Die approbierte Originalversion dieser Diplom-/Masterarbeit ist in der Hauptbibliothek der Technischen Universität Wien aufgestellt und zugänglich.
http://www.ub.tuwien.ac.at

The approved original version of this diploma or master thesis is available at the main library of the Vienna University of Technology.
http://www.ub.tuwien.ac.at/eng

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Angriffe gegen Neuronale Netzwerke

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Logic and Computation

eingereicht von

Martin Matak
Matrikelnummer 01635889

an der Fakultät für Informatik
der Technischen Universität Wien
Betreuung: Associate Prof. Dipl.-Ing. Georg Weissenbacher, D.Phil.

Wien, 30. April 2019

Martin Matak	Georg Weissenbacher

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Attacks against Neural Networks

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Logic and Computation

by

Martin Matak
Registration Number 01635889

to the Faculty of Informatics

at the TU Wien

Advisor: Associate Prof. Dipl.-Ing. Georg Weissenbacher, D.Phil.

Vienna, 30th April, 2019

Martin Matak	Georg Weissenbacher

Erklärung zur Verfassung der Arbeit

Martin Matak
Siebeneichengasse 16/6-7, 1150 Wien, Austria

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 30. April 2019

Martin Matak

Acknowledgements

Writing the master thesis is harder than I thought and more rewarding than I could have ever imagined. None of this would have been possible without my supervisor, Georg Weissenbacher, whose guidance and support helped me with writing this thesis. He also taught me to be aware of the implicit assumptions that I often made. I will leverage this way of thinking through the rest of my life.

I would like to thank Agata Ciabattoni who pointed me to the scholarship offered by the Master's Program in Logic and Computation when I was in the first semester and encouraged me to apply for it. Without that scholarship, this journey at TU Wien would take much longer.

I am grateful to my family who believed in me and supported my decision to move to Vienna for my master studies.

Kurzfassung

Anwendungen künstlicher Intelligenz (KI) in unserem Alltag gewinnen immer mehr an Bedeutung. Umso wichtiger ist es, dass KI für Menschen sicher ist, z.B. im Bereich der automatisierten Altersschätzung. Allerdings können nur wenige veränderte Pixel dazu führen, dass neuronale Netze ein Bild falsch klassifizieren.

Der Schwerpunkt dieser Arbeit liegt daher auf der Analyse von White-Box- und Black-Box-Angriffen gegen neuronale Netze im Bereich der Altersbestimmung. Bestehende Techniken werden evaluiert und ein neuer, semi-orientierter Ansatz entwickelt. Mit Hilfe dieses Ansatzes können Samples generiert werden, die zu einer widersprüchlichen Klassifizierung durch das neuronale Netz führen. Der Ansatz kann immer dann angewendet werden, wenn die Label der Klassifizierung geordnet sind oder zum Teil gebündelt werden können.

Obwohl sich diese Arbeit auf den Bereich der Altersschätzung konzentriert, impliziert die Fähigkeit, widersprüchliche Samples zu generieren, dass neuronale Netzwerke nicht das letzte Wort in sicherheitskritischen Anwendungen haben sollten.

Abstract

As Artificial Intelligence (AI) is getting a greater impact in our everyday life, it is of utmost importance that it is safe for humans. However, this thesis demonstrates that if only several pixels in an image are changed, neural networks produce incorrect results.

The focus of this thesis is on white-box and black-box attacks against neural networks in the age estimation domain. Existing techniques are evaluated and a new semi-targeted approach is developed. Using the semi-targeted approach, adversarial samples can be generated. This new approach can be used whenever labels can be somehow ordered or even clustered.

Although this thesis focuses on the age estimation domain, the ability to craft adversarial samples implies that neural networks should not have the final word in safety-critical applications.

Contents

Kurzfassung **ix**

Abstract **xi**

Contents **xiii**

1 Introduction **1**
1.1 Problem Definition 1
1.2 Aim of the Work 3
1.3 Methodological Approach 5
1.4 Outline 5

2 Background **7**
2.1 Feedforward neural networks 9
2.2 Gradient descent 11
2.3 Backpropagation 14
2.4 Convolutional neural networks 14

3 State-of-the-Art **17**
3.1 Fast Gradient Sign Method (FGSM) 18
3.2 Jacobian-based Saliency Map Attack (JSMA) 18
3.3 Carlini & Wagner (CW) 20
3.4 Transfer based approach 22
3.5 Ensemble approach 23
3.6 Boundary attack 23

4 Experiments **25**
4.1 Methodology 25
4.2 Whitebox attacks 28
4.3 Blackbox attacks 35

5 The Semi-targeted Approach **43**
5.1 Motivation 43
5.2 Evaluation 44

6 Threats to Validity and Future Work **47**
6.1 Threats to Validity 47
6.2 Future Work 48

7 Summary **49**

Bibliography **51**

CHAPTER 1

Introduction

As Artificial Intelligence (AI) is getting a greater impact in our everyday life, it is of utmost importance that it is safe for humans.

One of the tools used in AI are deep neural networks (DNNs) and neural networks in general. Deep neural networks are powerful learning models that achieve excellent performance on visual recognition problems [KSH12]. Those results imply that DNNs can be used in different industrial domains, e.g. traffic sign recognition, a domain in which DNNs even outperform humans [SSSI12b].

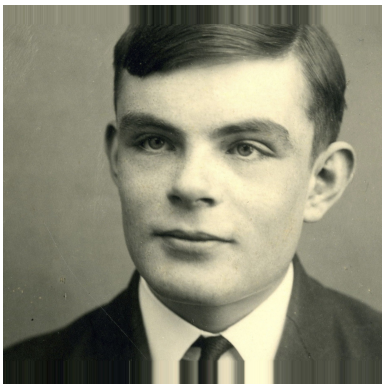
Nevertheless, neural networks tend to have some peculiar properties as well. If only a few pixels in an image are changed, some neural networks produce incorrect results [SZS+13]. Such a behaviour is not desirable in safety-critical systems. For example, if an autonomous car recognizes a *STOP* sign as anything else but a *STOP* sign, it can lead to deadly consequences. Therefore, it is important to verify the system.

Three original and three adversarial samples that are crafted in one of the experiments this thesis is presented in Figure 1.1.

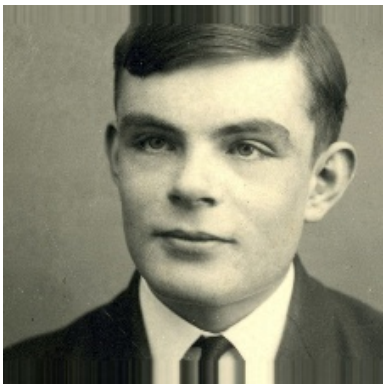
1.1 Problem Definition

The assumption in the rest of this thesis is that the neural network is performing a *classification task* - mapping an input (image) to a discrete output value (e.g. mapping an image of a traffic sign to the name of the traffic sign). Output values are often called labels or categories. There is a finite number of possible labels.

To find errors in a system where a neural network is used, we need a framework which can trigger every possible output of the neural network. We want to achieve that by using only one image as a starting point.



(a) Original sample classified as 28 years old



(b) Adversarial sample classified as 59 years old

(c) Original sample classified as 59 years old

(d) Adversarial sample classified as 28 years old

(e) Original sample classified as 28 years old

(f) Adversarial sample classified as 64 years old

Figure 1.1: Original samples (left) and adversarial samples (right)

Since for the same input we will always get the same output, modifications of that image are necessary. For each different desired output, a different modification of the image is performed. Then we repeat that process by changing our desired output in every iteration and in that way, cover all possible outputs. The idea is that the modified image is as close as possible to the original one. For example, an image of a *STOP* sign can be modified as long as it still is an image of a *STOP* sign to a human observer. In that case, with the variations of the *STOP* sign we cover all the possible outputs of the neural network - all other traffic signs. Using this technique, we can trigger errors in the system which can help us in the process of its verification.

Modifying an input for a neural network with a goal of reaching an output that is different than the output of an unmodified input is an attack called *misclassification*. If an output which an attacker wants to reach is one specific label, then a name of the attack is *targeted misclassification*.

Creating a framework for targeted misclassification in the domain of age estimation as well as comparison and evaluation of existing approaches is the main focus of this thesis. In other words, the focus is on approaches how to trick a classifier to classify a person as older or younger than he or she actually is.

1.2 Aim of the Work

The goals of this thesis, in order of chapters, are the following:

- **write a short survey on adversarial algorithms**.
  - Three different white-box attacks and three different black-box attacks are analyzed and discussed. All of them are used to attack the DNN used for age estimation.
- **create a classifier for age estimation**;
  - Implement a DNN which receives an image as an input and outputs how old the person in the image is.
- **attack the classifier in a white-box environment**;
  - In the *white-box* environment, an adversary has all the information about the DNN under attack. The internal structure of the neural network, all the implementation details and values of all the variables in any moment are known to the attacker. In other words, the attacker has access to the source code and nothing is hidden.
- **attack the classifier in a black-box environment**;

– In the *black-box* environment, an adversary doesn't have access to all the information. Depending on the precise definition of "black-box", more or less information is provided. In this thesis, the only capability of the black-box adversary is to observe the labels assigned by the DNN to chosen inputs.

The contributions of this thesis, sorted by priority, can be enumerated as follows:

1. **the new Semi-targeted Black-box Attack is introduced**

   - In the domain of age estimation, it makes sense to be less strict about the targeted label. For instance, if an image of a minor is classified as an image of a person over a fifty years old, this could lead to the same consequences (e.g. access to an age-restricted content), no matter if the minor is classified as a 55 or 65 years old person.

     Formally, if the goal is to hit any label outside a specific group of labels, I define that attack as *semi-targeted misclassification*. A group of labels must be greater than an empty set, otherwise the task is trivial. In general, this attack makes sense in any environment where labels can be ordered or at least clustered.

     To this end, I adapted one already existing black-box approach to this more relaxed setting. Such a relaxation is not yet introduced in the literature since it is very domain specific. Consequently, the results can't be compared against previous work of that kind, but results are compared to targeted black-box attacks.

2. **adversarial algorithms are evaluated in different environments**

   - A natural question is which algorithm is the best one for the white-box attack and which one for the black-box attack? Several algorithms are run in different settings and the results are compared. That provides an answer to the question which algorithm to use in which scenario.

3. **a framework for the white-box and the black-box attacks is developed**

   - I developed the framework for a white-box and a black-box targeted misclassification attack. In other words, while treating the DNN as a white-box or a black-box, images can be constructed in a way that the targeted DNN outputs a specific year. I also extended the framework with capability to craft semi-targeted black-box misclassification attacks.

To achieve the goals of this thesis and consequently provide the contributions, I had to face and overcome **the challenges**:

- **Can the already existing attacks be adapted on the age estimation task?**

  - None of the attacks from related work is performed in the age estimation domain.

- **Does image size have an impact on the attack?**
  - Most of the attacks in related work are performed against the images with the small dimensions.

- **How to measure attacks?**
  - Is the accuracy of the classifier the only measure?
  - If an image is changed too much, the attack becomes obvious.
  - If an API used in the black-box environment is queried too many times, the attack can easily get detected.

1.3 Methodological Approach

The methodological approach consists of the following steps:

1. I write about the background knowledge needed for training a neural network. The goal is to give a brief introduction to this area so that a non-expert reader can follow the rest of the thesis. Using that knowledge, I train a deep neural network for age estimation of a person in the image.
2. I conduct a research about different state of the art methods for generating *adversarial examples*, i.e. images which are not correctly classified by DNN. The focus here is on the approaches that address targeted misclassification. While treating a DNN from the first step as a white-box, I use those methods to construct adversarial examples. Afterwards, I present and analyze the results of the attacks.
3. I make a literature survey on targeted black-box attack methods, explain those methods and use them to generate adversarial inputs for the DNN from the first step, but this time while treating it as a black box. I compare the results of different algorithms.
4. As the last step, combining the domain knowledge and an existing attack method, I implement a new adversarial algorithm and using that algorithm, I construct images for semi-targeted misclassification. I compare results against targeted black-box approaches.

1.4 Outline

In this chapter an introduction to the problem is provided. In the next chapter background knowledge that is needed to follow the rest of the thesis is presented. In Chapter 3 state-of-the-art attacks are presented and explained. In Chapter 4 results of the experiments

are presented. The Semi-Targeted Approach is discussed in Chapter 5. Threats to validity and ideas for future work are presented in Chapter 6. Finally, summary of this thesis is presented in Chapter 7.

CHAPTER 2

Background

A formal definition of a machine learning algorithm is given by [Mit97]: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E." From such a vague definition, it is obvious that a comprehensive introduction to machine learning is a broad topic. Hence, in this thesis a brief introduction to the *image classification* problem is provided only. That should be enough for a non-expert to be able to read and to understand this whole thesis.

