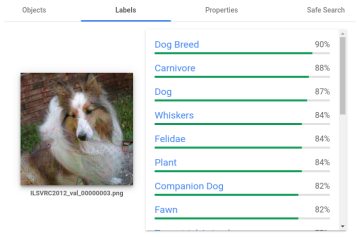
(d) $\alpha = 0.8$



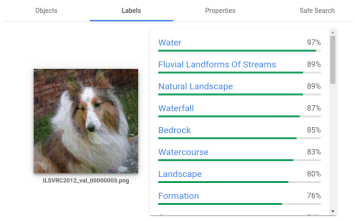
(e) $\alpha = 1.6$



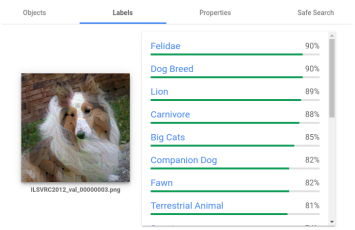
(f) $\alpha = 2.4$



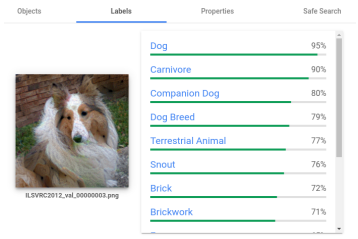
(g) $\alpha = 3.2$



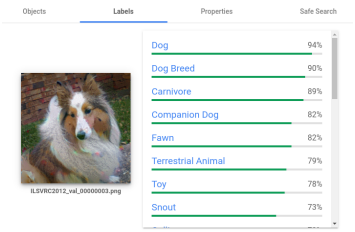
(h) $\alpha = 4.0$



(i) $\alpha = 6.4$



(j) $\alpha = 9$



(k) $\alpha = 12.8$

Figure 4.20: Elastic dog from ResNet-18 to GVision

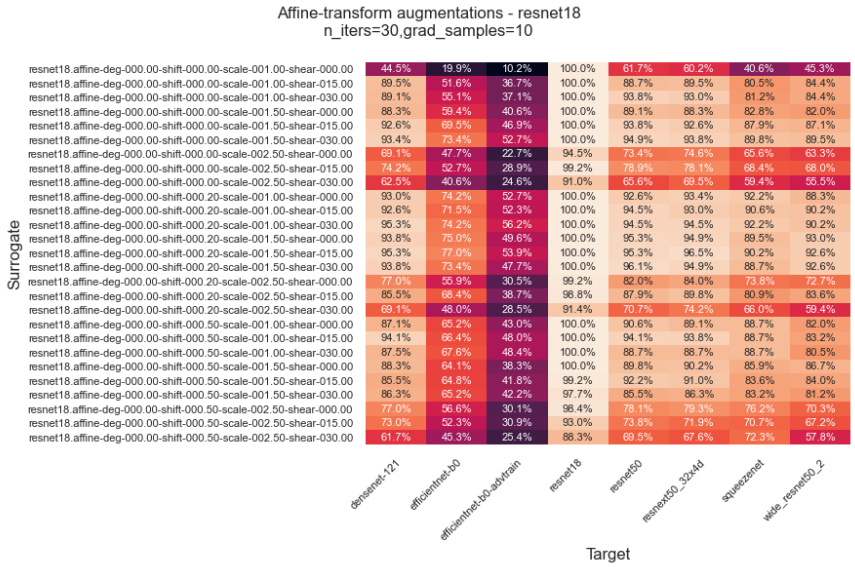


Figure 4.21: APGD elastic transformation augmentations (TODO - remove the 'deg' param and format the captions better)

