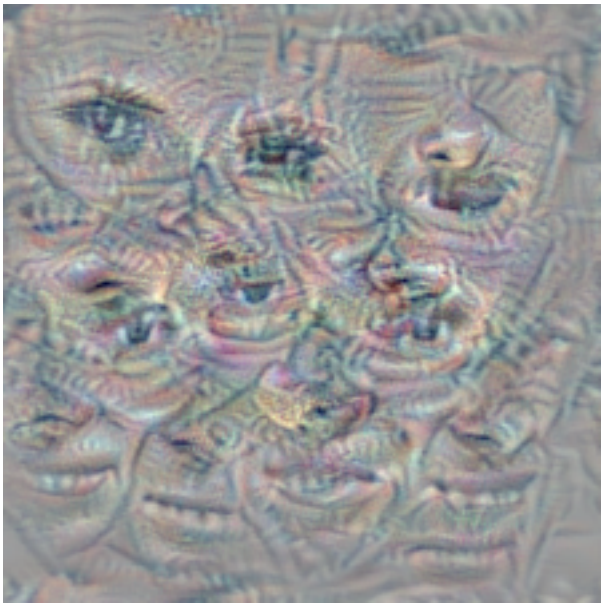
Imperial College London

Department of Electrical and Electronic Engineering

Final Year Project Report 2018



Project Title:	**Fooling Neural Networks using Adversarial Image Perturbations**
Student:	**Guillaume Ramé**
CID:	**00978741**
Course:	**EEE4**
Project Supervisor:	**Dr. Tae-Kyun Kim**
Second Marker:	**Dr. K.M. Mikolajczyk**

Abstract

While neural networks and their convolutional variants have provided state-of-the-art accuracy on many tasks, even outperforming expert humans, there has recently been large effort in investigating the existence of adversarial examples. Adversarial examples are deliberately crafted inputs, imperceptibly different from natural examples, which have the ability to push a classifiers output to an arbitrary value. This means that the classifier will misclassify an input that is visually only slightly different than an input taken from the data distribution. The fact that adversarial examples can be generated regardless of network architecture or dataset is in turn a major concern for safety-critical applications.

In this report, we will formally introduce various methods used to generate adversarial examples. We will analyze the threat model and provide in depth analysis of the behaviour of each technique, according to predefined metrics such as the perturbation norm. In addition, we describe a novel method of generating targeted universal adversarial perturbations. We will introduce a number of defensive methods used to improve the robustness of the classifiers, while showing that there has yet to be the development of a universal mitigation technique. We shall then perform quantitative evaluation of the strengths and weaknesses of the previously introduced attack methods using three datasets. A majority of the experiments will be performed on the well-known MNIST dataset for handwritten digit recognition to provide a sturdy baseline. The substantially harder CIFAR10 dataset for generalized object recognition will be used to provide results based on a more realistic use-case. Finally, we shall introduce a testing methodology focused on evaluating the implications of this study on models developed for face recognition, through analysis on the FaceScrub dataset.

Acknowledgements

Firstly, I would like to express my sincere gratitude for the time and support provided by my project supervisor, Dr. Tae-Kyun Kim, during the course of this project. I would also like to thank Mr. Baris Gecer, a PhD candidate in the ICVL team, who helped me tremendously in the early stages of this thesis, providing me with ideas and directions, and helping me make the final decision of going for adversarial machine learning.

Secondly, I would like to show my appreciation to my parents, sister and grand-parents, and their unconditional love and support in what I do. Even though it's not always easy living in different countries, I love you guys <3.

Finally, I would like to thank RemyBoyz *et al.* for the banter. More seriously, the support and motivation from my friends has been amazing, allowing me to get through this endeavour in one piece.

Vocabulary

- **Natural sample/example** A datapoint/image sampled from the ground truth distribution of the dataset.
- **Adversarial Perturbation** A vector of same dimensionality as the modelś image space, that when added to a natural sample/example forces the prediction of the model to be wrong. s
- **Adversarial Example** The result of applying an adversarial perturbation to a natural sample.
- **Perturbation Budget** The maximum allowed amount that the perturbation can modify the natural sample. This can be in terms of any arbitrary norm, but most commonly in this report, it will be in terms of the L_∞ norm of the perturbation. Unless indicated otherwise, we denote the L_∞ norm of the perturbation by ϵ.
- **Universal Adversarial Perturbation** An adversarial perturbation that is able to generalize to multiple different natural samples.

List of Figures

Contents

Chapter 1

Introduction

1.1 Motivation and Objectives

Machine learning is currently being employed to solve a wide range of problems. In classification problems, models can extrapolate the class of a previously unseen data point using information gained by analyzing the distribution of the training data. Intuitively, one would expect two sufficiently close data points (according to an arbitrary distance metric) to be classified similarly by a robust model. Recently however, this assumption has been proven false, as otherwise extremely accurate models can be fooled into mis-classifying a maliciously crafted input.

Machine vision has always been a task that has piqued the interest of many researchers. This most certainly comes from the fact that object recognition is such an elementary and simple task for humans. More specifically, face recognition, a task that humans particularly excel at, and something that starts to develop while still in the womb [4] is still a very difficult tasks even for the state of the art. Neural networks and their convolutional variants have to some extent reduced the gap between machine and human recognition, however humans intrinsic ability to use context means that this gap will possibly never be fully closed [5]. Improving the robustness of machine learning models also means identifying their shortcomings. Adversarial examples and their impressive ability to fool otherwise highly performing classifiers shows that high test accuracy on a dataset does not equal human performance. This in turn gives precious insight into the inner-workings of neural networks, whose black box nature has traditionally been one of their biggest drawbacks.

This project explores various methods of learning what are called adversarial perturbations. These perturbations are small modifications made to points sampled from the ground truth distribution. These perturbed data points, which are the sum of the original point and the adversarial perturbation,

are referred to as adversarial examples. These adversarial examples have the characteristic of 'fooling' a model into predicting the wrong class. The main goal of this project is to implement a number of these methods and compare their behavior on different datasets and models.

We will also provide a robust formulation and explanation for a novel method of generating targeted universal adversarial perturbations. This method will rely on the modification of a pre-existing attack, in order to transform it into a universal framework. In addition to this, we will provide various techniques to improve the generalizability of the resulting universal perturbations.

In addition to attacks, we will investigate what are known as mitigation methods, used to attempt to defend against these aforementioned attacks. These methods work in wide variety of ways, from modifying the data before it is passed through the model to changing the model itself to improve its robustness to adversarial attacks.

A large majority of the experiments will be performed on the MNIST dataset, a dataset commonly used for its simplicity and small size. We will briefly investigate the behavior of a model trained on the CIFAR10 dataset when subject to our attacks. This second batch of experiments will provide more results based on a more realistic use-case than MNIST. This is also an opportunity to analyze the influence of architecture on adversarial susceptibility, as solving CIFAR10 requires more complex (deeper) models. Finally, a final suite of experiments will focus on generating perturbations specifically tailored to fooling a face recognition model. The objective of these tests is to prove that face identification systems relying on undefended neural network classifiers can easily be fooled, hence making them insecure.

1.2 Project definition

This project can be divided in a number of parts:

- In the first part, we will build an understanding of the current state of the field. This means gaining an theoretical comprehension of the challenges and solutions involved in fooling a machine learning model. This theoretical basis will also help in justifying the rationale behind the novel universal perturbation method introduced in this report.
- In the second part, we will implement a number of attack and mitigation methods. In order to simplify the next step, it is important to formally define an interface, whether containerized in a function or a class. This will reduce the amount of boiler-plate code required to actually execute the experiments.

- Finally, we will introduce experimental protocols, designed to test various aspects of our implementations. Using the theoretical understanding gained from the first section, we will attempt to analyze and explain the behavior our models and attacks, in an attempt to produce a global interpretation of the subject. These experiments will also serve to provide empirical justification of the feasibility of our novel attack method.

1.3 Thesis Structure

The *Background* section introduces the theory necessary to understand the main subject of this report. This theory consists of a brief explanation of the structure and building blocks behind neural networks as well as the optimization framework used to train the networks. It will then formulate the adversarial perturbation discovery problem in a constrained optimization framework as well as a number of techniques used to solve this problem. Finally, this section will introduce several defensive techniques used to mitigate against some of the previously mentioned attacks.

The *Design & Analysis* and *Implementation* sections describe the design choices, details and challenges related to the implementation of the various attacks used in this comparative study. These sections will introduce the architecture and training procedures for the models used for the comparison, as well as their respective datasets. The metrics used for the comparison will also be formally introduced.

The *Testing* and *Results* sections will provide a complete set of test results and analysis of the pros and cons of the different methods contained in this comparative study. These parts will also contain in depth analysis of the behavior of the methods as a function of the type of data used and their respective model.

Finally, we shall look back upon this project in the *Evaluation* and *Conclusion* sections. We will perform an analysis of the work performed during the course of this project, as well as an in depth critic of what has been achieved versus what remains to be done.

Chapter 2

Background Theory

This section will aim to paint a complete picture of the current deep learning landscape. We will start out by introducing fully connected networks, moving onto convolutional neural networks and why they are superior to their dense counterparts. We shall then perform a survey of existing techniques used to learn perturbations. Finally, we will introduce a number of methods used to try and mitigate against these attacks.

2.1 Introduction

2.1.1 Classification Problems

Throughout this report, we shall consider the problem of classification. Given an input x, and a model whose inference function is denoted by f_θ, we want to find optimal θ, the model weights, such that $f(x) \approx y$, where y, a vector, is the ground truth label of x. We find θ by finding the optimal solution to some objective function, given several (point, label) pairs, which form the training set of our model. The trained model's accuracy can then be evaluated using a holdout set of (point, label) pairs, the test set. If the distribution of our train set is representative of the ground truth distribution of our data, then the model should be able to generalize, achieving sufficiently low error on our test data. This project focuses exclusively on cases where x are images in $\mathbb{R}^{n \times m}$, where each dimension of x is represented visually by a gray-scale or a red-green-blue pixel. The labels y will be either scalars representing the index of the corresponding class, or one-hot vectors, where the hot bit is at the index of the corresponding class.

2.1.2 Hand-crafted features and their shortcomings

Traditionally, object recognition has been performed using hand-crafted features. These hand-crafted features would then be classified using standard model types such as Nearest Neighbors, Linear Regression and Support Vector Machines. While good for low-dimensional data typically encountered in non-computer-vision tasks, the intrinsic variability present in visual data (pose, lighting, sensor differences, etc) means that hand-crafted features learned through Principal Component Analysis (PCA, such as that used for eigenfaces [6]) or Histogram Of Gradients (HOG) [7] tend to severely underperform when used for unconstrained object recognition. This motivates the use of a non-linear feature extraction methods, such as neural networks.

2.2 Neural Networks

2.2.1 Fully-connected Neural Networks

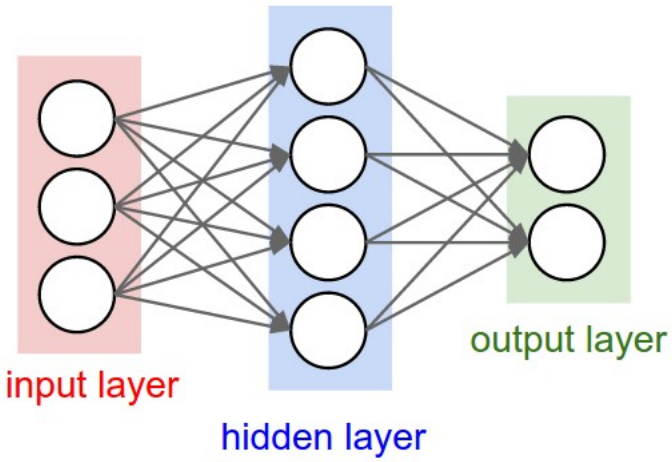


Figure 2.1: Basic fully connected network [1]

Standard neural networks, or multi-layer perceptrons (MLP), consist at their core of layers of neurons, separated by non-linear activation functions. Each neuron classifies the input features using a hyperplane defined by its relevant weight vector. As shown in figure 2.1, each neuron in each layer is connected to all neurons in the previous layer. By convention, the output of the input layer is assumed to be the data passed to the network. Each layer is itself a linear classifier, computing:

$$\forall i, \quad f_i(x) = w_i^T x + b_i$$

Where w_i and x are column vectors and b_i is a scalar. $f(x)$ is then passed through a non-linear

activation function which historically has been the sigmoid function: $\sigma(x) = \frac{1}{1+e^{-x}}$. Recently however, the rectified linear unit function, $\sigma(x) = max(0, x)$, has been used to combat sigmoid's vanishing gradient problem. This problem arises due to the fact that the gradient of the output of a sigmoid with respect to its input tend to zero as the input goes to $\pm\infty$. Using ReLU rather than sigmoid allowed [8] to achieve a factor of six convergence speedup. One issue with ReLU is the so-called dead-neuron problem. This is usually caused by large gradient flow pushing the activation of a given neuron into the negative region. This causes the neuron to 'die', meaning that once passed through the ReLU, the output will never move from zero. Leaky ReLU [9] solves this issue by respectively changing the activation function to $\sigma(x) = max(\alpha x, x)$ with fixed $\alpha << 1$, allowing small gradient to continue flowing even in the negative domain. Parametric ReLU [10] uses a similar idea and makes the α parameter learnable.

While the hidden layers all use the same activation function, the output layer is instead followed by an output activation. If probabilities are needed then the sigmoid function can be used in the binary classifier case, generalising to the softmax function, with the following expression, in the multiclass case:

$$\sigma(z_j) = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

2.2.2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs), whose application for visual recognition was pioneered by Yann LeCun in 1998 [2], have been the most popular model architecture forobject classification in computer vision since [8] considerably improved upon the state of the art during the ImageNet [11] competition in 2012. This type of architecture is much more efficient both in terms of computational and space complexity allowing extremely deep networks to be trained. Figure 2.2 shows the LeNet architecture from 1998. We can clearly see the hierarchal architecture, going from wide but shallow feature maps to very deep by very small maps at the end of the network, followed by the classification layers.

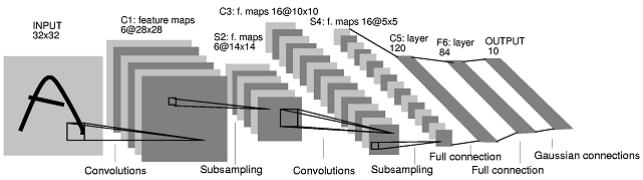


Figure 2.2: Basic Convolutional Neural Network using the LeNet architecture [2]

Convolutional Layer

While the weights for each layer of a fully connected network can be represented as a massive dense matrix, the weights for each convolutional layer can be visualized as a stack of $3d$-matrices. The difference in the case of the CNN is that each weight matrix has orders of magnitudes less parameters than the fully connected weight matrix. Each of these kernels is scanned across the input, applied using summed element-wise multiplication on each kernel-sized patch in the input. In a convolutional layer, each kernel has $K \times K \times D$ weights, where D is the depth of the input space. The activation matrices for each kernel are then concatenated together depth-wise, resulting in an output that has the shape of a rectangular cuboid.

If we take F as representing the kernel receptive field, S as representing the kernel stride, P as the padding size and W as the size of the input feature map, then the output feature map has side of size $O = \frac{W-F+2P}{S}+1$. In turn, this means that the output feature map has dimensions $height = width = O$ and $depth = K$, where K is the number of filters contained in the layer. Recently, most CNN implementations have used 3×3 receptive fields, with strides of 1. It has been shown [12] that while larger filters (5×5 or 7×7) are extremely expensive in terms of computation and lead to large increases in parameter counts (respectively $\frac{5^2}{3^2} = 2.7$ and $\frac{7^2}{3^2} = 5.4$ factor increase in parameters), they do not lead to measurable increases in accuracy.

Similarly to fully connected layers, each convolutional layer is followed by an arbitrary non-linearity. This non-linearity is precisely what enables a neural network to approximate any arbitrary function, such as that mapping our input data to their respective labels. The Rectified Linear Unit (ReLU) function $y = max(x, 0)$ will be used throughout this project, as it is very computationally efficient and has a proven track-record of helping train large models.

Pooling Layer

In order to reduce the computational complexity of the model, it is often interesting to reduce the dimensionality of the output at a certain layer in the network. The most common way to do this is through pooling operations. These pooling operations are characterized by their receptive field, the stride of this field when slid over the input, the function used to map the field values to each output value, and the type of padding used on the input (valid or same). For all experiments in this report, we will use the max pooling function, with a receptive field of 2×2 and strides of 1. We will also follow the consensus in terms of stacking convolutional blocks, then followed by a pooling layer. This pooling operation will serve to reduce the dimensionality of the current feature map, reducing the

computational cost of the rest of the network.

Classification Layer

After a certain number of blocks consisting of stacks of convolutional layers followed by a pooling layer, it is necessary to reduce the output of the network down to a single vector of size equal to the number classes considered. This is done by stacking a number of fully-connected layers and non-linearity functions at the end of the network. The very last layer is however stripped of non-linearity, instead being passed through the previously mentioned softmax function. This will constrain the activation to fall in $[0, 1]$, essentially acting like the probability that the input is classified as the corresponding logit label given the input. As a side-note, reducing the dimensionality of the features output from the convolutional network sections also helps reduce the number of parameters necessary to perform classification on those features. This is because all neurons in a fully connected layer need to be connected to all neurons in the previous layer, with one weight per connection. Reducing the number of neurons in the previous layer hence also reduces the number of parameters, making the model easier and faster to train.

2.2.3 Optimisation and Gradient Descent (GD)

We define an optimization problem as: minimize $f(x)$ subject to $x \in F$, where $f : \mathbb{R}^n \to \mathbb{R}^k$ and $F \in R^n$. A vector $d \in \mathbb{R}^n$ is a descent direction at x if $\exists \lambda > 0 \ni f(x + \lambda d) < f(x)$. In addition to this, if f is differentiable then a descent direction for f at x is simply the opposite of the gradient of f with respect to x. A suitably good solution to this optimization problem can be found by taking several steps along the descent direction evaluated at each step. A suitably good solution is defined as that which sufficiently minimizes our objective function f. We can hence formulate the standard gradient descent relation, for some starting point $x \in R^n$:

$$x_{k+1} = x_k + \alpha_k d_k$$

$$x_0 = x \quad d_k = -\nabla f(x_k)$$

2.2.4 Training a Neural Network

In order to teach the network to compute the desired mapping function between the inputs and the labels, it is first necessary to measure the quality of the networks current output with respect to what

it should be outputting. This can be done with a loss function, penalizing mistakes, encouraging the network to adjust its predictions to make them closer to the ground truth.

When training a network, we generally have a number of outputs equal to the number of classes. Each output from the softmax layer will be the networks confidence that the input has the associated label. A popular loss function used to penalize classification mistakes is cross-entropy:

$$H(p, q) = -\sum_i p_i(x) \log_2 q_i(x)$$

Where p corresponds to the ground truth and q the prediction/softmax logits. Clearly, if $p = q$ then this expression is simply the entropy of the stream of bits p. In order to use this loss, assuming that l is the original label in $\{0..N\}$, with N the number of classes, we first one-hot encode our labels, such that only the l^{th} bit is hot. This means that $p_i = 0$ for all $i \neq l$. In this case the cross-entropy expression reduces to $-log_2 q_l(x)$, which is zero when $q_l(x) = 1$, or in other words, the loss is minimized (at 0) when the model classifies the input correctly.

In order to push the networks predictions closer to the ground truth, we can compute the derivative of the loss with respect to all the parameters, taking a step along the descent direction. Neural networks are however extremely complex functions with millions of parameters. This can be represented as a composition of functions designed to linearly project and apply non-linear transformations to their input. Under this view, we can use back-propagation [13] to efficiently compute the gradient of the loss function with respect to certain parameters. This gradient can then be applied using the previously introduced GD formulation.

2.2.5 Advanced Training and Regularization Methods

Batch Normalization

While training shallow networks (i.e. only a couple layers) is extremely easy, most computer vision applications consist of data with clear hierarchical features. This encourages the use of deep and narrow architectures (such as ResNet [14]) with tens/hundreds of layers, where each layer will detect features of slightly higher-level than the layer before it. The issue with these networks is that the distribution of each layers input changes as the parameters of the previous layer change. This reduces the stability of the training procedure, requiring lower learning rates and extreme care in parameter initialization.

It has long been known that normalizing a models input feature-wise (i.e. along the depth of the data) speeds up convergence and improves the converged accuracy of the model [2]. Batch Normalization [15] takes this idea even further, as it integrates the normalization steps directly into the network's architecture. In a batch-normalized network, most of the convolutional layers are followed by a feature-wise normalization step performed across the batch, with an additional scale and shift to be able to represent the identity transformation. As shown in the paper, this technique dramatically improves the speed of convergence of the networks, enabling the training of very deep architectures that would otherwise be too unstable. Batch normalization also helps to regularize the model, as it introduces noise caused by the fact that the values for any given training example will be dependent on the rest of the mini-batch, in turn reducing the need to apply dropout. Given a d-dimensional vector x, each dimension of x will be normalized across the mini-batch of size m:

$$\hat{x}_k = \frac{x_k - E[x_k]}{\sigma_{x_k}} \quad y_k = \gamma_k \hat{x}_k + \beta_k$$

For $k \in \{0..d\}$. In order to apply this to convolutional layers, rather than performing normalization across the mini-batch for each activation of the layer, the statistics are computed across both the elements of the mini-batch and all spatial locations. As this is a standard technique used to train neural networks, robust implementations exist in most tensor frameworks.

Dropout

Dropout [16] is a regularization technique that adds noise to neuron activations in an effort to reduce overfitting, improving model generalization properties. This is done by randomly dropping activations according a certain probability p. One possible explanation regarding dropout's positive effect on accuracy is that it reduces the specificity of individual neurons to a given task, "making each hidden unit more robust and driving it towards creating useful features on its own without relying on other hidden units to correct its mistakes" [16]. This essentially reduces the specialization of each individual neuron, creating multiple similar pathways throughout the network.

In this report, this layer will only be used to regularize the bottleneck layer of our convolutional models, as batch normalization is usually preferred between the convolutional layers themselves.

Adam Optimizer

The Adam optimizer [17] (from a combination of Adagrad and RMSProp) is a slightly fancier formulation of the standard gradient descent algorithm. Like other advanced adaptive-learning rate methods, it uses per-parameter learning rate tuning to speed up convergence. It employs moving average estimates of the mean and variance of the gradients, hence it is especially adapted towards the optimization of problems with noisy objective functions, such as that of neural network training. The paper also mentions a bias correction mechanism used to correct the bias induced from the zero initialization of the mean and variance estimates. This improves the accuracy of the estimates during early stages in the training process, resulting in a more stable convergence. This optimizer (through the default implementation in TensorFlow [18] for analytic gradient attacks and through a custom implementation otherwise) will be used to both train the models necessary for the comparison, but also to improve the convergence properties of some of the iterative gradient methods.

2.3 Adversarial Perturbations

2.3.1 Introduction

As briefly mentioned in the introduction, adversarial perturbations are small changes made to a certain input (sampled from any arbitrary distribution, including that of the t) that fool a model into making a mistake in the prediction. Formally, we can define an adversarial perturbation $\delta \in \mathbb{R}^n$, as a vector in $\mathbb{R}^n$ that solves the following general-case constrained optimization problem:

$$min|\delta|_p$$

$$C(x+\delta) \neq C(x)$$

With $x \in F$, $x+\delta \in F$, where F is the feasible set of images (I.e. if integer pixels then $\{0 \dots 255\}$ or $[0,1]$ if float valued pixels), $C(x)$ is the classification function modeled by the network and we denote the L_p -norm of δ by $|\delta|_p$. Generally, attacks can be targeted or untargeted. If they are targeted, then the problem becomes:

$$min|\delta|_p$$

$$C(x+\delta) \neq C(x)$$

$$C(x+\delta) = l$$

Where l is our target class of interest. In other words, we want the model to predict the wrong class for the adversarial example, but we also want the predicted wrong class to be a specific class. In the untargeted case, we simply want the prediction to be wrong, without considering how exactly it is wrong. In this report, C will denote the index of the largest logit (before or after softmax, depending on whether cross-entropy loss is used).

2.3.2 Why do Adversarial Perturbations Exist?

Intuitively, one would expect small perturbations to have little effect on the classification performed by a trained model. This comes from the fact that small differences in pixel space (such as noise) does not make much of a difference to human viewers. The existence of adversarial examples is unintuitive to most people because while they generally have very good intuition of 2 or $3d$ space, that intuition generalizes very poorly to high dimensional spaces like the kind encountered in computer vision. In 2 or $3d$ space, small differences in coordinates between two points still mean that these points are 'close together' in space. In spaces with thousands of dimensions, distances are amplified, as small contributions in each dimension sum up and amount to large distances.

Normally, it can be easily shown that networks are also extremely robust to additive noise. An example of this can be seen in Figure 2.3. In this experiment, 0-mean Gaussian noise with increasing standard deviation is added to a reference image of a two digit, for twelve iterations, in steps of 0.05 (we do not sum cumulatively, each iteration reuses the same noise-free image, resampling noise according the current step). As can be seen, even for noise sampled from $N(0, 0.6)$, the network manages to classify the digit correctly with extremely high confidence, possibly outperforming a human, as it becomes extremely difficult for us to perceive the digit. This shows that even extremely large perturbations made to the input data doesnt lead in itself to adversarial examples if the perturbations arent structured.

