FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

German title

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Logic and Computation

eingereicht von

Pretitle Martin Matak, Posttitle
Matrikelnummer e1638889

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Pretitle Georg Weissenbacher, Posttitle

Wien, 1. Jänner 2001

Martin Matak	Georg Weissenbacher

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Attacks on Neural Networks

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Logic and Computation

by

Pretitle Martin Matak, Posttitle
Registration Number e1638889

to the Faculty of Informatics

at the TU Wien

Advisor: Pretitle Georg Weissenbacher, Posttitle

Vienna, 1st January, 2001

Martin Matak	Georg Weissenbacher

Erklärung zur Verfassung der Arbeit

Pretitle Martin Matak, Posttitle
Address

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, 1. Jänner 2001

Martin Matak

Acknowledgements

TODO

Kurzfassung

TODO

Abstract

TODO

Contents

Kurzfassung **ix**

Abstract **xi**

Contents **xiii**

1 Introduction **1**
1.1 Problem Definition 1
1.2 Aim of the Work 2
1.3 Methodological Approach 4
1.4 Outline 4

2 Background **5**
2.1 Feedforward neural networks 7
2.2 Gradient descent 9
2.3 Backpropagation 12
2.4 Convolutional neural networks 12

3 State-of-the-Art **15**
3.1 Fast Gradient Sign Method (FGSM) 16
3.2 Jacobian-based Saliency Map Attack (JSMA) 16
3.3 Carlini & Wagner (CW) 18
3.4 Transfer based approach 20
3.5 Ensemble approach 21
3.6 Boundary attack 21

4 Experiments **23**
4.1 Methodology 23
4.2 Whitebox attacks 26
4.3 Blackbox attacks 26

Bibliography **31**

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 industry 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 several 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.

TODO: Add more interesting examples

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.

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 no particular order, are the following:

- **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**; and
  - 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.
- **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.

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 from a specific group of labels, I define that attack as *semi-targeted misclassification*. Of course, a group of labels must be smaller than a whole set of possible labels, 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 with targeted black-box misclassification attacks. This comparison is explained in more details in Section 1.3.

2. **adversarial algorithms are evaluated in the different environments**; and

   - 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 the different settings and 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 trained a deep neural network for age estimation of a person in the image.

2. I conducted 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 did a literature survey on the targeted black-box attack methods, explain those methods and use them to generate adversarial inputs for a 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 implemented a new adversarial algorithm and using that algorithm, I construct images for semi-targeted misclassification. I compare results against targeted black-box approaches. The way I is compared them is the following: a target label for a targeted version of the attack is the median value of the set of labels in a semi-targeted version of the attack. If a DNN outputs any label outside the set of the labels, an attack is considered as a successful no matter if it is a targeted or a semi-targeted misclassification attack.

1.4 Outline

TODO

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, an image classification is a *task* of assigning a *label* or a *class* to an image. If an input is 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 must not 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 good 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, respectfully.

For every sample in a dataset, an information is provided that defines what is a class of that sample. 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 instance of some specific class or there is 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

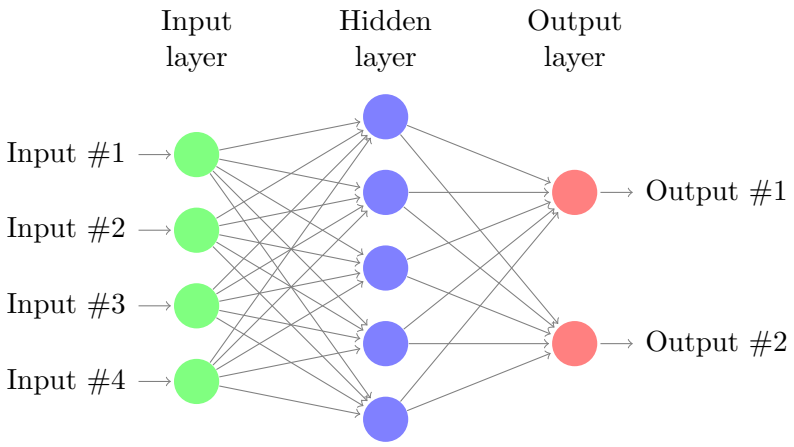


Figure 2.1: An example of the feedforward neural network architecture

repeated k times, with each of the k subsets used exactly once as the validation data. The k results can then be averaged to produce a 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 the 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 the architecture of the neural network when it comes to the computer vision domain.