As explained in Section 1.1, image classification is the *task* of assigning a *label* or a *class* to an image. If an input is an n-dimensional vector and there are k classes, then the learning algorithm is usually asked to produce a function $f : \mathbb{R}^n \to \{1, ..., k\}$. One other variant of a classification task would be to produce a function f which outputs a probability distribution over classes, i.e. how likely each class is. In such a scenario, the next step usually is to assign a label according to the most likely class, but this need not to be the case. A model that is used for classification is called a *classifier*.

After defining a task, it is important to know how to measure the performance of a model. Depending on the domain of a system, this measure can vary. The *accuracy* of the model is usually measured for a classification task. It is the fraction of correct predictions of the model. For example, if the prediction of the model was correct for 8 out of 10 samples, the accuracy of that model (on those 10 samples) is 0.8 or 80%. An equivalent information can be obtained by measuring the *error rate*. It is defined as the fraction of incorrect predictions of the model.

Usually we are interested how good the model is on previously unseen data, because that way we can see how well will it work in the real world application. Therefore, we measure its performance on a set of data that is not used during training. Such a set of data is called *test set*. A test set is subset of an entire *dataset*.

A dataset is a collection of samples. One of the oldest datasets used in Machine Learning is the Iris flower dataset [FIS]. That dataset consists of 50 samples from each of three species of Iris. For each sample, four measures are taken: the length and the width of the sepals and petals, in centimeters. A measurable property of a sample is called a *feature*. Hence, Iris dataset has 3 classes, 150 samples and each sample has 4 features. A classifier for that dataset could be modelled as a function $f : \mathbb{R}^4 \to \{0, 1, 2\}$ where $0, 1, 2$ encode the first, the second and the third type of Iris, respectively.

For every sample in a dataset, information is provided that defines what a class of that sample is. That information is called *ground-truth label*. It is usually used as a vector. Assume there are N different classes in a dataset. Then every class can be indexed from 0 to $N - 1$. Let there be a sample such that its ground-truth label has the index p. Then a ground-truth vector $\mathbf{1}_p$ represents a vector with dimensionality N and all the values are 0, except for the index p where the value is 1. In other words, a vector $\mathbf{1}_p$ encodes a ground-truth label.

Most machine learning algorithms have *hyperparameters*, settings that we can use to control the algorithms' behaviour. Hyperparameters are values which are set before the learning algorithm begins. In contrast, the values of other parameters are derived via training.

If a classifier has a high accuracy on a test dataset, we say it *generalizes* well. However, if it has a high accuracy on the training samples, but a low accuracy on a test dataset, we say it *overfits*. To avoid overfitting, training samples are split into two non overlapping subsets: the *training* subset and the *validation* subset.

The validation subset can be used to find good hyperparameters. The training subset is used to adjust parameters of a model. The validation subset is verifying that any increase in accuracy over the training dataset actually yields an increase in accuracy over a dataset that has not been shown to the model before, or at least the model hasn't been trained on it. If the accuracy over the training dataset increases, but the accuracy over the validation dataset stays the same or decreases, then a model is overfitting. The test accuracy, an accuracy we are typically interested in, is computed on the test dataset. That means that an original dataset is split into three disjoint subsets: training, validation and test subset.

A problem can occur if there is not enough data, i.e. an original dataset is too small. That makes the training, the validation and the test subset not big enough. For instance, it can happen that the training dataset doesn't have any instances of some specific class or there are too few of them. That problem can be solved by the *k-fold cross-validation* procedure.

That procedure is based on the idea of repeating the training and testing computation on different splits of the original dataset. The original set is split into k equal disjoint subsets. Of the k subsets, a single subset is retained as the validation set for testing the model, and the remaining $k - 1$ subsets are used as training data. The process is then

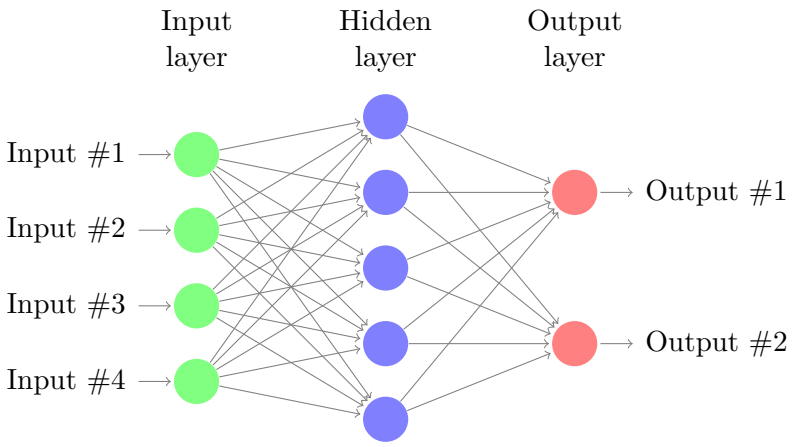


Figure 2.1: An example of the feedforward neural network architecture

repeated k times, with each of the k subsets used exactly once as the validation data. The k results can then be averaged to determine the test accuracy.

So far, the basics of the machine learning are covered. A goal for the rest of this chapter is to provide a brief introduction to the deep learning area. In Section 2.1 an introduction to basic neural networks is provided. After that, in Section 2.2 the Gradient Descent algorithm and variations of it are introduced. Gradient descent is used for optimization of functions based on the derivations of them. The backpropagation algorithm, an efficient algorithm to compute a derivation of a function represented by a neural network, is introduced in Section 2.3. Finally, in Section 2.4 convolutional neural networks are presented. A convolutional neural network represents a typical choice of architecture of the neural network when it comes to the computer vision domain.

2.1 Feedforward neural networks

Feedforward neural networks or *multilayer perceptrons* (MLPs) are the essential deep learning models. A feedforward neural network defines a mapping $f(\boldsymbol{x};\boldsymbol{\theta}) = \boldsymbol{y}$ and learns the value of the parameters $\boldsymbol{\theta}$ that result in the best function approximation.

A neural network consists of several *layers*. There is always one *input* layer, followed by one or more *hidden* layers and finally, there is an *output layer*. The number of layers (without the input layer) defines the *depth* of the model. There is no precise definition, but neural networks with more than three layers are called *deep* neural networks.

When it comes to feedforward neural networks, a network is a directed acyclic graph. Vertices of such a graph are called *units* or *neurons* and edges are called *weights*. Vertices represent scalar functions of an input. Weights are the parameters $\boldsymbol{\theta}$ which a neural network learns. Edges define data flow. One example of such a model is shown in the Figure 2.1.

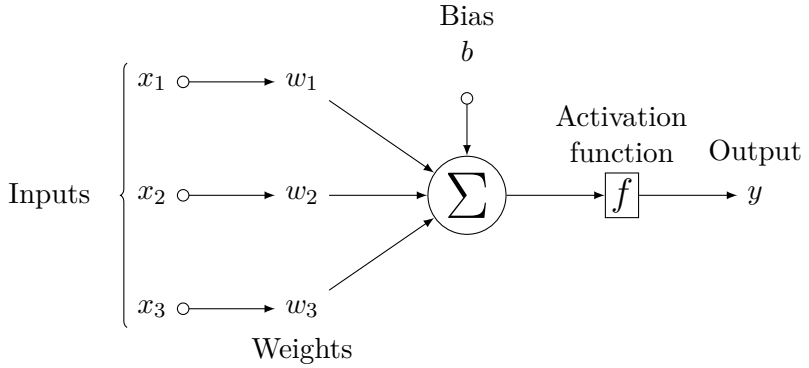


Figure 2.2: A single processing unit in a neural network

A typical architecture of one neuron can be seen in the Figure 2.2. That specific neuron performs an operation $f(net) = y$ where $net = x1 * w1 + x2 * w2 + x3 * w3 + bias$. Bias is usually modelled as an additional input with the corresponding weight $w = 1$. Then a neuron performs the operation $f(\boldsymbol{x} \times \boldsymbol{w}) = y$ where $\times$ denotes cross-product between two vectors. Function f is called an *activation function* and usually is the same for all neurons in the same layer. In the input layer the activation function is mapping from input to output, i.e. $f(x) = x$, but those in the other layers are usually non-linear functions.

Usually we are interested in *probabilities* for every class, i.e. what's the probability that a given sample belongs to a specific class. In that case, we want a vector of the probabilities as an output of a neural network. If there are N different classes, the vector will have dimensionality of N. The sum of the probabilities for all classes must be equal to 1. Then, if we want to find out a predicted label for a given sample, we take the class that corresponds to the index with the highest probability. The layer that provides such a vector as an output is called *softmax* layer and usually represents the last layer in a neural network that is used for classification. Sometimes the softmax layer is not even considered part of a neural network, but only as a function used for post-processing an output of a neural network. In this thesis, when it matters, it is clear from the context if the last layer is a softmax layer or not.

Convolutional Neural Networks (CNNs), which are described in Subsection 2.4, are a specific kind of feedforward neural networks. They are most popular in the computer vision domain. In domains where a context is important, for instance in Natural Language Processing, cycles are introduced in a computing graph so that the context can be stored in a state of a neural network. The *Recurrent Neural Networks* (RNNs) are an example of a neural network with cycles. However, RNNs are out of the scope of this thesis and will not be further discussed.

2.2 Gradient descent

The accuracy of a parametric model depends on the data provided to train it and the parameters used. The same holds for neural networks. The parameters in the neural networks are weights and during the training we are trying to find *the best* weights. To find them, we express how bad the model is using the *loss function* or *cost function*. It expresses how wrong the model is (for a given data) using the given weights. When such a function is defined, all we do is try to find an input to that function, i.e. weights, such that the output of that function, i.e. the loss, is minimal. Most machine learning algorithms have some kind of optimization built in. Optimization refers to the task of finding $\boldsymbol{x}$ s.t. $f(\boldsymbol{x})$ is maximal or minimal. In this thesis, optimization will always mean minimization, except when stated otherwise. Maximization can be accomplished by minimizing the function $-f(\boldsymbol{x})$.

Since training a neural network is actually minimizing a loss function that is represented by the neural network, two things need to be decided: the loss function that will be used and the optimization algorithm that will be used to find its minimum.

For the rest of this thesis, I assume that a reader is familiar with the basics of linear algebra, differential calculus, probability theory and statistics. If not, please consult [GBC16], for instance.

The most popular classification loss function is *cross-entropy*. Given two probability mass functions (a mass function is a function that gives the probability that a discrete random variable is exactly equal to some value) $\boldsymbol{u}$ and $\boldsymbol{v}$ in $\mathbb{R}^T$, i.e. $\boldsymbol{u} = (u_1, ..., u_T)$ and $\boldsymbol{v} = (v_1, ..., v_T)$, the cross-entropy between $\boldsymbol{u}$ and $\boldsymbol{v}$ is

$$H(\boldsymbol{u}, \boldsymbol{v}) = -\sum_{t=1}^{T} u_t \ln v_t \tag{2.1}$$

Let $\boldsymbol{u}$ encode the ground-truth label and let $\boldsymbol{v}$ be the predicted softmax class scores. Now H measures how dissimilar the true and predicted probabilities are for a single sample.

Let $\mathcal{D}$ be a dataset such that $\mathcal{D} = \{(\boldsymbol{x}_s, \boldsymbol{w}_s)\}_{s=1}^{S}$ where x_s is an input vector and w_s is a vector encoding the ground truth (i.e. all indices have value 0, except for the index of the class which x_s represents where the value is 1).

On this basis, *cross-entropy loss* on $\mathcal{D}$ is defined as

$$L(\boldsymbol{\theta}) = \frac{1}{S} \sum_{s=1}^{S} H(\boldsymbol{w}_s, \text{softmax}(f(\boldsymbol{x}_s; \boldsymbol{\theta}))) \tag{2.2}$$

Models trained with this loss function are called *softmax classifiers*. If $T = 2$, it is also called *logistic regression*. Classifiers learn to predict probabilities per class label.

After we defined the loss function $L(\boldsymbol{\theta})$ as in 2.2, we need to find an optimization algorithm to find its minimum. Since the function is not linear in $\boldsymbol{\theta}$, a nonlinear optimization algorithm is needed. A popular choice in deep learning is *Gradient Descent*.

Gradient Descent is an iterative optimization algorithm that is used for finding the minimum of a function. It is based on the first derivative of the function whose minimum needs to be found. This means that the function must be differentiable.

Let ∇f be a vector of all partial derivatives of a function $f : \mathbb{R}^N \mapsto \mathbb{R}^N$ and let f_{x_i} denote the partial derivative of f with respect to x_i, i.e. $f_{x_i} = \frac{\partial f}{\partial x_i}$. Then $\nabla f(\boldsymbol{x}) = (f_{x_1}(\boldsymbol{x}), ..., f_{x_n}(\boldsymbol{x}))$ encodes how fast f changes with all arguments $x_1, ..., x_n$, which is exactly what is needed for optimizing $L(\boldsymbol{\theta})$. Compute the direction of the greatest increase, i.e. $\nabla L(\boldsymbol{\theta})$, and move in the opposite direction. The size of that move is called *step size* and it is defined by the *learning rate* - hyperparameter α.

