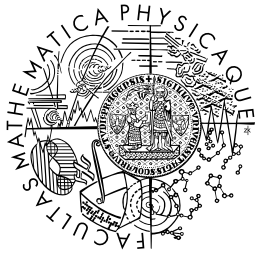


FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

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 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 on AI SaaS providers like Google Vision API, Amazon recognition, or Clarify. Understandably, attacking an unknown system will be more challenging than attacking a known neural network in the testing stage. We will try to measure this attack difficulty increase. 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 some of those services are proven to be vulnerable, this would 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.

2. Preliminaries

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 we define and explain adversarial attacks and defenses, adversarial examples, adversarial robustness and different adversarial attack threat models.

2.1 Machine learning

Goodfellow et al. [2016]

2.2 Deep neural networks

2.3 DNN training

2.4 Convolutional neural networks

2.5 Adversarial attacks on DNNs

2.6 Threat models

3. Related Work

3.1 Title of the first subchapter of the third chapter

TODO: rewrite your review papernotes.md here.

4. Our approach

4.1 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.

4.1.1 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.

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 $m_{local}(c)$:

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

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.

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].

4.2 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.

- 4.2.1 TREMBA (Huang and Zhang [2020])
- 4.2.2 RayS (Chen and Gu [2020])
- 4.2.3 SquareAttack (Andriushchenko et al. [2020])
- 4.2.4 Sparse-RS (Croce et al. [2020])

TODO: describe each attack briefly and show sample images

4.2.1 TREMBA

4.2.2 RayS

4.2.3 SquareAttack

4.2.4 Sparse-RS

We go into more details in the Experiments 5

4.3 The need for query-efficient attack

In the previous section 4.2 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.

4.4 Leveraging transferability

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

4.4.1 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.

4.4.2 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.

4.4.3 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.

4.4.4 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 4.4 using the same prioritization candidate scoring mechanism.

4.5 The need for attack pipeline

We argued in 4.4 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.

4.5.1 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.

4.5.2 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.

4.6 AdvPipe

4.6.1 The vision

Solves all our problems. At least in theory.

4.6.2 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.

4.7 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.

4.7.1 Attack regimes

Cooperative iterative regime

Transfer regime

Transfer regime with multiple targets

4.7.2 Attack algorithms

Whitebox

Blackbox

4.7.3 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 4.1.1, 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.

Preprocessing

In chapter 5 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]).

4.7.4 Configuration

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

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.

4.7.5 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.