Multiple publications have over the years tried to justify the existence of adversarial perturbations. Some speculate that these could be caused by the extreme non-linearity of deep neural networks, or insufficient regularization of the models, leading to overfitting [19]. In contrast, [20] affirm that the vulnerability of neural networks to adversarial examples cannot be due to their highly non-linear behavior as they show that even linear models can be vulnerable to adversarial examples. Figure 2.4 illustrates this claim, showing a picture of a six perturbed into being classified by a linear softmax classifier as all of the other MNIST classes. We can see that the model is consistently and very easily fooled, with the perturbations themselves being minimally intrusive. Interestingly, these adversarial examples are not necessarily marginal, as some of them are classified as the target class with extremely

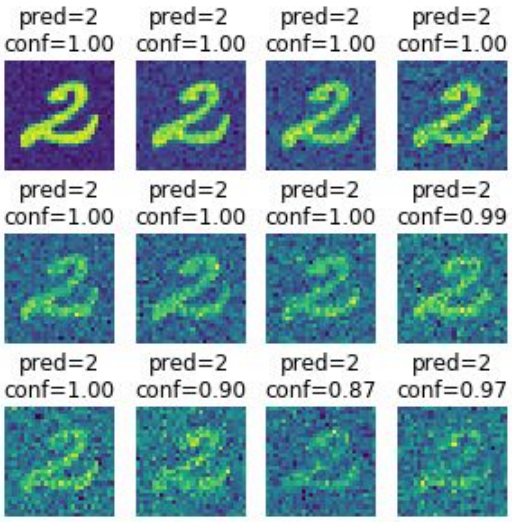


Figure 2.3: An image of a 2, with noise added of increasingly large variance

high confidence (the lowest being 0 with 47% confidence and highest being 2 with 97% confidence). The adversarial perturbations were found by simply performing a single step of gradient ascent of the targeted label (the step size is found by sweeping the space in powers of 2).

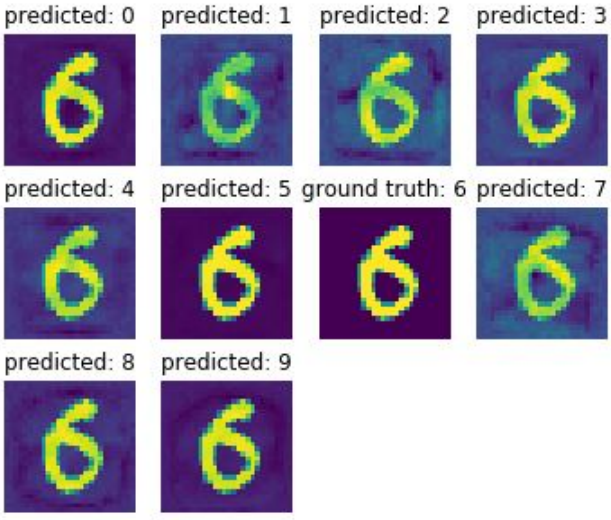


Figure 2.4: An image of a 6, perturbed to perform a targeted attack on all other MNIST classes, applied to a linear classifier

The reason this result is possible is due to the fact that given sufficient dimensionality, a large number of small variations in the input can lead to a large change in the output due to the cumulative effect of these perturbations when forwarded through the model. This leads [20] to conjecture that the reason neural networks are vulnerable to adversarial examples is precisely because they are conditioned to perform as linearly as possible in order to facilitate their training.

2.3.3 Threat Model

In this report, we will consider multiple different threat models. In threat analysis, it is important to define exactly the capabilities of the adversary, in terms of which information they have access to, and what capabilities they have. Most attacks can be broadly separated into two categories: white and black box attacks.

In white box attacks, the attacker has full access to a model, including the model's input and the ability to compute gradients of the model with respect to the input. This threat model allows attacks such as the gradient based methods that will introduce in the following sections. To facilitate the comparison, we assume that the undetectability (i.e. limit the L_∞-norm) of adversarial perturbations is a desirable feature, hence this will be a metric that will constantly be taken into account in the following experiments.

In contrast, a black-box adversary is any adversary that has limited or no access to the model itself. It is generally agreed upon that a black-box adversary at the very least does not have access to the model gradients [21], and only has access, in the best of cases, to the model logits. This threat model breaks all attacks that rely on gradient information. An argument can be made that a black box adversary is a more realistic use-case, for example someone gaining unauthorized access to a system by breaking the face recognition access control [22], or more interestingly similar to [23], where they attack cloud-based classification services by first learning a substitute model similar to the attacked model, then perform a white box attack on the substitute model, transferring it over to the cloud model.

Both threat models have their merits. While white-box attacks are typically the strongest and expose fundamental weaknesses in the absolute behaviour of state of the art models, black box attacks attempt to quantify the risks faced by current production systems.

2.4 White Box Attack Methods

2.4.1 Fast Gradient Sign (FGS) Method

One of the most basic attack methods is the FGS method [20]. This method takes a single L_∞-norm bounded step in the direction that increases the loss J of the network, exactly equivalent to reducing

the activation of the largest logit given x.

$$\delta = \epsilon \operatorname{sign}(\nabla_x J(x, y)) \Rightarrow \hat{x} = x + \epsilon \operatorname{sign}(\nabla_x J(x, y))$$

Where $\nabla_x J(x, y)$ is the gradient of the loss function (cross-entropy) with respect to the input of the model, y is the class predicted by the network given x, hence making this method untargeted. This is equivalent to minimizing the networks confidence in that $\hat{x}$ has class y. The rationale to limit the L_∞-norm of the step comes from the fact that we would like to make infinitesimally small changes to the input x, which combine to a very large change in the output of the model due to the compounding effects caused by the high dimensionality of x. By limiting the L_∞-norm of the perturbation, we intend to maintain the undetectability of the attack, while still causing large shifts in prediction.

2.4.2 Basic Iterative (BI) Method

The BI method [24] is similar in idea to FGS except it takes multiple small steps rather than a large single step. Similarly to FGS, the perturbation is L_∞-norm bounded to maintain the assumption that the perturbation is arbitrarily infinitesimally small. While slower than the previous method, by virtue of the iterative process, the BI method will usually find a better local minimum than FGS, which, especially for larger L_∞ bounds will have a tendency to overshoot. The expression for the BI process is the following:

$$\hat{x}_0 = x$$

$$\hat{x}_{k+1} = clip_{x,\epsilon}\{\hat{x}_k + \alpha \nabla_{\hat{x}_k} J(\hat{x}_k, y)\}$$

Where x is the original image, $\hat{x}_k$ is the perturbed image at iteration k and is the optimization step size. There are two ways of guaranteeing that we respect the L_∞ bound. The first is by using Projected Gradient Descent to project each new estimate for the perturbed image back onto the feasible set, accomplished in our case by clipping the estimate to an ϵ-interval around x. This can also be done by setting $\alpha = \frac{\epsilon}{n}$, n being the number of iterations, with a simple clip at the end of the iterative process to guarantee that the final estimate is still contained in valid image-space. BI, like FGS, is an untargeted attack, hence y is set to the original predicted label of x.

2.4.3 Target One Step (TOS) Method

Similar in idea to FGS, we instead set y to the least likely predicted class given x, formally:

$$y_{LL} = \text{argmin}_i \, F_i(x)$$

Where $F_i(x)$ is the i^{th} output of the networks softmax layer. The expression for the adversarial example crafted using TOS is:

$$\hat{x} = x - \epsilon \, \text{sign}(\nabla_x J(x, y_{LL}))$$

In other words, this will take a step in image-space such that the value of the target logit is increased. Once again, the resulting adversarial candidate $\hat{x}$ is clipped so that it is contained in $[0, 1]$. This attack is can be potentially far stronger than the simple FGS method. This is because it is usually more interesting to be able to control the models exact output. Considering an example of a face recognition used for access control, a targeted attack could allow an imposter to impersonate somebody with access.

2.4.4 Target Iterative (TI) Method

This attack combines the iterative nature of BI with the targeted nature of TOS. Similarly to BI vs FGS, we will see in the experiments that TI is a substantially stronger attack than TOS. This is because for large perturbation sizes, the single step have a tendency to overshoot the local-minima. Hence iterative techniques can be used to find more effective adversarial examples, at the cost of a higher computational complexity. Like BI, we set $\alpha = \frac{\epsilon}{n}$ where n is the number of iterations.

$$y_{LL} = \text{argmin}_i \, F_i(x)$$

$$\hat{x}_{k+1} = clip_{x,\epsilon}\{\hat{x}_k - \alpha \nabla_{\hat{x}_k} J(\hat{x}_k, y_{LL})\}$$

2.4.5 Carlini & Wagner (C&W) Method

Carlini & Wagner combine a number of ideas and enhancements to produce the strongest attack at the time of the publication of their paper [25]. Recall the formulation of the original adversarial learning problem, where l is either set by the attacker or determined automatically as a function of the logits given x, the original image:

$$\min |\delta|_p$$

$$C(x) \neq C(x + \delta) = l$$

$$x + \delta \in [0, 1]$$

Rather than enforcing the $[0, 1]$ box-constraint by clipping the result of the optimization like previous methods, they instead perform a change of variable, optimizing for w rather than for δ directly:

$$\delta_i = \frac{1}{2}(\tanh(w_i) + 1) - x_i$$

$$\Rightarrow 0 \leq x_i + \delta_i \leq 1 \; as \; -1 \leq \tanh(w_i) \leq 1$$

C&W define a $f : \mathbb{R}^n \to \mathbb{R}$ such that $f(x + \delta) \leq 0$ when $C(x + \delta) = l$. One formulation they give for f is shown below (called margin loss).

$$f(x) = \max_{i \neq l} Z_i(x) - Z_l(x)$$

Clearly, this function fits the previously mentioned requirement as:

$$f(x) \leq 0 \Rightarrow Z_l(x) \geq \max_{i \neq l} Z_i(x) \Rightarrow C(x) = l$$

This formulation is a targeted attack, as l can be set by the attacker to any of the models possible classes. It encourages the logit corresponding to the attacked label to be maximized, while minimizing any of the other logits. We can then reformulate the original problem to get the C&W attack method:

$$\hat{x} = x + \delta = \frac{1}{2}(\tanh w + 1)$$

$$\min_w |\hat{x} - x|_2 + cf(\hat{x})$$

This unconstrained optimization problem can be efficiently solved with gradient descent. As an additional optimization, one can use Adam optimizer instead of vanilla GD to achieve faster convergence. In all experiments, $c = 1$. We define that the method has converged once the logit for our targeted class is the larger than that of all other classes (i.e. our loss turns negative).

We also implement their L_∞ method, in order to use it for our generalized perturbation method:

$$\min_\delta |\delta|_\infty + cf(x + \delta)$$

Simply solving the non-differentiability of the max-norm by naively optimizing the above formulation

produces suboptimal solutions due to the instability of convergence [25]. Instead, we penalize the perturbation dimensions when they surpass a certain threshold τ, exponentially decaying τ when all the dimensions are under the threshold:

$$\min_{\delta} cf(x+\delta) + \sum_{i} \max(0, \delta_i - \tau)$$

Similarly to the paper, we start with $\tau = 1$, decaying by a factor of 0.9 when $\forall i, \delta_i \leq \tau$. We use Adam Optimizer to perform the gradient descent, with a base learning rate of 0.01, $c = 1$. There are however two fundamental issues with this formulation. The first is that it does not apply the necessary $[0, 1]$ box-constraint to the generated adversarial example. This means that this formulation will return solutions outside of the feasible set of images. This can be fixed by applying the change of variable method in a similar fashion to the L_2 method. Secondly, the perturbation L_∞ penalization only penalizes the positive values of δ. Clearly, we have:

$$0 \leq x + \delta \leq 1 \Rightarrow \forall i,\ \delta_i \in [-x_i, 1 - x_i] \subset [-1, 1]$$

Hence we need to penalize the absolute value of the perturbation, rather than the perturbation itself. This does not compromise differentiability of the objective function as we can simply approximate the derivative as:

$$\frac{d|x|}{dx} = \begin{cases} -1 & x < 0 \\ 0 & x = 0 \\ 1 & x > 0 \end{cases}$$

Judging solely from the paper, it is uncertain if the authors of the original paper considered these issues. Fortunately, they provide a link to their experiment code in the paper from which we can see that they do actually squeeze the adversarial example into the correct interval with tanh. In addition to this, they also penalize the absolute value of the perturbation. It is unclear why the authors did not mention these details in the paper, as without them the formulation of the objective is wrong. It is possible that they expect the reader to infer that the optimization variable for the L_∞ problem must be box-constrained similarly to that of the L_2 problem.

Regardless, if we adjust the formulation for this optimization problem using what we have just explained, we get:

$$\delta = \frac{1}{2}(\tanh(w) + 1) - x$$

$$\min_{w} cf(x+\delta) + \sum_{i} \max(0, |\delta_i| - \tau)$$

Where x is the original image, w is our optimization variable and δ is the learned perturbation. With this change, we get the results in Figure 2.5, where we apply the L_∞ method on an image of class 7 and target class 2. When we correctly apply the box-constraint during the optimization, we get the middle result. As expected, the pixel values are contained in $[0.479, 0.889] \subset [0, 1]$. When we use the original formulation, we get the right image, where the pixel values fall inside $[-0.284, 2.013] \not\subset [0, 1]$.

Figure 2.5: *Left*: Unperturbed image of a 7, *Middle*: perturbed image using the correct box constraint, *Right*: perturbed image without box constraint

2.5 Black Box Attack Methods

2.5.1 Simultaneous perturbation stochastic approximation (SPSA)

Lets consider a function $f : \mathbb{R}^n \to \mathbb{R}$ that we wish to minimize. We do not have access to analytical gradients of f, yet we wish to find x that minimizes it. The idea behind SPSA is to combine gradient descent with an approximation of the gradient of f by finite difference. The central difference of f at x, given a step-size h, is:

$$\delta_h f(x) = f(x + \frac{h}{2}) - f(x - \frac{h}{2})$$

In which case our approximation of the gradient of f at x is:

$$\nabla f(x)_h \approx \frac{\delta_h f(x)}{h}$$

Hence using the standard gradient descent formulation, with descent direction $d_k = \nabla f(x)_h$:

$$\hat{x}_{k+1} = \hat{x}_k - \alpha \nabla f(x)_h$$

We can build a good approximation of the analytical gradient by computing the central difference at random offsets in x-space. SPSA however requires that the noise distribution sampled has finite

first and second moments [26]. According to [27], the Rademacher distribution, which takes values ± 1 with equal probability 0.5, is a good choice. We then compute the feature-wise mean of the gradient approximations in order to get the overall approximation. Similarly to [27], we reuse the loss function from [25] as it provides (empirically) smoother convergence than cross-entropy loss for this optimization method.

Similarly to previously introduced methods, it is important to constrain the steps of the optimization to the feasible set of images. In order words, the perturbed image at all steps is projected back (or clipped) onto $[0, 1]$. This step also allows us to limit the norm of our perturbation, helping to keep this attack method comparable with previously introduced methods. Once again, we also use Adam optimizer to speed up the convergence when compared to vanilla gradient descent. Due to the fact that we are not using analytical gradients, a custom implementation of Adam is required. As a consequence, particular care is taken to make sure that the gradient moment estimations are bias-corrected (as they are initialized to zero) in order to stabilize early optimization iterations [17].

It is important to comment on the computational complexity of this optimization method. For each iteration of the algorithm, we must compute the average of the central difference for a batch of data points. Naturally, the higher the batch size, the more stable and robust the convergence, at the cost of an increasing computational cost, due to the fact that we must compute the loss, and hence infer on the model for each item in our batch. This also means that this method cannot be optimized to compute adversarial perturbations for different images simultaneously, in effect, we are restricted to fooling one input at a time. In summary, the advantage from the gradient-free nature of this optimization algorithm comes at a very large cost: the computational complexity.

2.5.2 Fooling Examples

Also called *Rubbish Class Examples* [20], this section outlines a method that isnt an attack *per se*. It is an interesting experiment that illustrates fundamental flaws in the models commonly used. Figure 2.6 shows samples of Gaussian distributed noise $N(0, 0.1)$. We can see that a model trained on the MNIST dataset tends to classify this noise as classes $\{1, 3, 5, 7\}$, even though these clearly look nothing like their predicted class. While these classifications are made with extremely low confidence, the variability in the predictions in this experiment shows that the model is unreliable when exposed to data that is completely disjoint from the intrinsic distribution of the training set.

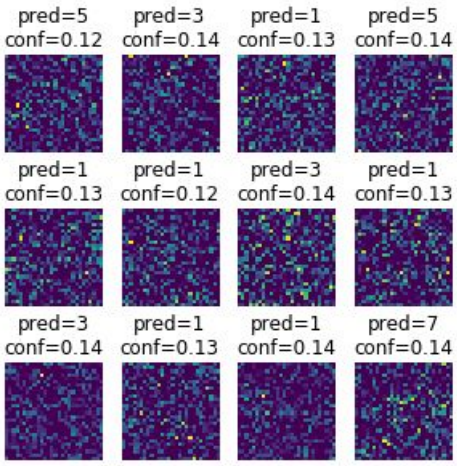


Figure 2.6: Samples of noise classified with low confidence as random classes

2.5.3 Adversarial Transferability

Many papers have also focused on the transferability of adversarial examples between models that have disjoint training sets, different architectures, or even use completely different classification algorithms [21]. The transferability property is extremely interesting as the attacker does not need access to the attacked model at all. This is different from other black box methods that require at least access to the logits of the model in order to determine how to reach the relevant objective.

Transferability of adversarial examples relies on the fact that given the same task, even fundamentally different models with have a tendency to learn similar mappings. This comes from the fact that machine learning models are heavily encouraged to generalize and extrapolate from the training data as much as possible. For example, if two models both learn to recognize digits, and both models generalize equally well on the task at hand, then it is likely that they will both learn similar decision functions.

As we will see in the experiments, FGS (as well as other attack methods) finds adversarial perturbations regardless of the step-size . This ultimately means that adversarial examples are spread across large sub-regions of the image space. Under the view that vulnerability to adversarial perturbations is caused by a models linear behavior, then it is normal that adversarial examples should generalize across different models, as it is extremely likely that models with have overlapping adversarial sub-spaces

In order to use this property to transfer an attack to a black box system, an adversary can generate adversarial examples for a model trained to perform the same task as the attacked system. These adversarial examples are then likely (with a non-negligible probability, dependent on the exact model

types and application) to fool the black box system in addition to the reference model.

2.5.4 Adversarial Generalization: Targeted Universal Adversarial Perturbations

We propose a method to generate adversarial perturbations that will generalize to an entire class of inputs. We denote this method as adversarial generalization, in which a single adversarial perturbation, when applied to multiple different natural samples, will cause all of them to be misclassified by a model. Using the previously introduced attack methods, we jointly optimize the adversarial perturbation over a set of images. Instead of focusing on generating adversarial perturbations that fool a single image, we will learn a single adversarial perturbation that will be optimized to fool a set of images. The desired outcome of this optimization is that by learning to fool a set of images, we generate a perturbation that captures attackable features of the targeted class, allowing it to generalize to natural samples outside of its 'training' set. In this section, we will constantly refer to the set of images used to generate the pertubation as its 'training' set, as we are effectively 'training' the perturbation to fool out-of-sample data.

Recall that the standard formulation for the adversarial optimization problem is:

$$\min |\delta|_p$$

$$C(x) \neq C(x+\delta) = l$$

$$x + \delta \in [0, 1]$$

Where $C(x)$ denotes the classification function of the attacked model. We replace $x \in \mathbb{R}^d$ with $X \in \mathbb{R}^{x \times d}$, where n is the number of images over which the optimization is performed, and l is the label targeted by the attack. The adversarial objective, in the generalized case, becomes:

$$\min |\delta|_p$$

$$\forall i, C(X_i) \neq C(X_i + \delta) = l$$

$$\forall i,\ X_i + \delta \in [0, 1]$$

If we apply this generalization to the C&W L_∞ attack, with $\hat{x}_i$ denoting the i^{th} adversarial example,

the formulation becomes:

$$\hat{x}_i = \frac{1}{2}(\tanh(X_i + \delta) + 1) - x, \ i = 0, \ldots, n$$

$$\min_w \ c\,\mathbb{E}_{x \sim X} f_l(\hat{x}) + \sum_i \max(0, |\delta_i| - \tau)$$

Where the expectation is performed across the training set of the perturbation and the sum is performed across the features of the perturbation vector. $f_l(x)$ represents the loss of the model's prediction given that we target class l. In order to train the perturbation on a dataset larger than GPU memory, we restrict the expectation to operate on batches of images rather than the whole dataset:

$$\min_w \frac{c}{n} \sum_{i=0}^{n} f_l(\hat{x}_i) + \sum_i \max(0, |\delta_i| - \tau)$$

Where each batch contains n images. Given a sufficiently large batch size, the average gradient of the loss across the batch should be close enough to the real average gradient over the entire data set. We can hence use this approximation to speed up the optimization process. A block diagram of the proposed method is shown in Figure 2.7.

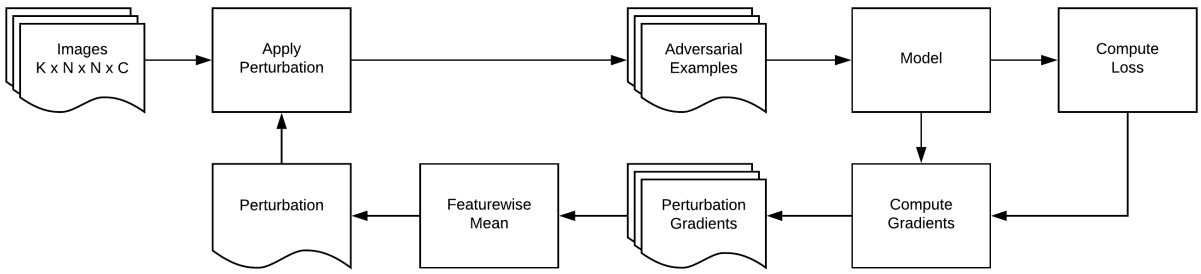


Figure 2.7: Block diagram of the proposed technique.

We reuse the C&W L_∞ objective function, maintaining the L_∞ penalization to minimize the absolute visual distortion caused by the generated perturbation. In contrast however, we maintain τ fixed in order to evaluate this attacks sensibility to the perturbation budget. Finally, we also employ the change of variable used by C&W to ensure that the resulting adversarial examples are contained within the models image space $[0, 1]$. This will also guarantee that the adversarial examples generated outside of the perturbation training set also fall into valid image space, without any additional processing (such as clipping).

In order to improve the generalization properties of the 'learned' perturbation, we validate the fooling properties of the perturbation on a holdout set at every epoch, saving the current perturbation if we

have achieved a new best. This prevents the otherwise uncontrolled overfitting of the perturbation on the train set. The optimization process is hence the following: for each batch of data points in our perturbation training set, we compute the value of our objective, and take a step along the descent direction. At the end of each epoch, we evaluate the target class accuracy of the perturbation when applied to the validation set. If this accuracy is larger than any of previously encountered, we save the current perturbation vector. We then repeat this process for a certain number of epochs.

It is important to investigate the computational complexity of this attack method. In order to generate the perturbations, the complexity is mostly equivalent to that of the single images L_∞-method. To generate the perturbation, the single image methods perform a backwards pass through the attacked model for each iteration of the optimization. In contrast, the generalized method computes the gradient averaged accross batches of images. While the perturbations learned on single images are mostly constrained to that image (and hence the perturbation needs to be recomputed for each image), the generalized method inherently allows us to reuse the perturbation. We hence have a very large up front cost required to 'train' the perturbation, however using that perturbation only involves adding it to a natural sample and squeezing the sum into the box constraint.

It shall be noted that this method has some resemblance to the work produced by [28]. Introducing 'Expectation over Transformation', [28] attempt to improve the robustness of adversarial perturbations to random transformations intrinsic to real life use cases such as scaling, rotation, crop, etc. To achieve this, [28] solve the optimization problem such that the perturbation is invariant to these transformations while still being close (according to some distance metric) to the natural sample. While we also attempt to generate robust perturbations, our method differs in that we expect the adversarial perturbation built with the proposed method to be resistant to *any* change in the input, simply by virtue of the fact that our method heavily encourages the perturbation to generalise accross the entire dataset. In addition to this, we also note the work produced by [29] and [30] who both generate untargeted adversarial perturbations. On one hand, [29] extend their previous work on Deepfool [31], an attack method relying on the iterative linearisation of the decision boundary, and instead learn the minimum perturbation such that a set of images is misclassified. On the other, [30] learn universal untargeted perturbations without using data itself, instead optimizing over the activations of the network's internal layers.

2.6 Mitigation Techniques

2.6.1 Random Transformations

The idea behind this method is to apply random transformations to the input before passing that to the classifier [32]. The reason that this works is that translation invariance is a property that convolutional networks inherently satisfy (due to how convolutional layers work), and can be made even stronger by actively augmenting the training set. During training, dataset augmentation, such as random translation, cropping, rotation and flip can improve the models general robustness to these variations. The reason why this could reduce the effectiveness of adversarial perturbations comes from the view that adversarial perturbations exist because of small subspaces in the input space, disjoint from the data distribution, where the network will be fooled. If we can apply random transformations to the input then it is possible that we can push the adversarial example out of this undesirable subspace. At the cost of extra computation, multiple random transformations using different techniques can be applied to the input of the model. For increased reliability, it is also possible to average together the predictions for multiple randomly perturbed versions of the same input, increasing the robustness of the classification.

2.6.2 Model Capacity

Under the view that neural networks are vulnerable to adversarial perturbations because they have linear behavior [20] (to make them easier to train), it is possible to increase the adversarial robustness of a model by making it more non-linear. This is the conclusion reached by [24], who prove that a model needs sufficient capacity to be able to resist adversarial perturbations. In neural networks, this translates directly to using larger/deeper architectures, with more layers. The rationale behind this is that a deeper network, while substantially more difficult to train (without keeping the architecture simple, i.e. without residual learning or batch normalisation), should be able to represent a more complex decision function, capable of capturing the intricacies of the natural data, while also learning to mitigate adversarial perturbations of certain strength. This effect is illustrated in Figure 2.8, where a very linear (middle) model will have extremely limited robustness to adversarial perturbations, while a non-linear model (right) will be able to learn a complex decision function with an increased robustness to adversarial examples.