The pseudo code for gradient descent is presented in Algorithm 2.1.

Algorithm 2.1: Gradient Descent

```
   Input: θ, L, α
1  while true do
2  |  θ' ← ∇L(θ);
3  |  if ||θ'|| ≈ 0 then
4  |  |  return;
5  |  end
6  |  θ ← θ − α ∗ θ';
7  end
```

This algorithm is simple and efficient. Simple since the only requirement is that f is differentiable and efficient because it requires only first derivatives. A problem that can occur is that $\|\boldsymbol{\theta}'\| \approx 0$ can happen at all critical points - minimum, maximum and saddle points.

To speed up convergence, *momentum* is introduced using hyperparameter β. It increases step size dynamically by using *velocity* $\boldsymbol{v}$. Velocity builds up momentum if successive gradients are similar. The pseudo code for gradient descent with momentum is shown in Algorithm 2.2

The idea behind gradient descent with *Nesterov Momentum*, which works often better than standard Gradient Descent with momentum, is not to evaluate the gradient at the current point, but at the point which would be the next, i.e. evaluate the gradient at $\boldsymbol{\theta} + \boldsymbol{v}$ instead of at $\boldsymbol{\theta}$. The pseudo code is shown in Algorithm 2.3

So far, $L(\boldsymbol{\theta})$ is computed based on the whole training dataset $\mathcal{D}$. Therefore, time complexity increases linearly with number of samples in $\mathcal{D}$. That can be problematic if $\mathcal{D}$ is large. Instead of having a whole dataset $\mathcal{D}$ as a *batch*, we can evaluate $L(\boldsymbol{\theta})$ on a subset of the training dataset with S number of samples. If $S = |\mathcal{D}|$, the gradient descent algorithm is called *Batch Gradient descent*. But we can split the training dataset $\mathcal{D}$ into several *minibatches* (subsets) of cardinality S and process them one by one (one per iteration). One full run through the training set is called an *epoch*. Usually training

Algorithm 2.2: Gradient Descent with momentum

Input: $\boldsymbol{\theta}, L, \alpha, \beta$

$\boldsymbol{v} \leftarrow \mathbf{0}$;
while *true* **do**
  $\boldsymbol{\theta}' \leftarrow \nabla L(\boldsymbol{\theta})$;
  if $\|\boldsymbol{\theta}'\| \approx 0$ **then**
    return;
  end
  $\boldsymbol{v} \leftarrow \beta \boldsymbol{v} - \alpha * \boldsymbol{\theta}'$;
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$;
end

Algorithm 2.3: Gradient Descent with Nesterov momentum

Input: $\boldsymbol{\theta}, L, \alpha, \beta$

$\boldsymbol{v} \leftarrow \mathbf{0}$;
while *true* **do**
  $\boldsymbol{\theta}' \leftarrow \nabla L(\boldsymbol{\theta} + \boldsymbol{v})$;
  if $\|\boldsymbol{\theta}'\| \approx 0$ **then**
    return;
  end
  $\boldsymbol{v} \leftarrow \beta \boldsymbol{v} - \alpha * \boldsymbol{\theta}'$;
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$;
end

takes many epochs. The resulting algorithm is called *Minibatch Gradient Descent* (or *Stochastic Gradient Descent* (SGD) if $S = 1$). In practice, it is often called SGD even if $S > 1$. The time for a single iteration is now independent of the dataset size and depends only on the size of a minibatch. Typically, S is 64, 128, 256, or 512. In general, S is usually 2^N because of efficiency reasons (data parallelism). Decreasing S decreases the computation time per iteration, the amount of memory required on GPU (minibatch processed as whole) and the accuracy of the gradient estimate. It is important to sample minibatches randomly to break (possible) ordering in a dataset. In practice, usually the training set is shuffled once or before every epoch and then processed sequentially in minibatches.

There are many alternatives to the gradient descent algorithm which have an advantage of not having to choose the learning rate. An interested reader can find an overview of gradient descent optimization algorithms in [Rud16].

2.3 Backpropagation

Now we already know how to train a neural network. We can use a cross-entropy loss function $L(\boldsymbol{\theta})$ and minibatch gradient descent with (Nesterov) momentum. For gradient descent, we must calculate $\nabla L(\boldsymbol{\theta})$.

One way to obtain $\nabla L(\boldsymbol{\theta})$ is *numerical differentiation*. Let $p \in [1, dim(\boldsymbol{\theta})]$, $\epsilon \in \mathbb{R}$ be arbitrary small, and $\mathbf{1}_p$ encode a ground-truth label. Then directly by definition of the derivative we obtain 2.3

$$\nabla L(\boldsymbol{\theta}) = (L(\boldsymbol{\theta} + \mathbf{1}_p\epsilon) - L(\boldsymbol{\theta}))/\epsilon \tag{2.3}$$

Sometimes is 2.4 used instead of 2.3.

$$\nabla L(\boldsymbol{\theta}) = (L(\boldsymbol{\theta} + \mathbf{1}_p\epsilon) - L(\boldsymbol{\theta} - \mathbf{1}_p\epsilon))/2\epsilon \tag{2.4}$$

This approach is easy to implement, but it's only an approximation (ϵ cannot be arbitrary small) and it is too inefficient in practice because L must be evaluated $dim(\boldsymbol{\theta})$ times (modern DNNs have millions of parameters). Instead of *numerical differentiation*, preferable way is to obtain the *analytic gradient*, i.e. obtain $\nabla L(\boldsymbol{\theta})$ analytically using calculus. This approach is more accurate (no approximation) and much more efficient (single evaluation).

A neural network is actually a computational graph composed of other functions. The loss function L of the neural network is again a computational graph. Derivatives in such graphs can be computed iteratively using the *chain rule*. The chain rule is used for computing the derivative of composition of two or more functions.

Let $f : \mathbb{R} \mapsto \mathbb{R}$, $g : \mathbb{R} \mapsto \mathbb{R}$ and $F : \mathbb{R} \mapsto \mathbb{R}$ s.t. $F(x) = f(g(x))$. Then by the chain rule, $F'(x) = f'(g(x))g'(x)$.

Based on the chain rule, the gradient of every weight can be efficiently computed and then updated as defined in the gradient descent method. The paper in which this famous algorithm is introduced is [RHW86].

2.4 Convolutional neural networks

Since the focus of this thesis is on the computer vision domain, this background chapter wouldn't be complete without mentioning the most popular choice of neural networks for image classification, namely *Convolutional Neural Networks* (CNNs/ConvNets).

Convolutional Neural Networks are a well-known deep learning architecture inspired by the natural visual perception mechanism of living creatures. Among different types of deep neural networks, CNNs have been most extensively studied. ConvNets are feed-forward neural networks which contain one or more *convolutional layers*. Neurons in

this architecture are purposely spatially arranged to form feature maps. The ConvNet architectures make the explicit assumption that the inputs are images, which allows a creator of it to encode certain properties into the architecture. These properties make the forward function more efficient to implement and vastly reduce the amount of parameters in the network.

Unlike a regular Neural Network as introduced in the previous subsection, the layers of a ConvNet have neurons arranged in 3 dimensions: width, height, depth (not the same as depth of the network). This makes sense, because an image has width, height and number of channels (depth). To reduce number of parameters, the neurons in a layer are only connected to a small fixed-size region of the layer before it and weights are shared among neurons in the same feature map (layer). Such a layer is called a convolutional layer. Except for convolutional layers, in the CNN architecture, there are usually *Pooling Layers* (POOL) and *Fully-Connected Layers* (FC), which are classic layers as introduced in a previous subsection. In CNNs, activation functions are also often referred to as separate layers. One of the most popular activation functions among CNNs is the *ReLU* activation function. It is defined as in 2.5

$$f(x) = max(0, x)^1 \tag{2.5}$$

and its graph can be seen in the Figure 2.3.

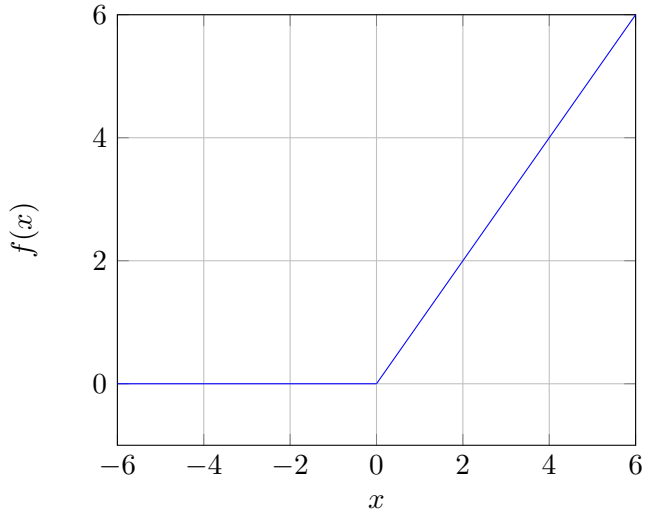


Figure 2.3: ReLU function

Pooling Layers are used to perform a downsampling operation along the spatial dimensions (width, height). After all, we must reduce an input image dimensions to produce an

[1]This function is not differentiable at x=0, but software implementations of neural network training usually return one of the one-sided derivatives rather than reporting that the derivative is not defined or raising an error. Such a solution works well in practice.

output of expected dimensions. It is common to periodically insert a Pooling layer in-between successive convolutional layers in a ConvNet architecture.

A simple CNN for classification could have the architecture [INPUT - CONV - RELU - POOL - FC]. A similar architecture is shown in the Figure 2.4.

For more information about CNNs, please consider the original paper[KSH12].

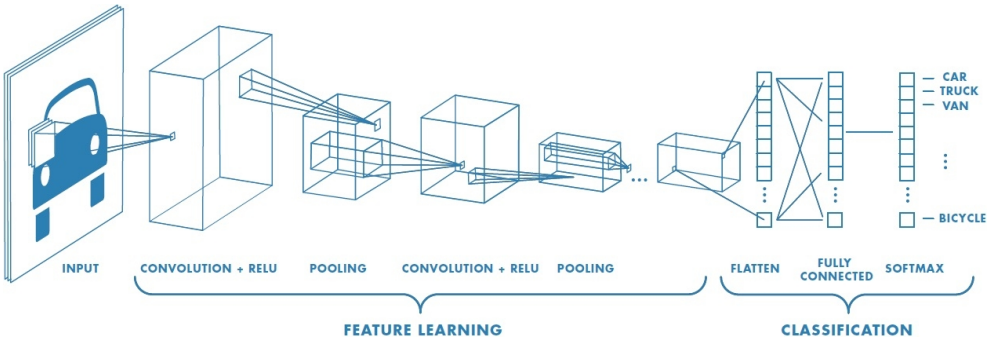


Figure 2.4: A simple deep neural network, image taken from `floydhub.com`

CHAPTER 3

State-of-the-Art

Machine learning (ML) is a rapidly evolving field and a lot of papers have been published in ML area in the last few years. However, since the focus of this master thesis is on generating adversarial examples, the related work can be separated into two categories, one in which DNNs are treated as a white-box and one where they are treated as a black-box.

In terms of white-box attacks, the *Fast Gradient Sign Method* (FGSM) attack is presented in [GSS15]. The attack computes an adversarial image for a non-targeted attack based on the direction of the gradient of a DNN. The FGSM attack is presented in more details in Section 3.1.

In [PMJ+15], the *Jacobian-based Saliency Map Attack* (JSMA) for generating adversarial examples is introduced. The attack is based on identifying regions in an image which have a higher impact on a DNN's output during the classification. JSMA attack is presented in more details in Section 3.2.

In [CW16], the *Carlini & Wagner* (CW) attack is presented. The attack is based on formulating an attack as an optimization problem and then using a state-of-the-art optimizer to solve it. The attack is presented in more details in Section 3.3.

All three attacks, FGSM, JSMA, and CW are evaluated in the experiments in this thesis.

On the black-box side of the attacks, the *transfer-based* approach introduced in [PMG+16] is a popular choice. This approach uses a subsitute DNN that is trained on a similar dataset as the targeted DNN. More details about the transfer-based approach can be found in Section 3.4.

In [LCLS16], the authors show that adversarial samples for a targeted misclassification don't transfer as well as in a pure misclassification attack. The authors suggest an *ensemble* approach which is described in Section 3.5.

In [B18], the authors implement a completely different attack and call it *Boundary Attack*. The attack starts with an image of a targeted class and then, step by step, the image is changed to an image of some other class while staying adversarial, i.e. classified as a target class by a DNN under the attack. The boundary attack is described in Section 3.6.

I direct the interested reader to the survey [AM18] of the different attack strategies and defenses for a more detailed overview.

3.1 Fast Gradient Sign Method (FGSM)

The FGSM attack perturbs an image to increase the loss of the classifier on the resulting image. The target label in the original paper [GSS15] is always a label with the least probability for an unmodified image. An adversarial example is crafted then by perturbing the unmodified image in a way that the cost function is being maximized.