5. Experiments

5.1 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

5.1.1 Baseline

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

Figure 5.1: Cat and Shark, GVision baseline

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

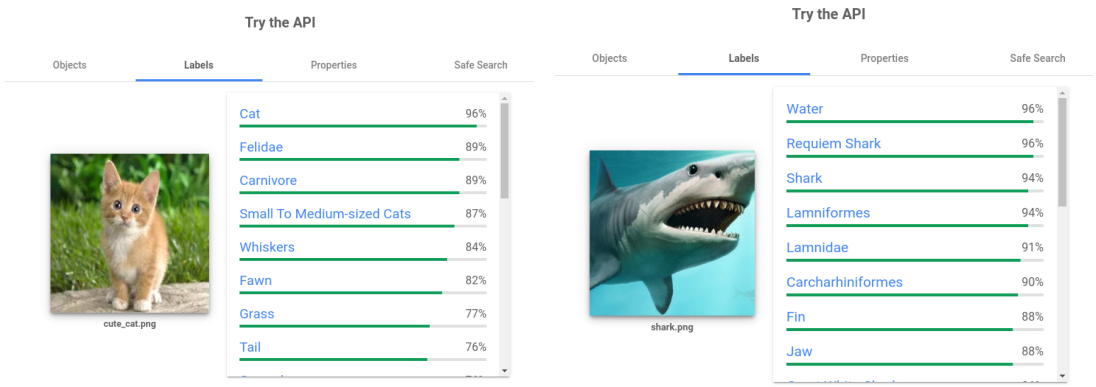


Figure 5.2: Cat and Shark, GVision baseline

5.1.2 TREMBA

5.1.3 RayS

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

5.1.4 SquareAttack

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

SquareAttack L2

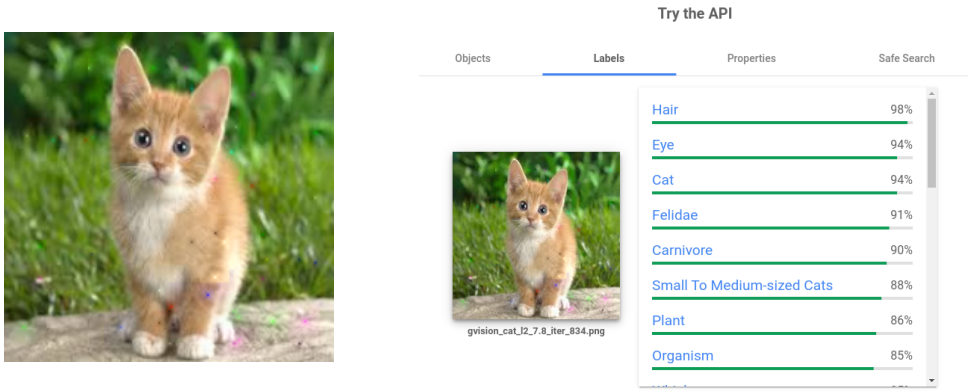


Figure 5.3: SquareAttack, 834 queries, pertubation norm $l_2 = 7.84$

SquareAttack Linf

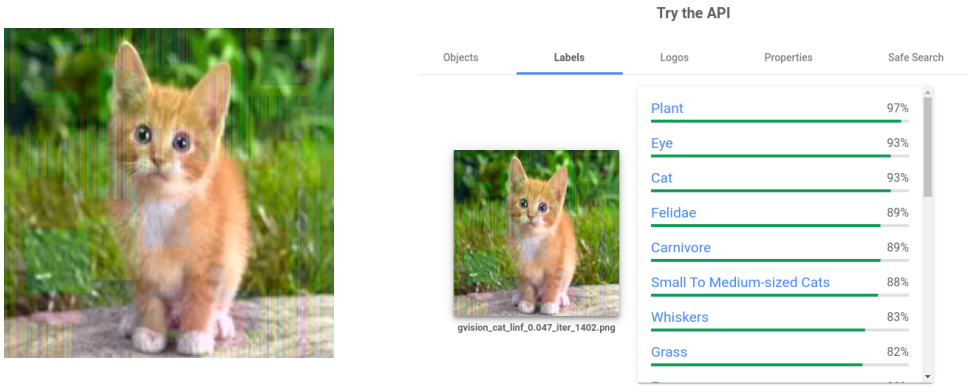


Figure 5.4: SquareAttack, 1402 queries, pertubation norm $l_{inf} = 0.047$

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.

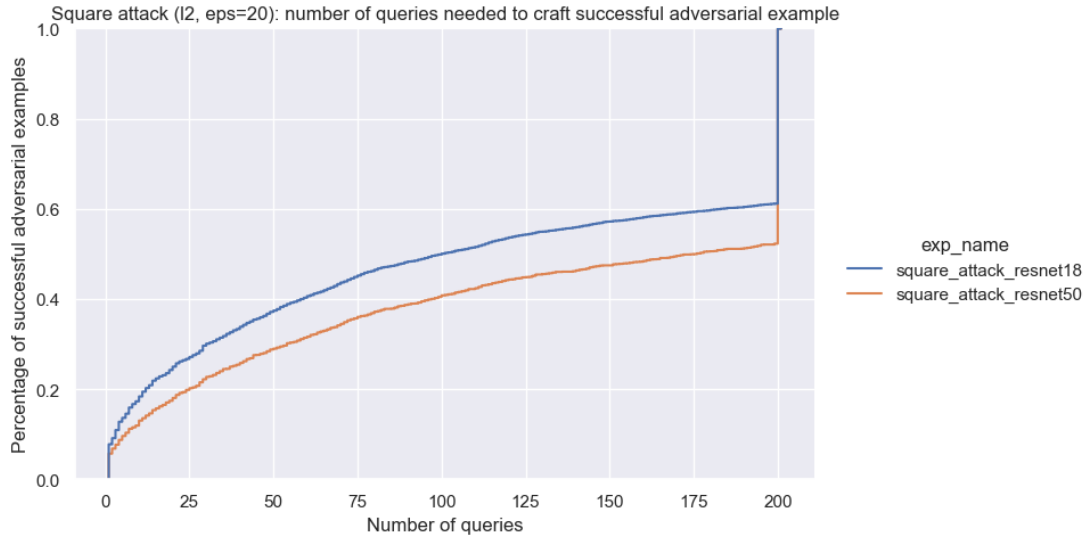
In this experiment of sample size = 1 the L2 version of SquareAttack seems to produce much less visually perceptible pertubation and is able to achive the top-1 misclassification with l_2 norm of only 7.84.

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.