2.1 Feedforward neural networks

Feedforward neural networks or *multilayer peceptrons* (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 an input layer) defines a *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.

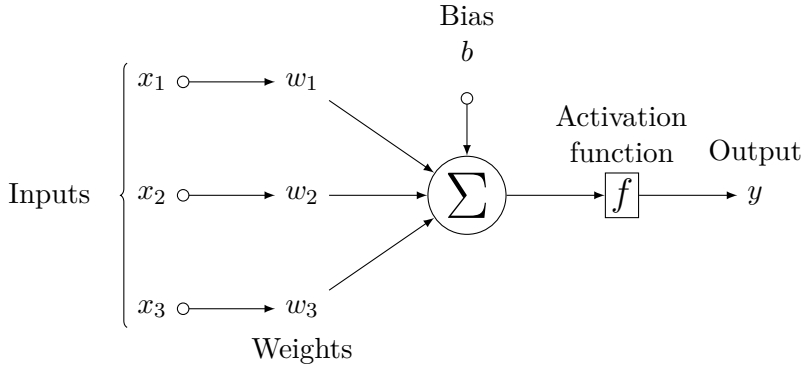


Figure 2.2: A single processing unit in a neural network

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 in the other layers those 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 the 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. Sum of the probabilities for all the 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 as a 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 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 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 output of that function, i.e. loss, is minimal. Most machine learning algorithms have some kind of optimization built in. Optimization refers to the task of find $\boldsymbol{x}$ 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, an assumption is that a reader is familiar with the basics of linear algebra, differential calculus, probability theory and statistics. If not, please consult [GBC16].

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 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 on 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} denotes 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
while true do
    θ' ← ∇L(θ);
    if ||θ'|| ≈ 0 then
        return;
    end
    θ ← θ − α * θ';
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 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 *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 Network is 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. 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.

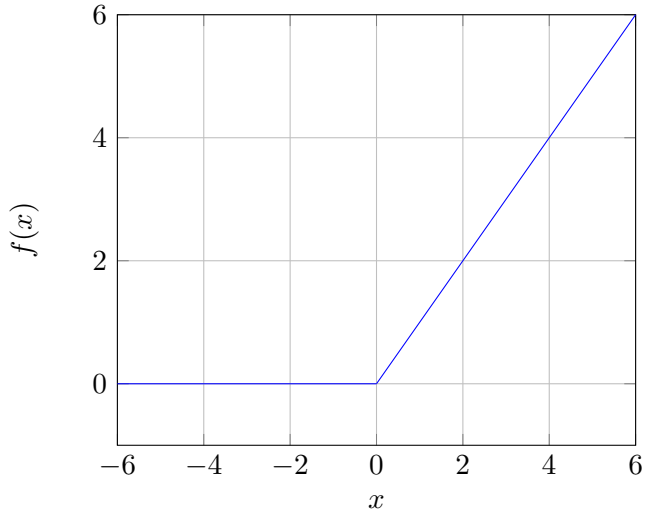


Figure 2.3: ReLU function

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].