Let $\boldsymbol{\theta}$ be the parameters of a model, $\boldsymbol{x}$ the input to the model, y the target associated with $\boldsymbol{x}$, and $J(\boldsymbol{\theta}, \boldsymbol{x}, y)$ be the cost fucntion used to train the neural network. Then an adversarial perturbation is computed as

$$\boldsymbol{\rho} = \epsilon * sign(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y)).$$

An adversarial example can be crafted then by adding the adversarial perturbation to the original input

$$\boldsymbol{x} = \boldsymbol{x} + \boldsymbol{\rho}.$$

The authors evaluate their method on the ImageNet dataset [DDS+09], a dataset used for a large-image recognition task with 1000 classes, and achieve good results for misclassification. Targeted misclassification was not evaluated. Similar results are achieved on the MNIST dataset [LC10], a dataset used for a digit-recognition task (0-9), and on the CIFAR-10 dataset [KNH], a dataset used for a small-image recognition task, also with 10 classes as in MNIST. From Figure 3.1, a reader can get the intuition for the attack. For more details, I refer the reader to the original paper.

3.2 Jacobian-based Saliency Map Attack (JSMA)

This attack is based on a greedy algorithm that picks pixels to modify one at a time, increasing the likelihood of the targeted class in each iteration. *Adversarial Saliency Maps* - maps that measure an impact of a pixel on an image being classified as a target class - are created. If a value in this map is large, it means that changing that pixel will increase the likelihood of the image being classified as a target class.

The idea is, given the saliency map for a target class, the algorithm picks the pixel with the highest impact and modifies it. In the next iteration, the second most important

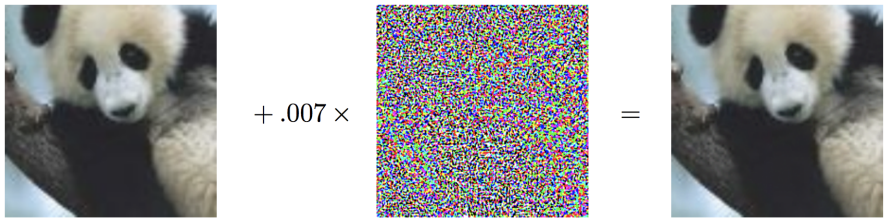


Figure 3.1: Image taken from the [GSS15]

pixel is changed and so on. This continues until either the attack succeeds to trick the classifier or too many pixels get changed and the attack becomes detectable.

In the actual implementation of the algorithm, instead of picking one pixel, a pair of pixels is picked because selecting pixels one at a time is too strict and very few pixels would meet the heuristic search criteria introduced below. A pair of pixels is more likely to match the conditions because one of the pixels can compensate a minor flaw of the other pixel. Larger group of pixels comes at a greater computational cost. For further details, I refer the reader to the original paper [PMJ+15].

Formally, let t be the target class, $\boldsymbol{x}$ be the input to the classifier and $\boldsymbol{F}$ be the output of the softmax layer s.t. $\boldsymbol{F}(\boldsymbol{x})_t$ denotes output of the softmax layer for a class t on the input $\boldsymbol{x}$, i.e. $\boldsymbol{F}(\boldsymbol{x})_t$ denotes the probability that $\boldsymbol{x}$ belongs to the class t. Then the adversarial saliency map in terms of pair pixels p, q is defined as:

$$\alpha_{pq} = \sum_{i \in \{p,q\}} \frac{\partial \boldsymbol{F}(\boldsymbol{x})_t}{\partial \boldsymbol{x}_i},$$

$$\beta_{pq} = \Big(\sum_{i \in \{p,q\}} \sum_{j \neq t} \frac{\partial \boldsymbol{F}(\boldsymbol{x})_j}{\partial \boldsymbol{x}_i} \Big) - \alpha_{pq}$$

so that α_{pq} represents the impact of changing the both pixels p and q on the input being classified as t, and β_{pq} represents how much changing p and q will change all the other outputs of the softmax layer.

Now the algorithm picks p and q such that the target class gets more likely ($\alpha_{pq} > 0$), but other classes get less likely ($\beta_{pq} < 0$) and that combination is as strong as possible, i.e. that $-\alpha_{pq} \cdot \beta_{pq}$ is as large as possible. This can be formalized as:

$$(p^*, q^*) = argmax_{p,q}(-\alpha_{pq} \cdot \beta_{pq})$$
$$\text{s.t. } \alpha_{pq} > 0 \text{ and } \beta_{pq} < 0$$

Starting with an original input, two by two pixels are picked and perturbed by a constant offset ϵ. The authors [PMJ+15] show that JSMA attack can effectively produce MNIST samples that are correctly classified by human subjects, but misclassified into a specific target class by a DNN with a high success rate.

3.3 Carlini & Wagner (CW)

To quantify similarity between two images, different distance metrics can be used. Quantification of similarity can be used when comparing how much an adversarial image is different from the original input. There are three widely-used distance metrics in the literature for generating adversarial examples, all of which are L_p distances. The L_p distance is written $||\boldsymbol{x} - \boldsymbol{x'}||_p$, where the p-norm for purposes of this thesis can be defined as

$$||\boldsymbol{v}||_p = \begin{cases} |\{i|v_i \neq 0\}|, & \text{if } p = 0 \\ (\sum_{i=1}^{n} |v_i|^p)^{1/p}, & \text{if } p \in [1, \infty) \\ max\{|v_1|, |v_2|, ..., |v_n|\} & \text{if } p = \infty \end{cases} \tag{3.1}$$

In other words, L_0 measures how many pixels are changed, L_2 measures standard euclidean distance and L_∞ measures the maximum change to any of the coordinates. It is open for discussion which metric performs the best job in measuring the human perceptual of similarity, but neither of the L_p metrics is optimal for that.

The authors [CW16] introduce three new attacks for the L_0, L_2, and L_∞ distance metrics. It is worth mentioning that their L_0 attack is the first published attack which can cause targeted misclassification on the ImageNet dataset. All three of them are based on optimization techniques.

In this thesis, L_2 is used in the attack and hence I explain it now.

The authors start by using the initial formulation of adversarial examples [SZS+13] and define the problem of finding an adversarial sample $\boldsymbol{x}$ as follows:

$$\text{minimize } \mathcal{D}(\boldsymbol{x}, \boldsymbol{x} + \boldsymbol{\delta})$$
$$\text{such that } \mathcal{C}(\boldsymbol{x} + \boldsymbol{\delta}) = t$$
$$\boldsymbol{x} + \boldsymbol{\delta} \in [0, 1]^n$$

where t is the target class, $\boldsymbol{\delta}$ is perturbation added to the original input $\boldsymbol{x}$, $\mathcal{C}$ is a function performed by the classifier, and $\mathcal{D}$ is either L_0, L_2 or L_∞.

Since the constraint $\mathcal{C}(\boldsymbol{x}+\boldsymbol{\delta}) = t$ is highly non-linear and therefore hard to solve directly for existing algorithms, the authors introduce the function f such that $\mathcal{C}(\boldsymbol{x}+\boldsymbol{\delta}) = t$ if and only if $f(\boldsymbol{x}+\boldsymbol{\delta}) \leq 0$. Now problem can be formulated as

$$\begin{aligned} &\text{minimize } \mathcal{D}(\boldsymbol{x}, \boldsymbol{x}+\boldsymbol{\delta}) \\ &\text{such that } f(\boldsymbol{x}+\boldsymbol{\delta}) \leq 0 \\ &\qquad\qquad \boldsymbol{x}+\boldsymbol{\delta} \in [0,1]^n \end{aligned}$$

or using the alternative formulation:

$$\begin{aligned} &\text{minimize } \mathcal{D}(\boldsymbol{x}, \boldsymbol{x}+\boldsymbol{\delta}) + c \cdot f(\boldsymbol{x}+\boldsymbol{\delta}) \\ &\text{such that } \boldsymbol{x}+\boldsymbol{\delta} \in [0,1]^n \end{aligned}$$

where $c > 0$ is a suitably chosen constant. The authors in their implementation use a modified binary search to find the optimal value of c.

Let $\boldsymbol{Z}$ be the output of the targeted DNN second-to-last layer, the logits, with $\boldsymbol{Z}_i$ as an output for the class i.

The function f that the authors find the most effective is:

$$f(\boldsymbol{x}+\boldsymbol{\delta}) = max(max(\{\boldsymbol{Z}(\boldsymbol{x}+\boldsymbol{\delta})_i : i \neq t\}) - \boldsymbol{Z}(\boldsymbol{x}+\boldsymbol{\delta})_t, 0). \tag{3.2}$$

To ensure the modification yields a valid image, there is a constraint $\boldsymbol{x}+\boldsymbol{\delta} \in [0,1]^n$. The authors refer to this constraint as a "box constraint". The Adam [KB14] optimizer does not support box constraints natively and the authors modify the box constraint as follows in order to be able to use the Adam optimizer. A new variable $\boldsymbol{\omega}$ is introduced and instead of optimizing over the variable $\boldsymbol{\delta}$, an optimization is done over $\boldsymbol{\omega}$, setting

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

Now the solution will automatically be valid since from $-1 \leq tanh(\omega_i) \leq 1$ it follows that $0 \leq x_i + \delta_i \leq 1$.

Finally, for $\mathcal{D} = L_2$, the attack can be formalized as follows. Given the original sample $\boldsymbol{x}$ and the target class t, search for $\boldsymbol{\omega}$ that solves [1]

[1] Here $\mathbf{1}$ represents a vector of same dimensionality as $\boldsymbol{x}$ and $\boldsymbol{\omega}$, it has value 1 at every index and it shouldn't be confused with a ground-truth vector $\mathbf{1}_p$.

$$\text{minimize } (||\boldsymbol{x} - \frac{1}{2}(tanh(\boldsymbol{\omega}) + \mathbf{1})||_2)^2 + c \cdot f(\frac{1}{2}(tanh(\boldsymbol{\omega}) + \mathbf{1}))$$

with f similar to the objective function defined in 3.2, but this time defined as

$$f(\boldsymbol{x}') = max(max\{\boldsymbol{Z}(\boldsymbol{x}')_i : i \neq t\} - \boldsymbol{Z}(\boldsymbol{x}')_t, -\kappa)$$

where κ is a parameter that controls the confidence with which the misclassification occurs. The authors in their implementation set $\kappa = 0$. The adversarial example is then crafted as $\boldsymbol{x}' = \boldsymbol{x} + \boldsymbol{\delta}$. For more details about the attack, please consider [CW16].

According to the authors, this attack is often much more effective (and never worse) than all the others presented in the literature. Attacks are evaluated on the three datasets: ImageNet, MNIST and CIFAR-10. They also report that the JSMA attack, an attack introduced in Section 3.2, is not able to craft an adversarial example when the ImageNet dataset is used due to memory complexity of the algorithm, i.e. dimensions of images in ImageNet dataset are too big for JSMA attack. This implies that the JSMA attack would not work in my thesis as well if an image of a person is too big. Reported results for the CW attack are showing 100% success against all three datasets.

3.4 Transfer based approach

This technique is used to attack the DNN in the black-box settings. The idea is to create a *substitute* DNN which should be similar to the targeted DNN. A precise definition of the similarity is omitted here because it's not well defined, but the substitute DNN should solve the same task as the targeted DNN.

Adversarial images are crafted then for a substitute DNN using a white-box approach, for instance the FGSM attack introduced in Section 3.1. Created adversarial images are used then as adversarial images for the black-box DNN as well. The main idea is that similar classifiers will have similar boundaries for a specific class and therefore the same adversarial example should be adversarial for both networks.

The dataset on which the substitute neural network is trained should be similar to the dataset on which the targeted neural network is trained. Ideally, that would be the same dataset, but the assumption is that an attacker doesn't have access to that data.

The attacker therefore generates a Synthetic Dataset. He or she starts generating the dataset by querying the targeted DNN with several examples and obtaining labels for them. Afterwards, he or she expands the dataset using the Jacobian-based Dataset Augmentation and trains the substitute neural network. For more details how to generate the synthetic dataset, please consult the original paper [PMG+16].

The authors present good results for misclassification attacks against the MNIST dataset and the GTSRD dataset [SSSI12a]. Targeted misclassification is not presented in the paper.

3.5 Ensemble approach

The authors of [LCLS16] show that using the transfer based approach, introduced in Section 3.4, the target labels don't transfer well for the targeted misclassification attack. The transfer based approach seems good for a misclassification task, but not for a targeted misclassification task. The goal of the ensemble-based approach is to solve this issue.

The ensemble-based approach is also based on transferability of an adversarial image, but instead of generating an adversarial image for one substitute neural network, an attacker generates it for several of neural networks - *the ensemble of the models.* The underlying assumption is that if an adversarial example works as expected among several models, there is a higher probability that it will work as expected for the one more as well.