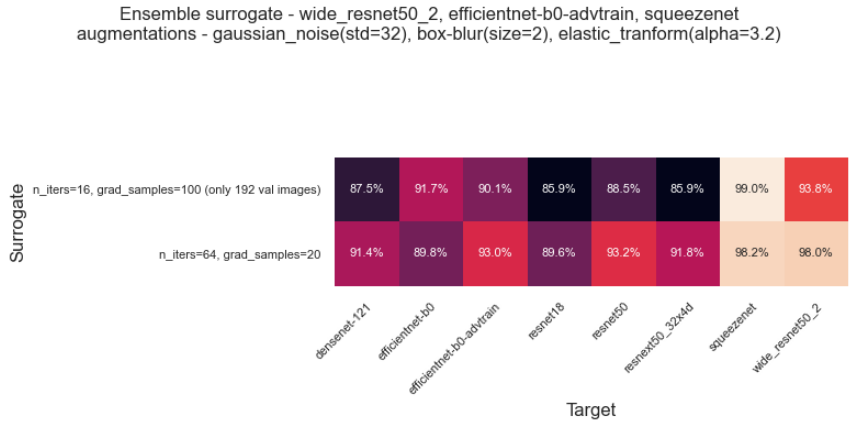


Figure 4.22: Ensemble with augmentations

4.2.5.5 Ensemble

To put it all together, we have created an ensemble of Wide-ResNet-50-2, EfficientNet-b0 and Squeezenet with equal voting weights as our surrogate. Their scores were combined by summing their output logits. We chose these 3 models because they appear to be complemenary in their transfer performance. We've added the Guassian noise augmentation with $\sigma = 32$, box-blur with kernel size $= 2$, and the elastic augmentation with $\alpha = 3.2$ to the mix, as those augmentations seemed to perform well.

The gaussian noise and the random elastic transform in the pipeline meant higher noise in the gradient, so we've also increased the N_{EoT} to 20 and 100 respectively.

Figure 4.22 shows the results, which are all above 90%. The $N_{EoT} = 20$ version compensates for the lower number of N_{EoT} by more APGD iterations. The version with extensive gradient sampling was run only for 16 iterations,

Figure 4.23: "Oh, you think darkness is your ally. But you merely adopted the dark; I was born in it, molded by it." - The Dark Knight Rises

which still adds up to more total ensemble queries. The performance difference shouldn't be compared too much, because we were forced to stop the 16-iteration experiment only after 192 validation images.

The attachment A.2 contains a couple of randomly chosen example adversarial images. Some of them are really creepy as can be seen in the case of a dog having a mask clearly inspired by the adversary Bane from the Batman movie (4.23).

4.3 Transferability evaluation on Google Cloud Vision

In this section we take the adversarial images that had good transferability across all of our local models and send them to the Google Cloud Vision for a final evaluation. More precisely, we take the 500 output images of the $N_{EoT} = 20$ version of 4.22 and compare the top-1 labels to the top-1 labels of the clean images

4.3.1 Choice of evaluation metrics

As it turns out, evaluating the transferability using the objective from 3.1.1 may not be the most informative evaluation metric. ImageNet dataset with its numerous dog breeds could be better decribed as a "pet dataset". What happened was that many adversarial animal images, which were successfully misclassified as "not animals" were in many cases assigned the top-1 Google Cloud Vision label "Plant". Our initial binary objective choice of "organism" vs. "object" meant that this misclassification wouldn't show up in the stats as successful a attack, because "Plant" is also an organism. That's why we chose to evaluate the transfer results on Google Cloud Vision differently. More details about the evaluation are described in the next section (4.3.2)