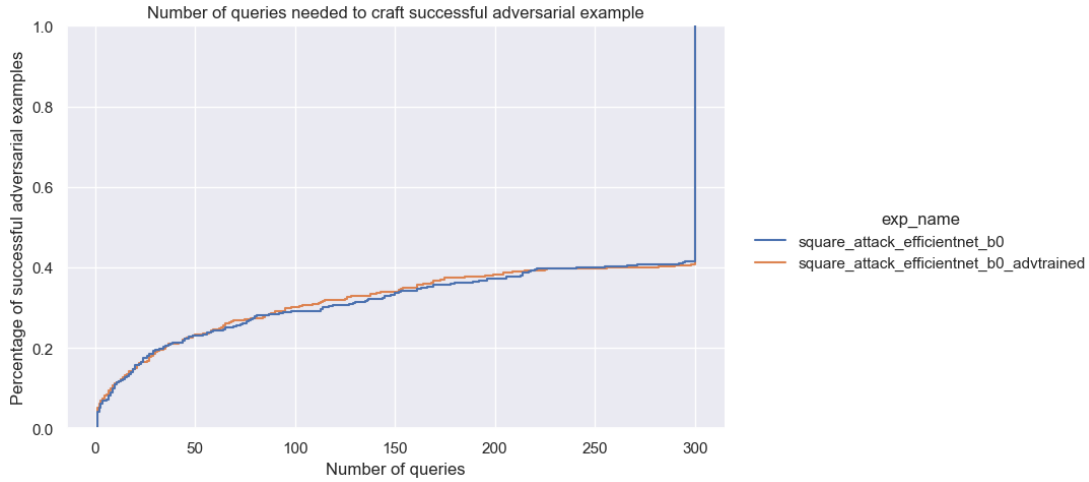
This observation made us focus more on the l_2 bounded attacks in later local experiments (5.2).

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 5.5: 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.

5.2 Local transferability experiments

Motivated by the not-ideal query-efficiency of pure-blackbox attacks (4.3), we moved to transfer attacks to see how far we can push the pure transfer threat model.

5.2.1 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 (5.2.2)

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.

Inference time

In the early days of AdvPipe we had supported only batch sizes of one. We wondered what kind of difference does batch size make in such a memory-limited hardware setting. As we can see in the following figure, 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).

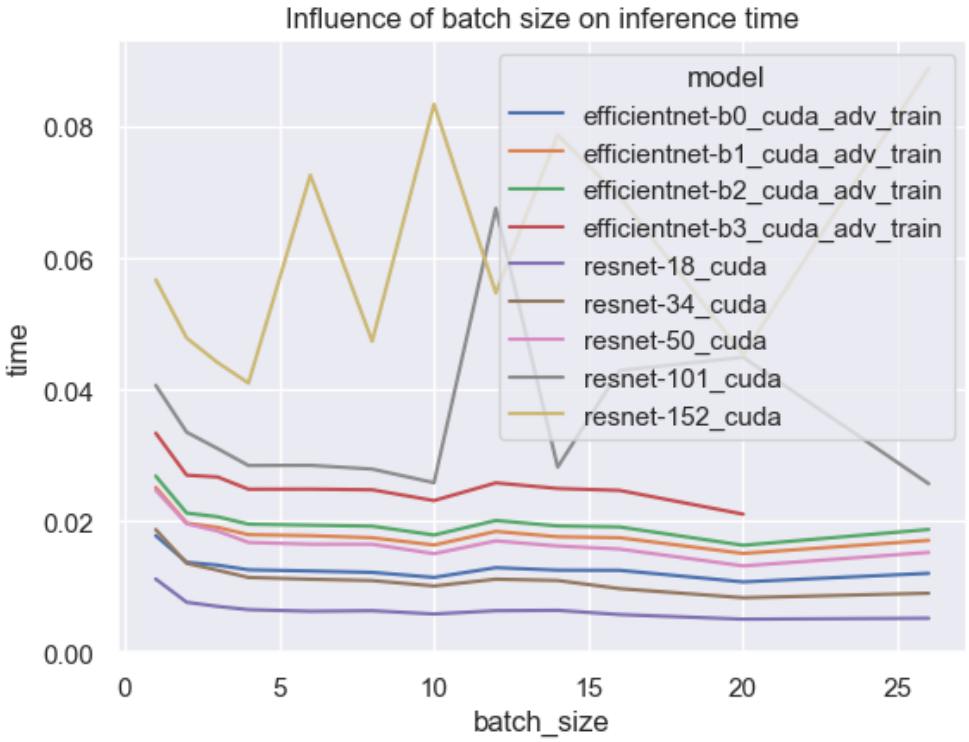


Figure 5.6: ResNet vs EfficientNet inference

5.2.2 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 4.1.1 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.

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.

5.2.3 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%)

5.2.4 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

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 4.1.1 instead? Well, we don't. We just pretend our surrogate is outputing excatly what it should and everything is fine. Figure 5.7 shows the transferability performance across different local models with varying amount of pertubation size ϵ.