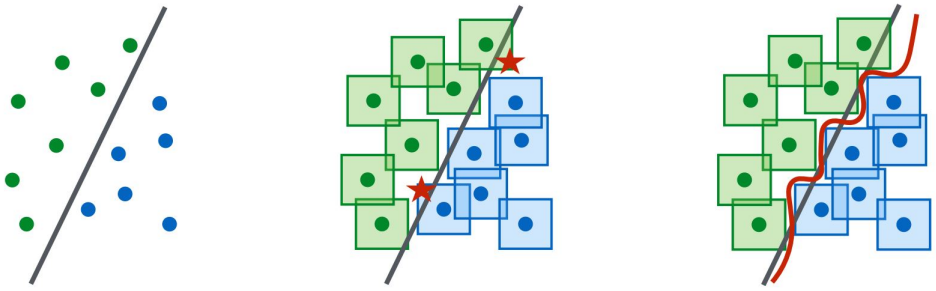


Figure 2.8: Illustration of why a model with higher capacity can be taught to be more robust to adversarial perturbation (taken from [3]). The L_∞ neighbourhoods are represented by squares surrounding the data points.

2.6.3 Hard positive Mining

This method is also called adversarial training. When training object detectors, it is very common to perform hard-negative mining. We take this notion and perform it in the other direction. Hard positive mining consists of replacing some training examples in each batch with FGS-generated adversarial examples effective on the current model [33]. We then minimize the training loss in the same way as when training undefended models. The rationale behind this technique is that the network will learn to ignore adversarial perturbations made to its inputs.

When the FGS method is used to build adversarial examples for adversarial training, it is better to set y as being the most likely predicted class given x rather than using the ground truth label of x. Discovered by [33], this is to prevent a form of label leaking where the accuracy of the adversarially trained network becomes artificially high on adversarial examples. They conjecture that this happens because FGS generates predictable transformations to the input. They show that this does not happen with iterative methods, possibly as these generate a more diverse set of transformations. In order to stabilize the model training, [33] use a weighted-average of the cross-entropy loss, to reduce the penalization of mistakes on adversarial examples compared to that on natural examples:

$$L = \frac{2}{n}\sum_{i=1}^{n/2} l(x_i, y_i) + \lambda \sum_{n/2+1}^{n+1} l(x_i + \delta_i, y_i)$$

Where $l(x, y)$ is the cross-entropy loss given a point-label pair (x, y) and δ is the perturbation generated from x. We have formulated the loss such that in any given batch, the ratio of the number of natural examples to adversarial examples is 1 (i.e. batch is half of one, half of the other). In the experiments, we also investigate training on perturbations generated by the BI attack, in accordance with [3].

2.6.4 Network Distillation

Network distillation is a technique that attempts to solve the issue that model requirements are different during training and deployment phases [34]. In the training phase, the model must only learn to extract a structure from the training data. It does not need to operate in real time and can require extreme computational resources. In contrast, deployment of a model to production often has requirements for latency and throughput in order to serve the model to a large number of users. The idea behind distillation is to first train one or multiple large models on the training set using the hard labels (i.e. one-hot), without considering computational requirements. The knowledge extracted from the training set can then be distilled into a single, more efficient network.

One way to achieve this is to train the student network on the soft labels produced by inferring the larger models on the training data. It is hypothesized that training the student model on the softmax probabilities of the larger models allows the student model to better understand the training set by taking advantage of the relative magnitudes of each class probability [34] conjecture. One problem is that the softmax layer will tend to push low probabilities close to zero even closer to zero, in turn making the magnitude of the gradient too small to be useful. One solution to this problem is to train the teacher and student models at a high softmax temperature T, setting the temperature back to 1 after the student network has been trained:

$$\sigma(z_j) = \frac{e^{\frac{z_j}{T}}}{\sum_{k=1}^{K} e^{\frac{z_j}{T}}}$$

When the student/distilled network is trained at temperature T, it will learn to scale the logits of the layer just before softmax such that its predicted probabilities are similar to the soft labels. When the temperature is reset back to one (at test time), the network will make extremely confident predictions, as the logit for the strongest class will completely overpower all the others. This phenomenon can be used to break attacks that rely on cross-entropy loss and the associated gradient, as the latter will generally be very close to zero and hence smaller than the precision of a floating point number. This is because the large logit values will completely saturate the softmax activation before the loss,

After the softmax activation, the operating regions for the logits will be in the extremely flat part at extremes on the x-axis. Because of this, the gradient for that logit with respect to the input will be extremely small, close to zero and very likely smaller than the precision of a floating point number. Clearly, this can be used to break attacks like FGS or TOS that depend on the training loss of the network to function. This should not, however, prevent pure-logit based attacks (such as C&W or

SPSA) from working, as they do not rely on softmax, and hence their gradients should not be affected by the distillation (apart from scaling).

2.6.5 Defending against Adversarial Examples is inherently hard

In this section, we have introduced a number of techniques that can be used to attempt to mitigate against adversarial attacks. The issue with the aforementioned mitigation techniques is that they do not allow us to make any guarantees with respect to model robustness once applied. Gradient masking, defined by [21], is a defense that results in the model that does not have 'useful gradients'. This is exactly what is achieved by the adversarial training and network distillation methods. Gradient masking is not a complete defense, as it does not defend against strong black-box attacks that do not make use of gradient information.

A concrete example of gradient masking is a model that does not expose its logits, instead returning the `argmax` of the logits. This is equivalent to breaking the differentiability of the model, as now an adversary cannot use output variations to figure out how to vary the input, as the model will always output the hard label. [21] show that it is still possible to break a defense of this type, through the training of a substitute model that can be used to perform the same predictions as the attacked model. By virtue of generalizability of machine learning models, the substitute model will likely learn the same decision function as the attacked model. It is then possible to perform a white box attack on the substitute model, which should transfer extremely well to the attacked model. This effect is illustrated in Figure 2.9, taken from [35], who perform a disciplined analysis of the threat model of adversarial examples. We also rely on the insightful analysis provided by [36].

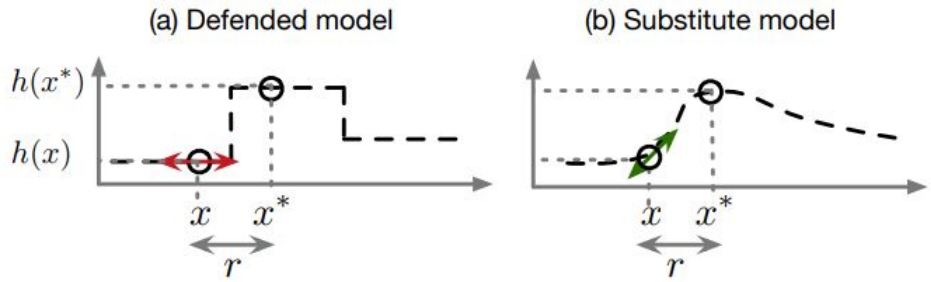


Figure 2.9: Illustration of why model substitution breaks a gradient masking defense. x is the natural sample, x^* is the adversarial example, resulting from perturbing x with r.

In the left image of 2.9, we have the decision function of the defended model, which has zero gradient for all x. On the right, we have the decision function of the substitute model. In essence, the decision function of the substitute is a smoothed out version of that of the defended model. Because the

substitute model's gradients are not masked, we can attack it and transfer the attack to the defended model.

These thought experiments show that while there have been a number of defenses that have been invented in to order to attempt to mitigate against adversarial perturbation, there is a fundamental imbalance in the capabilities of adversaries and defenders. However, it is still interesting to investigate the behaviour of the mitigation methods that we have previously introduced. Even if we cannot make any guarantees as to the absolute robustness of a defended model with respect to adversarial examples, we can at least attempt to do so empirically, in an effort to analyze the behaviour of these methods.

Chapter 3

Analysis, Design and Implementation

3.1 Analysis & Design

Designing a convenient interface for the attack algorithms was necessary in order to minimize the development time and amount of boiler-plate code necessary to execute these tests. This allows one to iterate upon different models and attack methods at an extremely fast pace. Fundamentally, a fooling algorithm can be visualized as a black box residing in between the user input and the input to the network. The algorithm takes the input from the user, applies an arbitrary transformation to that input and passes it on to the model. Figure 3.1 contains a block diagram presents a more detailed picture of the system, where both model blocks represent the same object. This is because most attacks need to forward the original images through the network once to compute the perturbation. This perturbation is applied inside the attack block, and the resulting adversarial example is then passed through the model block to measure the models response to the perturbation.

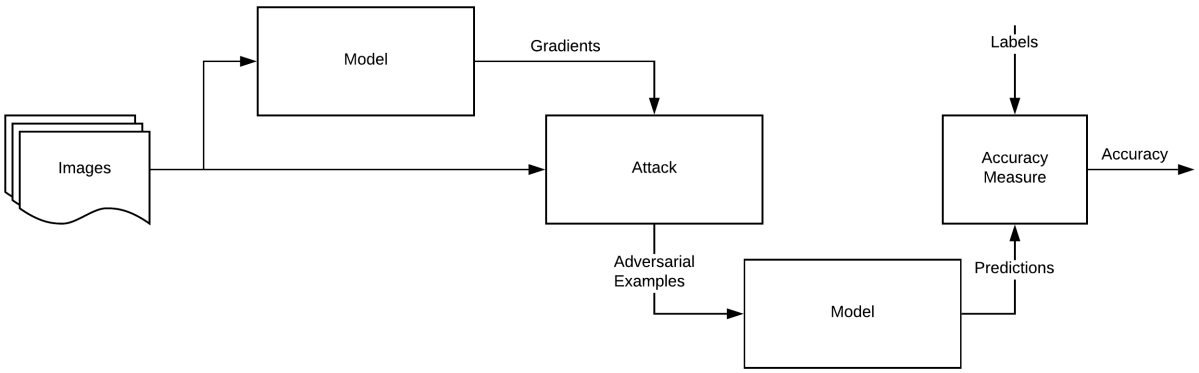


Figure 3.1: Block diagram of the system.

This vision was constantly considered when designing the interface for any one of the attack methods

used in this report. Ultimately, each attack method is hidden behind a function that takes the input of the entire system, the targeted model, and any other parameters specific to each method such as the bound on the norm of perturbation or the number of iterations for iterative methods. Each method will then build a graph representing the logic behind the attack, returning the output of the attack block. This output, which will contain the perturbed images, can then be connected to the input of our model of interest. We also decide to design flexibility of the accuracy measurement method into this model of the system, as this will likely vary depending on the nature of the applied attack method.

3.2 Implementation

3.2.1 Framework

The experiments for this project were implemented entirely in TensorFlow, an extremely high performance framework designed for tensor manipulation, for example that necessary when working with neural networks. TensorFlow was chosen over the plethora of other tensor manipulation frameworks due to prior experience, the fact that is an industry standard and because it is well supported on windows. Building a system in TensorFlow consists of defining a set of operations on a number of Variables or placeholders. Variables are mutable state contained in the graph and can also be marked as optimizable using one of the gradient descent optimizers contained in the framework (neural network weights are defined as trainable variables). Placeholders are inputs or access ports to the graph, where data can be passed into by the user. When using the hardware acceleration capabilities built-into TensorFlow, it is very important to reduce the amount of unnecessary communication between the host and the GPU, as this process involves large amounts of latency due to the limited bandwidth of the channel.

Implementing the attacks in addition to the models in TensorFlow is beneficial for a number of reasons. Testing the attacks on the relevant test data and evaluating their sensitibility to their parameters is extremely computationally intensive and usually requires a large amount of vectorised operations. It also largely simplifies the code for these attacks as it is not necessary to manually handle the generation of the adversarial perturbations. Instead, the attacks are designed as blocks of TensorFlow operations that can simply slotted together as desired, for example when testing the robustness of a model to any particular attack. This also has a large speed benefit for iterative methods, as there is no need to communicate data from the host to the GPU in between iterations, reducing the computational overhead of this class of methods. For example, the starting images (for which we need to fool a different class) can be passed into the graph using placeholders. Variables can then be initialized from

these placeholders, representing the mutable state of the iterative process. On every iteration, the graph performs a series of operations, updating the mutable state in a certain way. As a consequence, it is only required to communicate twice with the device: once at the beginning to load the starting images onto the device, and once at the end to read the perturbed images or to fetch the networks prediction given the generated adversarial examples.

TensorFlow, in itself, is an extremely verbose framework. It requires a substantial amount of boiler-plate code and formality in order to express the graph definition in a robust way. In order to simplify the definition of the neural networks used for the experiments, Keras, a high-level framework that can use TensorFlow as its backend, was used for its convenient model definition and inference routines. Keras [37] abstracts out a large amount of the low-level details necessary to TensorFlow, such as sessions and graph manipulation, instead presenting a simplified interface best compared to that popularized by Scikit-Learn [38].

3.2.2 Datasets and Models

Modified National Institute of Standards and Technology (MNIST) database for handwritten digit recognition

The MNIST dataset is by far the most popular and well-known dataset used to test the performance of machine learning methods in a computer vision context. It consists of 60000 28×28 grayscale images of digits from 0 to 9. It is a very easy dataset, on which many state of the art computer vision methods perform nearly perfectly. This dataset is extremely useful for debugging, as it is a small dataset, of relatively low dimensionality compared to the more complex image datasets such as ImageNet. As a consequence, it can be solved easily using a convolutional model with a small number of layers whose exact architecture has little effect on the overall accuracy. These small networks can be trained in a couple seconds, and their overall computational efficiency was extremely useful when debugging the attack implementations.

The following table shows the architecture of the network used in all the following MNIST experiments. The format of parameters for the convolutional layers is 'number of filters - filter size - filter stride - padding - activation function'. The format for the maximum pooling layers is 'pool size - pool stride - padding'. The format for the fully connected layers is 'number of neurons - activation function'. The below network achieves a test error of just above 1% after 4 epochs of training, using ADAM optimizer and cross-entropy loss. Using 'same' padding rather than 'valid' padding for the convolutions

is extremely important due to the fact that 'valid' padding produces zero-gradients for the k-1 right-most columns and bottom-most rows, where k is the size of the filter.

Layer	Output Activation Size
Convolutional - 32 - 3×3 - 1 - same - ReLU	$28 \times 28 \times 32$
Max Pooling - 2×2 - 1 - valid	$14 \times 14 \times 32$
Convolutional - 64 - 3×3 - 1 - same - ReLU	$14 \times 14 \times 64$
Max Pooling - 2×2 - 1 - valid	$7 \times 7 \times 64$
Fully Connected - 512 - ReLU	512
Fully Connected - 10 - Softmax	10

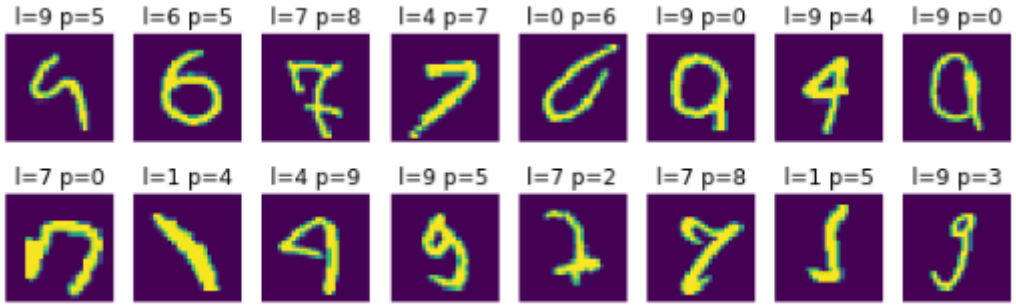


Figure 3.2: Mistakes made by our MNIST model

Figure 3.2 shows 16 examples of misclassified test examples. l denotes the ground truth label while p denotes the predicted labels. While some are examples where the network was legitimately wrong (e.g. upper row, second image from the left), some digits are extremely misleading (e.g. upper row, first from the right), or seem to be entirely mislabeled (upper row, second from the right). This ultimately shows that our model has learned a robust decision function and genuinely seems to have learned to generalize from our training set.

Canadian Institute for Advanced Research-10 (CIFAR10)

The CIFAR10 dataset [39] is similar in idea to the MNIST dataset in that it consists of 10 small and well-balanced classes of relatively low dimensionality $32 \times 32 \times 3$ images. Rather than digits, CIFAR10 contains images of 10 types of objects[1]. This dataset is extremely useful to test a given method on a more realistic use case: general object recognition. Objects contained in CIFAR10 are structurally more complicated than digits and the images themselves have large variations in terms of pose and quality expected of real life images. This dataset is substantially more difficult than MNIST,

[1] *airplane*, *automobile*, *bird*, *cat*, *deer*, *dog*, *frog*, *horse*, *ship*, *truck*

requiring more advanced network architectures and training methods to perform well. The below table shows the architecture for the model used for the CIFAR10 experiments. When compared to that used for MNIST, it is far deeper in order to better capture the hierarchal information inherent to natural images. It also follows the recommendation from [40] of going from wide but shallow (in terms of filters/activations) at the head of the network to narrow but deep at the tail. In this model, each convolutional layer is followed by a batch normalization layer in order to speed up (due to being able to use a high learning rate) and stabilize the training. Each block of two convolutional layers is followed by a maximum pooling layer to subsample the activations by a factor of 2×2. Dropout is used at the very end with a probability of 0.5, in order to further regularize the training procedure. This model is trained in 20 epochs, with ADAM optimizer and a 0.001 learning rate. Further training of the model does not bring any measurable improvements in validation accuracy. Horizontal flipping and random height and width crops are used to reduce train set overfitting, improving generalization. By using the predetermined 50k-10k train-test split, the model achieves a train error of 13% and a test error of 15%. Many different publications have gotten substantially better test error by using deeper and more complex architectures (such as residual networks), however it was chosen to keep the architecture complexity to a minimum in order to reduce the iteration time required for experiment runs.

Layer	Output Activation Size
Convolutional - 32 - 3x3 - 1 - same - ReLU - BN	$32 \times 32 \times 32$
Convolutional - 32 - 3x3 - 1 - same - ReLU - BN	$32 \times 32 \times 32$
Max Pooling - 2×2 - 1 - valid	$16 \times 16 \times 32$
Convolutional - 64 - 3×3 - 1 - same - ReLU - BN	$16 \times 16 \times 64$
Convolutional - 64 - 3×3 - 1 - same - ReLU - BN	$16 \times 16 \times 64$
Max Pooling - 2×2 - 1 - valid	$8 \times 8 \times 64$
Convolutional - 128 - 3×3 - 1 - same - ReLU - BN	$8 \times 8 \times 128$
Convolutional - 128 - 3×3 - 1 - same - ReLU - BN	$8 \times 8 \times 128$
Max Pooling - 2×2 - 1 - valid	$4 \times 4 \times 128$
Fully Connected - 256 - ReLU	256
Dropout - $p = 0.5$	256
Fully Connected - 10 - Softmax	10

FaceScrub

The final dataset used for the experiments is the FaceScrub [41] dataset. This dataset consists of 106k+ high quality mostly-frontal images of more than 500 celebrities. Originally, this dataset is distributed as a list of URLs in order to prevent copyright infringement by the creators (as the images are not owned by them). Conveniently, the creators of another dataset, MegaFace [42], distribute a pre-downloaded version of the FaceScrub dataset, designed to be used as a test set for the MegaFace challenge. While this prevents us from having to download the images ourselves, a large number of the images are either corrupt or entirely non-existent. To ensure that the dataset used for our experiments is as clean as possible, and to prevent problems caused a path pointing to a missing/corrupt image, each image in the dataset is checked, and removed from the selectable set for the experiments if there is any problem with it. After this processing step, of the original images, only 93.7k images remain.

In order to reduce the computational cost of using this dataset for the experiments, the dataset is subsampled to just 2000 randomly selected images of 20 people. This subsample is then split again in a 80%-train-20%-test fashion. We ensure that all selected classes are present in both splits by using a stratified split. The selected images are then extracted from the other images and saved in two new folders for train and test in order to guarantee the reproducibility and reduce the computational cost of the experiment runs.

Recognizing faces is an inherently difficult task. Deep learning methods designed to tackle this problem consist of extremely deep networks trained on millions of face samples. These networks are very expensive to train in terms of time and hardware and hence it was decided at the start of this project to use one of the many pre-trained models available online [2]. These models are generally trained to minimize a similarity loss in order to learn features that can be visualized as points contained in an n-dimensional hyper-sphere [43]. This way, faces can be converted to their respective embedding using the network, and compared in the feature-space by using a distance metric such as Euclidean/L_2 distance. Another way of using this model is to attach an additional fully-connected layer (called *bottleneck* layer) at the very end with a number of neurons equal to the number of classes we wish to classify. This is exactly equivalent to training a multi-class linear classifier on the face embeddings. We retrain a bottleneck layer on the train split of our FaceScrub subset, achieving 1.5% error on the test split. The weights for this linear classifier are then saved along with the weights for the rest of the network. The confusion matrix for the test split can be found in the appendix (Figure 14).

It has been proven that face-alignment, or aligning facial landmarks to a canonical pose, improves the

[2] https://github.com/davidsandberg/facenet

accuracy of face recognition by constraining the model to recognize the position of features relative to a fixed coordinate system. It is also a step that is generally ignored by the face recognition models themselves and hence must be performed manually before passing the faces through the model. In this project, a pre-trained Multi-Task-CNN [3] [44] was used to detect, crop and align faces to a universal pose. The output of this step consists of one face thumbnail per image, resized to the dimension of input of the face recognition network ($224 \times 224 \times 3$), an example of which can be seen in Figure 3.3.

Figure 3.3: (left) Original picture of *Kit Harrington*, (right) Cropped, aligned and centered image of *Kit Harrington* used for recognition.

3.2.3 Attack Optimization

In order to be able to efficiently experiment with the previously introduced attack methods, it is extremely important to formulate and implement the attacks such that they can be vectorized as easily as possible. Essentially, this means that they should be able to attack multiple images at the same time. Performing this optimization for FGS, BI, TOS and TI is simple, as it is easy to extend the TensorFlow graph to compute and apply gradients to multiple candidates at the same time. Subject to similar restrictions (in terms of memory usage) as that encountered when training the models, we can compute fooling images in batches of 128 for the MNIST model, and 32 for the CIFAR10 model. The maximum batch size for CIFAR10 is smaller than that of MNIST because the network is much deeper, requiring substantially more memory per batch item. FGS and TOS simply take the image tensor passed to the attack functions, and immediately apply the relevant operations, returning a tensor that will contain adversarial examples once executed. This tensor can be directly plugged into the classification model to measure how well any particular attack performs.

The iterative methods BI and TI require taking some care in the implementation as they require a looping mechanism to update the mutable state containing the current adversarial candidate. While it is possible to build a graph that mimics the unrolled optimization loop, it was ultimately decided to use the `tf.while_loop` [4] construct, which enables building a looping mechanism into the graph, simplifying the code for the attack and reducing the complexity of the graph itself.

[3] https://github.com/davidsandberg/facenet/tree/master/src/align
[4] assumes: `import tensorflow as tf`

Optimizing the C&W method was however far more challenging. The C&W method stops the optimization process once the network has been fooled by the current adversarial candidate. This means that there is an indeterminate number of iterations per image, with some processes converging faster than others. This means that if we naively vectorize the process, the speed of the implementation will depend on the number of iterations needed for the slowest converging instance. One idea was to keep track of which instances have converged, and only compute the loss (and hence forward the candidates through are model) for instances that have not converged. This if/else mechanism can be achieved using the `tf.cond` construct. Unfortunately, this construct always evaluates both of its possible branches, defeating the point of using it in the first place. The final implementation instead relies on the fact that a large majority of instances will converge in very few iterations. Using this information, we keep track of the proportion of converged instances at each iteration, stopping the execution completely when that surpasses 80%. We then extract the instances that have not converged (which is usually only a small portion of the batch) and start the process once again, iterating until all of the left over instances have converged. With this technique, on batches of size 128 on MNIST, we drop the batch time from 30 seconds to 2 seconds, a speedup of 15×.

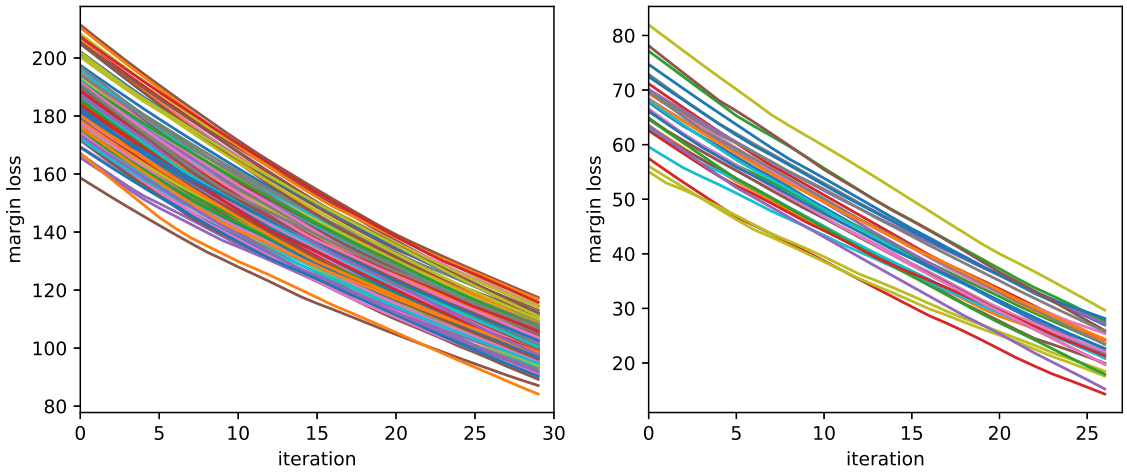


Figure 3.4: Loss (margin + L_2 penalization) as a function of the number of iterations for the first round (left) and second round (right) of optimization. The second round consists solely of optimization instances that did not converge in the first round.