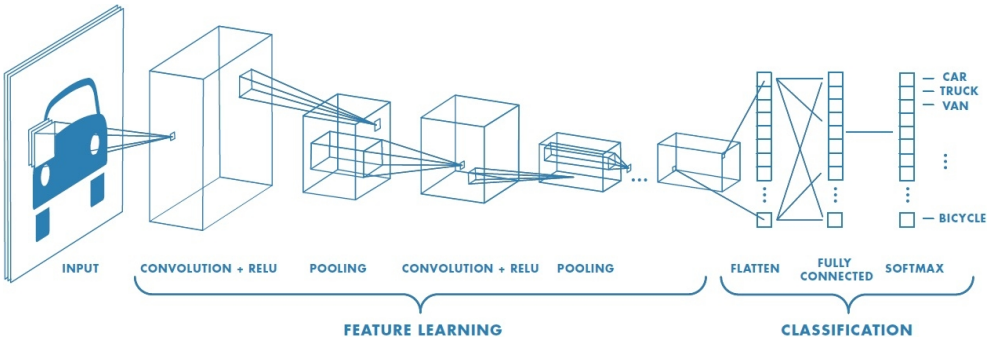


Figure 2.4: A simple deep neural network, image taken from `floydhub.com`

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 used 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] was 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. As soon as misclassification occurs (any label different than the original one is predicted, not necessarily the target label), the attack is finished.

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, please consult 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.

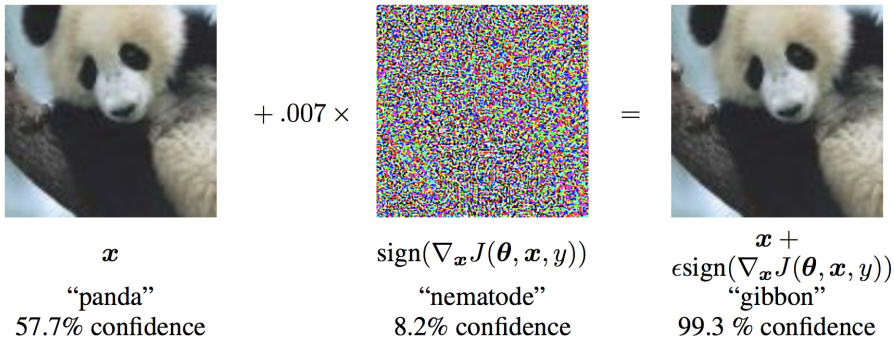


Figure 3.1: Image taken from the [GSS15]

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 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.

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. Then the adversarial saliency map in terms of pair pixels p, q is defined as:

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

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

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 $-\alpha_{pq} * \beta_{pq}$ is as large as possible. This can be formalized as:

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

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:

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

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

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

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:

$$f(\boldsymbol{x}') = max(max(\{\boldsymbol{Z}(\boldsymbol{x}')_i : i \neq t\}) - \boldsymbol{Z}(\boldsymbol{x}')_t, 0). \tag{3.2}$$

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]

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

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

[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 $\mathbf{1}_p$.

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

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 [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 boundary attack with 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 methods proposed in Chapter 3 on their effectiveness in producing adversarial examples.

First, I present 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 train, 1500 valid and 1978 test images. For each image in the dataset, there is also the corresponding image which contains the cropped face. 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 a face[2] from it. For face detection, I used Dlib [Kin09] library. Distribution of samples over age is presented in Figure 4.1.

In constructed my training dataset as union of all the 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