Formally, given k white-box models with softmax outputs being $\boldsymbol{J}_1$, ..., $\boldsymbol{J}_k$, an original image $\boldsymbol{x}$, and its ground-truth vector $\boldsymbol{y}$, the ensemble-based approach solves the following optimization problem:

$$argmin_{\boldsymbol{x}^*}(-log((\sum_{i=1}^{k} \alpha_i \boldsymbol{J}_i(\boldsymbol{x}^*)) \cdot \boldsymbol{y}^*) + \lambda d(\boldsymbol{x}, \boldsymbol{x}^*)) \tag{3.3}$$

where $\boldsymbol{y}^*$ is a vector encoding the target label specified by the adversary, $\sum_{i=1}^{k} \alpha_i \boldsymbol{J}_i(\boldsymbol{x}^*)$ is the ensemble model, and α_i are the ensemble weights s.t. $\sum_{i=1}^{k} \alpha_i = 1$.

The authors use two approaches to solve this problem: optimization-based and fast gradient-based. The optimization-based approach uses a state-of-the-art optimizer to solve the optimization problem defined in Equation 3.3. The authors observe that the optimization-based approach outputs a large proportion of targeted adversarial images whose target labels transfer. The exact percentage depends on the architectures used for the ensemble models and for the targeted DNN and can vary from 11% up to 99%. The fast gradient-based approach using the ensemble model gives result no better than using the single model.

3.6 Boundary attack

This approach is also used in black-box settings and it is completely different from the attacks introduced in Sections 3.4 and 3.5. The boundary attack has nothing to do with either a substitute DNN or transferability of the adversarial examples.

The attack starts with an image of a targeted class and then, step by step, it changes it to an image of some other class while staying adversarial, i.e. classified as a target class by a DNN under the attack. In every iteration of the attack, the image is changed a little bit towards a class which will be in the image in the end, at least according to a human observer. More specifically, in every iteration of the attack, a perturbation that reduces the distance of the perturbed image (adversarial sample) towards the original

input (an image of a class that is presented to a human observer) is added. For specific details how this perturbation is selected, please consult the original paper [B18].

After every change, the targeted DNN is queried to check if the image is still adversarial, i.e. classified as a target label. If not, the change is reverted. In this way, the attacker doesn't need any substitute neural network.

However, this attack comes at cost of a large number of queries to the targeted DNN. For the targeted attack, the authors needed around 10^4 queries to get an adversarial example. The real world systems could notice such intensive querying of their APIs and detect the attack. On top of that, the attacker needs both an image of the targeted class and an image of the class that will be presented to a human observer. That could be an obstacle when the number of classes is high because it can happen that it is not easy to find an image of a particular class.

The authors compare their boundary attack with the white-box CW attack, introduced in Section 3.3, on MNIST and CIFAR-10 dataset and produce only a bit worse results, although this attack is treating a targeted DNN as a black-box. For more information about this approach, please consult the original paper [B18].

CHAPTER 4

Experiments

In this chapter, I evaluate the effectiveness of the methods proposed in Chapter 3 for producing adversarial examples.

First, I present my evaluation methodology. Then I present evaluation results for whitebox attacks. Finally, I present results for blackbox attacks.

4.1 Methodology

To evaluate effectiveness of the attacks, I trained a classifier for age estimation. First I describe the dataset I used to train the classifier and then I present results of training.

For training data, I created a dataset based on two datasets: the APPA-REAL dataset [EA17] and the UTK Face dataset [1].

The APPA-REAL dataset contains 7,591 images with associated age labels. The dataset is split into 4113 training, 1500 valid, and 1978 test images. For each image in the dataset, there is also the corresponding image which contains the cropped face. The distribution of samples over age for training dataset is presented in Figure 4.2.

The UTK Face dataset consists of 23252 images with associated age labels. I preprocessed every image to extract the face[2] from it. For face detection, I used the Dlib [Kin09] library. The distribution of samples over age is presented in Figure 4.1.

I constructed my training dataset as union of all images from the UTK Face dataset and training images from the APPA-REAL dataset. For my validation dataset and my test dataset I used validation images and test images from the APPA-REAL validation and test dataset, respectively.

[1] https://susanqq.github.io/UTKFace/

[2] every face is cropped with 40% margin

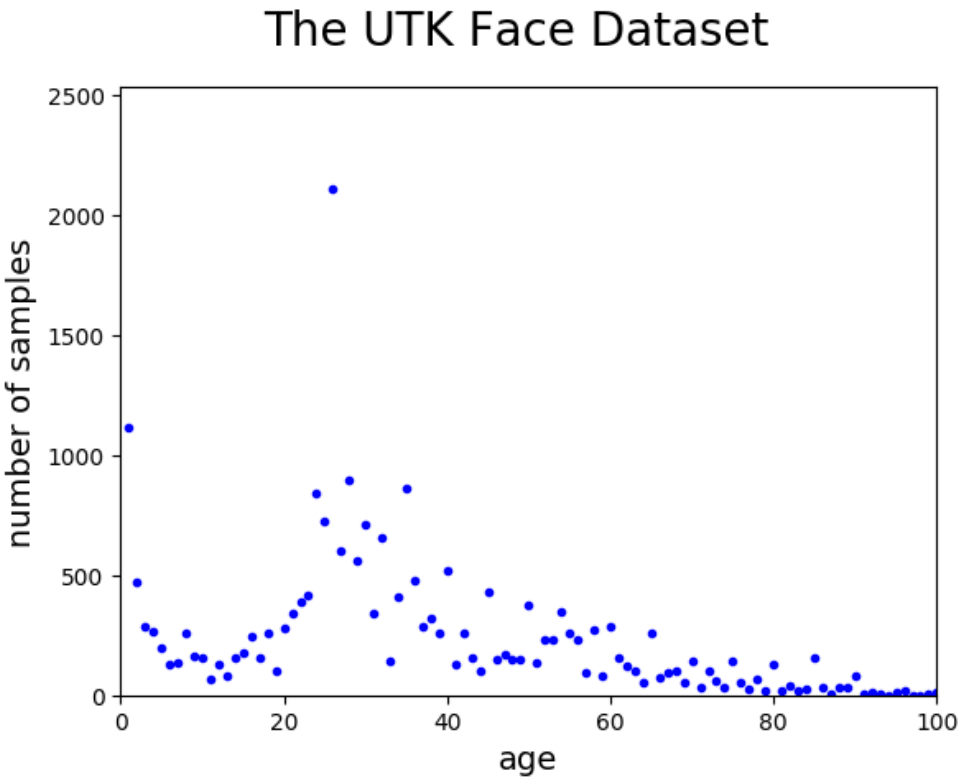


Figure 4.1: Number of samples per age in the UTK Face Dataset

Instead of training a DNN from scratch, I used a pretrained DNN that is trained on the ImageNet dataset and fine-tuned it for age estimation. In other words, I downloaded already existing model of a DNN and re-trained it for my specific task, i.e. age estimation. This technique is called *transfer learning* and more information about it can be found in [YCBL14].

If a person who is 65 years old gets classified as 66 years old, a classifier did actually a good job although it didn't predict the correct class. However, if a person who is 65 years old gets classified as 6 years old, that is a significantly bigger mistake than classifying it as 66 years old. Yet, when computing an accuracy, there is no difference if a predicted class is 6 or 66 if a ground truth is 65. That is the reason why in age estimation usually a mean absolute error (MAE) of a classifier is measured instead of accuracy.

When it comes to (targeted) misclassification, there is something in the age estimation domain that needs to be taken care of. If a person is classified a year or two older, it is not a huge success for a (targeted) misclassification. In the experiments that follow, targeted version of attacks is used, but a label is set as follows. If a person is under 50 years old, the target label is 90 years old. If a person is 50 years old or older, the target label is 10 years old.

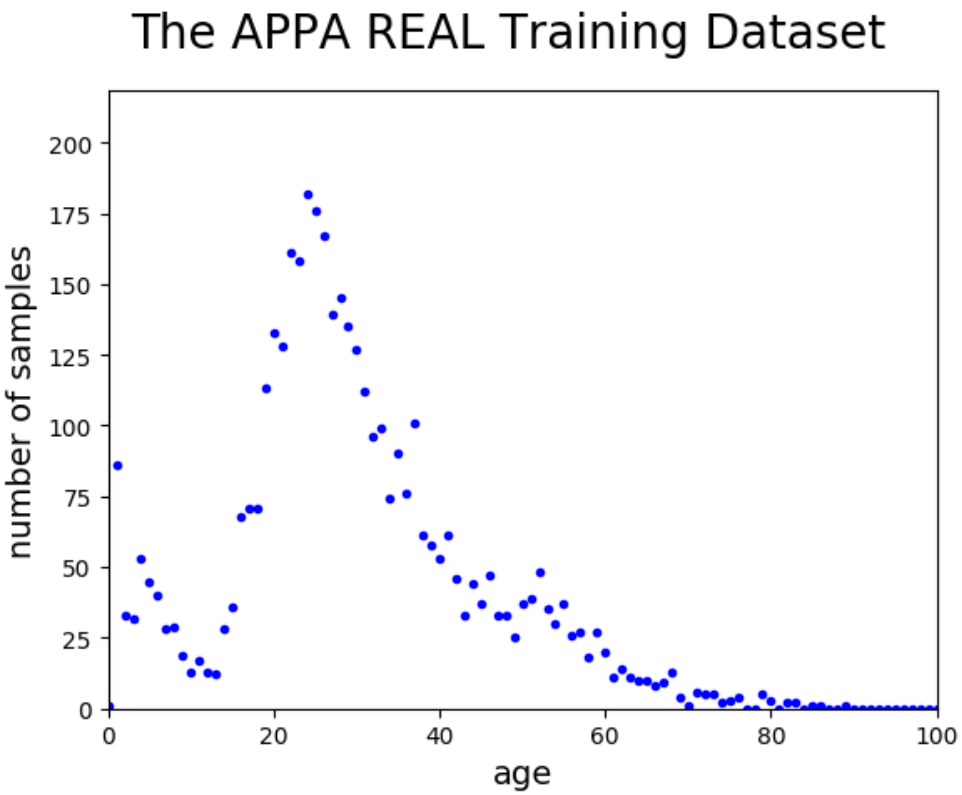


Figure 4.2: Number of samples per age in the APPA-REAL Training Dataset

Formally, let y be the ground truth label of an input sample $\boldsymbol{x}$. Then function $f(y)$ is a function that outputs the target label for the corresponding sample $\boldsymbol{x}$. The function f is defined as

$$f(y) = \begin{cases} 10, & y \geq 50 \\ 90, & \text{otherwise} \end{cases} \tag{4.1}$$

The idea behind such a setting is to maximize the mean absolute error, i.e. make the classification as wrong as possible, using the targeted version of known attacks. This is how I reused targeted attacks in a novel setting. Initially I put targeted labels to 0 and 100 instead of 10 and 90, respectively, but I observed a worse performance, probably because they are on the edge of the considered age range.

Two different optimizers are used in the experiments for minimizing a loss function: SGD, introduced in Section 2.2, and Adam [KB14]. Three different DNN architectures are used for training: VGG16 [SZ14], ResNet-50 [HZRS15], and InceptionResNetV2[SIV16]. Results of training the models based on the ResNet-50 architecture and the Inception-ResNetV2 architecture are presented in Table 4.1.

Id	Architecture	Optimizer	Validation Loss	**Validation MAE**
1	ResNet-50	SGD	3.436	5.151
2	ResNet-50	Adam	3.456	6.772
3	InceptionResNetV2	SGD	3.086	4.505
4	InceptionResNetV2	Adam	3.268	3.922

Table 4.1: Different models trained

The VGG16 architecture didn't show nearly as good results as the other two did, i.e. it always predicted the value of the most frequent label. Hence, I do not present the results of training the VGG16 in Table 4.1 and I do not evaluate any adversarial attack against it. However, if the VGG16 architecture is trained on the significantly larger IMDB-WIKI dataset, it can produce good results [RTG16].

For the evaluation of attacks, 100 random samples are taken from the APPA-REAL test dataset. Neither of the models from Table 4.1 has ever been trained on any of these samples before.

4.2 Whitebox attacks

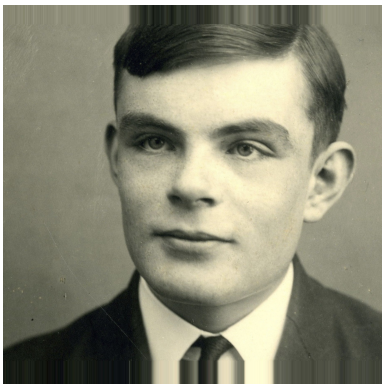
When it comes to whitebox attacks, I evaluated three approaches: FGSM, CW and JSMA. Hyperparameters used in the FGSM and the CW attack are listed in Table 4.2 and Table 4.3, respectively. In Table 4.5 and 4.6 results are presented for FGSM attack and CW attack, respectively. In Table 4.4 results of all three attacks are presented.