Figure 4.24: Top-1 baseline labels

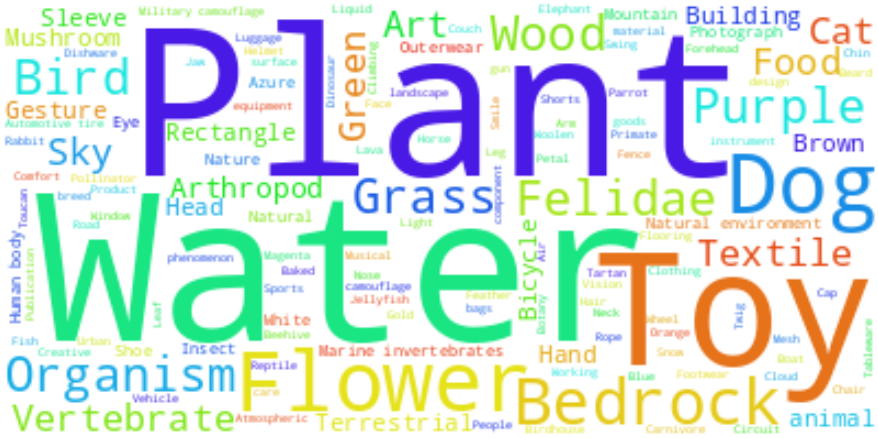


Figure 4.25: Top-1 labels of adversarial images produced by our ensemble with augmentations (4.2.5.5)

4.3.2 Wordcloud

As the objective used to guide the whitebox optimization doesn't illustrate properly the transferability success we have made, we have decided to use maybe a little unorthodox evaluation criteria.

In figure 4.24 and 4.25 we use a wordcloud visualization to show how the top-1 label assigned by the Google Cloud Vision changes when the ImageNet organism validation images are pertubed by our augmented ensemble transfer attack.

As we can see, the cheapest option for the attack is to make the prevailing dogs nd birds look like a "Plant". This is probably caused by the fact that animal images are often taken in a nutural environment with the "Plants" often being a part of the original image. The "Plant" semantics often appears in the clean image already and is thus the easiest category to target. The ImageNet dataset doesn't contain almost any plant species, so the whitebox optimization often doesn't have an any problems of making the adversarial images look like some kind of a plant species.

5. Future work

5.1 Hybrid attacks

Explore the combination of transfer-based attack with iterative methods mentioned in 3.4.1

5.1.1 Finetuning the surrogate during the attack

Some blackbox attacks train their surrogates to mimick the outputs of a target blackbox. The way this is usually done is that the target is queried multiple times with synthetic data and the outputs are collected to form a training set. Surrogates are then trained on this synthetic dataset and as a result their gradient are more alinged with the target. Papernot et al. [2017]

The problem with this approach is that to clone the target in this way requires way too much queries.

Alternative approach can be to only tune the voting weights in ensemble. This parameter space is orders of magnitude smaller than the space of all model weights. The hope is that this could achive fast ensemble adaptation with only a few queries.

5.2 Combining augmentations

As we have said in 4.2.5.4, the particular configuration of the augmentation pipeline definitely isn't optimal, but we simply didn't have enough time to make further experiments. We especially didn't experimented with multiple augmentations stacked on top of each other, which may be a promising direction for the future.

5.3 Stronger models in the ensemble

To keep the local experiments comparable we have sticked to the model selection made in the beginning. That said, the only adversarially robust model in the pool is the EfficientNet-b0-advtrain. It might be interesting to see the difference of having an ensemble of only adversarially trained models for example.

Stronger surrogate could be also achived with the use of technique described in Li et al. [2019], where they make up for the low surrogate ensemble diversity by creating a group of dynamically generated "ghost-networks". Similarly to dropout regularization, which can be interpreted as a regularization method creating an always-changing ensemble with randomly dropped feed-forward connections, these "ghost-networks" take this idea a step further. They randomly drop (amongst other connetions) the skip-connections that are frequent in the ResNet style networks, generating diverse ensemble of surrogates on the fly without any additional resource requirements.

6. Conclusion

Bibliography

Anaconda software distribution, 2020. URL `https://docs.anaconda.com/`.

Maksym Andriushchenko, Francesco Croce, Nicolas Flammarion, and Matthias Hein. Square attack: a query-efficient black-box adversarial attack via random search. In *ECCV*, 2020.

Anish Athalye, Logan Engstrom, Andrew Ilyas, and Kevin Kwok. Synthesizing robust adversarial examples, 2018.