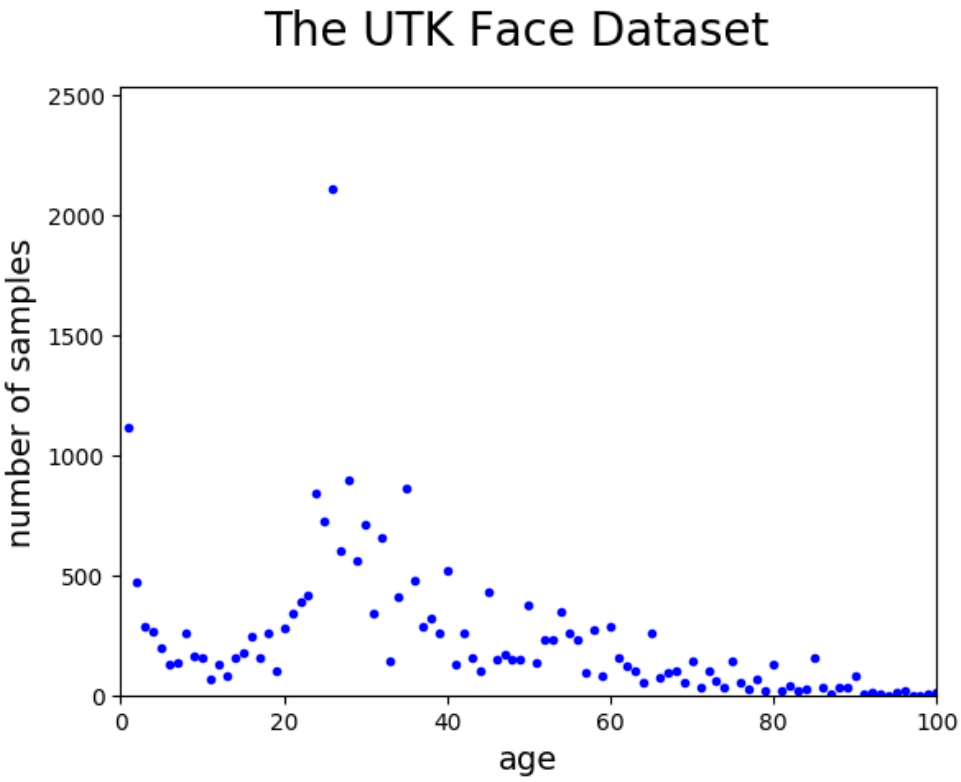


Figure 4.1: Number of samples per age in the UTK Face Dataset

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].

Instead of measuring accuracy of a classifier, in the age estimation domain usually a mean absolute error is measured. 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 intuition why mean absolute error is used instead of accuracy as a metric.

When it comes to (targeted) misclassification, there is something in 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