Figure 3.4 shows both splits of the iteration process. The plot on the left shows the converged of the entire batch of 128 images. We can notice that while we wait for the 80^{th} percentile to converge, the top instances converge to the point of pushing the loss into the negative domain, meaning that those instances are pushing the adversarial candidates even further into the fooled class, increasing the networks confidence of those classifications. The plot on the right shows the convergence of the extracted batch of candidates. It consists of only $128 - \lceil 0.8 * 128 \rceil = 128 - 103 = 25$ images, all of

which converge in ≤ 70 iterations. Clearly, the models accuracy on this batch of adversarial examples is exactly 0, as all instances of the optimization have converged.

Implementing the SPSA attack is straight-forward. It was decided to implement it in a targeted rather than untargeted framework due to the increased desirability of targeted attack methods. Ultimately, two different implementations of SPSA were designed. The first does not make use of `tf.while_loop`, and hence the optimization must be manually stepped through using a python for loop. The other implementation uses `tf.while_loop`, with the update and stopping logic integrated directly in the TensorFlow graph. The second implementation is simpler and more elegant, and doesnt require the use of explicit `tf.Variable` to keep state of the optimization. In contrast, the first implementation is more verbose, and slightly more complex. There is however little to no speed difference between the two. The Adam optimizer logic was implemented from scratch, as the one contained within TensorFlow forces the use of analytic gradients.

3.3 Attack Comparison Metrics

In order to compare the various attack methods contained in this report, it is necessary to constrain them in some way. Usually, each attack is given a perturbation budget in terms of a certain norm (i.e. L_∞) that must be strictly respected. It is then possible to vary this budget in order to further analyze the behaviour of any given method. The main test metrics will be test accuracy on adversarial examples, target class accuracy for targeted methods and perturbation norm. The test accuracy is simply the model's accuracy on the test set of each dataset. For an adversarial example, a robust model should be able to correctly identify the class of the original nature sample. Generally however, we expect the test accuracy to decrease monotonically as we increase the perturbation budget as the attack will have broader range of options.

In order to compare attack methods, it is convenient to measure the p-norms of the perturbations with p different than that used to define the budget. In this report, we will use L_1, L_2, and L_∞ norms, defined as:

$$L_1(x) = \sum_i |x_i| \qquad L_2(x) = \sqrt{\sum_i x_i^2} \qquad L_\infty(x) = \max_i |x_i|$$

These metrics can be used to compare the exact effects of the various attacks and how they differ in terms of tactics. While measuring the accuracy (or lack thereof) of a model on adversarial examples generated using untargeted attacks is simple, it is also important to make sure that targeted attacks

force the network to predict the target class, rather than more generally making the wrong prediction. The target class accuracy is defined as the probability that the model with predict the class targeted by the attack given an adversarial example, given that the model predicted the correct class for the original sample. As a consequence, untargeted attacks such as FGS or BI will only measure the accuracy of the network given the perturbed images and the labels, while targeted attacks such as TOS, TI, C&W L_2 & L_∞ and SPSA will only be considered successful if the predicted label is that targeted by the attack.

Chapter 4

Testing & Results

4.1 Attacking the MNIST Model

4.1.1 Standard gradient methods

In this section, we will show the behavior of various attack methods as a function of the L_∞ budget allocated to the adversarial perturbation. We will compare targeted and untargeted attacks, including more advanced methods such as C&W Wagner and SPSA on more specialized benchmarks.

The first experiment consists of using each attack method to maximize the adversarial perturbation for the entire MNIST test set, given the convolutional model introduced in prior sections. For information, The BI and TI attacks are both performed for 10 iterations, with a step size of $\frac{\epsilon}{10}$. All methods also include a clipping step to make sure that the generated adversarial examples fall in valid image space. For targeted methods, we do not use the ground truth label, instead opting to target the least likely predicted class given the natural sample.

When comparing FGS and BI methods, it is clear that the iterative method is superior to the one step method. This is because the iterative method is able to better converge towards the local minima. For large perturbation norms, the single step method has a tendency to overshoot the local minima, causing attacks with large perturbations to perform worse than the same with a smaller step size. This doesnt happen for iterative methods, pushing the accuracy of the attacked model towards zero even for very large perturbations.

For targeted attacks, it is important to ensure that, in addition to encouraging the model to make a mistake, the attack should also push the model to predict the targeted class. Under a simple accuracy

Epsilon	FGS	BI	TOS	TI
0	99%	99%	99%	99%
0.01	99%	99%	99%	99%
0.1	86%	**77%**	95%	96%
0.2	41%	**6%**	69%	37%
0.3	13%	1%	23%	**0%**
0.5	7%	1%	9%	**0%**

Epsilon	TOS	TI
0.01	0%	0%
0.1	0%	0%
0.2	1%	**23%**
0.3	9%	**97%**
0.5	14%	**100%**

Figure 4.1: Test accuracy (left) and targeted class accuracy (right) of the attacked model for the four baseline attack methods as a function of the perturbation L_∞-norm

Epsilon	L_1				L_2			
	FGS	BI	TOS	TI	FGS	BI	TOS	TI
0	0	0	0	0	0	0	0	0
0.01	4.1	**3.5**	4.0	3.7	0.2	0.2	0.2	0.2
0.1	38.7	**31.5**	40.7	33.1	2.0	**1.6**	2.0	1.7
0.2	76.7	**58.8**	77.3	64.8	3.9	**3.0**	3.91	3.2
0.3	114.0	**82.1**	115.3	87.4	5.8	**4.2**	5.8	4.3
0.5	183.8	**121.7**	192.6	123.3	9.5	6.3	9.7	**6.2**

Figure 4.2: L_1 and L_2 norms of adversarial perturbations four baseline attack methods as a function of the perturbation L_∞-norm

test, the single step and iterative methods produce similar results to the untargeted attacks. It is only when we look at target class accuracy that we see large differences in the methods performance. As we can see in the results, the targeted one-step (TOS) attack has trouble getting the adversarial example predicted as the intended class, reaching a maximum of 14% target class accuracy at $\epsilon = 0.2$. We yet again see the single step method overshoots local minima of the objective function, causing worsening performance for large perturbation. The targeted iterative (TI) method performs remarkably, going from 23% target class accuracy at $\epsilon = 0.2$ to 97% target class accuracy at $\epsilon = 0.3$. We can clearly see that the iterative methods superior convergence abilities enables it to far outperform the single step method on this benchmark. Table 4.2 also shows that iterative methods outperform single step methods in terms of perturbation 1 or 2-norm given a fixed L_∞-norm.

Figure 3 shows a test image of class 1 that is adversarially perturbed with all attack methods, for varying perturbation norms. The targeted class, corresponding to the index of the smallest logit given the original image, is class 3. While the quality of the adversarial examples generated by single step methods quickly deteriorates to being essentially only noise, the iterative methods warp the image in a more precise way. With this result, it is not surprising that the target one step method has such low target class accuracy over the whole range of perturbation norms as for most examples, the one-step methods generate similar blobs for all images, regardless of class.

4.1.2 Carlini & Wagner L_2 Attack

In this series of experiments, we shall bound the maximum number of optimization iterations by 1000. We set the Adam learning rate to $\alpha = 0.1$. We define the convergence rate as the portion of individual optimization instances that produce an adversarial example inside of the allowed iteration budget. We will perform the C&W attacks in a targeted framework, as that is the most difficult requirement. The following image shows examples of the C&W L_2 attack applied to one MNIST image per class. Each row represents the results for each image, whereas each column corresponds to each targeted digit. In other words, for each image, we target of the other MNIST classes. The images in Figure 4.3 were generated with $c = 1.5$.

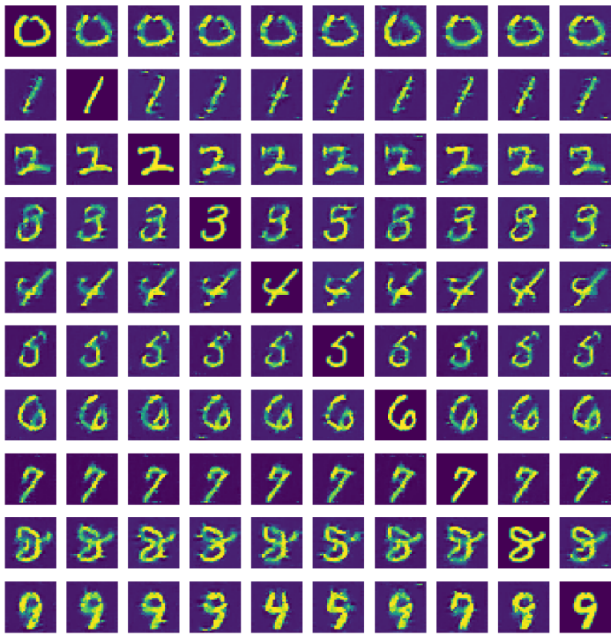


Figure 4.3: Grid generated from 10 MNIST images. Each column holds perturbed examples for targeted attacks to the given class. Each row holds a different attacked image, one for each MNIST class

We can see that the generated adversarial perturbations are extremely smooth and lack any noise entirely. This is in stark contrast to the baseline techniques which have a tendency to introduce substantial amounts of noise. We also see a phenomenon that doesn't appear for other adversarial methods, that is, when targeting a certain class, the digits themselves are warped to somewhat resemble the target class. An extreme example is when attacking an image of a 9, targeted as a 4 (last row, 5^{th} from the left). In this case, the upper 'bar' of the 9 disappears entirely, resulting in a convincing 4.

An interesting experiment consists of studying the influence of the norm penalty c on the outcome of the optimization. The results for this can be found in Table 4.4. For each run, we attack a small sub-sample of the MNIST test set, recording the number of iterations required for each optimization

instance to converge, as well as various statistics concerning the generated adversarial perturbations. For each experiment run, the statistics are computed from the instances that converge. From the table, we discover a general trend, in that as we increase the norm penalty c, we get substantially improved convergence properties at the cost of increasing perturbation norms. We also see that the confidence of the adversarial examples tends to increase with c until a point (between 2 and 3 for this particular example) where the generated examples are less confidently classified by the model. In this case, a good general trade-off between the different measured parameters can be found for $c = 1$.

c	E / σ iter. conv.	Convergence rate	L_1	L_2	L_∞	E adver. conf.
0.01	-	0%	-	-	-	-
0.1	-	0%	-	-	-	-
0.5	209/202	62%	37.1	2.2	0.68	76%
1	100/99	97%	84.3	3.8	0.77	83%
2	40/21	100%	175.4	6.6	0.71	77%
3	26/15	100%	223.2	8.4	0.69	66%
10	12/6	100%	297.6	11.2	0.61	58%

Figure 4.4: Mean & standard deviation of the number of required iterations to convergence, the convergence rate (portion of optimization instances that find an adversarial example), the $L_{1,2,\infty}$-norms of the generated perturbation and the mean adversarial confidence as a function of the C&W constant c for the C&W L_2 attack method, applied to the MNIST dataset

4.1.3 SPSA Attack

Unless explicitly mentioned, these experiments use a perturbation size budget of $|\delta|_\infty = 0.3$ and bound the number of iterations by 1000. The iteration process will stop when the candidate image is detected as fooling the targeted model. We use a base learning rate of $\alpha = 0.1$ for the Adam optimizer. We use a batch size (to compute the gradient approximation) of 64. We define the convergence rate as the portion of optimization instances that result in an adversarial example under the maximum iteration count of 1000. We define adversarial confidence as the confidence of the model given an adversarial example.

Figure 4.5 contains adversarial examples resulting from an SPSA attack applied to one test image per MNIST class. Each row corresponds to a different starting image of each digit from 0 to 9. The columns correspond to a different targeted class. The matrix diagonal contains the clean/unperturbed images. We can see that SPSA is able to build extremely convincing adversarial examples in all cases for this dataset, albeit with some noticeable noise present in most adversarial examples.

An interesting experiment is to vary the learning rate of the optimization process. From the results in Table 4.6, we can see that as we increase the learning rate from 0.01 to 0.1, we achieve faster

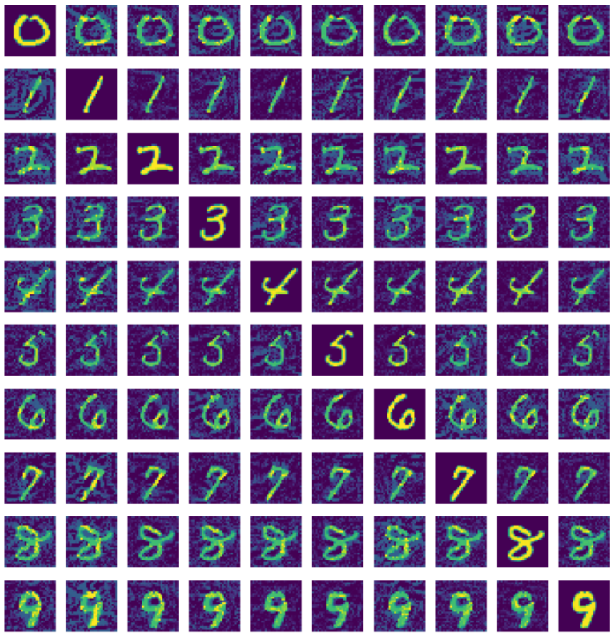


Figure 4.5: Grid generated from 10 MNIST images. Each column holds perturbed examples for targeted attacks to the given class. Each row holds a different attacked image, one for each MNIST class

convergence, at the cost of increased L_1 and L_2 norms of the resulting perturbation. This trend doesnt maintain itself when going from 0.1 to 1, as it is possible that the convergence process has become too unstable due to the very high learning rate. Regardless, we see higher average confidences on adversarial examples when we increase learning rate. It is possible that this is a direct side effect of the higher learning rate, as the optimization process is overshooting the minimum required perturbation, giving a higher adversarial confidence as a result.

α	E / σ iter. conv.	Convergence rate	L_1	L_2	E adver. conf.
0.01	133/113	100%	88.2	4.3	46%
0.1	56/130	98%	118.7	5.5	50%
1	80/224	96%	129.8	6.1	56%
0.01^+	-	99%	117.5	5.46	98%
0.1^+	-	97%	125.7	5.77	94%

Figure 4.6: Mean & standard deviation of the number of required iterations to convergence, the convergence rate (portion of optimization instances that find an adversarial example), the $L_{1,2}$-norms of the generated perturbation and the mean adversarial confidence as a function of the Adam learning rate with the SPSA attack method, applied to the MNIST dataset

We denote α^+ as the results achieved when the optimization process isnt stopped when fooling is achieved, but rather continued for a fixed 1000 steps. At the cost of a substantially more expensive optimization process, we can craft adversarial examples that are extremely confident. In these experiments, we can yet again see the effects of increasing convergence instability as we increase the learning

rate: for $\alpha = 0.1$, we achieve a lower average adversarial confidence and convergence rates and higher perturbation norms than for $\alpha = 0.01$.

In addition to the above experiments, we also measure the sensibility of the convergence stability to the batch size used when computing the gradient approximation. We prove empirically the importance of batch size on the accuracy of the approximation by evaluating the Euclidean distance between the analytical and SPSA-approximated gradients as function of the batch size. To improve the statistical significance of these results, we average the results for each batch size over 10 runs. As we can see in Figure 4.7, the error decreases exponentially with batch size, with diminishing returns for batch sizes larger than 256 (note the logarithmic x-axis).

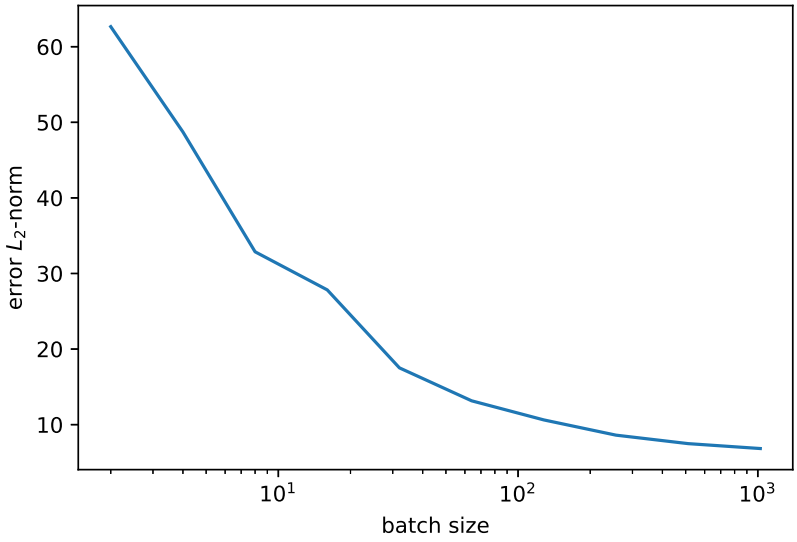


Figure 4.7: L_2 norm of the gradient approximation as a function of the batch size used to generate the approximation

As we can see from the table below, larger batch sizes tend to improve all aspects of the optimization, providing quicker convergence (in terms of iterations) and generating adversarial examples that are less perturbed. Larger batch sizes also stabilize the optimization process when using high learning rates. This is caused directly by the fact that the accuracy of the gradient approximation increases as we increase the number of offsets we sample. This can be seen in the last row of the table, where a learning rate of $\alpha = 1$ was used instead of the normal $\alpha = 0.01$. Compared to the same experiment with a batch size of 64, we achieve convergence on average in $8\times$ less steps (with a standard deviation $13\times$ smaller). When taking into account the batch size itself, using a batch size of 256 at a learning rate of $\alpha = 1$ instead of 64 at the same learning rate results at approximately half the computational complexity ($256 \times 11 = 2816$ model/loss evaluations $64 \times 80 = 5120$ model/loss evaluations, $\frac{5120}{2816} = 1.81$).

Batch size	Adam lr α	E / σ iter. conv.	Convergence rate	L_1	L_2	E adver. conf.
16	0.01	268/249	94%	97.2	4.56	48%
64	0.01	133/113	100%	88.2	4.3	46%
256	0.01	77/56	100%	**83.6**	**4.2**	47%
64	1	80/224	96%	129.8	6.1	56%
256	1	11/17	100%	131.9	6.1	**60%**

Figure 4.8: Mean & standard deviation of the number of required iterations to convergence, the convergence rate (portion of optimization instances that find an adversarial example), the $L_{1,2}$-norms of the generated perturbation and the mean adversarial confidence as a function of the batch size and Adam learning rate for the SPSA attack method, applied to the MNIST dataset

4.1.4 Adversarial Generalizability

In this experiment, we investigate the possibilities of generating universal perturbations through the application of the adversarial generalization method. In order to test the effectiveness of this method, we devise a simple experimental protocol. As we perform these attacks in a targeted setting, we generate one perturbation vector for each class in the dataset. In order to generate said perturbations, we perform an optimization over a small subset (1000 images) of the training set. For each class, we can evaluate the effectiveness of the attack by measuring the average target class accuracy for that class. That is, given images that do not belong to the targeted class, we measure the probability that the formed adversarial example is classified as the targeted class. We also make sure to compute the confidence statistics using only images whose prediction has been pushed towards the correct target class.

The following results will all be computed on the MNIST test set, which is both disjoint from the model training set and from the perturbation training set. In order to perform the optimization, we use Adam optimizer with a learning rate of 0.001 (although from empirical results, the exact learning rate matters little). We set $c = 1$ (the margin loss scale factor) and do not change it during the course of these experiments. We train the perturbation for 100 epochs, where each epoch means optimizing the perturbation on the entirety of the chosen subset. To reduce the number of variables, we perform this test solely for the generalized C&W method, as it is sufficient to show a general trend in the results. For this method, we employ both the C&W loss function, where l denotes the targeted class (in both expressions):

$$f(x) = \max_{i \neq l} Z_i(x) - Z_l(x)$$

And the cross-entropy of the softmax logits, where $q_i(x)$ is the i^{th} softmax logit:

$$f(x) = -\log_2 q_l(x)$$

The results for this experiment are shown in Table 4.9. We can see that regardless of which loss function is used, given a sufficient perturbation budget, we can generate a perturbation that is able to reliably push the prediction towards any targeted class. As we reach a perturbation L_∞-norm of $\epsilon = 0.5$, the attack success is dramatically increased, reaching 100% accuracy when using either classification loss. From these results, this attack is extremely effective, and shows that, at least on the MNIST dataset, we can reliably learn perturbations that target an arbitrary class.

Epsilon	Target class acc.	Mean conf.
0.01	2%	30%
0.1	18%	37%
0.2	68%	55%
0.3	96%	74%
0.5	100%	96%

Epsilon	Target class acc.	Mean conf.
0.01	2%	30%
0.1	20%	37%
0.2	72%	59%
0.3	95%	79%
0.5	100%	97%

Figure 4.9: Targeted class accuracy and mean confidence of adversarial predictions for the generalized attack method using cross-entropy loss (left) and margin loss (right), applied to the MNIST dataset.

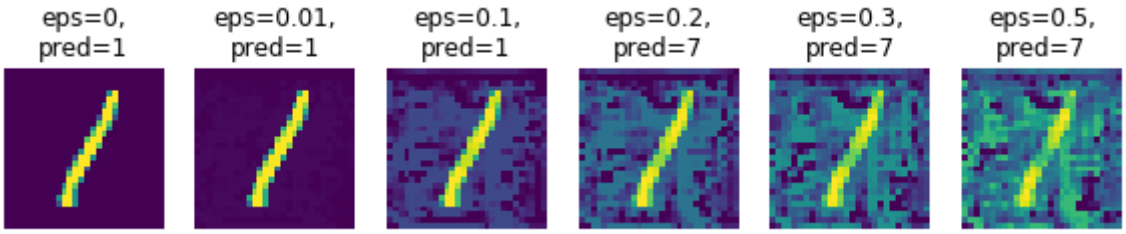


Figure 4.10: Adversarial examples generated using the generalized attack method. Each image corresponds to attacks of increasing perturbation budget ϵ

Figure 4.10 shows adversarial examples of a 1 digit. As we increase the perturbation budget ϵ, we get a large increase in background noise in the image, while the prediction produced by our model shifts from being correct at 1 to wrong at 7. Looking closely, we can see that the general shape of the perturbation is similar across the images. This shows that when given a larger budget, the optimization can venture farther away inside the hypercube centered at the natural sample but also that regardless of budget, the optimization seems to always converge in the same direction, showing that our objective function is very smooth. This is a possible explanation as to why the Adam learning rate had little effect on the overall convergence.

Figure 4.11 contains examples of adversarial examples for $\epsilon = 0.3$. Like before, each column represents a different targeted class, whereas each row represents a different base image. The diagonal of the matrix holds the unperturbed/natural examples. Even though we can clearly see the perturbations in all examples, these are not sufficient to overpower the digit itself, and the resulting adversarial examples are extremely convincing. It is important to note that all of the adversarial examples are

valid images (i.e. they lie in $[0, 1]^d$) and the perturbations all respect the claimed perturbation budget (albeit with negligible overshoot due to floating point rounding errors). Notice in the columns, the perturbations all have exactly the same shape.

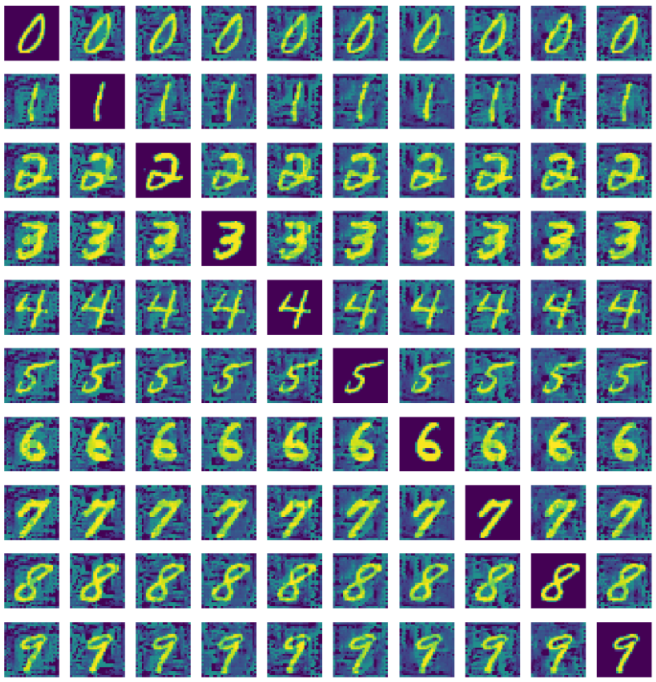


Figure 4.11: Grid generated from 10 MNIST images. Each column holds perturbed examples for targeted attacks to the given class. Each row holds a different attacked image, one for each MNIST class

This series of experiments enables us to further analyze the behavior of adversarial examples. Due to the fact that we can reliably target any class from any other class, we can see that the model becomes essentially blind to the contents of the image, instead focusing on the perturbations. In Figure 4.12, the top row consists of the natural sample and the adversarial sample repeated. The bottom row contains saliency maps [45], which correspond to the magnitude of the gradient of a certain logit with respect to the image. The left map is that of the original image with respect to the logit representing class 1. The middle and right maps are that of the adversarial example with respect to the 1 and 7 classes respectively. We can see that in the natural example, the network focuses on the pixels where the digit is located. On the adversarial example however, while there still seems to be some activity around the digit itself, there is substantial attention on the outside of the image. As expected, this shows that the network is forced into paying more attention to pixels that have nothing to do with the digit in the original image. In essence, the network stops being able to see the digit itself among the surrounding perturbations. Clearly, a human viewer is still fully capable of classifying the digit correctly.