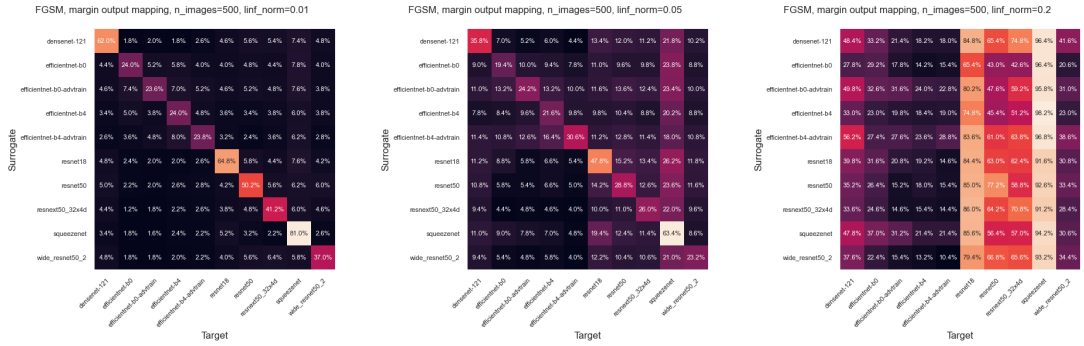


Figure 5.7: FGSM, margin output mapping

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 5.8 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.

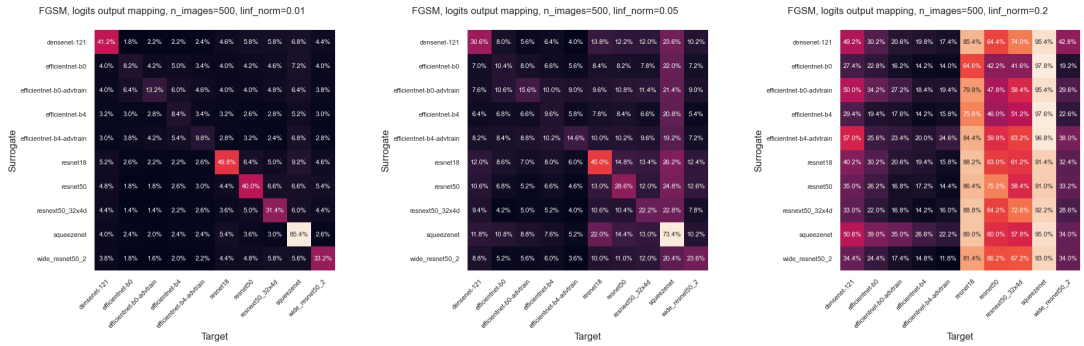


Figure 5.8: FGSM, probability sum mapping

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 4.4.2, 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]).

The need for a better optimizer

We argued in 5.2.4 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.

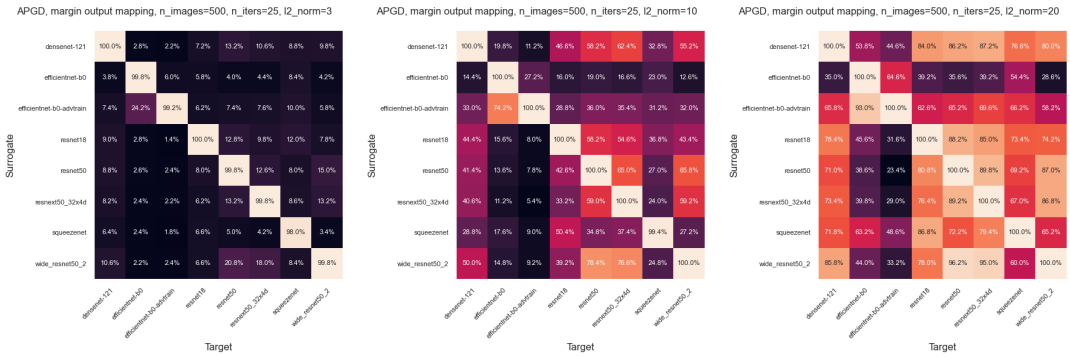
The only knobs to we can tweak are:

- 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.

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 5.9: 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 5.10, 5.11 and 5.12 show some examples of the adversarial images produced:

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

Figure 5.10: 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 5.11: APGD L2 baseline ResNet-50

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

Figure 5.12: APGD L2 baseline EffecientNet-b0-advtrain

5.2.5 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

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 (5.13) 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 5.14 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 5.14 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.