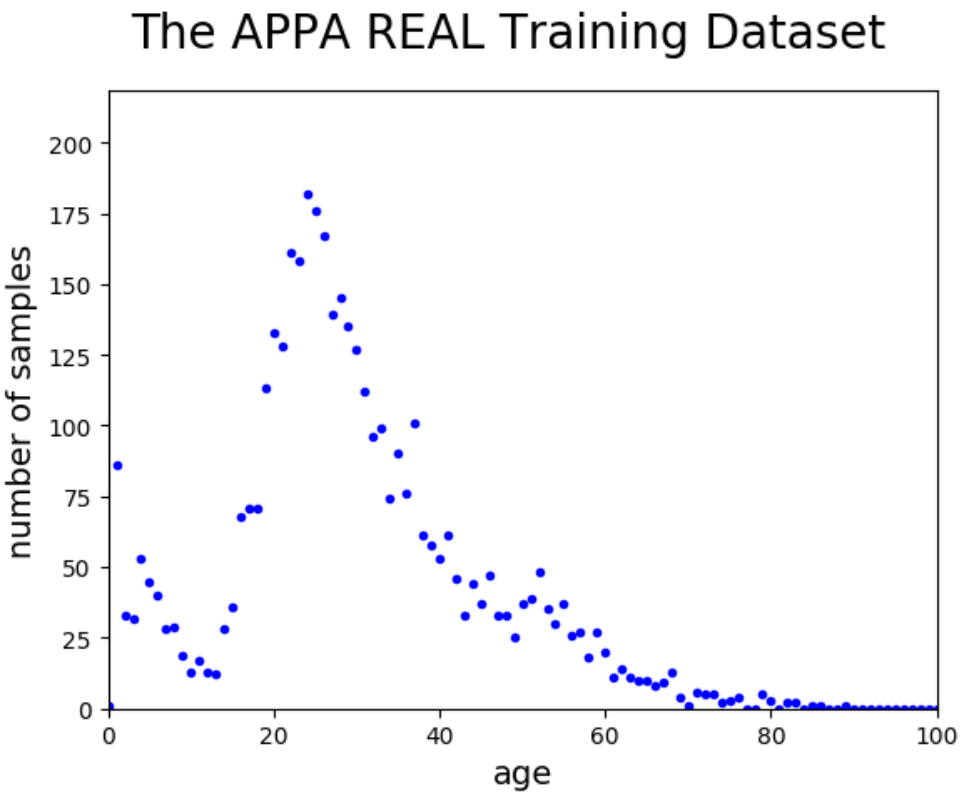


Figure 4.2: Number of samples per age in the APPA-REAL Training Dataset

years old.

The idea behind such a setting is to maximize mean average error, i.e. make classificator 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]. Two different DNN architectures are used for training: ResNet-50 [HZRS15] and VGG16 [SZ14]. Their performance is presented in Table 4.1.

For evaluation, 100 random samples are taken from the APPA-REAL test dataset. Neither of models from Table 4.1 has never seen any of these samples before.

Id	Architecture	Optimizer	Validation Loss	**Validation MAE**
1	ResNet-50	SGD	3.436	5.151
2	ResNet-50	Adam	3.456	6.772
3	VGG16	SGD	16.603	11.255
4	VGG16	Adam	15.494	11.685

Table 4.1: Different models trained

		FGSM		CW		JSMA	
model id	clean MAE	adv MAE	avg L2	adv MAE	avg L2	adv MAE	avg L2
1	6.21	29.89	1879.63	19.73	195.49	-	-
2	8.61	41.48	1879.49	8.61	0.0	-	-
3	12.68	12.68	1910.48	12.68	0.0	-	-
4	15.14	15.14	1900.47	15.14	0.0	-	-

Table 4.2: Results of different adversarial attacks. The "-" sign means that an attack couldn't be executed.

4.2 Whitebox attacks

When it comes to whitebox attacks, three approaches are evaluated: FGSM, CW and JSMA. FGSM is used with $eps = 5$ (the parameter that sets how large the perturbation may be). Number of iterations in CW is bounded to 1000. In Table 4.2 results are presented. The experiment for the CW attack with 10000 iterations yielded very similar results as with 1000 iterations. The results are discussed in the next chapter.**TODO: reference to the next chapter**.

4.3 Blackbox attacks

When it comes to blackbox attacks, two existing approaches are evaluated: the Transfer based approach and the Boundary attack.

In the transfer based approach, I take 1000 samples that are previously unseen to target neural network and obtain labels for them by querying the target neural network. Then a substitute network that is pretrained on the ImageNet dataset is trained on those 1000 samples with labels that are obtained from the target network. The idea of such a process is to achieve the similar decision boundaries for the substitute network as they are in the targeted network.

Next, I take random 100 samples that are yet unseen by both networks and evaluate MAE of the targeted neural network and the substitute network on those 100 images. Then I craft adversarial samples for the substitute neural network using those 100 images. Target labels for adversarial samples are set in the same manner as in whitebox attacks.

	Clean Samples		Adversarial samples	
Target model id	Blackbox MAE	Substitute MAE	Blackbox MAE	Substitute MAE
1	6.21	14.17	6.61	14.20
2	8.61	13.20	8.17	13.39

Table 4.3: Results using the ResNet-50 architecture for the substitute network and FGSM attack for crafting adversarial samples for it

	Clean Samples		Adversarial samples	
Target model id	Blackbox MAE	Substitute MAE	Blackbox MAE	Substitute MAE
1				
2				