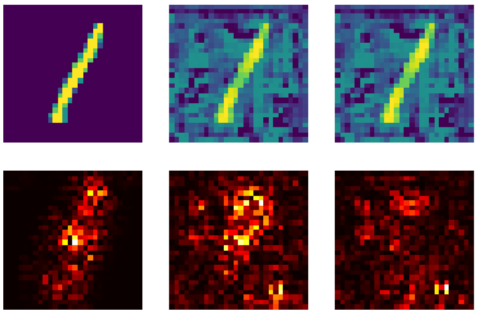


Figure 4.12: Saliency maps for three images. Left: natural sample, saliency with respect to class 1, Middle/Right: Adversarial example, saliency with respect to classes 1 and 7

4.2 Mitigating Adversarial Attacks

4.2.1 Adversarial Training

In this experiment, we will show that it is possible to defend against some of the attack methods introduced previously be actively training the model on adversarial perturbations, using the following methodology. We reuse a MNIST model trained on the entire training set. We first measure this models undefended robustness to various generation techniques, while varying the L_∞-norm budget of the perturbation. We then retrain the model (from scratch) for three epochs by using 50%-50% normal-adversarial batches. In all experiments, the adversarial example loss weight $\lambda = 0.3$. The first experiment consists of training on BI-generated perturbations, testing the robustness of the model to FGS attacks. We vary both the training perturbations and testing perturbation L_∞-norms respectively in $\{0, 0.1, 0.2, 0.3, 0.5\}$ and $\{0, 0.01, 0.05, 0.1, 0.2, 0.3, 0.5, 0.6, 0.7\}$. Figure 4.13 shows the test accuracy and fooling probability for this grid, where each curve represents the results for a given training perturbation ϵ.

Epsilon	$\epsilon_{train} = 0$	0.1	0.3	0.5
0	**0.989**	0.983	0.974	0.978
0.1	0.849	0.928	**0.944**	0.855
0.2	0.314	0.731	**0.848**	0.302
0.3	0.113	0.406	**0.609**	0.214
0.5	0.063	0.037	0.074	**0.104**

Epsilon	$\epsilon_{train} = 0$	0.1	0.3	0.5
0	0	0	0	0
0.1	0.151	0.067	**0.046**	0.141
0.2	0.691	0.269	**0.147**	0.702
0.3	0.894	0.599	**0.393**	0.789
0.5	0.940	0.972	0.942	**0.900**

Figure 4.13: MNIST test accuracy (left) and target class accuracy (right) as a function of the training perturbation budget (columns) and testing perturbation budget (rows)

We can see that the undefended model has trouble predicting the generated perturbations, and, as we increase ϵ, the models accuracy on the adversarial examples increases as well, while the accuracy on natural examples is relatively constant. Adversarial training greatly benefits the robustness of the

model to adversarial perturbations. For example, when we target a L_∞ budget of $\epsilon_{train} = 0.3$ during training, the fooling probability at test time drops from 0.894 to 0.393. We also notice that given a model trained at a certain L_∞ budget, it becomes extremely robust to adversarial perturbations generated with norms lower or equal to that budget. The robustness of the model then drops substantially when the norm of the perturbation surpasses its training budget. This makes sense as given perturbations with a certain norm, the model is able to adjust its decision boundary to go around a ball of radius the norm used for training (as illustrated in Figure 2.9). As a consequence there is no reason why it would be any more robust to adversarial perturbations of norm larger than the models training budget when compared to the undefended model.

Interesting behavior arises when we train the model with a adversarial training budget of $\epsilon = 0.5$. In this case, we get a drop in robustness to adversarial perturbations when compared to lower training budgets. This result is extremely interesting as the accuracy on natural examples is still constant around 98%. We can conjecture that this happens because the model is overfitting these adversarial-perturbed training examples, as they are so heavily perturbed (due to the large L_∞ norm from natural examples) that the network is able to infer the label this way. In a way, this is a form of label leakage, with consequence that the model has much higher accuracy at train time than test time.

4.2.2 Model Distillation

In this experiment, we show that while model distillation can be used to defend against training-loss based attacks such as FGS and BI, distilled models are still vulnerable to logit and gradient free attack methods such as C&W and SPSA. First of all, we train a teacher network at temperature T on the hard labels, represented as one hot vectors. We then map the training set onto the soft labels by passing it through the teacher network. We train another identical network at temperature T on the soft labels, this network is the distilled network. To make the distilled network robust to training-loss perturbations, we simply set the softmax temperature to 1, and produce predictions like before.

Unsurprisingly, the distilled network is completely immune to FGS and BI attacks, as the accuracy on the perturbed images is constant as a function of the perturbation L_∞-norm ϵ. When the distilled model is trained at high temperature, it simply scales its outputs in order to make its predictions correct. At test time, when the temperature is set back to 1, the logits will be much larger than they actually need to be. To illustrate this claim, we measure the average value of the largest logit for both undefended and distilled networks (with temperature set to 1). While the undefended model has average value 13.3 on the MNIST test set, the distilled model has average value 1077.7. If we compute the ratio between these two means ($\frac{1078}{13} = 82.9$), we can clearly see that the distilled network has

approximately scaled its logits by the temperature. It should be noted that distillation should not affect the networks test accuracy as the predicted class is independent of the logit magnitude.

When the logits are passed through the softmax, the output is either close to zero (and hence zero when quantized to machine precision), or close to 1. This is because large positive or negative inputs produce outputs in the softmaxs saturated region, meaning that the gradient is close to 0 for every output, preventing the attack methods from making any progress. This echoes the vanishing gradient problem characteristic of the logistic activation function, a reason why most modern deep learning applications use rectified units instead. If we set the temperature back to what was used during training, the distilled model becomes vulnerable once again, as if it were undefended.

In order to attack a distilled model, we can still use logit based attack methods rather than methods relying on the training loss function. With C&Ws L_2 method, the distilled model achieves 0% accuracy on adversarial examples. Figure 4.14 shows an example of the same C&W attack performed on both undefended and distilled models. The left image is the ground truth image, along with the corresponding predicted image (by the undefended network). We then perform the optimization using the C&W objective, until we detect that the model under attack has been fooled. As we can see, the resulting adversarial examples are extremely similar. Both perturbations have $|\delta|_\infty = 1$ and the perturbations for the undefended and distilled models have respectively $|\delta|_2 = 3.27$ and $|\delta|_2 = 3.33$. We notice that the C&W loss weighting factor c needs to be scaled approximately by the inverse of the ratio of the scaling factor between the undefended and distilled network.

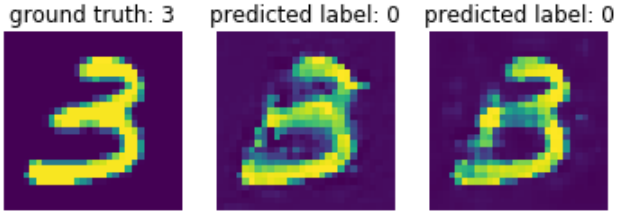


Figure 4.14: Example of the same C&W attack performed on an undefended (middle) and distilled (right) model on the same natural sample (left)

4.2.3 Adversarial Transferability

This experiment will try to measure how well adversarial examples generated using different attack methods transfer between models trained to perform the same task. The objective is to show the influence of architecture and training data on the robustness or susceptibility to adversarial transferability between different models. In order to achieve this, we will train three models: a convolutional network (using the previously mentioned architecture), a linear softmax classifier and a 2 hidden-layer

multi-layer-perceptron (MLP) with $256-256-10$ neurons in each layer and ReLU activation function. All three models are trained with cross-entropy loss. Each model will be trained on disjoint subsets (each $\frac{1}{3}$ of the original set) of the training set, for three epochs. The linear, MLP and convolutional models respectively achieve 90.8%, 96.1%, and 98.3% test accuracy on the MNIST test set.

We will generate adversarial examples from the MNIST test set using the convolutional model, evaluating the test accuracy probability of each of the three models on the perturbed examples as a function of the L_∞-norm of the perturbation δ, $\epsilon = |\delta|_\infty$. The results with $\epsilon = 0$ denote the accuracy achieved on natural test samples (without any perturbation). The perturbed examples are also clipped to $[0, 1]$ to ensure that they fall on feasible set of images. We will run this experiment for each white box attack method. The following tables summarize the full results that can be found in the appendix.

ϵ	Model 1	Model 2	Model 3
0	0.976	0.963	0.909
0.1	0.825	0.922	0.864
0.2	0.406	0.741	0.682
0.3	0.073	0.499	0.427

ϵ	Model 1	Model 2	Model 3
0	0.976	0.963	0.909
0.1	0.746	0.928	0.870
0.2	0.080	0.801	0.734
0.3	0.017	0.576	0.488

ϵ	Model 1	Model 2	Model 3
0	0.983	0.961	0.908
0.1	0.940	0.920	0.874
0.2	0.643	0.646	0.698
0.3	0.136	0.352	0.411

ϵ	Model 1	Model 2	Model 3
0	0.983	0.961	0.908
0.1	0.945	0.924	0.875
0.2	0.353	0.740	0.781
0.3	0.012	0.504	0.613

Figure 4.15: MNIST test accuracy as a function of the model architecture and attack perturbation budget for the FGS (top-left), BI (top-right), TOS (bottom-left) and the TI (bottom-right) attacks. Models $1, 2$ and 3 are respectively the convolutional, MLP and linear models.

As we can see from the results in Table 4.15, adversarial perturbations learned on the convolutional model are able to transfer relatively well to the MLP and linear models. This result is understandable as all models are trained on exactly the same task. When combined with the fact that they achieve similar scores on natural examples, this means that it is highly likely that the models have learned similar decision boundaries, meaning that they should be similarly vulnerable to adversarial perturbations. From these results, it would seem that the linear model is slightly more susceptible to adversarial transferability than the MLP. It is difficult to give an extract explanation for this behavior, however we can conjecture that this happens because the linear model generally has the least capacity of the three studied models, meaning that it will likely be the least robust to adversarial perturbations.

Out of the four methods, the single step targeted (TOS) attack seems to transfer the best, achieving fool probabilities on the MLP and linear models of respectively 63.9% and 56.0% at $\epsilon = 0.3$. While FGS achieves similar transferability on the linear model, the MLP transfer result is especially impressive considering that TOS beats FGS by 15 points.

ϵ	Model 1	Model 2	Model 3
0	0	0	0
0.1	0	0	0
0.2	0.003	0.002	0.004
0.3	0.051	0.009	0.011

ϵ	Model 1	Model 2	Model 3
0	0	0	0
0.1	0.052	0.047	0.049
0.2	0.353	0.334	0.245
0.3	0.865	0.639	0.560

Figure 4.16: MNIST target class accuracy as a function of the model architecture and attack perturbation budget for the TOS (left) and TI (right). Models $1, 2$ and 3 are respectively the convolutional, MLP and linear models. Recall that the perturbations are learned on model 1, transfered to models 2 and 3.

When working with targeted attacks, it is also important to measure the target class accuracy. In order to measure this, we compute the common correctly classified test examples by each pair of models in {(conv, MLP), (conv, Linear)}. We then compute the target class accuracy derived from a models predictions on adversarial samples by checking whether it has predicted the same class as was originally targeted by the convolutional model. As before, the target class given a natural sample and the convolutional network, is that which has the smallest logit. The results for this are contained in Table 4.16. We can see that while TOS fails completely at transferring the target class of an adversarial sample, TI manages to transfer the targeted class for larger perturbation norms. The fact that the targeted class of an adversarial example transfers between model seems to confirm the hypothesis that the models are learning very similar decision functions, even though they have fundamentally different architecture and capacity.

4.2.4 Input Randomization

A very promising technique to make the networks robust to adversarial perturbations is to add noise to the input. As seen previously, neural networks are extremely robust to additive noise and random translations. We can take advantage of the fact that the effectiveness of adversarial perturbations is highly dependent on the precise location of the perturbed pixel. This is because the perturbed pixel values correspond to the optimal gradient necessary to push our classification in the desired direction. Hence when the pixel values change, even slightly, the classification will likely be completely different. For these experiments, we will investigate four setups:

- Clean: we do not apply defensive randomization to the network input. This is equivalent to attacking an undefended network.
- Additive Gaussian Noise (AGN): add $N(0, 0.2)$ to the input and clip the result to $[0, 1]$.

- Random Translation and Cropping (RTC): assuming $N \times N$ pixel image, we first resize to $(N+3) \times (N+3)$, then take a random $N \times N$ crop.
- AGN + RTC: we first apply AGN then RTC.

In addition to this, we can either generate and apply the adversarial perturbations on the same noisy model or more realistically, generate our adversarial perturbations on the undefended model and apply them to the models with randomized inputs. This second method is equivalent to best case adversarial transferability, where the attacker trained model is exactly the same (in terms of architecture, training data, and weights), but the black box model is defended with randomized perturbations. The results for the latter, most importantly, can be found in Figure 4.17. The rest of the results can be found in Appendix .7.

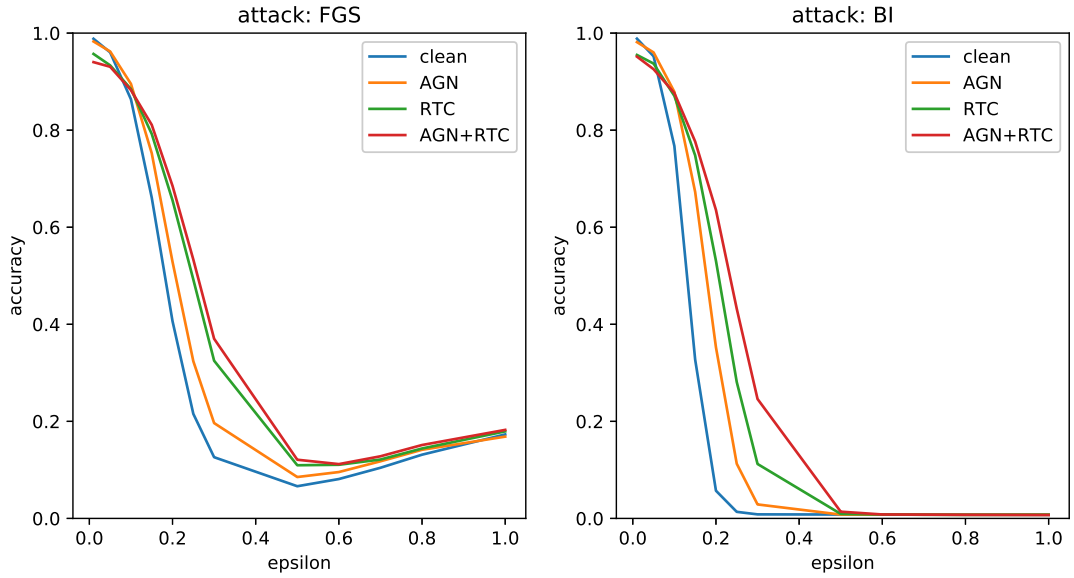


Figure 4.17: Test accuracy on the MNIST dataset as a function of the perturbation L_∞-norm given clean inputs and inputs perturbed using the three randomization methods.

Regardless of the which methodology we use, we notice that randomizing the inputs to our network improves the networks robustness to adversarial attacks with small perturbation norms, whether single step (FGS) or iterative (BI). As can be seen in the following table, this increase in adversarial robustness however comes at a slight cost in accuracy on natural examples and an increase in computational complexity of the model.

As we increase the strength of the attack, the effectiveness of this defense drops and even worsens results in the case of the FGS attack from the first method. Normally, the larger step sizes turn out to be suboptimal as the gradient algorithm effectively jumps over the local minima. Randomizing

Method	Clean	AGN	RTC	AGN+RTC
Natural Accuracy	99%	98.5%	96.3%	94.9%
Time	$45\mu s$	$53\mu s$	$62\mu s$	$73\mu s$

Figure 4.18: Natural accuracy and time per inference on the MNIST test set as a function of the randomisation method

the inputs to the network seems to prevent this behavior, and the FGS attack becomes even more effective. It is unsurprising that the other results show reduced effectiveness of this defense for larger perturbation sizes when using single step methods (see Figure 23), as the visual aspect of the data begins to deteriorate extremely fast when $\epsilon \geq 0.5$, becoming extremely distant from natural samples. The outcome of the tests using iterative is more interesting however, as the perturbation seems to be more precise. Rather than simply generating an adversarial example that is complete noise like the large norm single step methods, the perturbations arising from iterative methods seem to contain more intricate features, reminiscent of the MNIST classes.

The above results seems to disprove our original hypothesis that randomly perturbing an input before passing it through the network should break adversarial attacks, as it was assumed that these rely on specific perturbations at specific pixel locations to work. We can clearly see that while we get a substantial improvement in model adversarial robustness for attacks of small perturbation budget, as we increase the budget, the attacked model's accuracy drops back to that of the undefended model. This would indicate that the way adversarial perturbations function is more complicated as expected. The results seem to show that the exact location of the perturbed pixels doesn't matter, as long as the general aspect of the adversarial example is maintained. This begs the question of whether the inherent translation invariance of convolutional networks is causing this behaviour, however additional experiments would be required in order to verify this hypothesis.

4.2.5 The Influence of Model Capacity on Adversarial Robustness

In this series of experiment, we will evaluate the importance of model capacity in defending against adversarial perturbations. A models capacity is defined as the complexity of the relationship it can model. For this application, we would expect the robustness conferred by adversarial training to increase with model capacity. This is because more complex models will be able to better represent the correct decision function necessary to protect the subspace around natural examples. The protocol for this experiment is to first evaluate the robustness of each undefended model (there should not be much difference between the three). We will then perform simple adversarial training with a BI trainer and a FGS adversary. We also limit the L_∞-norm of the perturbations to $\epsilon = 0.3$.

In order to modulate the capacity of our three models, we reuse the same architecture as previously introduced and vary the number of filters and neurons in the two convolutional layers and bottleneck layer by a certain factor. The following table details the exact architectures:

Model	# Layers	Layer filters	# Bottleneck neurons
Small	2	$32 - 64$	128
Medium	3	$32 - 32 - 64$	256
Large	4	$32 - 32 - 64 - 64$	512

Figure 4.19: The three convolutional model architectures to be used in this experiment

ϵ	Small	Medium	Large
0	0%	0%	0%
0.1	15%	15%	15%
0.3	92%	**90%**	88%
0.5	98%	**94%**	**94%**

ϵ	Small	Medium	Large
0	0%	0%	0%
0.1	5%	3%	**2%**
0.3	35%	**23%**	27%
0.5	**87%**	88%	95%

Figure 4.20: Fool probabilities for undefended models (left) and the adversarially trained models (right) as a function of the perturbation budget ϵ

We train each model for three epochs on the entire MNIST training set. All models reach test errors with less that 0.1% difference with each other. The left results in Table 4.20 show that all three models are extremely vulnerable to the perturbations, with fooling probabilities easily reaching over 90%. Interestingly, the medium sized models is more robust than the largest model, followed by the smallest model. The next step is to retrain each model with adversarial training. Similarly to the adversarial training experiment, we train each model for three epochs using batches of half natural half adversarial samples. The results, shown on the right of Table 4.20, show that all models benefit from adversarial training. We can see that the most robust model is the medium sized one, while the smallest sized behaves similarly with an accuracy curve shifted down. The largest models accuracy starts out well, following that of the medium model, but drops sharply when $\epsilon \geq 0.3$ (see Figure 22). In this case, it is possible that the largest model is yet again overfitting on the training adversarial samples. However of the three models, once adversarially trained, the largest model has the highest accuracy of the three (small: 96.96%, medium: 97.86%, large: 98.17%). This is because the large models capacity advantage over when compared to the others allows it to simultaneously represent an accurate mapping for natural examples while being relatively robust to the perturbations.

4.3 Attacking the CIFAR 10 Model

4.3.1 Baseline Attacks

In this experiment, we shall apply a number of the attack methods introduced in this report on the CIFAR10 trained convolutional network. We will identify possible differences in this models behavior when under attack compared to that of the MNIST model. Intuitively, one would expect this model to be easier to fool than the MNIST model, solely due to the fact that this dataset is more complicated, leading to less robust and confident classifiers. In addition, the MNIST dataset is essentially noise free, while the CIFAR10 dataset contains noise intrinsic of generalized object recognition, due to pose and appearance variation of the objects, but also due to the crippling quantization noise caused by the reduction in resolution from the original images down to the dataset resolution.

Epsilon	FGS	BI	TOS	TI
0	85%	85%	85%	85%
0.01	52%	**42%**	62%	67%
0.1	14%	12%	6%	**2%**
0.2	13%	12%	7%	**2%**
0.3	12%	11%	8%	**3%**
0.5	12%	11%	9%	**3%**

Epsilon	TOS	TI
0.01	0%	**3%**
0.1	7%	**80%**
0.2	6%	**81%**
0.3	4%	**79%**
0.5	4%	**76%**

Figure 4.21: Test accuracy (left) and target class accuracy (right) of the attacked model for the four baseline attack methods as a function of the perturbation L_∞-norm

Epsilon	L_1				L_2			
	FGS	BI	TOS	TI	FGS	BI	TOS	TI
0.01	30.6	**22.8**	30.59	23.1	0.6	**0.5**	0.6	**0.5**
0.1	303	127.6	302.8	**126.7**	5.5	**2.8**	5.5	3.0
0.2	596.8	**175.0**	596.7	223.2	10.9	**3.9**	10.9	5.6
0.3	880.4	**212.2**	880.3	337.5	16.1	**4.8**	16.1	8.1
0.5	1408.6	**296.3**	1407.7	549.7	26.1	**6.6**	26.1	13.3

Figure 4.22: L_1 and L_2 norms of adversarial perturbations four baseline attack methods as a function of the perturbation L_∞-norm

As we can see from Table 4.21, the attacks are very effective given this model, dataset pair. Regardless of whether the method is single step or iterative, the attacks are able to reduce the test accuracy of the model to that of a random classifier. Interestingly, both untargeted methods have trouble reducing the models accuracy lower than 10%, while both targeted attacks are not affected by this issue. In the general case, iterative methods are still stronger than their single step counterparts. We also notice that the single step methods do not reduce in strength as the perturbation budget is increased

past $\epsilon = 0.5$. An explanation for this could be that the optimization landscape is smoother for the CIFAR model than the MNIST model, allowing large-stepped single step methods to preserve their effectiveness.

Analyzing the target class accuracy of the targeted methods shows similar results to that found with MNIST. That is, the single step method is generally unable to push the classifier to predict the targeted class, while the iterative method achieves this very easily. We notice that the large bump in the targeted iterative attacks target class accuracy happens far sooner for the CIFAR10 model than the MNIST model. We only require an L_∞ budget of 0.1 to achieve 80% target class accuracy on CIFAR10, compared to an approximate L_∞ budget of 0.2 to achieve the same on MNIST. Finally, we also see that the target class accuracy seems to be upper-bounded by the CIFAR10 models test set accuracy on natural examples.

A possible explanation as to why this model is less robust to adversarial perturbations could be linked to the models capacity and the difficulty of this dataset. We analyze the visual aspect of a perturbed sample originally correctly classified by our model as cat but with targeted class ship (see appendix). We can see that especially for high perturbation norms, the perturbations generated by single step methods are similar between the two studied datasets, in that they both produce mostly noise, and any visual aspect belonging to the original class has disappeared. In comparison, the iterative methods give quite different results for the two datasets. The perturbations for the MNIST dataset mostly warp the digit until it is unrecognizable (but not noisy), however, the CIFAR10 perturbations dont seem to add substantial amounts of noise, even for very high perturbation norms. It should be noted that the visual comparison of adversarial examples isnt completely fair as perturbations are CIFAR are less visible due to it consisting of color images. In contrast, the absolute black background of the MNIST dataset means that even minute changes to the image are immediately visible, as we can see for all attack methods for perturbations as small as $\epsilon = 0.05$.

4.3.2 Adversarial Generalizability

We perform an experiment regarding the behaviour of our proposed adversarial generalizability method with regards to the CIFAR10 dataset and its substantially more complex model. This experiment will also serve to show the effect of dimensionality of this method, as CIFAR10 has approximately $4\times$ the number of features of MNIST ($\frac{32\times32\times3}{28\times28} = 3.92 \approx 4$). As this model was trained with images mapped to $[-1, 1]$, we modify the formulation slightly such that the adversarial examples $\hat{x}_i$ are squeezed to the correct interval:

$$\hat{x}_i = \tanh\left(X_i + \delta\right)$$

We select a subset of 1000 images from the model's training set to perform the attack. We also sample an extra 100 images from the train set to form the attack validation set. We vary the L_∞-norm of the perturbation in $\{0.01, 0.1, 0.2, 0.3, 0.5\}$, and target each CIFAR10 class, in order to analyze the target class accuracy achieved by the generated perturbations. To reduce the computational cost of the experiments, we only investigate the optimization using C&W margin loss.

Epsilon	Target class acc.	Mean Conf.
0.01	1.7%	63.5%
0.1	80.2%	94.9%
0.2	99.4%	100%
0.3	99.5%	100%
0.5	99.6%	100%

Figure 4.23: Mean target class accuracy and mean confidence over all classes as a function of the perturbation L_∞ budget.