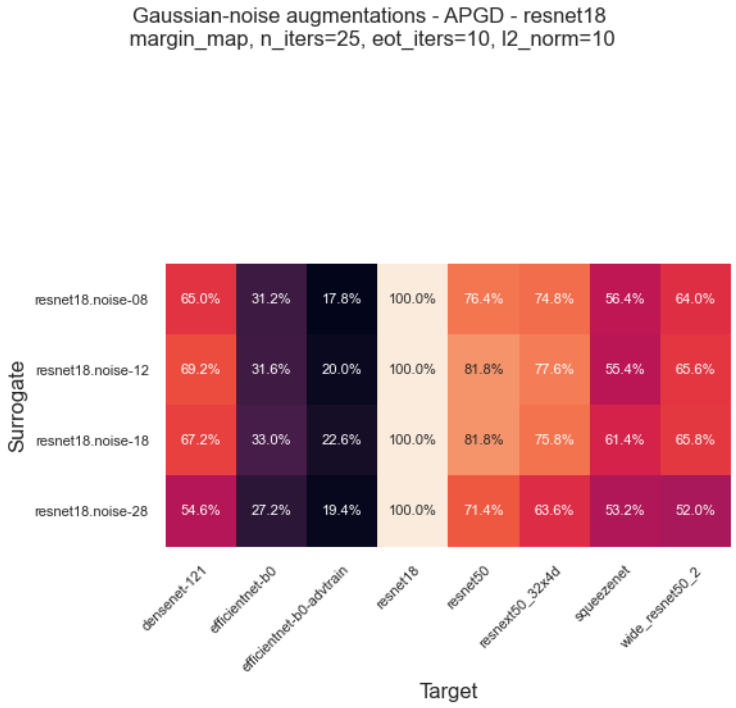
To explore how much the additional gradient sampling helps, we have set σ to an excessive $\sigma = 35$ and ran experiments with $N_{EoT} \in \{1, 3, 10, 30, 100\}$.

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

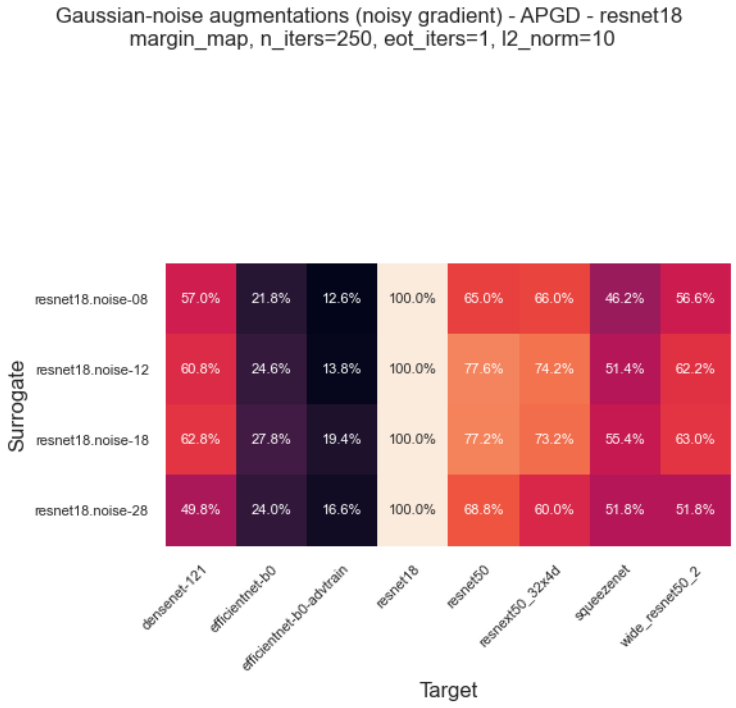
(d) $\sigma = 12$ (e) $\sigma = 16$ (f) $\sigma = 20$