It is interesting to notice that the FGSM attack managed to change the MAE for the model with id 2 from 9.5 to 47.79. In other words, the targeted model on average predicted that a person in the adversarial image is 38 years younger or older than a person in the original image!

Several original samples and the corresponding adversarial samples crafted using the FGSM attack are presented in Figure 4.3. It can be observed that the attack adds a similar pattern to every image. However, when hyperparameter *eps* is reduced to 1.0, it is almost impossible to notice the perturbation as it can be seen in Figure 4.4. But it still can be noticed that there is some perturbation. Lines in adversarial images are not that sharp as in original images and some pixels are not natural, e.g. the line between left cheek and the background in the image (d).

The CW attack, given the hyperparameters in Table 4.3, managed to move the MAE for the model with id 1 from 6.69 to 29.35. In other words, the targeted model on average predicted that a person in the adversarial image is 23 years younger or older than a person in the original image. This is not that much as FGSM managed for the model with id 2, but it is significant.

It is also interesting to observe that amount of perturbation added to an original sample varies a lot when the CW attack is used. From the value 0.00 for minimal perturbation



(a) Original sample classified as 28 years old

(b) Adversarial sample classified as 85 years old

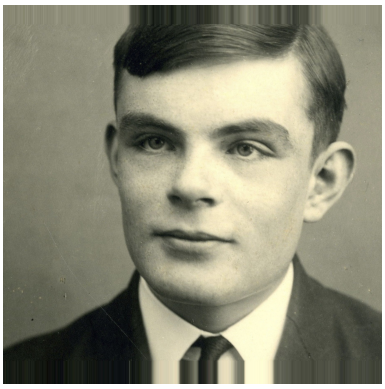
(c) Original sample classified as 59 years old

(d) Adversarial sample classified as 23 years old

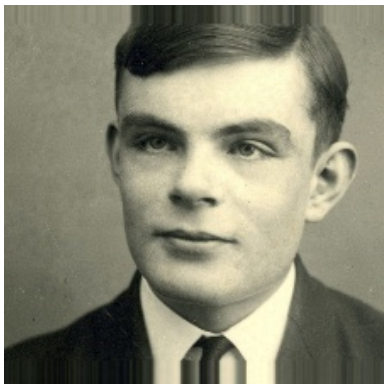
(e) Original sample classified as 28 years old

(f) Adversarial sample classified as 85 years old

Figure 4.3: Original samples (left) and adversarial samples (right) crafted using the FGSM algorithm (*eps* set to 5.0) and evaluated against the model with id 2



(a) Original sample classified as 28 years old



(b) Adversarial sample classified as 59 years old

(c) Original sample classified as 59 years old

(d) Adversarial sample classified as 28 years old

(e) Original sample classified as 28 years old

(f) Adversarial sample classified as 64 years old

Figure 4.4: Original samples (left) and adversarial samples (right) crafted using the FGSM algorithm (*eps* set to 1.0) and evaluated against the model with id 2

eps	5
clip min	0
clip max	255
y_target	10 or 90

Table 4.2: Values of the hyperparameters used in the FGSM attack

it can be observed that attack sometimes does not manage to find an adversarial sample and sometimes the added perturbation is rather huge. For example, the maximal value for L2 distance between the original and the adversarial sample using CW attack for model 1 is 16419.60, but when FGSM is used, that value is 1939.89.

Several original samples and the corresponding adversarial samples crafted using the CW attack are presented in Figure 4.5. It can be observed that the attack adds a very strong pattern to every image, i.e. introduces high perturbation. This is a consequence of chosen values for the number of iterations and the learning rate.

Since the CW attack wasn't able to find any adversarial sample for the model with id 2 with the given hyperparameters in Table 4.3, further experiments are performed against that model with different values for learning rate and maximum number of iterations.

Results of those experiments are presented in Table 4.7. The results show that it is not easy for the CW attack to find any adversarial sample for the specific model. Since no adversarial sample is found, I performed no further exploration of combination of learning rate and number of iterations. Further argumentation of this decision can be found in Section 6.1.

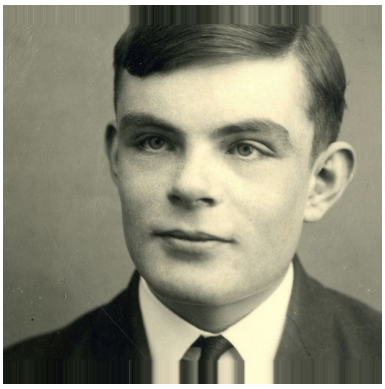
It is also interesting to notice that for the models with id 1 and 2, the attacks find stronger adversarial samples (i.e. bigger change in MAE) than for models with id 3 and 4. Could it be that bigger models[3] are more resistant to adversarial samples?

As expected from the analysis of the paper [CW16] in Section 3.3, the JSMA attack failed due to memory complexity of the algorithm.

Finally, I upload several original samples and corresponding adversarial samples to Microsoft's service for age estimation[4] to check if there is any transferability between adversarial samples. Usually, results are as expected as it can be seen in Figure 4.6. However, as it can be seen in 4.7 , sometimes an adversarial sample manages to trick Microsoft's service for age estimation as well!

[3]ResNet50 architecture that is used for model 1 and 2 has 25,636,712 parameters and InceptionResNetV2 architecture that is used for model 3 and 4 has 55,873,736 parameters.

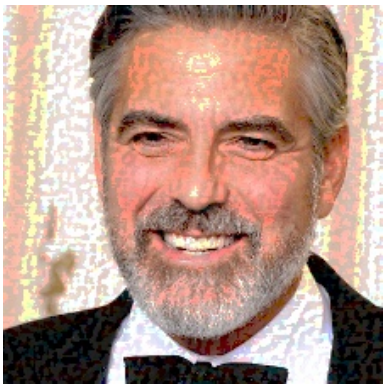
[4]https://www.how-old.net/



(a) Original sample classified as 36 years old

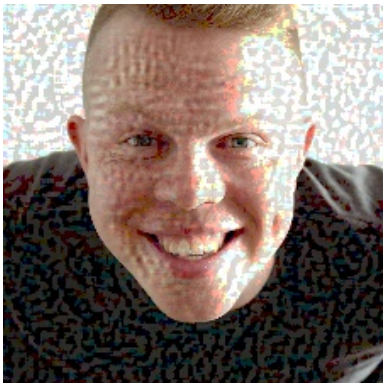
(b) Adversarial sample classified as 79 years old

(c) Original sample classified as 55 years old



(d) Adversarial sample classified as 10 years old

(e) Original sample classified as 35 years old



(f) Adversarial sample classified as 84 years old

Figure 4.5: Original samples (left) and adversarial samples (right) crafted using the CW algorithm and evaluated against the model with id 1

binary_search_steps	8
y_target	10 or 90
abort_early	True
max_iterations	5000
learning_rate	1
clip_max	255
clip_min	0
initial_const	0.1

Table 4.3: Values of the hyperparameters used in the CW attack

		FGSM		CW		JSMA	
model id	clean MAE	adv MAE	avg L2	adv MAE	avg L2	adv MAE	avg L2
1	6.69	32.25	1879.63	29.35	2977.28	-	-
2	9.50	47.79	1879.49	9.50	0.00	-	-
3	5.40	19.32	2510.92	7.51	1307.51	-	-
4	4.62	15.03	2511.10	6.79	1104.78	-	-

Table 4.4: Results of different adversarial attacks. The "-" sign means that an attack couldn't be executed.

model id	clean MAE	adv MAE	avg L2	std dev L2	min L2	max L2
1	6.69	32.25	1879.63	77.23	1637.33	1939.89
2	9.50	47.79	1879.49	77.26	1649.43	1939.90
3	5.40	19.32	2510.92	100.00	2201.44	2589.42
4	4.62	15.08	2511.10	99.53	2204.20	2589.41

Table 4.5: Results of FGSM attack

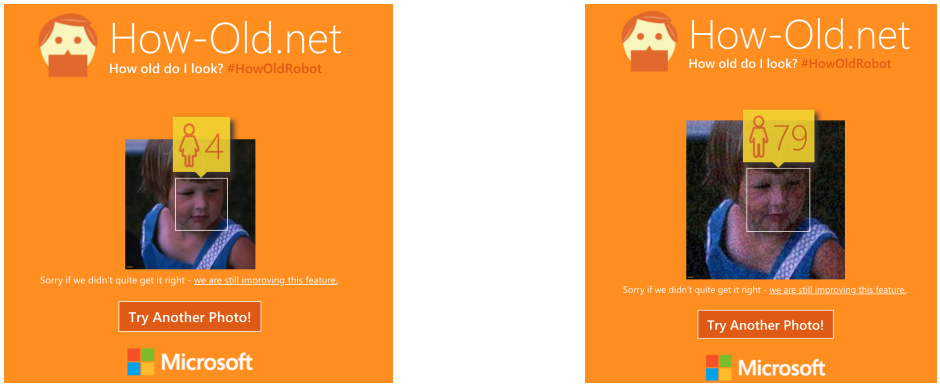
model id	clean MAE	adv MAE	avg L2	std dev L2	min L2	max L2
1	6.69	29.35	2977.28	3532.85	0.00	16419.60
2	9.50	9.50	0.00	0.00	0.00	0.00
3	5.40	7.51	1307.51	2969.74	0.00	11071.88
4	4.62	6.79	1104.78	3070.10	0.00	17673.81

Table 4.6: Results of CW attack

max iterations	learning rate	clean MAE	adv MAE	avg L2
5 000	1	9.5	9.5	0.0
10 000	1	9.5	9.5	0.0
10 000	0.1	9.5	9.5	0.0
10 000	10.0	9.5	9.5	0.0
100 000	0.01	9.5	9.5	0.0

Table 4.7: Results using the CW attack with different values of hyperparameters against model with id 2

Figure 4.6: Correct evaluation by Microsoft's service for age estimation



(a) Original sample (b) Adversarial sample

Figure 4.7: Adversarial sample tricked Microsoft's service for age estimation

4.3 Blackbox attacks

When it comes to blackbox attacks, the transfer based approach introduced in Section 3.4 is evaluated as well as the boundary attack from Section 3.6. It is worth mentioning that in this thesis a *blackbox attack* means that the adversary can only query the targeted neural network and obtain an associated label. The adversary has no information about the confidence of the network in the prediction or any kind of list of most probable labels.

In the transfer based approach, as described in Section 3.4, the goal is to train a substitute network that will learn similar decision boundaries for every class as the targeted network. This implies that the total number of possible classes must be the same, but an architecture of the substitute network may be different from the architecture of the targeted network.

To get information about boundaries of the targeted neural network, I take 778 samples that are previously unknown to the targeted neural network, obtain labels for those samples and train a substitute neural network based on those results. The distribution of samples over age is presented in Figure 4.8.

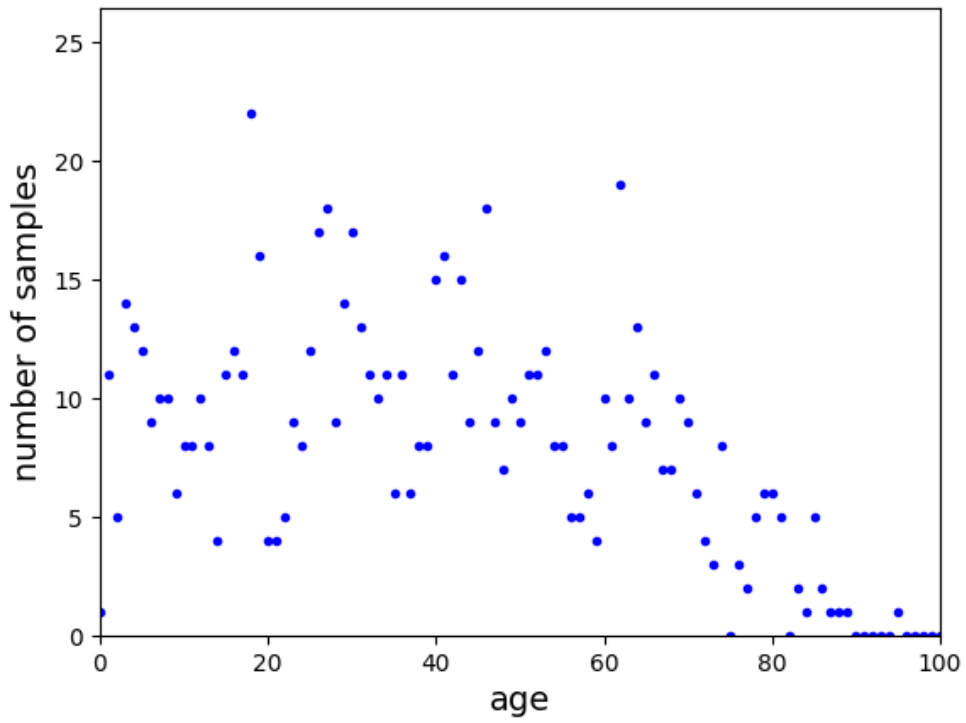


Figure 4.8: Number of samples per age in the dataset that is used for training a substitute neural network by querying a targeted blackbox neural network

Next, I take 200 random samples that are yet unseen by both networks and evaluate MAE of the targeted neural network and the substitute network on those 200 images.