B. Biggio, I. Corona, Davide Maiorca, Blaine Nelson, Nedim Srndic, P. Laskov, G. Giacinto, and F. Roli. Evasion attacks against machine learning at test time. *ArXiv*, abs/1708.06131, 2013.

Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006. ISBN 0387310738.

Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks, 2017.

R. Caruana, S. Lawrence, and C. Lee Giles. Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. In *NIPS*, 2000.

J. Chen and Quanquan Gu. Rays: A ray searching method for hard-label adversarial attack. *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020.

Pin-Yu Chen, Huan Zhang, Yash Sharma, Jinfeng Yi, and Cho-Jui Hsieh. Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, 2017.

Francesco Croce and Matthias Hein. Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks, 2020.

Francesco Croce, Maksym Andriushchenko, Naman D. Singh, Nicolas Flammarion, and Matthias Hein. Sparse-rs: a versatile framework for query-efficient sparse black-box adversarial attacks. *ArXiv*, abs/2006.12834, 2020.

Francesco Croce, Maksym Andriushchenko, Vikash Sehwag, Edoardo Debenedetti, Nicolas Flammarion, Mung Chiang, Prateek Mittal, and Matthias Hein. Robustbench: a standardized adversarial robustness benchmark, 2021.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

Yinpeng Dong, Fangzhou Liao, Tianyu Pang, Hang Su, J. Zhu, Xiaolin Hu, and Jianguo Li. Boosting adversarial attacks with momentum. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9185–9193, 2018.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks, 2018.

Z. Huang and Tong Zhang. Black-box adversarial attack with transferable model-based embedding. *ArXiv*, abs/1911.07140, 2020.

Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and ¡0.5mb model size, 2016.

Andrew Ilyas, Logan Engstrom, Anish Athalye, and Jessy Lin. Black-box adversarial attacks with limited queries and information. *ArXiv*, abs/1804.08598, 2018.

J. Z. Kolter and Eric Wong. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *ICML*, 2018.

Alexey Kurakin, I. Goodfellow, and S. Bengio. Adversarial examples in the physical world. *ArXiv*, abs/1607.02533, 2017.

Y. LeCun and Yoshua Bengio. Convolutional networks for images, speech, and time series. 1998.

Yingwei Li, Song Bai, Yuyin Zhou, Cihang Xie, Zhishuai Zhang, and Alan Yuille. Learning transferable adversarial examples via ghost networks, 2019.

Jiadong Lin, Chuanbiao Song, Kun He, Liwei Wang, and John E. Hopcroft. Nesterov accelerated gradient and scale invariance for adversarial attacks, 2020.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks, 2019.

Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012. ISBN 0262018020.

Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings, 2015.

Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning, 2017.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019.

Han Qiu, Yi Zeng, Tianwei Zhang, Yong Jiang, and Meikang Qiu. Fencebox: A platform for defeating adversarial examples with data augmentation techniques, 2020.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

Jonas Rauber, Roland S. Zimmermann, M. Bethge, and W. Brendel. Foolbox native: Fast adversarial attacks to benchmark the robustness of machine learning models in pytorch, tensorflow, and jax. *J. Open Source Softw.*, 5:2607, 2020.

Edgar Riba, Dmytro Mishkin, Daniel Ponsa, Ethan Rublee, and Gary Bradski. Kornia: an open source differentiable computer vision library for pytorch, 2019.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge, 2015.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

Fnu Suya, Jianfeng Chi, David Evans, and Y. Tian. Hybrid batch attacks: Finding black-box adversarial examples with limited queries. *ArXiv*, abs/1908.07000, 2020.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks, 2014.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision, 2015.

Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks, 2020.

Florian Tramèr, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. The space of transferable adversarial examples, 2017.

Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. Ensemble adversarial training: Attacks and defenses, 2020.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Huggingface's transformers: State-of-the-art natural language processing, 2020.