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

Figure 5.13: Guassian noise dog augmentation sample



(a) 25 iterations, 10 gradient estimates per step



(b) 250 iterations, 1 gradient estimate per step

Figure 5.14: Guassian noise augmentation

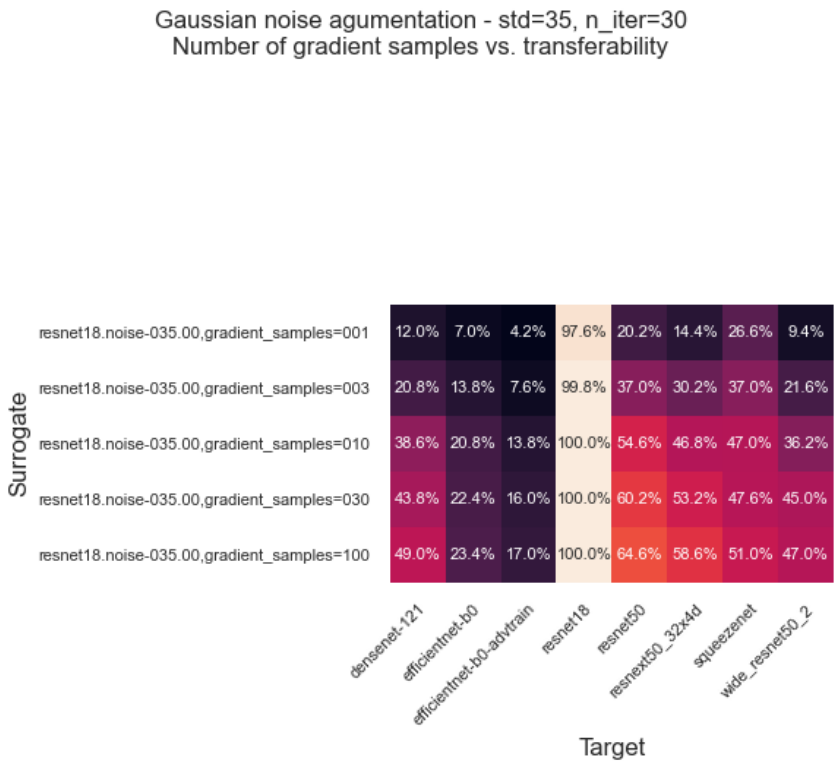


Figure 5.15: The effect of N_{EoT} in noisy gradient situation

Figure 5.15 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.

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 5.16 that a small amout of blur is beneficial, but larger amounts are essentially strong low-pass filters that introduce large surrogate bias. The 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 5.17 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.

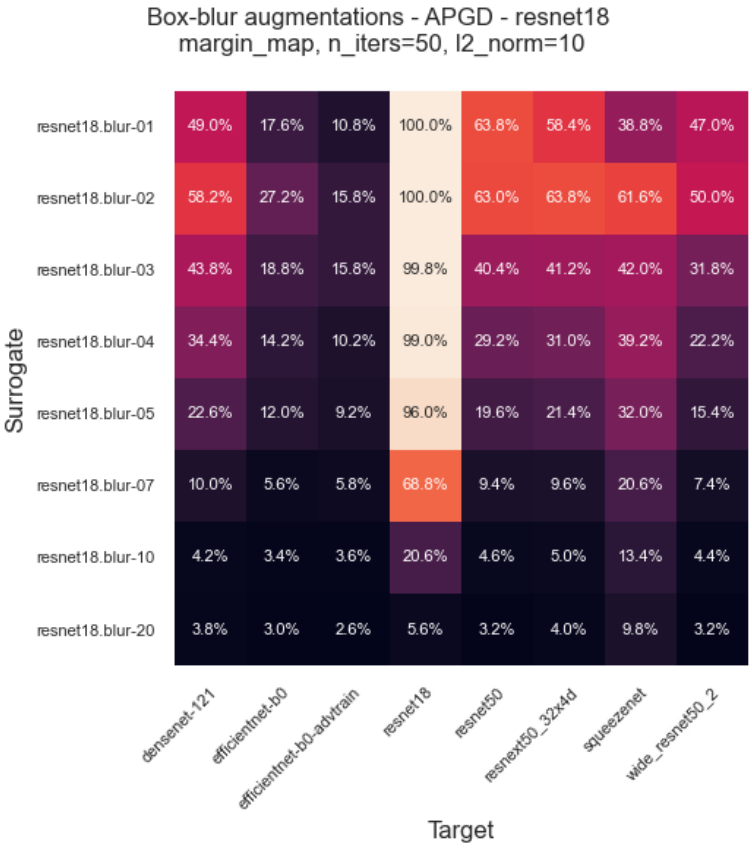


Figure 5.16: APGD box-blur augmentation

(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 5.17: Effect of box-blur kernel size on the frequency of adversarial pertubation

