

# Bilderzeugungsverfahren zur Überlistung einer Verkehrsschilder erkennenden KI

## Ausarbeitung

Master-Studiengang *Informatik*

Technische Hochschule Georg Simon Ohm

von

Leonhard Applis, Peter Bauer, Andreas Porada und Florian Stöckl

Abgabedatum: 15.01.2019

# Abstract

This paper uses several scientific approaches to show how convolutional neural networks can be tricked into recognizing and correctly classifying road signs. In the field of autonomous driving, neural networks have drastically increased the recognition rate, but they are still susceptible to errors in the face of deliberately generated false images. Even without information about the underlying architecture (so-called black box attacks), attacks by otherwise generated false images should be possible. The presented methods *Degeneration*, *Saliency Maps* and *Gradient Ascent* are successfully applied to generate false images for attacks on an unknown neural network with the help of an own neural network which serves as *Substitute*. An attack is considered “successful” if the image is recognized as a street sign with a confidence greater than 90%, which a human observer would not recognize as such.

**title:** Fooling a TrafficSign-AI

**authors:** Leonhard Applis, Peter Bauer, Andreas Porada und Florian Stöckl

# Kurzfassung

Diese Arbeit zeigt anhand von mehreren wissenschaftlichen Ansätzen, wie Convolutional Neural Networks zur Erkennung und korrekten Klassifikation von Straßenschildern überlistet werden können. Im Bereich des Autonomen Fahren wurde mit Neuronalen Netzen die Erkennungsrate drastisch gesteigert, allerdings sind diese immer noch fehleranfällig gegenüber gezielt erzeugten Irrbildern. Selbst ohne Informationen über die unterliegende Architektur (sog. Black Box Angriffe), sollen Angriffe durch anderweitig erzeugte Irrbilder möglich sein. Die vorgestellten Verfahren *Degeneration*, *Saliency Maps* und *Gradient Ascent* werden erfolgreich angewendet, um mithilfe eines eigenen Neuronalen Netz, welches als *Substitute* dient, Irrbilder für Angriffe auf ein unbekanntes Neuronales Netz zu erzeugen. Ein Angriff gilt als "erfolgreich", wenn das Bild mit einer Konfidenz größer als 90% als Straßenschild erkannt wird, welches ein menschlicher Betrachter nicht als solches erkennen würde.

Titel: Bilderzeugungsverfahren zur Überlistung einer Verkehrsschilder  
erkennenden KI

Autoren: Leonhard Applis, Peter Bauer, Andreas Porada und Florian Stöckl

# Inhaltsverzeichnis

<b>Abbildungsverzeichnis</b>	<b>V</b>
<b>Abkürzungsverzeichnis</b>	<b>1</b>
<b>1 Einleitung</b>	<b>2</b>
1.1 Problemstellung . . . . .	3
1.2 Ziel der Arbeit . . . . .	4
1.3 Aufbau der Arbeit . . . . .	4
<b>2 Anforderungsanalyse und Rahmenbedingungen</b>	<b>7</b>
2.1 Funktionale Anforderungen . . . . .	7
2.2 Nichtfunktionale Anforderungen . . . . .	7
2.3 Eigenschaften des Neuronalen Netzes des GI Wettbewerbs . . . . .	7
<b>3 Technisches Konzept</b>	<b>9</b>
3.1 Verwendete Technologien . . . . .	9
3.2 Transferierbarkeit von Adversarial Attacks . . . . .	10
3.3 Implementierung eines eigenen Modells zur Klassifizierung von Straßenschildern (Aphrodite) . . . . .	11
<b>4 Degeneration</b>	<b>13</b>
4.1 Konzept . . . . .	13
4.2 Implementierung Remote . . . . .	17
4.3 Ergebnisse Remote . . . . .	19
4.4 Implementierung Lokal . . . . .	23
4.4.1 Batch-Degeneration . . . . .	25
4.4.2 Parallel-Degeneration . . . . .	26
4.4.3 Tree-Degeneration . . . . .	27