We can see in Table 4.23 that this technique works extremely well with this particular model. With only minimal perturbation made to the images, we can reliably build extremely strong adversarial examples from a universal targeted perturbation. Figure 4.24 gives an example of one such adversarial sample. For perturbations of approximately less than 0.1, the prediction is still correct. For larger perturbations, while it is clear that the image is being perturbed, the classification is extremely stable and confident. We can also see that the visual aspect of the perturbations is completely different to the ones learnt on MNIST. In this case, we have extremely fine features and lines, spanning the entirety of the image. From Figure 4.25, we can see that in fact all of the targeted perturbations we generate consist of these fine features, the difference being the exact positioning of the lines and corners over the image. A possible explanation for this phenomenon is that these sets of parallel lines and interesting shapes mimic visual features found in real life data, which often consists straight lines (i.e. boundaries) and sharp corners. This explanation also fits our hypothesis claiming that the reason this technique works is because the perturbation learns to mimic sets of unrecognizable features for each target class.

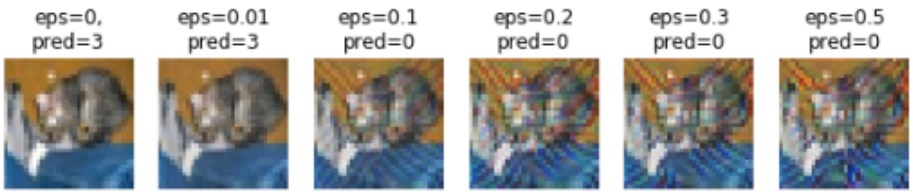


Figure 4.24: Adversarial Examples built from a natural sample of *cat* class, victim of a targeted attack towards the *airplane* class, using perturbations of varying norm.

The results in this section show that this method is able to craft adversarial perturbations reliably

for CIFAR10 (as we can see in Figure 4.25). Experimenting on this dataset provides a stronger guarantee, in terms of its behaviour when applied to other datasets and models, than was provided by the MNIST results. This is because the MNIST dataset is extremely easy to solve and is essentially the best case scenario of computer vision. The result of this experiment shows that if a network is left undefended, a white box attacker using this attack could generate perturbations that amount to a complete compromise of the model, as they are able to push any input to any targeted class.

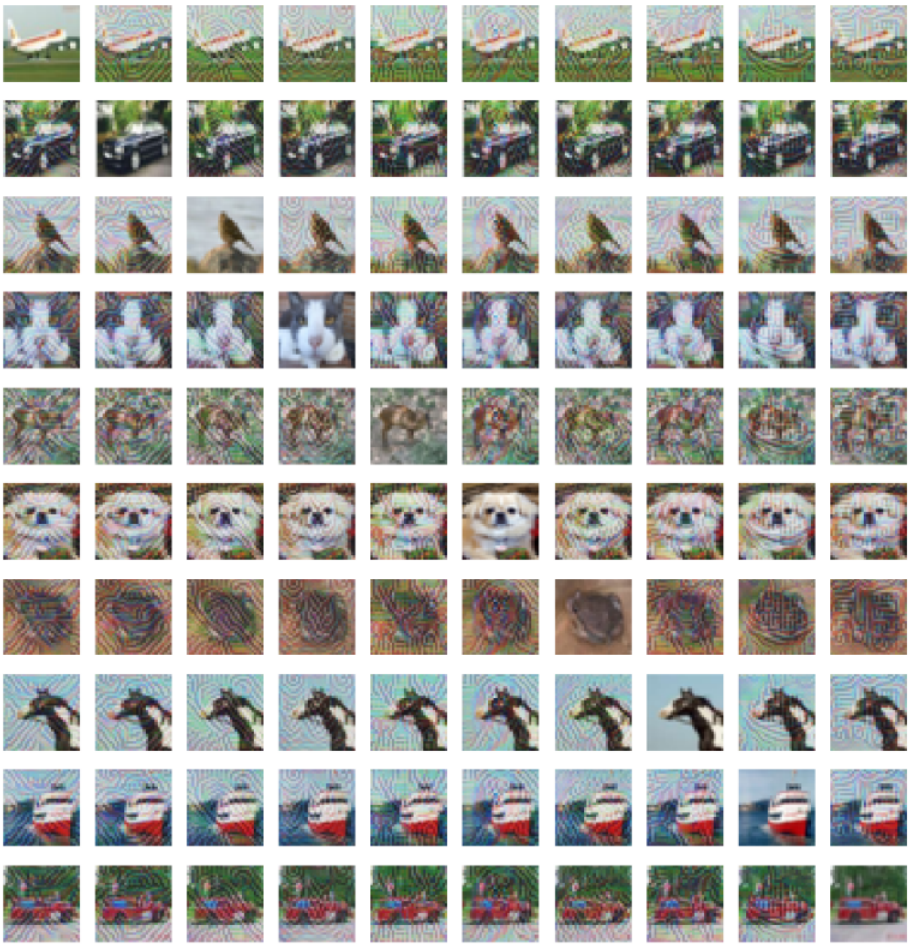


Figure 4.25: Adversarial Examples built from 10 different natural CIFAR10 samples (rows), targeted to each of the CIFAR10 classes (columns). The matrix diagonal contains the natural samples.

4.4 Attacking the Face Recognition Model

4.4.1 Baseline Attacks

In this section, we attack the face recognition model mentioned previously. The model itself uses an extremely deep Inception-Resnet v1 [40] architecture, similar in style to that which would be used in production for industrial applications. This type of model combines approaches from residual

networks, which allow the training of extremely deep models due to improved gradient flow, and from inception, which uses a clever network in network architecture to attempt to reduce to the number of model parameters and the associated computational cost while conserving representational power. The experiment will consist on attacking the model using a number of different attacks. Similarly to previous experiments, we compare the attacks according the various accuracies as a function of the perturbation L_∞-norm. Our test set consists of 400 images from 20 classes. For the untargeted attacks, we simply measure the test accuracy on perturbed examples, expecting this to drop to zero for large perturbation budgets. For targeted attacks, we measure the target class accuracy in addition to the test accuracy. The following tables report a subset of the results, the rest can be found in the appendix.

Epsilon	FGS	BI	TOS	TI
0	98%	98%	98%	98%
0.01	54%	**33%**	94%	90%
0.1	1%	**0%**	8%	**0%**
0.2	2%	**0%**	9%	**0%**
0.3	3%	**0%**	7%	**0%**
0.5	3%	**0%**	7%	**0%**

Epsilon	TOS	TI
0.01	1%	**7%**
0.1	76%	**100%**
0.2	62%	**100%**
0.3	47%	**100%**
0.5	23%	**100%**

Figure 4.26: Test accuracy (left) and targeted class accuracy (right) of the attacked model for the four baseline attack methods as a function of the perturbation L_∞-norm

Immediately, we can see from the results in Table 4.26 that all the attacks are extremely successful. Even when compared to the results on the CIFAR10 dataset, it would seem that this face recognition model is even more susceptible to even the smallest of adversarial perturbations. Similarly to previous experiments, we yet again see that iterative methods outperform single step methods at the cost of more computation. In fact iterative methods are so effective that the test accuracy of the model drops to approximately zero for perturbation budgets larger than $\epsilon = 0$.

In terms of targeted attacks, we yet again notice that the model is very easily fooled into predicting the targeted class. Interestingly, for the single step targeted attack, the targeted accuracy increases extremely quickly between $\epsilon = 0.01$ and $\epsilon = 0.1$, with a tendency to drop as we increase the perturbation norm farther. The reason this happens probably has to do with the fact that single step methods overshoot the local minimum of the objective function, meaning that smaller stepped attacks can outperform larger stepped attacks, as we see here. When switching to the iterative targeted attack, we yet again notice its substantially increased effectiveness over the single step version. It is extremely potent both in terms of general test accuracy but also in terms of target class accuracy, achieving respectively 0% and 100% for $\epsilon = 0.1$. While the test and target class accuracies follow similar trends to that seen for the CIFAR10 model, ultimately, the results for this model are more extreme than

previously seen.

In the appendix, we can see examples of a sample perturbed using all four attack methods. While the perturbations for the single steps attacks are clearly detectable as early as $\epsilon = 0.2$, iterative methods are much more stealthy, required $\epsilon = 0.3$ to be detectable. These images also give a visual representation as to why iterative methods are stronger than single step. We can see that for large perturbation sizes, the single step methods generate large amounts of noise, making the image unrecognizable. In contrast, the iterative methods both maintain a certain degree of recognizability until the maximum possible $\epsilon = 1$. Finally, if we study the perturbed examples for $\epsilon = 0.1$, examples which cause the model to misclassify 100% of its inputs are hardly differentiable from natural examples.

4.4.2 Adversarial Generalizability

Our final set of experiments involve applying the general perturbation method to our face recognition model. The face images have dimensions $224 \times 224 \times 3$ equivalent to $50\times$ more features, and hence that amount more degrees of freedom when compared to the CIFAR10 dataset. We investigate the target class accuracy achieved by our generalized perturbation as a function of the L_∞ budget it is allowed to occupy. Due to the extreme computational cost of performing this type of experiment on a model of this size/complexity, we restrict the scope, targeting only a small subset of the classes.

In order to boost our chances of success, we maximize the size of the perturbation train/validation set to the train/validation set of the bottleneck layer we trained previously. We hence have 1591 total images to use for training, of which 2% ($\approx$ 32 images) are held-out as our validation set. Similarly to the CIFAR10 experiment, we only use C&W margin loss, and we train each perturbation for 10 epochs. Importantly, we set the C&W constant to $c = 30$ in order to compensate for the large increase in image dimensionality, such that the L_∞ penalization doesn't drown out the classification part of the loss. Finally, instead of sweeping the space of all possible targets, we randomly sample three target classes, generating perturbations for each budget in $\{0.1, 0.3, 0.5\}$. Table 4.27 contains the results for this experiment. As we can see, both the target class accuracy and mean confidence of the targeted class are high, even for minimal perturbations such as $\epsilon = 0.1$.

Figure 4.28 contains adversarial examples generated by applying a generalized perturbation learned on our training subset of the FaceScrub dataset. We can immediately see that while the perturbations are clearly visible, the network is convincingly fooled, classifying the adversarial examples as the targeted class extremely confidently. To a human viewer, the perturbed images still clearly belong to the same class as the natural sample. Interestingly, both these samples and the CIFAR10 samples generate

Epsilon	Target class acc.	Mean Conf.
0.05	7.7%	36.7%
0.1	84.3%	62.7%
0.3	99.7%	89.7%
0.5	99.3%	89.7%

Figure 4.27: Mean target class accuracy and mean confidence over all classes as a function of the perturbation L_∞ budget for perturbations learned on our train subsample of the FaceScrub dataset.

perturbations that contain very fine features, such as lines and curves. This contrasts immensely with that generated on MNIST, which were mostly noise. The similarity in resolution between MNIST and CIFAR10 proves that this phenomenon isn't caused by increased dimensionality, but is likely to be necessary to capture the much more complex features of the target classes of the CIFAR10 and FaceScrub datasets.

Figure 4.28: Natural test samples (left most) from class 0 and 5 (*Allison Janney* and *Christian Slater*) attacked using a generalized perturbation targeted towards class 3 (*Linda Evans*) of increasing L_∞ budget.

If we compare this attack method to the baseline attack methods, we can see that the target class accuracy of the generalized perturbation is competitive even with the TI method. While TI has a measurable advantage for low perturbation sizes, our method manages to rival the target class accuracy when the budget surpasses $\epsilon = 0.1$. We also note a substantial advantage in terms of computational complexity when compared to any of the other methods. Once the perturbation for a given class is generated, we can apply it to any natural sample in O(1) time, requiring only to add the perturbation to the natural sample and pass it through `tanh`. In contrast, even though single step methods also have O(1) complexity, they still require substantially more computation. This comes from the fact that they need to compute both a forward pass, to compute the loss, and a backward pass, to compute

the gradient of the loss with respect to the input.

In this final experiment, we test the universal perturbations on an image that belongs to an identity that is disjoint from the training sets of the embedding extractor, the bottleneck and the perturbation itself. Figure 4.29 contains an example of a targeted attack on an image of myself, targeted towards class *Kristin Davis*. As we can see, the natural sample (technically also a very low confidence targeted adversarial example) is classified as a random class. When we overlay the universal perturbation, generated with $\epsilon = 0.3$, and apply the necessary transformation, we build an extremely potent adversarial example, that ends up being classified more confidently as the target class than real samples from the target class itself. This shows that by heavily encouraging the perturbation to generalize during the training process, we are able to make it generalize to samples that are completely disjoint from the training data, and even to samples that belong to identities/classes that have never been seen before. This essentially shows that with a universal targeted perturbation, we can perform targeted attacks using images of arbitrary identities.

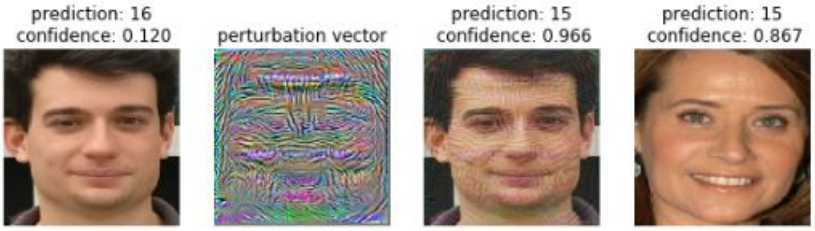


Figure 4.29: From left to right: cropped and aligned picture of myself, universal targeted adversarial perturbation targeted towards class *Kristin Davis* (15), adversarial example of myself, example natural sample from the targeted class.

Chapter 5

Evaluation, Future Work, and Conclusion

5.1 Evaluation

This project has produced a global overview of the current state of affairs in adversarial machine learning. We have analyzed and produced robust implementations of various attack methods and compared their relative behaviour on specific datasets and models. We have analyzed various ways of defending against these attacks, and showed that even if we can improve the local robustness of our models, we have yet to develop a method of guaranteeing global robustness against the strongest of attacks at our disposition.

The focus of this project has shifted since the interim stage. The original objective mostly consisted of performing a feasibility analysis of using Generative Adversarial Networks (GANs) [46] in order to generate adversarial perturbations. As we know, GANs are currently extremely popular, and are used for a wide range of tasks linked with data generation. They are however notoriously difficult to train as they are highly unstable, and are very commonly affected by 'mode collapse' which is a phenomenon where a GAN's output collapses to the mean of the desired output distribution. This is ultimately the problem that was encountered, and no suitable solution was found. In the last few months, there have been a number of publications related to using GANs both in attack and defense, hence showing that this was a feasible technique. It is possible that a lack of experience and knowledge in GAN training prevented me from making the technique work as expected. Regardless, we shifted away from using GANs, instead focusing on more direct ways of achieving adversarial perturbations.

We have introduced a new method of generating perturbations that generalize to any natural sample in a dataset. By modifying the formulation for the Carlini & Wagner L_∞ attack, we optimize a perturbation to simultaneously fool a set of images. We employ standard methods used in regular model training to improve the generalizability of the generated perturbation. We have verified the validity of the method by applying it to two similar datasets: MNIST and CIFAR10. We have also used this method to attack a face recognition system, consisting of a complex Inception-Resnet-v1 model, in order to prove that this method's performance isn't an artifact of the simplicity and low-dimensionality of the MNIST and CIFAR10 datasets. The results we achieve are extremely encouraging. At the very least, the generalized perturbations are competitive with the best iterative methods in fooling power. While they have a tendency of being visible even for moderate perturbation budgets, they can be used to convincingly fool a model into classifying any natural sample as any target class.

5.2 Future Work

As any field related to deep learning, the field of adversarial machine learning is advancing extremely fast, with new state of the art attacks and defenses being invented every couple of months. There is also a large corpus of work on the subject, with large conferences such as NIPS also starting to hold competitions that compare attacks and defenses on standardized benchmarks (unavailable to the public to prevent cheating). This factor alone shows that this is an extremely dynamic field, and there is plenty of work yet to be done.

We have previously introduced what seems to be a novel method for generating universal adversarial perturbations. That we know of, this is the first technique able to specifically generate targeted universal adversarial perturbations. As universal adversarial perturbations weren't the original subject of interest of this project, more research would be required in order to build a robust vision of the current state of the field. This will also enable us to accept or reject the claim that this method is novel, as it is possible that other work has already solved this problem. Of the publications that we are aware of, there are a number of methods that can be used to generate universal *untargeted* adversarial perturbations while also providing in depth analysis of their behaviour.

We define the future work in the continuation of this project in the following sections.

5.2.1 Benchmarking

In this report, we have proven, at the very least, the feasibility of our targeted universal adversarial perturbation method. We have compared it to existing generic attacks and found it to be extremely competitive in terms of fooling capacity. We are also aware of a number of other publications that have produced methods that generate robust universal adversarial perturbations. Conveniently we have framed the optimization problem for our method in such a way that it is easy to compare to existing techniques through perturbation norm analysis. [29] and [30] both evalute the fooling probability of their methods on an ImageNet classifier, using a predefined perturbation budget. Benchmarking our method against theirs on this dataset/model pair would give us valuable insight into the possible benefits of this method versus others. Submitting this attack to a conference competition such as the one hosted by NIPS would also allow this method to be compared to the current state of the art in the field.

5.2.2 Transferability of Universal Perturbations

The existing publications on universal perturbations also deeply analyze the transferability of these generalized perturbations by analyzing the test accuracy of models with differing architectures subject to attacks by transfer. Unfortunately, due to time constraints, we do not perform this test, however analysis of this aspect of the attack is important to increase the angles of comparison with existing work.

If given sufficient time, we would study the transferability accross different model types, such as convolutional to linear or convolutional to MLP (similar to the transfer test produced in this report). We would also attempt to study the transferability accross convolutional network architectures, selecting pairs of models between {VGG [47], ResNet [14], Inception [40]}. Production systems use convolutional models for computer vision tasks. Hence when adversarial transferability is used to perform a black box attack, it is likely that a convolutional to convolutional transfer test will be more realistic than convolutional to MLP for example. This test would most likely be performed on the ImageNet dataset, as this is the standard dataset for pre-trained models regardless of architecture.

5.2.3 Absolute generalizability

Face recognition models are usually structured as feature extractors. Given an input image, they generate embeddings/feature vectors of a certain dimensionality that can be classified using a bottleneck

layer or another classifier such as an SVM. These embeddings also enable us to recognize if a face is part of a certain set of 'known' people, or is otherwise 'unknown'. The experiments in this report have so far focused on perturbing images of 'known' people. That is, we only attack samples that have a corresponding logit in the bottleneck layer. We have proved that we can generalize the perturbations to different images of the people that were used to generate the perturbation. We have also proved in a single image experiement that we can also build targeted adversarial examples from images of identities that we have never seen before (i.e. 'unknowns'). It is however necessary to perform a more in depth analysis of the behaviour of the perturbation when applied to identities disjoint from its 'training' set. That is, can we reliably perturb images of people that are neither in the training set of the model's bottleneck layer or in the perturbation training set?

5.3 Conclusion

In this report, we have performed a robust investigation of methods used to generate adversarial examples. We have presented the threat model for these attacks as well as datasets, models and protocols used to compare and contrast the benefits of each technique. We have introduced mitigation techniques able to improve the local robustness of the models to adversarial examples. We have however proved both theoretically and empirically that these mitigation's do not provide robust guarantees, nor do they provide solid lower-bounds on the adversarial risk that a model is exposed to.

We have introduced a novel method of generating universal targeted adversarial perturbations. We have proven the feasibility of this method on three different datasets and models, of increasing complexity. Through an extensive paper review, we have attempted, without success, to accept or reject the novelty of this method. In addition to the experiments already carried out, we have formulated, as future work, a large number of additional experiments to be used to formally evaluate the robustness of the method when compared to that which currently exists.

Bibliography

[1] cs231n, "2-layer fully connected network," 2017. [Online; accessed January 28, 2018].

[2] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, pp. 2278–2324, Nov 1998.

[3] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards Deep Learning Models Resistant to Adversarial Attacks," *ArXiv e-prints*, June 2017.

[4] V. M. Reid, K. Dunn, R. J. Young, J. Amu, T. Donovan, and N. Reissland, "The human fetus preferentially engages with face-like visual stimuli," *Current Biology*, vol. 27, no. 12, pp. 1825 – 1828.e3, 2017.

[5] R. Mottaghi, X. Chen, X. Liu, N. Cho, S. Lee, S. Fidler, R. Urtasun, and A. Yuille, "The role of context for object detection and semantic segmentation in the wild," pp. 891–898, 1 2014.

[6] M. A. Turk and A. P. Pentland, "Face recognition using eigenfaces," 1991.

[7] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," vol. 1, pp. 886–893 vol. 1, June 2005.

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, pp. 84–90, May 2017.

[9] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," 2013.

[10] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," *CoRR*, vol. abs/1502.01852, 2015.

[11] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009.

[12] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," *CoRR*, vol. abs/1512.00567, 2015.

[13] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Neurocomputing: Foundations of research," ch. Learning Representations by Back-propagating Errors, pp. 696–699, Cambridge, MA, USA: MIT Press, 1988.

[14] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015.

[15] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015.

[16] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, pp. 1929–1958, Jan. 2014.

[17] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014.

[18] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: A system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pp. 265–283, 2016.

[19] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *CoRR*, vol. abs/1312.6199, 2013.

[20] I. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *International Conference on Learning Representations*, 2015.

[21] N. Papernot, P. D. McDaniel, I. J. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against deep learning systems using adversarial examples," *CoRR*, vol. abs/1602.02697, 2016.

[22] A. Greenberg, *Hackers Say They've Broken Face ID a Week After iPhone X Release*, Dec. 11, 2017 (accessed May 25, 2018). https://www.wired.com/story/hackers-say-broke-face-id-security/.

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

[24] A. Kurakin, I. J. Goodfellow, and S. Bengio, "Adversarial machine learning at scale," *CoRR*, vol. abs/1611.01236, 2016.

[25] N. Carlini and D. A. Wagner, "Towards evaluating the robustness of neural networks," *CoRR*, vol. abs/1608.04644, 2016.

[26] J. C. Spall, "Multivariate stochastic approximation using a simultaneous perturbation gradient approximation," *IEEE Transactions on Automatic Control*, vol. 37, pp. 332–341, Mar 1992.

[27] J. Uesato, B. O'Donoghue, A. van den Oord, and P. Kohli, "Adversarial risk and the dangers of evaluating against weak attacks," *CoRR*, vol. abs/1802.05666, 2018.

[28] A. Athalye, L. Engstrom, A. Ilyas, and K. Kwok, "Synthesizing robust adversarial examples," *CoRR*, vol. abs/1707.07397, 2017.

[29] S. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard, "Universal adversarial perturbations," *CoRR*, vol. abs/1610.08401, 2016.

[30] K. R. Mopuri, U. Garg, and R. V. Babu, "Fast feature fool: A data independent approach to universal adversarial perturbations," *CoRR*, vol. abs/1707.05572, 2017.

[31] S. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "Deepfool: a simple and accurate method to fool deep neural networks," *CoRR*, vol. abs/1511.04599, 2015.

[32] C. Xie, J. Wang, Z. Zhang, Z. Ren, and A. L. Yuille, "Mitigating adversarial effects through randomization," *CoRR*, vol. abs/1711.01991, 2017.

[33] A. Kurakin, I. J. Goodfellow, and S. Bengio, "Adversarial machine learning at scale," *CoRR*, vol. abs/1611.01236, 2016.

[34] G. E. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *CoRR*, vol. abs/1503.02531, 2015.

[35] N. Papernot, P. D. McDaniel, A. Sinha, and M. P. Wellman, "Towards the science of security and privacy in machine learning," *CoRR*, vol. abs/1611.03814, 2016.

[36] U. . BBC, "Attacking machine learning with adversarial examples," Feb. 24, 2017 (accessed May 17, 2018). https://blog.openai.com/adversarial-example-research/.

[37] F. Chollet *et al.*, "Keras." `https://keras.io`, 2015.

[38] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[39] A. Krizhevsky, "Learning multiple layers of features from tiny images," 2009.

[40] C. Szegedy, S. Ioffe, and V. Vanhoucke, "Inception-v4, inception-resnet and the impact of residual connections on learning," *CoRR*, vol. abs/1602.07261, 2016.

[41] S. W. H.-W. Ng, "A data-driven approach to cleaning large face datasets," *IEEE*, 2014.

[42] *MegaFace*. http://megaface.cs.washington.edu.

[43] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," *CoRR*, vol. abs/1503.03832, 2015.

[44] K. Zhang, Z. Zhang, Z. Li, and Y. Qiao, "Joint face detection and alignment using multitask cascaded convolutional networks," *IEEE Signal Processing Letters*, vol. 23, pp. 1499–1503, Oct 2016.

[45] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep inside convolutional networks: Visualising image classification models and saliency maps," *CoRR*, vol. abs/1312.6034, 2013.

[46] I. J. Goodfellow, "NIPS 2016 tutorial: Generative adversarial networks," *CoRR*, vol. abs/1701.00160, 2017.

[47] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014.