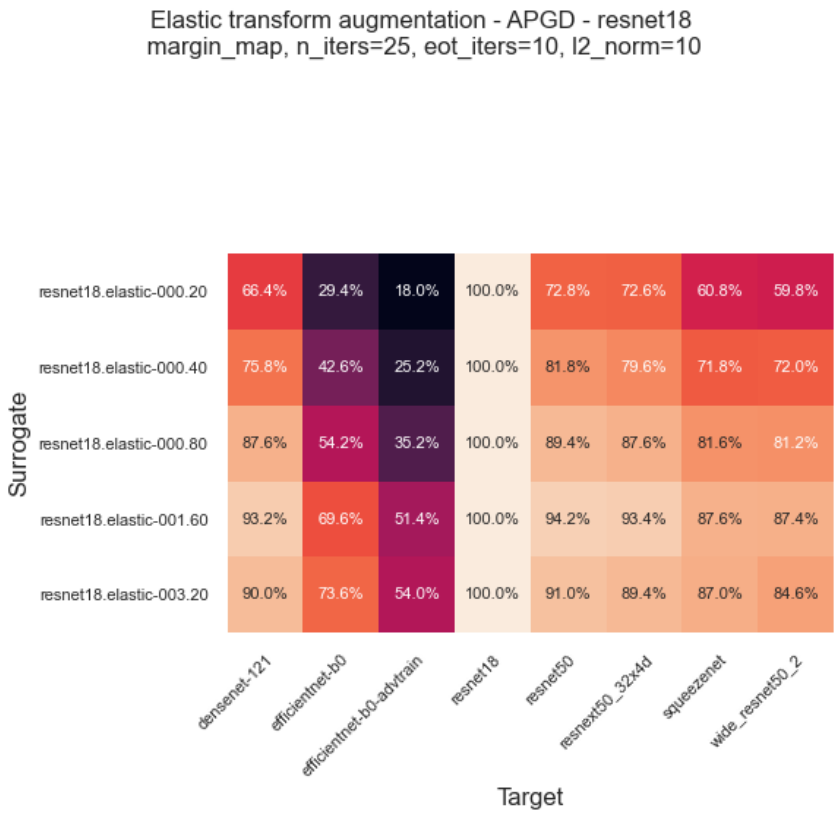


Figure 5.18: APGD elastic transformation augmentations

Elastic transformation

We have also tried to regularize our optimization with stochastic elastic-transformation (figure 5.19). This particular transformation had proved to be a powerful regularization element, which brought the transferability from ResNet-18 to EfficientNet-b0-advtrain to 54.0%.

TODO: describe in more detail what exactly it's the elastic transform...

Affine transformation

Ensemble

TODO:

5.3 Transferability evaluation on GVision

TODO: just run the images against GVision

5.3.1 Choice of evaluation metrics

5.3.2 Wordcloud

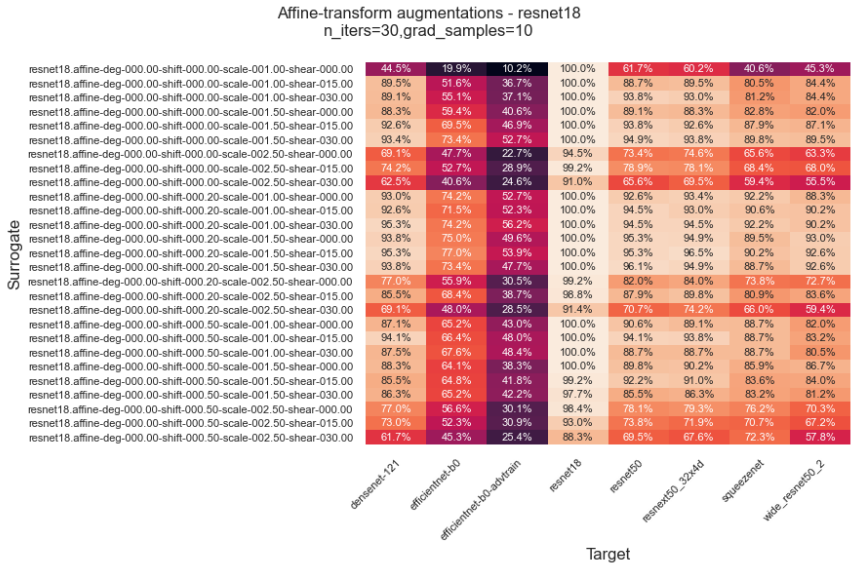


Figure 5.19: APGD elastic transformation augmentations

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.

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.

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.

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.

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.

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, 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, 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.

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.