Lei Wu and Zhanxing Zhu. Towards understanding and improving the transferability of adversarial examples in deep neural networks. In *ACML*, 2020.

Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks, 2017.

Han Xu, Yao Ma, Haochen Liu, Debayan Deb, Hui Liu, Jiliang Tang, and Anil K. Jain. Adversarial attacks and defenses in images, graphs and text: A review, 2019.

Heng Yin, Hengwei Zhang, Jindong Wang, and Ruiyu Dou. Boosting adversarial attacks on neural networks with better optimizer. *Secur. Commun. Networks*, 2021:9983309:1–9983309:9, 2021.

F. Yu, Zirui Xu, Yanzhi Wang, Chenchen Liu, and X. Chen. Towards robust training of neural networks by regularizing adversarial gradients. *ArXiv*, abs/1805.09370, 2018.

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks, 2017.

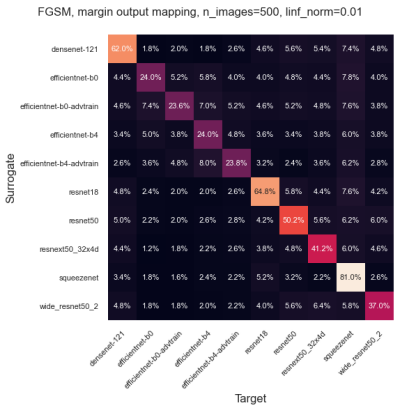
List of Figures