<b>5</b>	<b>Saliency Maps</b>	<b>29</b>
5.1	Konzept . . . . .	29
5.2	Implementierung . . . . .	30
5.3	Ergebnisse . . . . .	31
<b>6</b>	<b>Gradient Ascent</b>	<b>35</b>
6.1	Konzept . . . . .	35
6.2	Implementierung . . . . .	36
6.3	Ergebnisse . . . . .	37
<b>7</b>	<b>Fazit</b>	<b>39</b>
7.1	Zusammenfassung . . . . .	39
7.2	Diskussion . . . . .	40
7.3	Weiterführende Arbeiten . . . . .	40
	<b>Literaturverzeichnis</b>	<b>42</b>

# Abbildungsverzeichnis

4.1	Degeneration Tiefe 600 . . . . .	19
4.2	Degeneration Tiefe 4000 . . . . .	20
4.3	Plot Degeneration . . . . .	20
4.4	Degeneration overfit . . . . .	21
4.5	Batch-Degeneration-Plot . . . . .	25
4.6	Tree-Degeneration . . . . .	28

# Abkürzungsverzeichnis

<b>SQL</b>	Structured Query Language
<b>KI</b>	Künstliche Intelligenz
<b>NN</b>	Neurones Netz
<b>DNN</b>	Deep Neural-Network
<b>CNN</b>	Convolutional Neural-Network
<b>GTSRB</b>	German Traffic Sign Recognition Benchmark
<b>PPM</b>	Portable Pixmap Image
<b>PNG</b>	Portable Network Graphic
<b>SGD</b>	Stochastic-Gradient-Descent
<b>GI</b>	Gesellschaft für Informatik
<b>SVM</b>	Support-Vector-Machine

# 1 Einleitung

Der Traum von einem autonom gelenkten Automobil ist so alt wie das Automobil selbst [14]. Fabian Dröger schreibt dazu in seinem Beitrag „Das automatisierte Fahren im gesellschaftswissenschaftlichen und kulturwissenschaftlichen Kontext“ in dem Sammelwerk „Autonomes Fahren“ von Markus Maurer et al., wie die zunehmende Anzahl an Verkehrstoten in den USA zu Beginn des 20. Jahrhunderts in Verbindung mit den technischen Errungenschaften in der frühen Flugzeug- und Radiotechnik den Wunsch nach einem selbstfahrenden Automobil aufkommen ließen. Die Vision war, dass ein Automobil ähnlich wie ein Flugzeug durch einen Autopiloten in der Spur gehalten und gesteuert werden könnte. Für die Ansteuerung der mechanischen Teile setzte man auf eine Fernsteuerung mit Funk, die zu dieser Zeit im Bereich der *Radioguidance* erforscht wurde.

Der aktuelle Stand der Technik zeigt, dass sich die Umsetzung dieser Vision schwieriger gestaltet als zunächst angenommen. Anstelle einer autonomen Steuerung finden sich in heutigen Automobilen verschiedene Techniken zur Erhöhung der Fahrsicherheit und des Komforts. Beispiele sind Spurhalteassistenten, automatische Abstandshalter oder Einparkhilfen. Diese Funktionen unterstützen einen menschlichen Fahrer, ermöglichen jedoch noch kein selbstständiges Fahren.

Es wird aber weiterhin an der Entwicklung eines autonomen Fahrzeugs geforscht, wie die Vergabe von Forschungsgeldern [4] und Berichte von Automobilherstellern [5] und der Presse [7] zeigen. Die Forschung im Bereich Künstliche Intelligenz (KI) hat mittlerweile einen Stand erreicht, der für die Automatisierung des Autos genutzt werden kann. Ein besonderer Fokus liegt hierbei auf der automatischen Erkennung von Bildern aus der Umwelt mittels einer KI. Im Straßenverkehr ist besonders die Erkennung von Straßenschildern von Bedeutung.



## 1.1 Problemstellung

Bei der Untersuchung von **KIs**, welche auf die Erkennung gewisser Muster mit einem Datensatz trainiert wurden, fiel auf, dass auch Muster, die für den Menschen in keinem erkennbaren Zusammenhang mit dem Trainingsmuster stehen, von der **KI** mit hoher Konfidenz anerkannt werden. Das gezielte Ausnutzen dieses Fehlers wird in der Forschung als *Adversarial Attack* bezeichnet [11].