Appendices

.1 MNIST Baseline Results

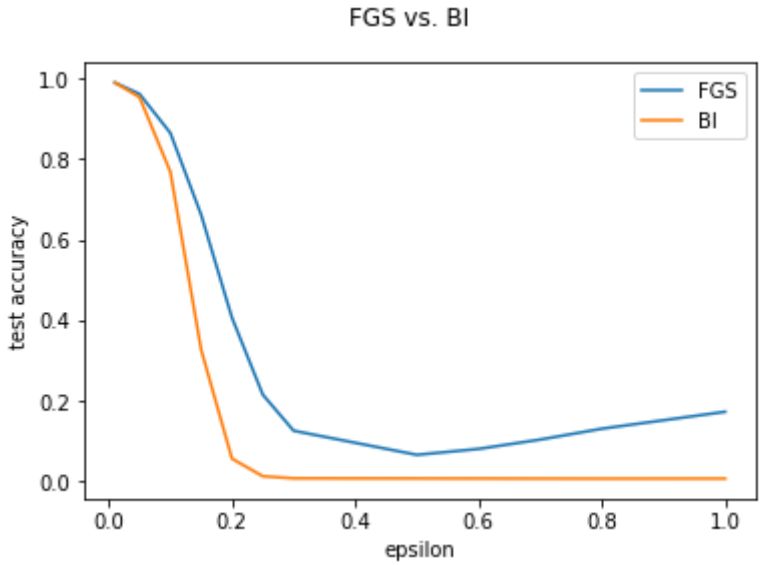


Figure 1: MNIST test accuracy of model when attacked using FGS and BI methods

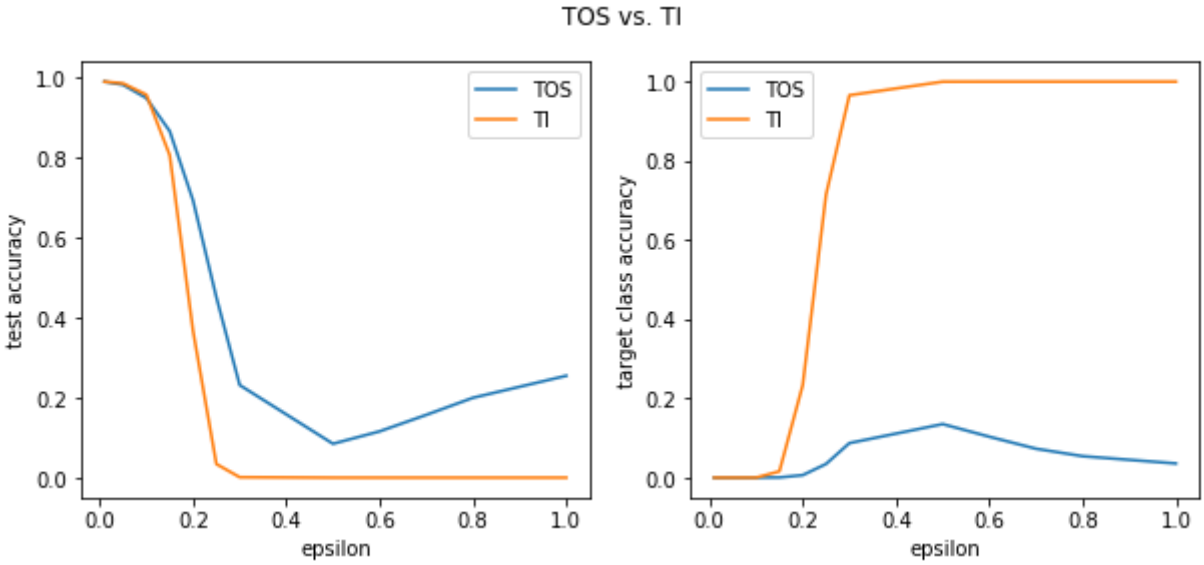


Figure 2: MNIST test accuracy of model when attacked using TOS and TI methods

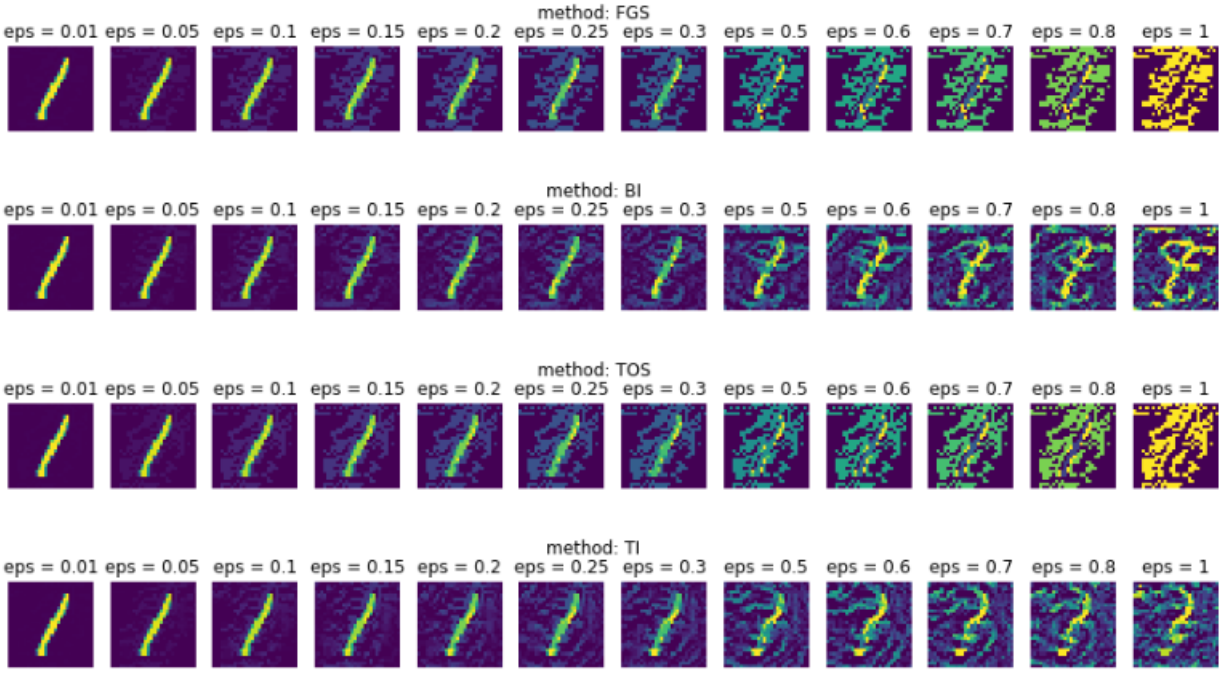


Figure 3: Adversarial samples for FGS, BI, TOS and TI as a function of perturbation norm

[Test Accuracy]

Method / eps	0.01	0.05	0.10	0.15	0.20	0.25	0.30	0.50	0.60	0.70	0.80	1.00
FGS	0.99	0.96	0.86	0.66	0.41	0.22	0.13	0.07	0.08	0.10	0.13	0.17
BI	0.99	0.95	0.77	0.33	0.06	0.01	0.01	0.01	0.01	0.01	0.01	0.01
TOS	0.99	0.98	0.95	0.87	0.69	0.45	0.23	0.08	0.12	0.16	0.20	0.25
TI	0.99	0.99	0.96	0.81	0.37	0.03	0.00	0.00	0.00	0.00	0.00	0.00

[targeted - Target Class Accuracy]

Method / eps	0.01	0.05	0.10	0.15	0.20	0.25	0.30	0.50	0.60	0.70	0.80	1.00
TOS	0.00	0.00	0.00	0.00	0.01	0.03	0.09	0.14	0.10	0.07	0.05	0.04
TI	0.00	0.00	0.00	0.02	0.23	0.71	0.97	1.00	1.00	1.00	1.00	1.00

[Perturbation - L1]

Method / eps	0.01	0.05	0.10	0.15	0.20	0.25	0.30	0.50	0.60	0.70	0.80	1.00
FGS	4.06	19.66	38.72	58.29	76.65	95.08	113.96	183.81	217.45	253.54	287.47	353.58
BI	3.48	16.49	31.52	45.58	58.84	70.48	82.12	121.68	136.80	153.54	172.61	202.50
TOS	4.00	19.54	40.69	59.51	77.26	97.65	115.28	192.63	226.99	258.07	294.96	369.02
TI	3.69	17.00	33.13	47.96	64.76	75.71	87.39	123.28	139.29	151.90	166.50	192.92

[Perturbation - L2]

Method / eps	0.01	0.05	0.10	0.15	0.20	0.25	0.30	0.50	0.60	0.70	0.80	1.00
FGS	0.20	0.99	1.95	2.94	3.89	4.84	5.80	9.48	11.28	13.12	14.92	18.37
BI	0.18	0.84	1.60	2.31	2.99	3.59	4.18	6.34	7.25	8.24	9.32	11.14
TOS	0.20	0.98	2.01	2.97	3.91	4.91	5.84	9.72	11.54	13.26	15.12	18.79
TI	0.18	0.86	1.67	2.41	3.20	3.76	4.33	6.21	7.06	7.73	8.48	9.79

Figure 4: Raw results used to build Figures 1 & 2

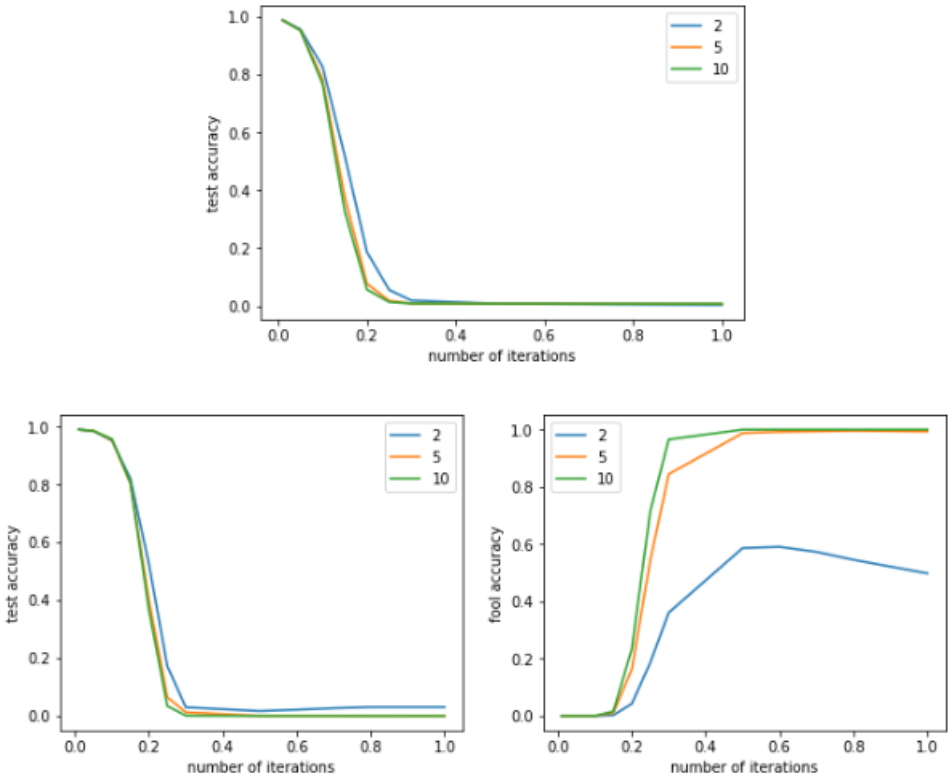


Figure 5: Top: Test accuracy as a function of the perturbation budget and the number of iterations for the BI method. Bottom: Test (left) and target class accuracies (right) as a function of the perturbation budget and the number of iterations for the TI method.

```
[Test Accuracy - 2 iterations]
Method / eps 0.01   0.05   0.10   0.15   0.20   0.25   0.30   0.50   0.60   0.70   0.80   1.00
BI           0.99   0.96   0.83   0.52   0.19   0.06   0.02   0.01   0.01   0.01   0.01   0.00
TI           0.99   0.99   0.95   0.82   0.53   0.17   0.03   0.02   0.02   0.03   0.03   0.03

[Test Accuracy - 5 iterations]
Method / eps 0.01   0.05   0.10   0.15   0.20   0.25   0.30   0.50   0.60   0.70   0.80   1.00
BI           0.99   0.95   0.78   0.38   0.08   0.02   0.01   0.01   0.01   0.01   0.01   0.01
TI           0.99   0.99   0.95   0.80   0.40   0.06   0.01   0.00   0.00   0.00   0.00   0.00

[Test Accuracy - 10 iterations]
Method / eps 0.01   0.05   0.10   0.15   0.20   0.25   0.30   0.50   0.60   0.70   0.80   1.00
BI           0.99   0.95   0.77   0.33   0.06   0.01   0.01   0.01   0.01   0.01   0.01   0.01
TI           0.99   0.99   0.96   0.81   0.37   0.03   0.00   0.00   0.00   0.00   0.00   0.00

[TI - Target Class Accuracy]
Iters / eps  0.01   0.05   0.10   0.15   0.20   0.25   0.30   0.50   0.60   0.70   0.80   1.00
2            0.00   0.00   0.00   0.00   0.04   0.19   0.36   0.59   0.59   0.57   0.55   0.50
5            0.00   0.00   0.00   0.01   0.16   0.54   0.84   0.99   0.99   0.99   1.00   0.99
10           0.00   0.00   0.00   0.02   0.23   0.71   0.97   1.00   1.00   1.00   1.00   1.00
```

Figure 6: Raw data for Figure 5

.2 CIFAR10 Baseline Results

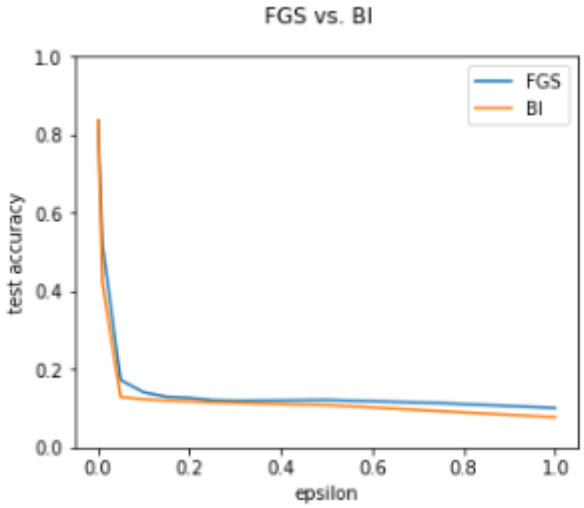


Figure 7: Cifar10 test accuracy of model when attacked using FGS and BI methods

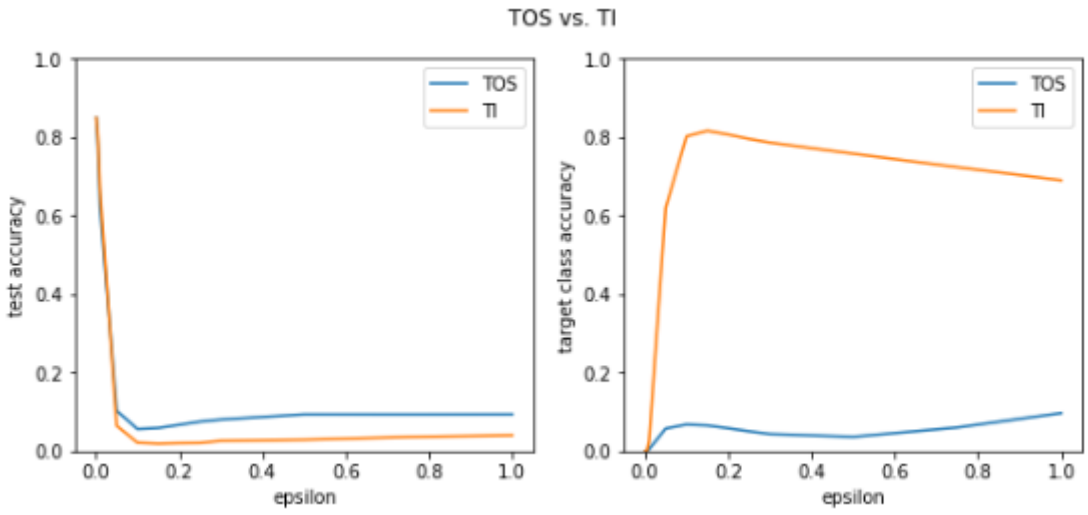


Figure 8: Cifar10 test accuracy of model when attacked using TOS and TI methods

[Test Accuracy]

Method / eps	0.00	0.01	0.01	0.05	0.10	0.15	0.20	0.25	0.30	0.50	0.75	1.00
FGS	0.84	0.69	0.52	0.17	0.14	0.13	0.13	0.12	0.12	0.12	0.11	0.10
BI	0.84	0.65	0.42	0.13	0.12	0.12	0.12	0.12	0.11	0.11	0.09	0.08
TOS	0.85	0.78	0.62	0.10	0.06	0.06	0.07	0.08	0.08	0.09	0.09	0.09
TI	0.85	0.80	0.67	0.06	0.02	0.02	0.02	0.02	0.03	0.03	0.04	0.04

[targeted - Target Class Accuracy]

Method / eps	0.00	0.01	0.01	0.05	0.10	0.15	0.20	0.25	0.30	0.50	0.75	1.00
TOS	0.00	0.00	0.00	0.06	0.07	0.07	0.06	0.05	0.04	0.04	0.06	0.10
TI	0.00	0.00	0.03	0.62	0.80	0.82	0.81	0.80	0.79	0.76	0.72	0.69

[Perturbation - L1]

Method / eps	0.00	0.01	0.01	0.05	0.10	0.15	0.20	0.25	0.30	0.50	0.75	1.00
FGS	3.06	15.31	30.60	152.36	302.82	450.98	596.79	740.05	880.41	1408.57	1979.39	2427.03
BI	2.83	12.60	22.75	80.35	127.62	153.71	175.04	193.69	212.18	296.32	402.28	524.03
TOS	3.06	15.30	30.59	152.35	302.81	450.95	596.73	739.97	880.28	1407.73	1977.79	2424.85
TI	2.86	12.81	23.10	74.06	126.72	179.51	233.18	284.21	337.48	549.72	789.62	1018.03

[Perturbation - L2]

Method / eps	0.00	0.01	0.01	0.05	0.10	0.15	0.20	0.25	0.30	0.50	0.75	1.00
FGS	0.06	0.28	0.55	2.76	5.49	8.19	10.87	13.51	16.11	26.06	37.23	46.52
BI	0.05	0.24	0.45	1.71	2.80	3.42	3.90	4.37	4.80	6.64	9.08	11.86
TOS	0.06	0.28	0.55	2.76	5.49	8.19	10.87	13.51	16.11	26.05	37.21	46.49
TI	0.05	0.25	0.46	1.67	2.97	4.28	5.59	6.86	8.14	13.27	19.15	24.67

Figure 9: Raw results used to build Figures 7 & 8

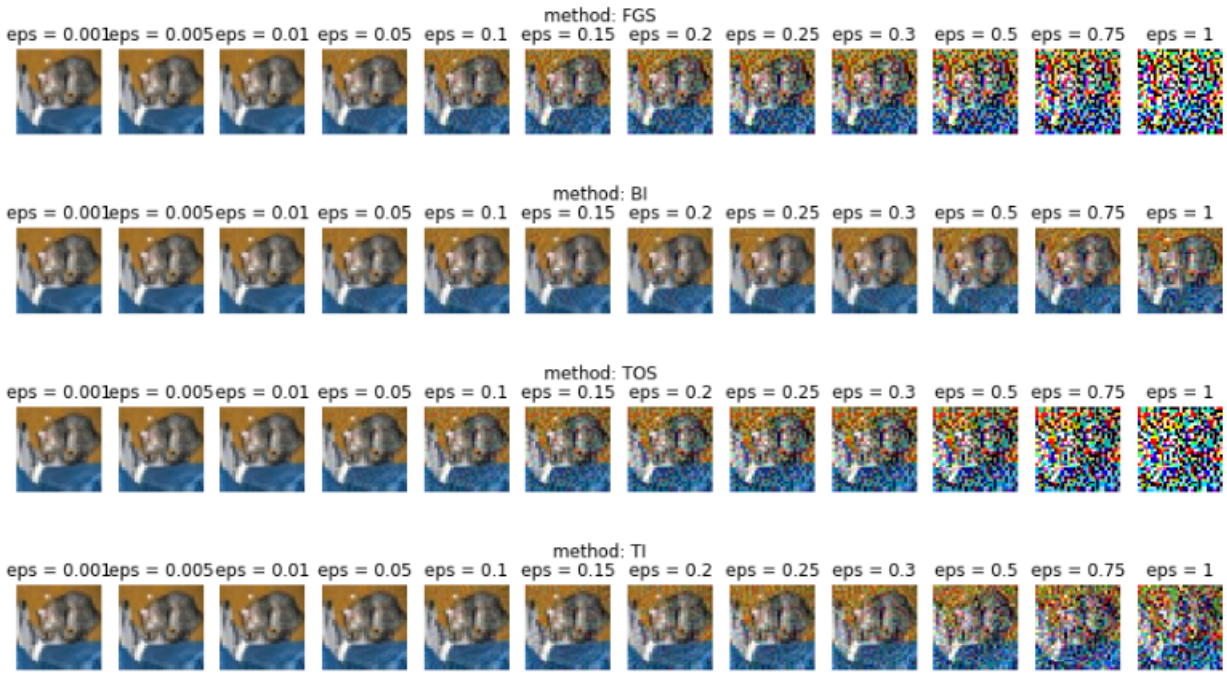


Figure 10: Adversarial samples for FGS, BI, TOS and TI as a function of perturbation norm

.3 FaceScrub Raw results

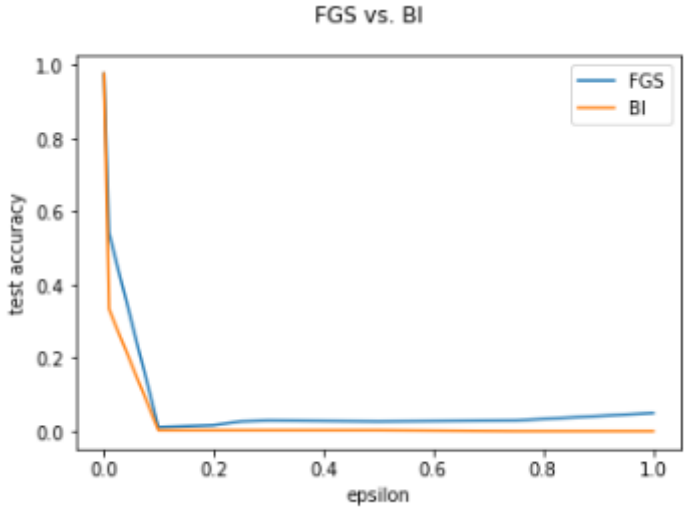


Figure 11: Facescrub test accuracy of model when attacked using FGS and BI methods

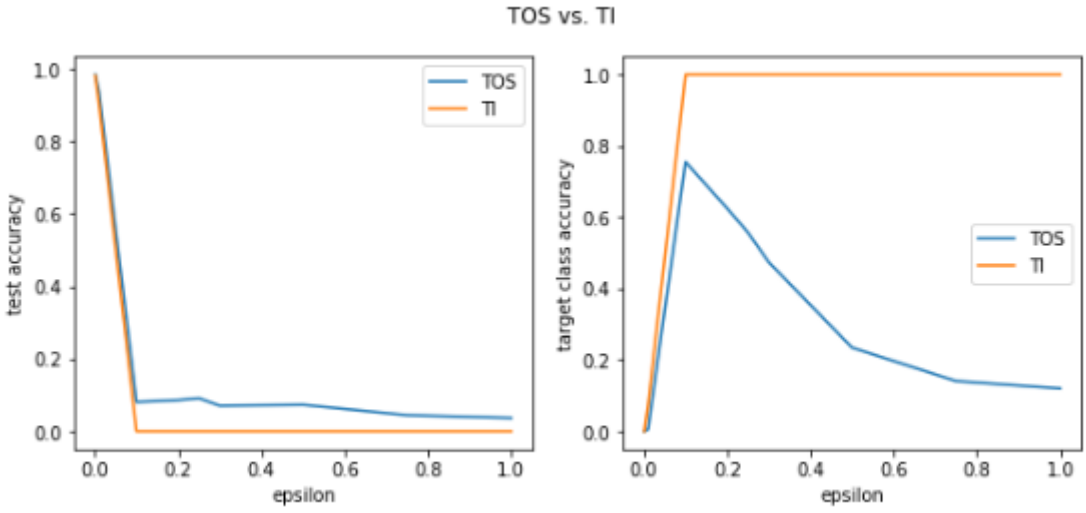


Figure 12: Facescrub test accuracy of model when attacked using TOS and TI methods

Figure 13: Adversarial samples for FGS, BI, TOS and TI as a function of perturbation norm

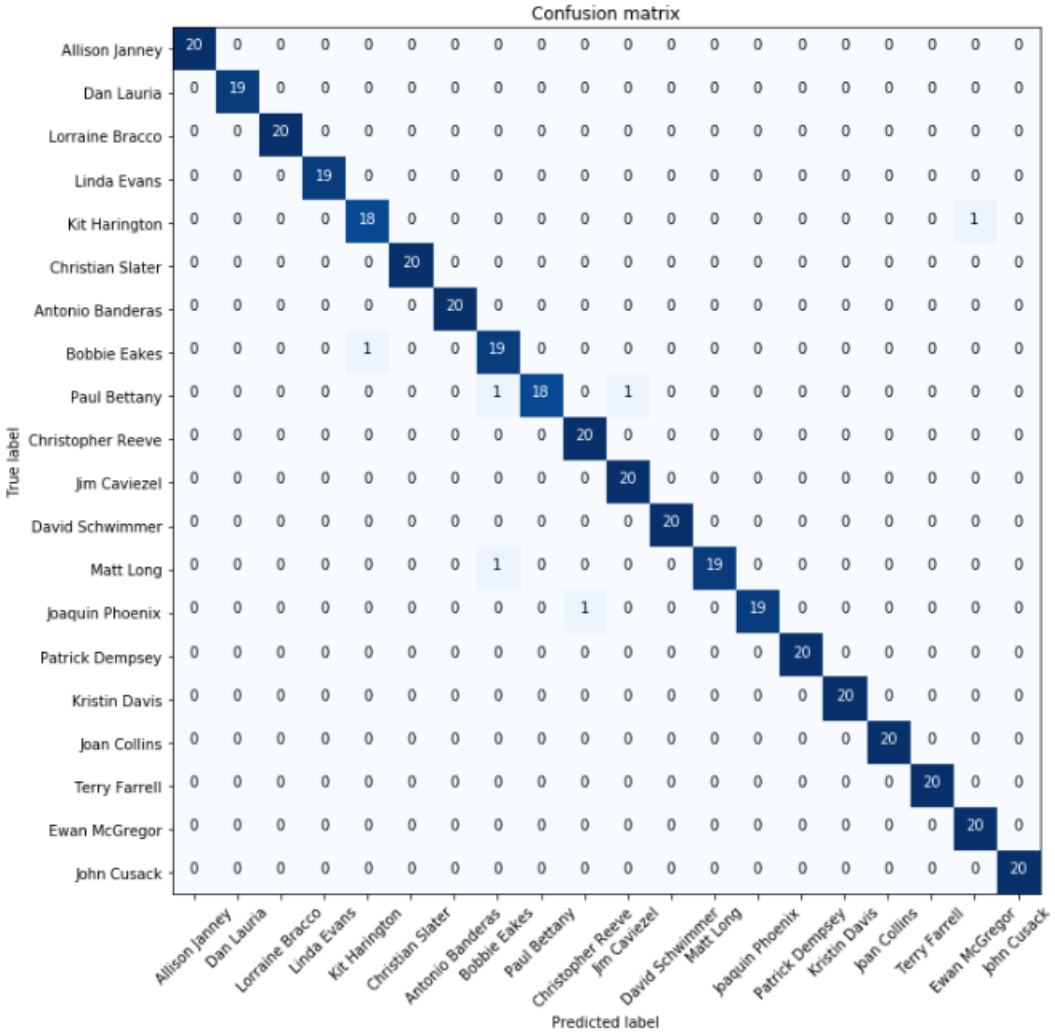
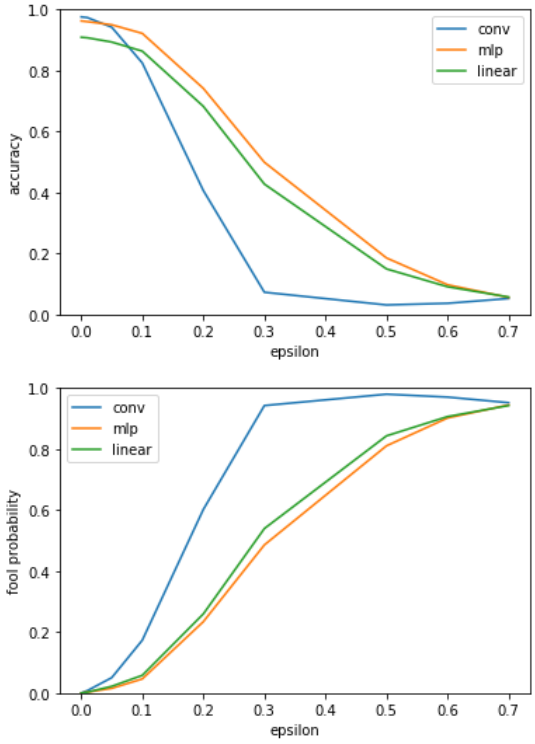


Figure 14: Confusion matrix for our subset of the FaceScrub dataset when classified by the face recognition model.