Then I craft adversarial samples for the substitute neural network using those 200 images. Target labels for adversarial samples are set in the same manner as in whitebox attacks, i.e. label 90 if a person is under 50 years old and label 10 if a person is over 50 years old.

Finally, I evaluate the MAE of the targeted neural network and the substitute network on the 200 adversarial samples crafted in the previous step.

In my first attempt, I tried to use the InceptionResNetV2 architecture as a substitute network. However, the experiment failed due to memory consumption when the Jacobian augmentation of the dataset was performed.

In my second attempt, I used the ResNet50 architecture for the substitute network instead of the InceptionResNetV2 architecture. However, the experiment again failed due to the memory exhaustion when the Jacobian augmentation of the dataset was performed.

In my third attempt, I used a vanilla CNN with three convolutional layers followed by a flatten layer and a softmax layer used for classification. However, the experiment failed again due to the same reason.

Finally, I skipped the step of the Jacobian augmentation of the dataset in the transfer based approach and managed to run the experiments. The Jacobian augmentation is a heuristic that computes how to expand the existing dataset so that new samples are closer to the boundaries of being classified as a certain class. Instead of using this heuristic, I query once the targeted neural network, train the substitute network based on the associated labels provided by the targeted neural network and execute the attack. This way the substitute network does learn information about classification boundaries of the targeted neural network, but less than it would learn using the Jacobian augmentation. Further justification of this decision can be found in Section 6.1.

Results of this modified transfer based approach are presented in Tables 4.8, 4.9, 4.10, 4.11, 4.12, 4.13, 4.14, and 4.15. Two things can be observed.

The first observation is that the InceptionResNetV2 architecture with the SGD optimizer is a combination of the neural network architecture and the optimizer that learned the most from the black-box network in terms of age estimation task. The substitute network managed to have an MAE under 10.0 on clean samples as can be seen in Tables 4.12 and 4.14.

The second observation is regarding the black-box accuracy after the attack, i.e. against the adversarial samples. Only the FGSM attack managed to increase the MAE of the black-box above 10.0 and only against the black-box model with id 2 as can be seen in Tables 4.12 and 4.8. However, since the MAE of the black-box model against clean samples is 6.99, this is not a large increase in MAE. This observation implies the result of the experiment.

The result of the experiment is that this approach is not successful. No matter which combination of an attack, a substitute network, and an optimizer used for training a network is used, there was no huge impact on the performance of a black-box model.

	clean samples		adversarial samples					
blackbox model id	blackbox MAE	substitute MAE	blackbox MAE	substitute MAE	avg L2	std dev L2	min L2	max L2
1	6.08	12.42	6.92	19.14	1890.86	58.77	1580.84	1939.89
2	6.99	11.52	**10.20**	23.15	1891.01	59.07	1565.25	1939.89
3	5.29	14.28	5.79	25.69	1891.34	58.79	1559.34	1939.89
4	3.75	11.40	4.92	19.08	1891.43	58.15	1575.75	1939.89

Table 4.8: Substitute network: ResNet50 architecture with SGD optimizer; Attack: FGSM

	clean samples		adversarial samples					
blackbox model id	blackbox MAE	substitute MAE	blackbox MAE	substitute MAE	avg L2	std dev L2	min L2	max L2
1	6.08	20.28	6.99	19.91	1890.10	60.31	1555.87	1939.89
2	6.99	11.94	7.44	12.15	1921.91	21.20	1809.24	1939.89
3	5.29	24.99	4.91	26.1	1893.30	58.41	1534.87	1939.89
4	3.75	10.53	4.0	9.75	1900.11	48.29	1662.20	1939.89

Table 4.9: Substitute network: ResNet50 architecture with Adam optimizer; Attack: FGSM

	clean samples		adversarial samples					
blackbox model id	blackbox MAE	substitute MAE	blackbox MAE	substitute MAE	avg L2	std dev L2	min L2	max L2
1	6.08	13.73	6.17	14.90	369.75	1407.87	0.0	6269.29
2	6.99	10.54	6.99	10.54	0.00	0.00	0.00	0.00
3	5.29	16.58	5.37	17.67	369.31	468.55	0.00	7710.35
4	3.75	11.83	4.00	11.58	33.56	473.54	0.00	6713.73

Table 4.10: Substitute network: ResNet50 architecture with SGD optimizer; Attack: CW

	clean samples		adversarial samples					
blackbox model id	blackbox MAE	substitute MAE	blackbox MAE	substitute MAE	avg L2	std dev L2	min L2	max L2
1	6.08	17.45	6.08	17.45	0.00	0.00	0.00	0.00
2	6.99	10.53	6.99	10.53	0.00	0.00	0.00	0.00
3	5.29	21.43	5.29	21.43	0.00	0.00	0.00	0.00
4	3.75	11.92	3.75	11.92	0.00	0.00	0.00	0.00

Table 4.11: Substitute network: ResNet50 architecture with Adam optimizer; Attack: CW

	clean samples		adversarial samples					
blackbox model id	blackbox MAE	substitute MAE	blackbox MAE	substitute MAE	avg L2	std dev L2	min L2	max L2
1	6.08	10.20	7.57	13.63	2524.84	77.62	2097.67	2589.41
2	6.99	11.09	**10.66**	18.59	2525.10	76.92	2105.15	2589.41
3	5.29	21.02	5.57	25.09	2527.32	73.56	2143.89	2589.41
4	3.75	8.25	4.25	14.75	2525.40	76.66	2111.09	2589.41

Table 4.12: Substitute network: InceptionResNetV2 architecture with SGD optimizer; Attack: FGSM

	clean samples		adversarial samples					
blackbox model id	blackbox MAE	substitute MAE	blackbox MAE	substitute MAE	avg L2	std dev L2	min L2	max L2
1	6.08	10.31	6.32	9.50	2529.29	71.93	2142.99	2589.41
2	6.99	10.73	9.83	6.99	2535.28	63.24	2227.84	2589.41
3	5.29	25.88	5.19	24.74	2544.61	49.88	2368.79	2589.41
4	3.75	11.82	3.92	10.50	2537.30	79.94	1996.58	2589.41

Table 4.13: Substitute network: InceptionResNetV2 architecture with Adam optimizer; Attack: FGSM

	clean samples		adversarial samples					
blackbox model id	blackbox MAE	substitute MAE	blackbox MAE	substitute MAE	avg L2	std dev L2	min L2	max L2
1	6.08	18.55	6.08	18.55	0.00	0.00	0.00	0.00
2	6.99	14.82	6.99	4.82	0.00	0.00	0.00	0.00
3	5.29	30.20	5.29	30.20	0.00	0.00	0.00	0.00
4	3.75	8.17	3.83	8.09	157.46	1276.19	0.00	10773.81

Table 4.14: Substitute network: InceptionResNetV2 architecture with SGD optimizer; Attack: CW

	clean samples		adversarial samples					
blackbox model id	blackbox MAE	substitute MAE	blackbox MAE	substitute MAE	avg L2	std dev L2	min L2	max L2
1	6.08	11.70	6.08	11.70	0.00	0.00	0.00	0.00
2	6.99	11.37	6.99	11.37	0.00	0.00	0.00	0.00
3	5.29	24.83	5.29	24.83	0.00	0.00	0.00	0.00
4	3.75	10.25	3.75	10.25	0.00	0.00	0.00	0.00

Table 4.15: Substitute network: InceptionResNetV2 architecture with Adam optimizer; Attack: CW

Regarding the boundary attack, it's hard to do any quantitative analysis because the attack, as described in Section 3.6, is fundamentally different than the transfer based approach.

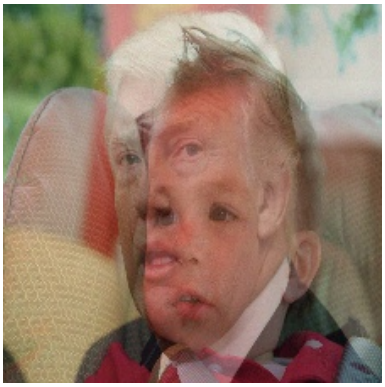
One sample can be seen in Figure 4.9 and the corresponding results in Figure 4.12. It is interesting to observe what is happening during the attack.

In the beginning of the attack, as presented in Figure 4.10, the original sample is classified as 68 years old and the targeted image, as presented in Figure 4.11, is classified as 4 years old.

As the attack performs more queries, the adversarial sample is looking more and more like the targeted image, but the values that the network is predicting also start to correspond to the targeted image. This can be seen in Figure 4.12.

Finally when the attack is finished, the adversarial sample seems as the targeted image and the network is very aware of it, i.e. there is a high probability that the adversarial sample is classified as 4 years old. However, probability that the person in the adversarial image is 68 years old is just a bit higher than the probability that the person is 4 years old and that is enough for the network to classify the person as 68 years old.

(a) The starting image



(b) 1000 queries

(c) 2000 queries

(d) 4000 queries

(e) 8000 queries

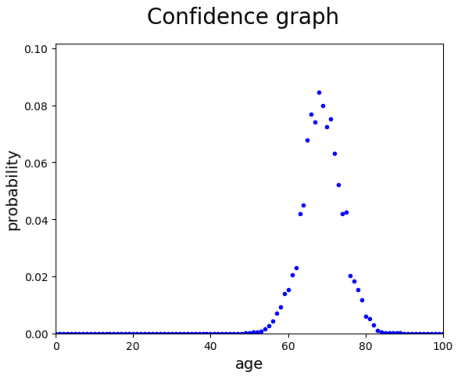
(f) 12000 queries

(g) 16000 queries

(h) Final adversarial sample

Figure 4.9: Although an adversary is changing the image, the blackbox classifier is not changing the prediction (68 years old)

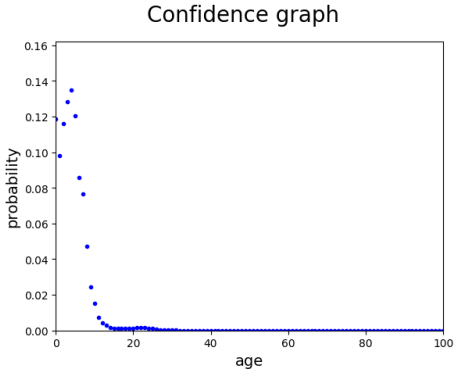
(a) Starting image



(b) Predictions for the starting image

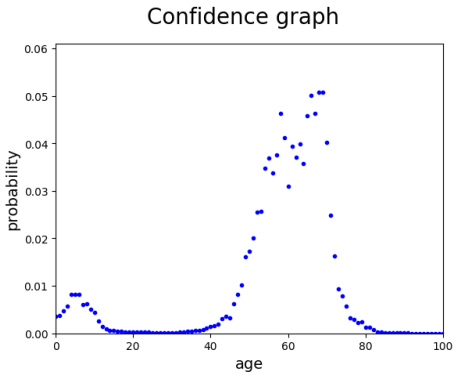
Figure 4.10: Starting image and the corresponding prediction

(a) The targeted image

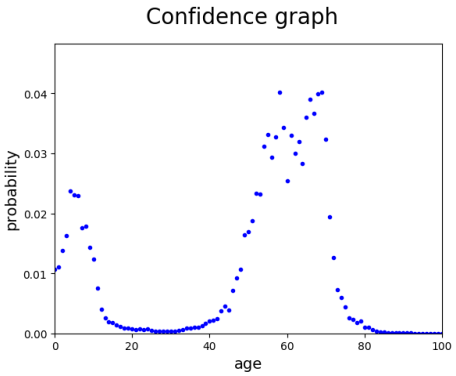


(b) Predictions for the targeted image

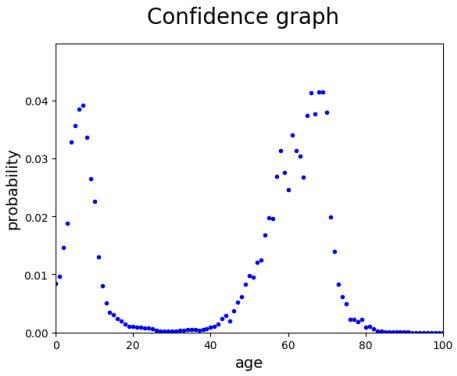
Figure 4.11: Targeted image and the corresponding predictions



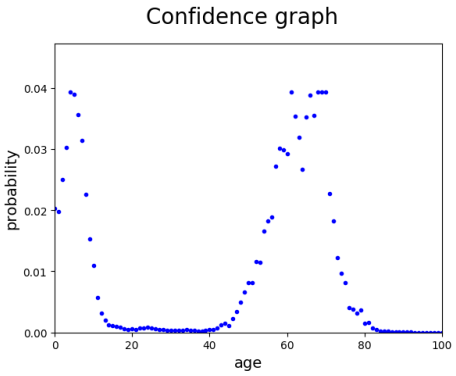
(a) 3000 queries



(b) 4000 queries



(c) 8000 queries



(d) The final adversarial sample (77 000 queries)

Figure 4.12: Predictions of the blackbox classifier for images corresponding to Figure 4.9. In all the graphs, age 68 is predicted with the maximum probability.