Der Erfolg von *Adversarial Attacks* liegt in der gezielten Stimulation von Gewichten, die die **KI** basierend auf ihrem Trainingssatz festgelegt hat, beziehungsweise die vom Trainer definiert wurden. Auf diese Weise ist es möglich ein gewünschtes Feedback eines **NNs** zu erhalten, obwohl das Muster für einen Menschen kaum oder gar nicht mit den Mustern des Trainingssatzes in Verbindung gebracht werden kann. Zum Beispiel haben erzeugte Fragmente zur Täuschung einer **KI**, die für die Erkennung von Bildinhalten trainiert ist, selten etwas mit einem *echten* Bild zu tun. Sie wirken eher wie Rauschen oder moderne Kunst.

Das Problem an den präparierten Mustern einer *Adversarial Attack* ist, dass ein Mensch schwer erkennen kann, ob ein derartiger Angriff auf die von ihm genutzte **KI** unternommen wird. Er ordnet das manipulierte Muster nicht den relevanten Mustern zu, die seine **KI** anerkennt.

Durch den steigenden Einsatz von Machine Learning in verschiedenen sensiblen Sektoren des täglichen Lebens, wie selbstfahrenden Autos, Terrorismusbekämpfung oder Betrugserkennung können Angriffe verheerende Schäden erzeugen und stellen ein lohnendes Ziel dar. Vor allem im Bereich des autonomen Fahrens, welcher aktuell geprägt ist durch eine Debatte über das *Vertrauen in Technik* [6], können erfolgreiche Angriffe zu einem forschungsschädlichen Misstrauen führen und die Nutzung der Technik verhindern.

Um gegen *Adversarial Attacks* vorzugehen, werden zunächst einige Bilder benötigt, die gezielt Gewichte einer **KI** stimulieren und mit hoher Konfidenz anerkannt werden.

## 1.2 Ziel der Arbeit

Ziel dieser Arbeit ist es, Methoden und Herangehensweisen vorzustellen, mit denen eine KI überlistet werden kann, die Straßenschilder auf Bildern erkennt. Die Ausrichtung dieser Arbeit orientiert sich an der Aufgabenstellung des InformatiCups<sup>1</sup> der Gesellschaft für Informatik (GI) e.V. aus dem Jahr 2019. Der InformatiCup ist ein Wettbewerb der GI für Studierende, bei dem sich diese in neue Technologien einarbeiten und in Teams Problemlösungen entwickeln [8].

Bei der Aufgabe 2019 soll ein Neuronales Netz (NN), welches sich hinter einer Webschnittstelle verbirgt und Verkehrsschilder erkennt, erfolgreich überlistet werden. Dazu sollen Bilder erzeugt werden, welche für den Menschen nicht als Verkehrsschild erkennbar sind, aber mit einer Konfidenz von über 90% von der KI als solche erkannt werden.

Die gefundenen Methoden sollen reproduzierbar und in einem Maße flexibel sein, um beliebig viele Irrbilder zu erzeugen.

Die Arbeit umfasst eine Dokumentation verschiedener Methoden sowie Verbesserungen und Schlussfolgerungen aus den Implementierungen.

Ebenfalls geliefert werden alle Elemente, um die erzielten Ergebnisse zu reproduzieren und zu variieren.

Nicht Ziel dieser Arbeit ist, einen Überblick über neuronale Netze, künstliche Intelligenz oder Bildbearbeitung zu vermitteln.

Ebenfalls außerhalb dieser Arbeit liegt eine Auswertung, welche Bilder von einem Menschen als Verkehrsschilder erkannt werden. Die Aussagen über solche stützen sich ausschließlich auf die persönliche Einschätzung des Projektteams.

## 1.3 Aufbau der Arbeit

Innerhalb dieser Arbeit werden zunächst in Kapitel 2 Anforderungen analysiert, die Lösungen für die unter 1.1 genannten Probleme erfüllen sollen, sowie die Rah-

---

<sup>1</sup><https://gi.de/informaticup/>

menbedingungen der Arbeit festgelegt.