.4 MNIST Adversarial Transfer Results



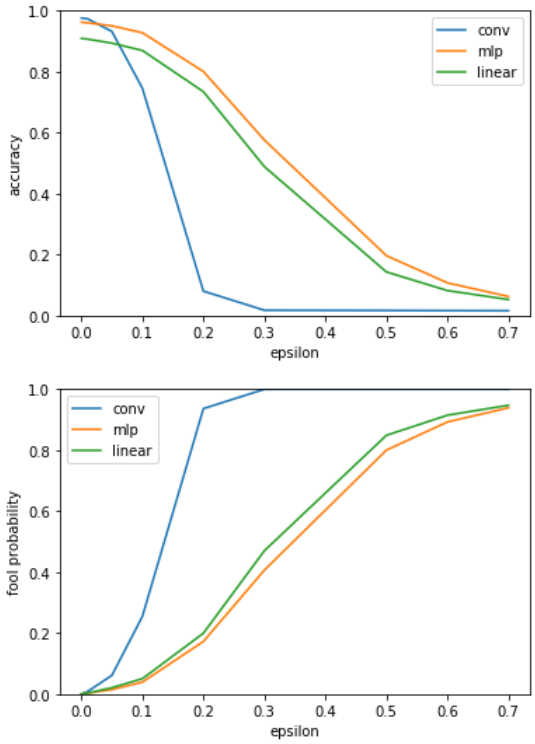
[Accuracies]

epsilon	conv	mlp	linear
0.000	0.976	0.963	0.909
0.010	0.974	0.960	0.907
0.050	0.942	0.950	0.893
0.100	0.825	0.922	0.864
0.200	0.406	0.741	0.682
0.300	0.073	0.499	0.427
0.500	0.031	0.185	0.149
0.600	0.036	0.097	0.091
0.700	0.052	0.056	0.057

[Fool Probs]

epsilon	conv	mlp	linear
0.000	0.000	0.000	0.000
0.010	0.008	0.003	0.003
0.050	0.050	0.015	0.022
0.100	0.173	0.046	0.058
0.200	0.602	0.234	0.259
0.300	0.943	0.486	0.539
0.500	0.980	0.811	0.843
0.600	0.971	0.902	0.907
0.700	0.952	0.945	0.943

Figure 15: Plot and raw results for the MNIST adversarial transfer tests using FGS attack



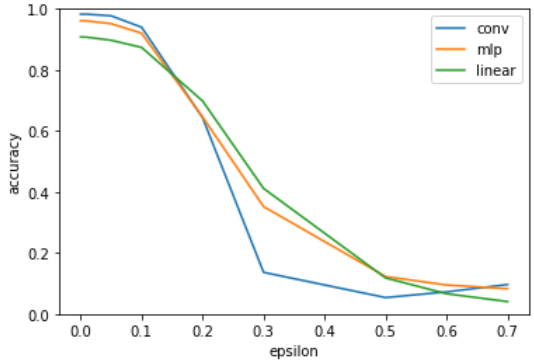
[Accuracies]

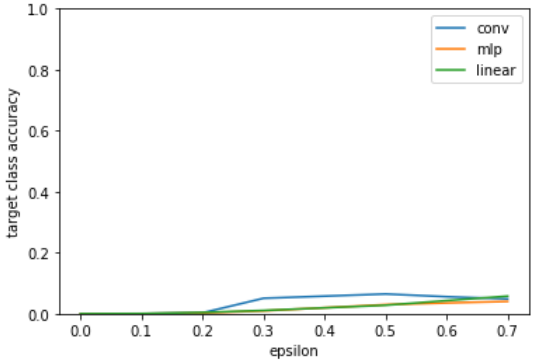
epsilon	conv	mlp	linear
0.000	0.976	0.963	0.909
0.010	0.974	0.960	0.907
0.050	0.932	0.950	0.894
0.100	0.746	0.928	0.870
0.200	0.080	0.801	0.734
0.300	0.017	0.576	0.488
0.500	0.017	0.196	0.143
0.600	0.016	0.107	0.082
0.700	0.016	0.062	0.052

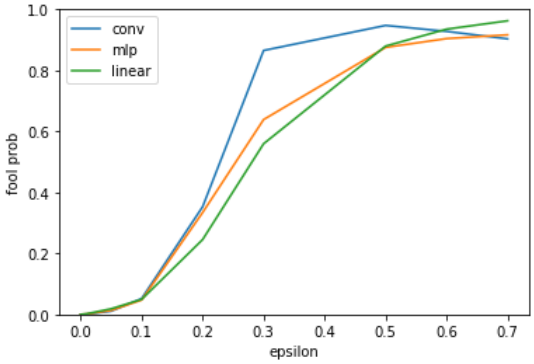
[Fool Probs]

epsilon	conv	mlp	linear
0.000	0.000	0.000	0.000
0.010	0.008	0.003	0.003
0.050	0.061	0.015	0.021
0.100	0.255	0.039	0.050
0.200	0.936	0.172	0.200
0.300	1.000	0.407	0.470
0.500	1.000	0.800	0.849
0.600	1.000	0.893	0.915
0.700	1.000	0.939	0.947

Figure 16: Plot and raw results for the MNIST adversarial transfer tests using BI attack







[Accuracies]

epsilon	conv	mlp	linear
0.000	0.983	0.961	0.908
0.010	0.983	0.961	0.907
0.050	0.977	0.952	0.897
0.100	0.940	0.920	0.874
0.200	0.643	0.646	0.698
0.300	0.136	0.352	0.411
0.500	0.054	0.123	0.118
0.600	0.072	0.095	0.066
0.700	0.097	0.083	0.041

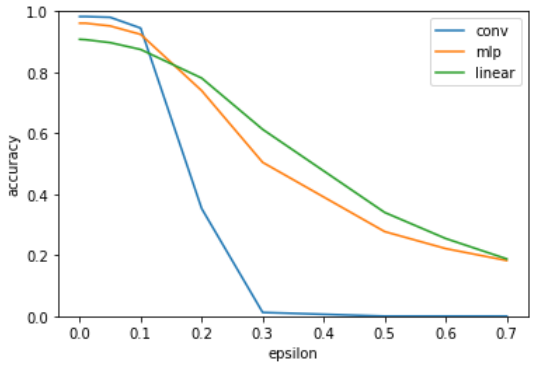
[Target class Accuracy]

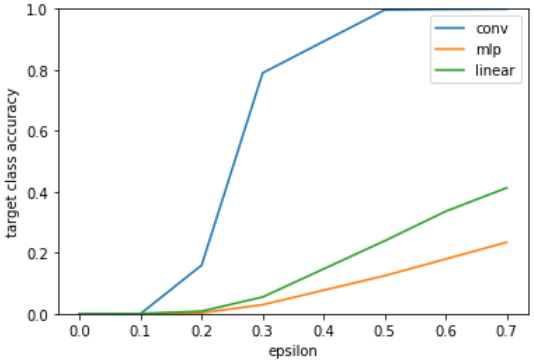
epsilon	conv	mlp	linear
0.000	0.000	0.000	0.000
0.010	0.000	0.000	0.000
0.050	0.000	0.000	0.000
0.100	0.000	0.000	0.000
0.200	0.003	0.002	0.004
0.300	0.051	0.009	0.011
0.500	0.065	0.030	0.028
0.600	0.056	0.035	0.043
0.700	0.048	0.040	0.058

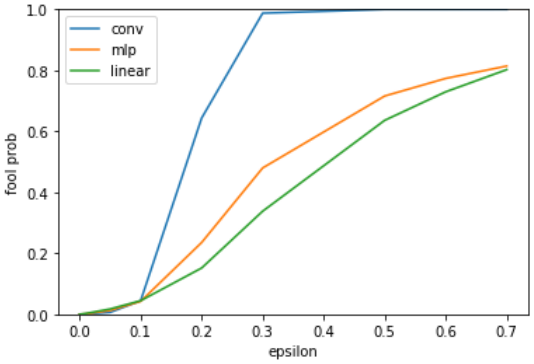
[Fool Prob]

epsilon	conv	mlp	linear
0.000	0.000	0.000	0.000
0.010	0.001	0.001	0.002
0.050	0.011	0.013	0.018
0.100	0.052	0.047	0.049
0.200	0.353	0.334	0.245
0.300	0.865	0.639	0.560
0.500	0.947	0.875	0.880
0.600	0.928	0.904	0.935
0.700	0.903	0.917	0.963

Figure 17: Plot and raw results for the MNIST adversarial transfer tests using TOS attack







[Accuracies]

epsilon	conv	mlp	linear
0.000	0.983	0.961	0.908
0.010	0.983	0.961	0.907
0.050	0.980	0.952	0.897
0.100	0.945	0.924	0.875
0.200	0.353	0.740	0.781
0.300	0.012	0.504	0.613
0.500	0.000	0.277	0.340
0.600	0.000	0.222	0.255
0.700	0.000	0.182	0.188

[Target class Accuracy]

epsilon	conv	mlp	linear
0.000	0.000	0.000	0.000
0.010	0.000	0.000	0.000
0.050	0.000	0.000	0.000
0.100	0.000	0.000	0.000
0.200	0.159	0.003	0.008
0.300	0.790	0.029	0.055
0.500	0.997	0.125	0.239
0.600	0.998	0.179	0.336
0.700	0.999	0.234	0.413

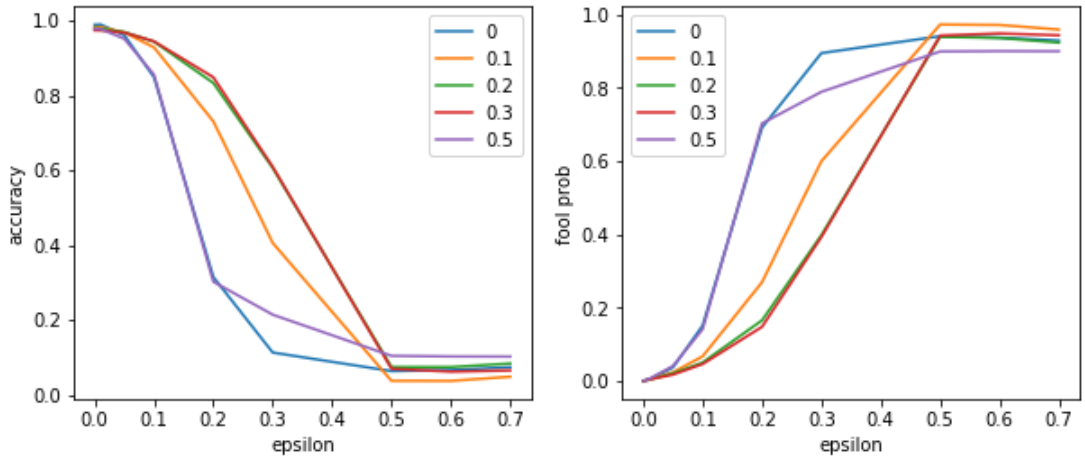
[Fool Prob]

epsilon	conv	mlp	linear
0.000	0.000	0.000	0.000
0.010	0.001	0.001	0.003
0.050	0.006	0.012	0.017
0.100	0.044	0.042	0.044
0.200	0.644	0.235	0.152
0.300	0.988	0.480	0.338
0.500	1.000	0.716	0.636
0.600	1.000	0.774	0.730
0.700	1.000	0.814	0.802

Figure 18: Plot and raw results for the MNIST adversarial transfer tests using TI attack

.5 Adversarial Training raw results

.5.1 BI training - FGS testing



[Accuracies]

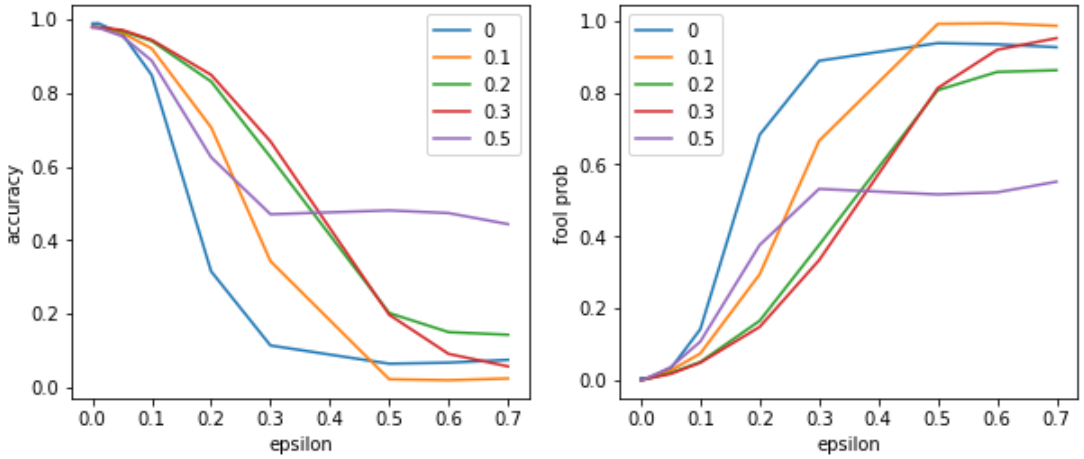
te / tr	0	0.1	0.2	0.3	0.5
0.000	0.989	0.983	0.979	0.974	0.978
0.010	0.990	0.983	0.978	0.974	0.978
0.050	0.961	0.968	0.969	0.968	0.951
0.100	0.849	0.928	0.944	0.944	0.855
0.200	0.314	0.731	0.833	0.848	0.302
0.300	0.113	0.406	0.606	0.609	0.214
0.500	0.063	0.037	0.074	0.069	0.104
0.600	0.066	0.037	0.074	0.062	0.102
0.700	0.073	0.048	0.083	0.065	0.102

[Fool Probs]

te / tr	0	0.1	0.2	0.3	0.5
0.000	0.000	0.000	0.000	0.000	0.000
0.010	0.003	0.004	0.004	0.004	0.005
0.050	0.037	0.023	0.020	0.017	0.041
0.100	0.151	0.067	0.050	0.046	0.141
0.200	0.691	0.269	0.165	0.147	0.702
0.300	0.894	0.599	0.398	0.393	0.789
0.500	0.940	0.972	0.939	0.942	0.899
0.600	0.937	0.971	0.936	0.948	0.900
0.700	0.928	0.959	0.923	0.943	0.899

Figure 19: Plot and raw results for the MNIST adversarial training tests using BI attack for training perturbation and FGS for testing

.5.2 FGS training - FGS testing



[Accuracies]

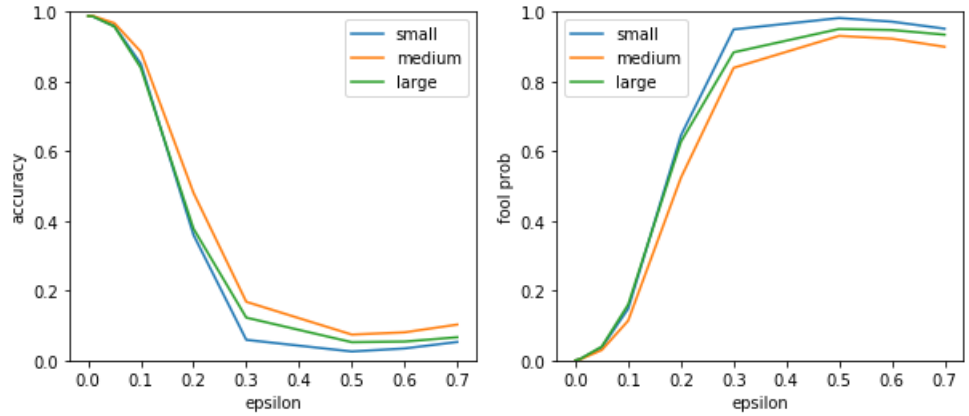
te / tr	0	0.1	0.2	0.3	0.5
0.000	0.989	0.981	0.980	0.979	0.979
0.010	0.990	0.979	0.979	0.978	0.977
0.050	0.961	0.964	0.969	0.972	0.955
0.100	0.849	0.921	0.941	0.945	0.888
0.200	0.314	0.706	0.831	0.849	0.625
0.300	0.113	0.342	0.625	0.668	0.470
0.500	0.063	0.021	0.201	0.196	0.480
0.600	0.066	0.018	0.149	0.090	0.473
0.700	0.073	0.023	0.142	0.056	0.443

[Fool Probs]

te / tr	0	0.1	0.2	0.3	0.5
0.000	0.006	0.000	0.000	0.000	0.000
0.010	0.006	0.005	0.003	0.004	0.005
0.050	0.032	0.027	0.020	0.018	0.036
0.100	0.142	0.074	0.051	0.049	0.107
0.200	0.683	0.295	0.165	0.148	0.376
0.300	0.889	0.666	0.376	0.334	0.532
0.500	0.938	0.991	0.807	0.813	0.517
0.600	0.934	0.993	0.858	0.919	0.523
0.700	0.926	0.986	0.862	0.951	0.552

Figure 20: Plot and raw results for the MNIST adversarial training tests using FGS attack for training perturbation and FGS for testing

.6 Influence of model capacity results



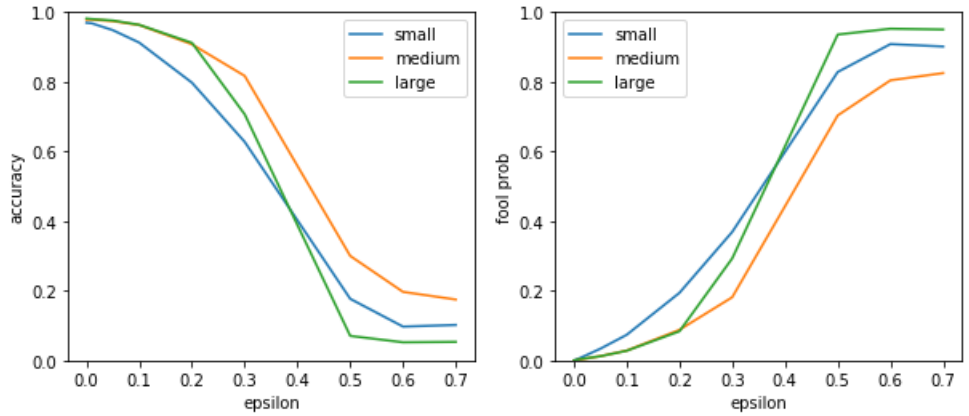
[Undefended - Test Accuracy]

Method / eps	0.00	0.01	0.05	0.10	0.20	0.30	0.50	0.60	0.70
small	0.99	0.99	0.96	0.85	0.40	0.09	0.03	0.05	0.07
medium	0.99	0.99	0.96	0.85	0.44	0.11	0.06	0.07	0.10
large	0.99	0.99	0.96	0.86	0.39	0.12	0.07	0.08	0.09

[Undefended - Fool Probability]

Method / eps	0.00	0.01	0.05	0.10	0.20	0.30	0.50	0.60	0.70
small	0.00	0.00	0.04	0.15	0.60	0.92	0.98	0.96	0.94
medium	0.00	0.00	0.04	0.15	0.56	0.90	0.94	0.93	0.91
large	0.00	0.00	0.03	0.14	0.61	0.88	0.94	0.93	0.91

Figure 21: Plot and raw results for the MNIST capacity tests on undefended models with FGS attack.



[Defended - Test Accuracy]

Method / eps	0.00	0.01	0.05	0.10	0.20	0.30	0.50	0.60	0.70
small	0.97	0.97	0.96	0.93	0.82	0.65	0.14	0.08	0.07
medium	0.98	0.98	0.97	0.96	0.90	0.77	0.13	0.08	0.09
large	0.99	0.99	0.98	0.97	0.93	0.73	0.06	0.03	0.03

[Defended - Fool Probability]

Method / eps	0.00	0.01	0.05	0.10	0.20	0.30	0.50	0.60	0.70
small	0.00	0.00	0.02	0.05	0.17	0.35	0.87	0.93	0.93
medium	0.00	0.00	0.01	0.03	0.10	0.23	0.88	0.93	0.92
large	0.00	0.00	0.01	0.02	0.07	0.27	0.95	0.97	0.97

Figure 22: Plot and raw results for the MNIST capacity tests on adversarially trained models with FGS attack.

.7 Input Randomization Results

Figure 23: Adversarial samples for each input randomization method.

FGS

method/eps	0.01	0.05	0.1	0.15	0.2	0.25	0.3	0.5	0.6	0.7	0.8	1
clean	0.99	0.96	0.86	0.66	0.41	0.22	0.13	0.07	0.08	0.10	0.13	0.17
AGN	0.98	0.97	0.95	0.89	0.77	0.59	0.40	0.04	0.02	0.02	0.01	0.01
RTC	0.96	0.94	0.87	0.77	0.64	0.49	0.35	0.13	0.12	0.13	0.16	0.21
AGN+RTC	0.94	0.93	0.92	0.89	0.82	0.74	0.63	0.22	0.13	0.09	0.07	0.05

BI

method/eps	0.01	0.05	0.1	0.15	0.2	0.25	0.3	0.5	0.6	0.7	0.8	1
clean	0.99	0.95	0.77	0.33	0.06	0.01	0.01	0.01	0.01	0.01	0.01	0.01
AGN	0.98	0.98	0.95	0.90	0.79	0.62	0.40	0.01	0.01	0.01	0.01	0.01
RTC	0.96	0.94	0.87	0.75	0.54	0.31	0.14	0.03	0.04	0.03	0.03	0.02
AGN+RTC	0.95	0.94	0.92	0.89	0.83	0.76	0.67	0.14	0.06	0.04	0.04	0.04

Figure 24: Raw results for 'normal' experimental protocol.

epsilon	clean	AGN	RTC	AGN+RTC
0.01	0.99	0.98	0.96	0.95
0.05	0.96	0.96	0.94	0.93
0.10	0.86	0.90	0.88	0.88
0.15	0.66	0.75	0.79	0.80
0.20	0.41	0.54	0.65	0.69
0.25	0.22	0.33	0.48	0.53
0.30	0.13	0.19	0.32	0.37
0.50	0.07	0.09	0.11	0.12
0.60	0.08	0.10	0.11	0.11
0.70	0.10	0.12	0.12	0.13
0.80	0.13	0.14	0.14	0.15
1.00	0.17	0.17	0.18	0.18

epsilon	clean	AGN	RTC	AGN+RTC
0.01	0.99	0.98	0.95	0.95
0.05	0.95	0.96	0.93	0.92
0.10	0.77	0.88	0.87	0.87
0.15	0.33	0.67	0.74	0.79
0.20	0.06	0.36	0.53	0.64
0.25	0.01	0.12	0.28	0.44
0.30	0.01	0.03	0.13	0.24
0.50	0.01	0.01	0.01	0.01
0.60	0.01	0.01	0.01	0.01
0.70	0.01	0.01	0.01	0.01
0.80	0.01	0.01	0.01	0.01
1.00	0.01	0.01	0.01	0.01

Figure 25: Raw results for 'best case transfer' protocol.