List of Tables

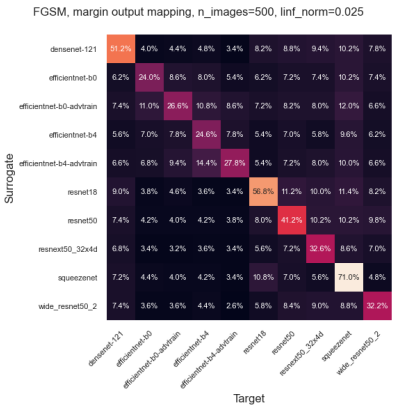
List of Abbreviations

A. Attachments

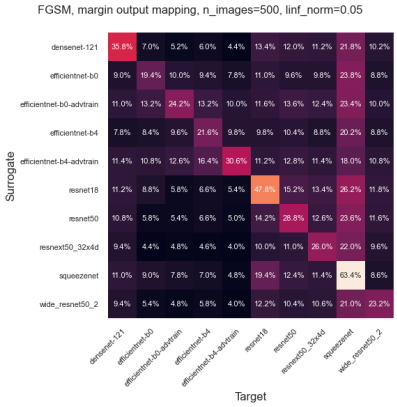
A.1 FGSM local transfer experiments



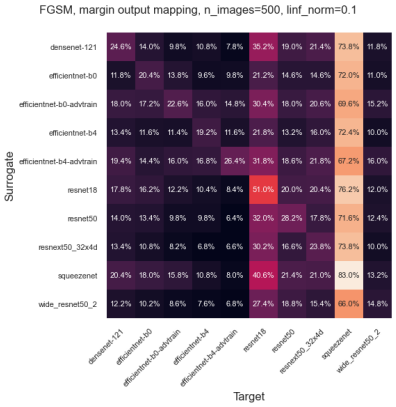
(a) $\epsilon = 0.01$



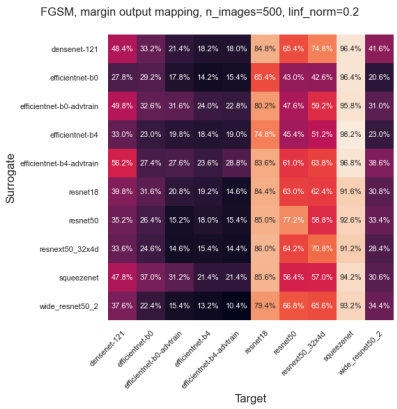
(b) $\epsilon = 0.025$



(c) $\epsilon = 0.05$

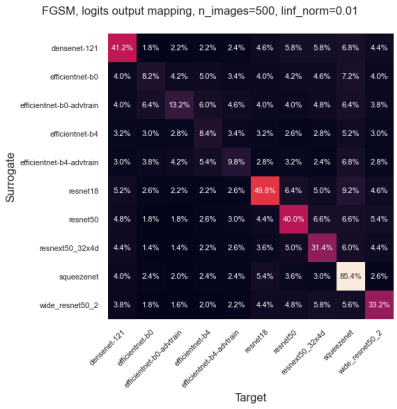


(d) $\epsilon = 0.1$

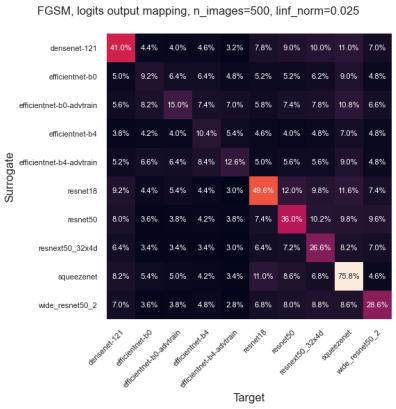


(e) $\epsilon = 0.2$

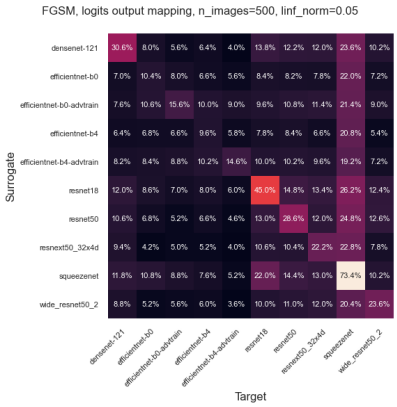
Figure A.1: FGSM, margin output mapping



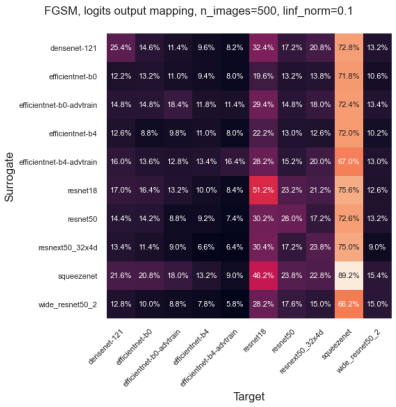
(a) $\epsilon = 0.01$



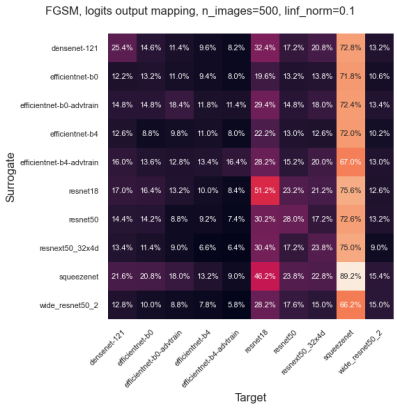
(b) $\epsilon = 0.025$



(c) $\epsilon = 0.05$



(d) $\epsilon = 0.1$



(e) $\epsilon = 0.2$

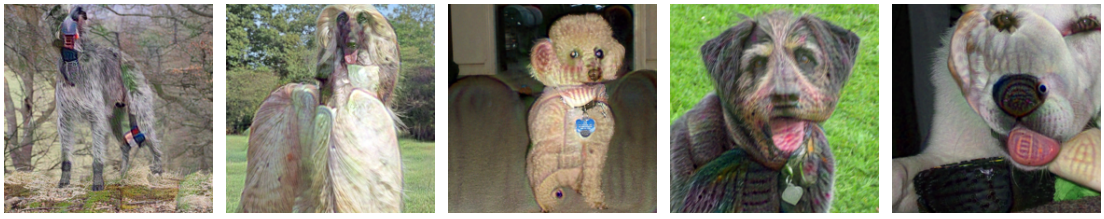
Figure A.2: FGSM, probability sum output mapping