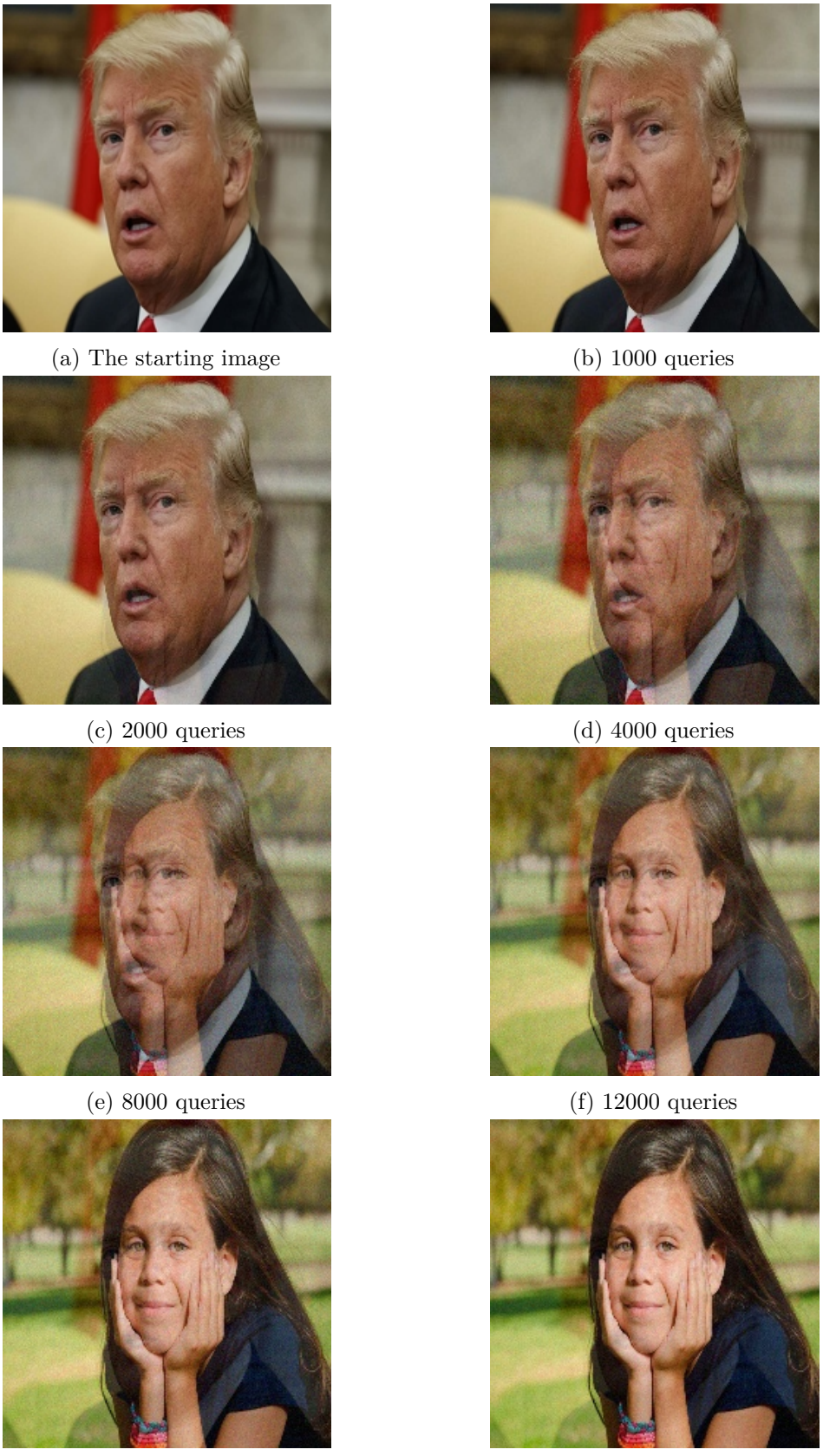
Table 4.4: Results using the ResNet-50 architecture for the substitute network and CW attack for crafting adversarial samples for it

Finally, I evaluate MAE of the targeted neural network and the substitute network on those 100 adversarial samples.