CHAPTER 5

The Semi-targeted Approach

In Section 4.3 I made the observation that the transfer based approach expects more memory than is available to me. In the same section I also present poor results of the transfer-based approach without Jacobian Augmentation. In this chapter I introduce an adaptation of the transfer-based approach.

5.1 Motivation

High memory expectation, poor results without Jacobian augmentation, and higher transferability of adversarial samples crafted in misclassification attacks than transferability of adversarial samples crafted in targeted misclassification attacks [LCLS16] motivated me to modify the transfer-based approach.

In the modified approach the substitute network has a lower number of classes than the black-box model is expected to have and the goal of an attack against the substitute network is not a targeted misclassification, but only a misclassification. I call the modified approach *the semi-targeted approach.*

In the semi targeted approach a substitute neural network has only several classes and every class represents a certain age interval. If a misclassification occurs in such a scenario, that means that the classifier is tricked at least for the amount of years corresponding to the age interval. In this thesis all age intervals have the same length, but in general this is not necessary.

Let me provide an example. Assume a substitute network with only three classes that represent age intervals 0-33, 34-66 and 67-99 years. Now if a person who is 50 years old gets misclassified, that means the person got classified as 0-33 years old or as 67-99 years old. In either case, the mistake is greater than getting classified as 51 or 49 years old as it would be the case when the substitute network would have 100 classes. Finally, if that

	clean samples		adversarial samples					
blackbox model id	blackbox MAE	substitute accuracy	blackbox MAE	substitute accuracy	avg L2	std dev L2	min L2	max L2
1	6.08	0.52	6.16	0.31	1891.38	56.60	1668.86	1939.89
2	6.99	0.56	8.40	0.28	1891.56	57.99	1577.01	1939.89
3	5.29	0.41	5.39	0.32	1890.05	61.06	1540.43	1939.89
4	3.75	0.54	3.92	0.30	1892.46	55.13	1666.18	1939.89

Table 5.1: Substitute network: ResNet50 architecture with SGD optimizer; Attack: FGSM

	clean samples		adversarial samples					
blackbox model id	blackbox MAE	substitute accuracy	blackbox MAE	substitute accuracy	avg L2	std dev L2	min L2	max L2
1	6.08	0.60	6.17	0.60	1856.08	271.20	0.00	1939.89
2	6.99	0.60	8.41	0.60	1871.71	153.83	0.00	1939.89
3	5.29	0.59	5.50	0.60	1213.41	911.17	0.00	1939.89
4	3.75	0.60	3.75	0.60	0.00	0.00	0.00	0.00

Table 5.2: Substitute network: ResNet50 architecture with Adam optimizer; Attack: FGSM

	clean samples		adversarial samples					
blackbox model id	blackbox MAE	substitute accuracy	blackbox MAE	substitute accuracy	avg L2	std dev L2	min L2	max L2
1	6.08	0.67	6.00	0.37	2525.36	76.90	2102.57	2589.41
2	6.99	0.68	9.25	0.30	2525.20	76.96	2103.83	2589.41
3	5.29	0.69	7.75	0.35	2525.43	76.61	2108.77	2589.41
4	3.75	0.74	4.83	0.34	2525.48	76.57	2105.55	2589.41

Table 5.3: Substitute network: InceptionResNetV2 architecture with SGD optimizer; Attack: FGSM

adversarial sample transfers to targeted black-box network, then the black-box model will also have a large error.

5.2 Evaluation

In Section 4.3 it is shown that FGSM has better results and it is used in this approach. To make the semi-targeted approach and the transfer based approach comparable, hyperparameters of the FGSM attack as well as training and test samples are completely the same as in Section 4.3.

Jacobian Augmentation is performed for three iterations before executing an attack as described in Section 3.4 that describes the transfer based approach. I also tried with four iterations for Jacobian Augmentation, but the system crashed.

In my experiments, the substitute network recognizes four classes: 0-25, 26-50, 51-75, and 76-99 years old. The number of epochs used for training the substitute network is set to 40.

Since the substitute network is trained on four classes, accuracy is used as a measure for evaluation of the network. For the targeted black-box neural network, MAE is used as a measure. Results are presented in Tables 5.1, 5.2, 5.3, and 5.4.

	clean samples		adversarial samples					
blackbox model id	blackbox MAE	substitute accuracy	blackbox MAE	substitute accuracy	avg L2	std dev L2	min L2	max L2
1	6.08	0.07	6.08	0.07	0.00	0.00	0.00	0.00
2	6.99	0.06	7.5	20.54	2525.54	77.47	2102.78	2589.41

Table 5.4: Substitute network: InceptionResNetV2 architecture with Adam optimizer; Attack: FGSM

The experiment corresponding to Table 5.4 is not evaluated further since the substitute network did not manage to learn, i.e. the accuracy was under 10%.

From the results in Tables 5.1, 5.2, and 5.3 one can observe that although the substitute neural network achieves a decent accuracy (around 60%) and get attacked successfully (accuracy reduced to 30%), the adversarial samples do not transfer significantly. That means the substitute neural network did not manage to learn the same boundaries as the targeted neural network.

CHAPTER 6

Threats to Validity and Future Work

6.1 Threats to Validity

- **Decision:** Skipping the Jacobian-based Dataset Augmentation step in the transfer-based approach.
  - **Argumentation:** One could argue that without the Jacobian augmentation, the transfer based approach is an incomplete algorithm. Given the hardware resources[1], every time during the call for Jacobian augmentation, the process would get killed by the kernel due to memory consumption. The implementation of the Jacobian augmentation that is used is the offical one provided by the author [PFC+18]. I tried to contact the author [2], but even he couldn't find a solution [3]. However, I observed that when the number of potential classes is lower, the jacobian-based augmentation can be performed for the same network.
  - **Potential consequences:** Since the transfer-based approach is not evaluated using the hardware that would support a complete execution of the attack, the results differ from those reported when a complete execution of the attack is supported.
- **Decision:** Number of iterations for Jacobian-based Dataset Augmentations is too low in semi-targeted approach.

[1] Intel(R) Core(TM) i5-8500 2 CPU @ 3.00GHz, 16GB RAM, GeForce GTX 1080 8GB
[2] https://github.com/tensorflow/cleverhans/issues/974
[3] https://stackoverflow.com/questions/54580105/memory-consumption-of-jacobian-dataset-augmentation/54718059

  - **Argumentation:** Using the given resources, that are described above, it was not possible to perform more iterations.
  - **Potential consequences:** The results are probably not as good as they would be when more iterations would be executed.

- **Decision:** Not trying more than 100 000 iterations for CW attack in the whitebox experiment discussed in Section 4.2.

  - **Argumentation:** The attack using 100 000 iterations takes a week to finish given the hardware resources. I believe that the hardware resources used in this thesis are hardware resources that an average adversarial user might have at hand and I don't see a motivation to run the attack for one sample for more than one week.
  - **Potential consequences:** Given more iterations, an optimizer might do a better job.

6.2 Future Work

Reducing the complexity of Jacobian-based dataset augmentation would allow more people to experiment in this field without the need for too expensive resources. If the complexity can not be reduced, maybe another, memory efficient, heuristic can be used for obtaining boundaries of the targeted network. Eventually, this would allow attacks from cheap computers and raise the awareness about the security of neural networks.

If the resources allow, it would be interesting to evaluate the ensemble approach on this problem. A cluster can be constructed that would train different networks which would train based on the results of the black-box network. Then an adversarial sample for all of them would probably be adversarial against the black-box network as well.

It would be interesting to evaluate the black-box attacks presented in this thesis against the real world systems for age estimation that are publicly available. Microsoft Azure [4] is one such system which offers age estimation service. Before the evaluation against the real system, the experiments for black-box attacks should show better results.

[4] https://azure.microsoft.com

CHAPTER 7

Summary

This thesis describes several adversarial attacks, trains four classifiers for age estimation and presents the results of the adversarial attacks against them in a white-box and a black-box scenario.

Three white-box attacks are evaluated. The JSMA attack was not able to craft a single adversarial sample because image dimensions were too large. This is confirmation of the result presented in [CW16].

The FGSM attack presents the best performance by successfully increasing the mean absolute error of one of the classifiers from 9.5 years to 48 years! Depending on the allowed perturbation in the image, adversarial samples are sometimes imperceptible from the original samples.

The CW attack is sometimes successful, but sometimes it fails to find an adversarial sample for a given neural network entirely. This optimization-based approach is significantly slower than the FGSM attack, but not necessarily more successful. It does not seem to be easy to find correct hyperparameters that should be used in this attack.

Two existing black-box approaches are evaluated. The Boundary attack is successful for every sample, but the cost is a high number of queries for every sample to the targeted black-box network. Around 16 000 queries are needed for one sample to become adversarial and adversary has to have two images to execute the attack.

The transfer-based approach trains a substitute network, attacks the substitute network and the adversarial samples then transfer to the targeted black-box network up to a certain extent. However, training of the substitute network such that the decision boundaries are similar as in the targeted network does not seem to scale up successfully with higher number of classes.

A new black-box attack, the semi-targeted approach, is introduced that is based on the transfer based approach.

Instead of having a substitute network with the same number of classes as the targeted network, the number of classes is reduced. If a sample is misclassified by the substitute network and misclassification transfers to the targeted network, the range of misclassification by the targeted network will be greater than if the substitute network would have the same number of classes as the targeted network. This approach can be used whenever labels can be ordered or even clustered. Results show that there is still a place for improvement in this approach.

Bibliography

[AM18] Naveed Akhtar and Ajmal Mian. Threat of adversarial attacks on deep learning in computer vision: A survey. *CoRR*, abs/1801.00553, 2018.

[B18] Wieland Brendel *, Jonas Rauber *, and Matthias Bethge. Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. In *International Conference on Learning Representations*, 2018.

[CW16] Nicholas Carlini and David A. Wagner. Towards evaluating the robustness of neural networks. *CoRR*, abs/1608.04644, 2016.

[DDS+09] 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.

[EA17] S Escalera X Baro I Guyon R Rothe. E Agustsson, R Timofte. Apparent and real age estimation in still images with deep residual regressors on appa-real database. In *12th IEEE International Conference and Workshops on Automatic Face and Gesture Recognition (FG), 2017*. IEEE, 2017.

[FIS] R. A. FISHER. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(2):179–188.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[GSS15] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*, 2015.

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[KB14] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[Kin09] Davis E. King. Dlib-ml: A machine learning toolkit. *Journal of Machine Learning Research*, 10:1755–1758, 2009.

[KNH] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research).

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[LC10] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[LCLS16] Yanpei Liu, Xinyun Chen, Chang Liu, and Dawn Song. Delving into transferable adversarial examples and black-box attacks. *CoRR*, abs/1611.02770, 2016.

[Mit97] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.

[PFC+18] Nicolas Papernot, Fartash Faghri, Nicholas Carlini, Ian Goodfellow, Reuben Feinman, Alexey Kurakin, Cihang Xie, Yash Sharma, Tom Brown, Aurko Roy, Alexander Matyasko, Vahid Behzadan, Karen Hambardzumyan, Zhishuai Zhang, Yi-Lin Juang, Zhi Li, Ryan Sheatsley, Abhibhav Garg, Jonathan Uesato, Willi Gierke, Yinpeng Dong, David Berthelot, Paul Hendricks, Jonas Rauber, and Rujun Long. Technical report on the cleverhans v2.1.0 adversarial examples library. *arXiv preprint arXiv:1610.00768*, 2018.

[PMG+16] Nicolas Papernot, Patrick D. McDaniel, Ian J. Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. Practical black-box attacks against deep learning systems using adversarial examples. *CoRR*, abs/1602.02697, 2016.

[PMJ+15] Nicolas Papernot, Patrick D. McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. *CoRR*, abs/1511.07528, 2015.

[RHW86] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1. chapter Learning Internal Representations by Error Propagation, pages 318–362. MIT Press, Cambridge, MA, USA, 1986.

[RTG16] Rasmus Rothe, Radu Timofte, and Luc Van Gool. Deep expectation of real and apparent age from a single image without facial landmarks. *International Journal of Computer Vision (IJCV)*, July 2016.

[Rud16] Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016.

[SIV16] Christian Szegedy, Sergey Ioffe, and Vincent Vanhoucke. Inception-v4, inception-resnet and the impact of residual connections on learning. *CoRR*, abs/1602.07261, 2016.

[SSSI12a] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural Networks*, (0):–, 2012.

[SSSI12b] Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural Networks*, 32:323–332, 2012.

[SZ14] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[SZS+13] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[YCBL14] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328, 2014.