A.2 Ensemble with augmentations - sample images

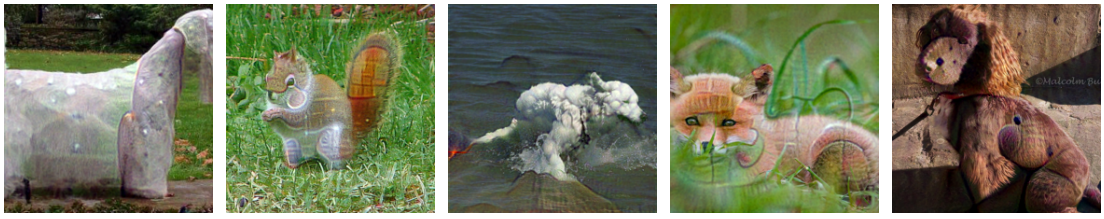
(a) 3 (b) 48 (c) 57 (d) 77 (e) 79

(f) 102 (g) 170 (h) 247 (i) 248 (j) 283

(k) 300 (l) 355 (m) 370 (n) 391 (o) 399



(p) 405 (q) 424 (r) 425 (s) 441 (t) 450

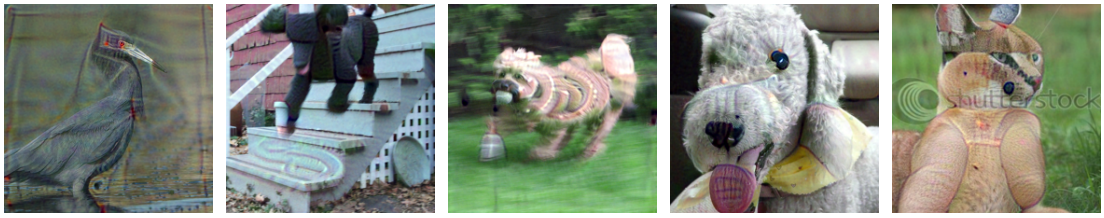


(u) 474 (v) 502 (w) 507 (x) 517 (y) 539

Figure A.3: Ensemble with augmentations sample images 1)

(a) 588 (b) 597 (c) 613 (d) 694 (e) 708

(f) 716 (g) 719 (h) 756 (i) 774 (j) 788

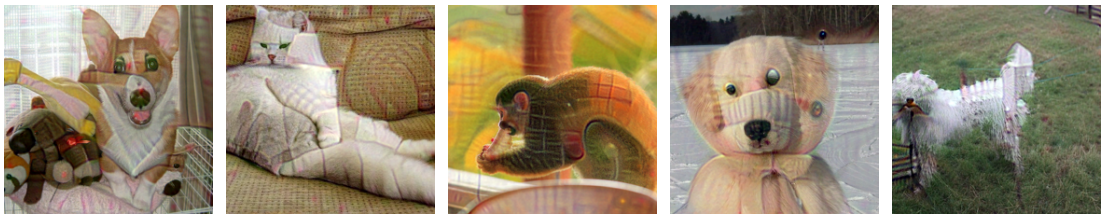


(k) 794 (l) 800 (m) 882 (n) 883 (o) 898

(p) 909 (q) 945 (r) 946 (s) 974 (t) 975

(u) 997 (v) 1028 (w) 1032 (x) 1065 (y) 1069

Figure A.4: Ensemble with augmentations sample images 2)



(a) 1096 (b) 1101 (c) 1104 (d) 1112 (e) 1113

(f) 1121 (g) 1132 (h) 1136 (i) 1138 (j) 1152

Figure A.5: Ensemble with augmentations sample images 3)

A.3 AdvPipe source code

Source codes for AdvPipe along with the experiment results are freely available at github.com/kubic71/bachelors-thesis.