From Table 4.1 it can be observed that the VGG16 architecture doesn't perform as good as the ResNet-50 architecture. Since the idea behind the attacks is to trick a classifier that performs good, in this section only attacks against the ResNet-50 architecture are considered.

Results are presented in Tables 4.3 and 4.4.

When it comes to the boundary attack, it's hard to do any quantitative analysis because because the attack, as described in Section 3.6, is fundamentally different that the transfer-based approach. Instead, I generate several samples and discuss them. One sample can be seen in Figure 4.3 and the corresponding results in 4.4. The other samples can be found in Appendix. discussion follows in the chapter TODO.



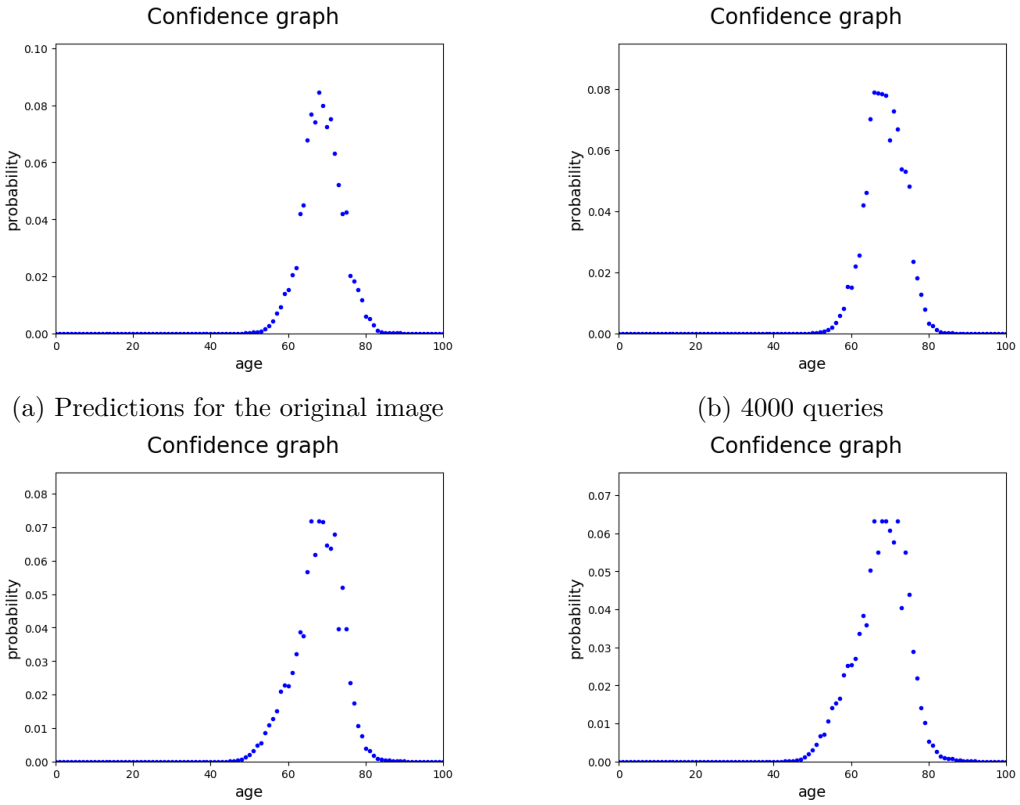
(a) The starting image (b) 1000 queries

(c) 2000 queries (d) 4000 queries

(e) 8000 queries (f) 12000 queries

(g) 16000 queries (h) Final adversarial sample

Figure 4.3: Although an adversary is changing the image, the blackbox classifier is not changing the prediction (68 years old)



(a) Predictions for the original image

(b) 4000 queries

(c) 16000 queries

(d) Predictions for the final adversarial sample

Figure 4.4: Predictions of the blackbox classifier for images corresponding to Figure 4.3

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.

[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.

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