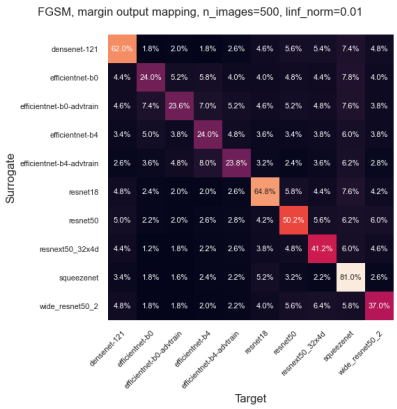
List of Figures

List of Tables

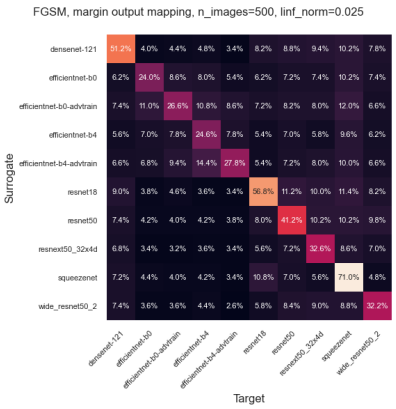
List of Abbreviations

A. Attachments

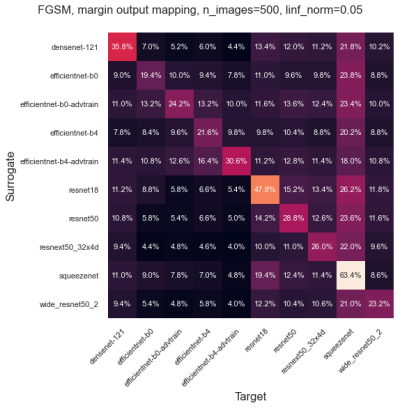
A.1 FGSM local transfer experiments



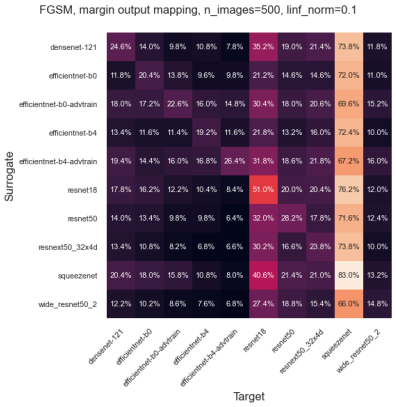
(a) $\epsilon = 0.01$



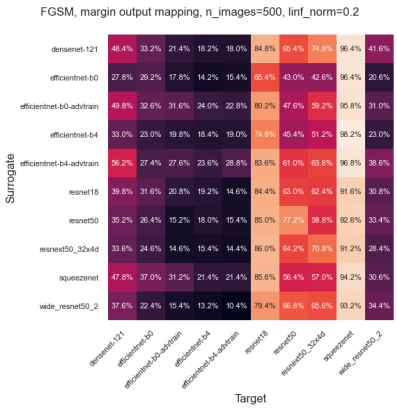
(b) $\epsilon = 0.025$



(c) $\epsilon = 0.05$

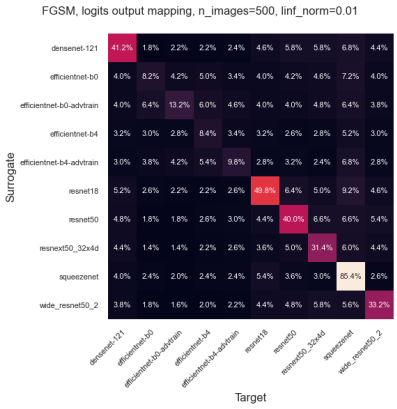


(d) $\epsilon = 0.1$

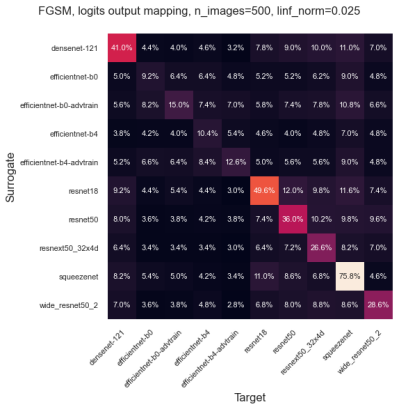


(e) $\epsilon = 0.2$

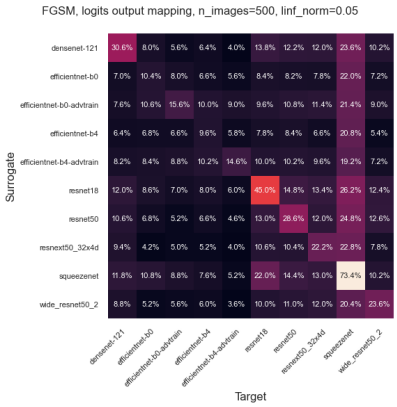
Figure A.1: FGSM, margin output mapping



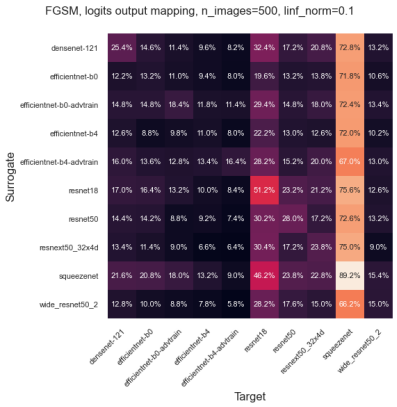
(a) $\epsilon = 0.01$



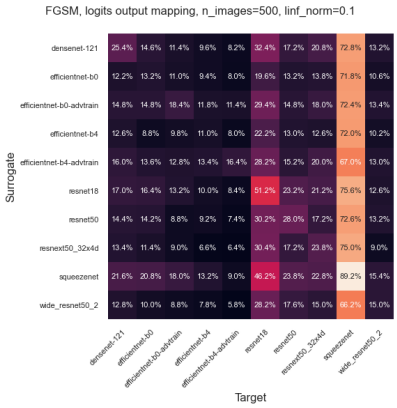
(b) $\epsilon = 0.025$



(c) $\epsilon = 0.05$



(d) $\epsilon = 0.1$



(e) $\epsilon = 0.2$

Figure A.2: FGSM, probability sum output mapping

A.2 AdvPipe source code

TODO github