Eine zentrale Rahmenbedingung stellt die Webschnittstelle des Wettbewerbs dar. Um Informationen über die Webschnittstelle zu sammeln, werden alle verfügbaren Quellen des Wettbewerbs genutzt, sowie eigene Untersuchungen durchgeführt. Eine davon ist die Analyse des German Traffic Sign Recognition Benchmark ([GTSRB](#)) in Abschnitt [2.3](#), dem Trainingsdatensatz, der für die [KI](#) hinter der Webschnittstelle verwendet wurde. Dieses Datenset bildet ebenfalls einen zentralen Ausgangspunkt für einige der verfolgten Ansätze der Arbeit.

Anschließend werden verschiedene Lösungsansätze vorgestellt, beginnend mit der *Degeneration* in Kapitel [4](#).

Dieser Ansatz verändert iterativ ein Verkehrsschild und behält die Änderungen bei, sollte der erzielte Score im akzeptablen Bereich liegen. Mit passenden Bildveränderungen erzielen höhere Iterationen für den Menschen unkenntliche Ergebnisse, die von der [KI](#) weiterhin mit hoher Konfidenz der Klasse des ursprünglichen Bildes zugeordnet wird.

Innerhalb des Kapitels wird zunächst im Abschnitt [4.1](#) die Idee anhand von Pseudocode weiter erläutert und anschließend in Abschnitt [4.2](#) die Implementierung für die Kommunikation mit der Webschnittstelle des Wettbewerbs gezeigt. Die Ergebnisse liegen gesondert im Abschnitt [4.3](#) vor.

Neben der Implementierung für die Webschnittstelle werden zum Abschluss des Kapitels in Abschnitt [4.4](#) noch weitere Verbesserungen für eine lokale Implementierung vorgestellt, welche allerdings nicht für die Webschnittstelle tauglich sind, sondern nur als Ausblick dienen.

Des Weiteren werden in Kapitel [5](#) verschiedene Methoden zur Erzeugung von sogenannten *Saliency Maps* (dt. Ausprägungskarte) vorgestellt. Bei dieser Methode werden unveränderte Bilder mit einer hohen Konfidenz ausgewählt, um die einzelnen Pixel hervorzuheben, welche für die Klassifikation den höchsten Einfluss haben.

In Kapitel [6](#) wird das Gradient Ascent Verfahren beschrieben und evaluiert. Hierbei wird anhand der Targeted Backpropagation Methode zunächst ein Zufallsbild erzeugt und dieses so lange verändert, bis es der angegebenen [GTSRB](#)-Zielklasse

entspricht.

Abschluss dieser Arbeit bildet im Kapitel [7](#) ein Fazit über die gefundenen Methoden, sowie ein Ausblick auf weiterführende Arbeiten.

## 2 Anforderungsanalyse und Rahmenbedingungen

Im nachfolgenden Kapitel werden ausgehend von der Aufgabenstellung des InformatiCups 2019 Anforderungen an die Bilderzeugungsverfahren zur Überlistung einer Verkehrsschilder erkennenden KI analysiert. Die Anforderungen untergliedern sich dabei in funktionale Anforderungen und nichtfunktionale Anforderungen. Die analysierten Anforderungen werden durch die Rahmenbedingungen des Wettbewerbs ergänzt, die im wesentlichen aus dem bereitgestellten Neuronales Netz besteht.

### 2.1 Funktionale Anforderungen

Die Kernaufgabe der implementierten Lösungen der nachfolgenden Kapitel 4 bis 6 besteht darin Bilder zu generieren. Als Eingangsparameter dürfen dafür Bilder verwendet werden, die anschließend durch Algorithmen modifiziert werden.

Die implementierten Lösungen müssen nachvollziehbar und reproduzierbar sein. Das bedeutet, dass eine Modifikation mit den selben Eingangsparametern zu dem selben Ergebnis führen muss. Grundvoraussetzung für diese Anforderung

### 2.2 Nichtfunktionale Anforderungen

### 2.3 Eigenschaften des Neuronales Netzes des GI Wettbewerbs

33 verschiedene aufgezeichnete klassenlabels (im vergleich GTSRB datensatz 43)

1. Aus der aufgabenstellung 64x64x3
2. bilder aus dem GTSRB [quelle] datensatz
3. Gekürzte Klassen: Aus der analyse geht die Vermutung hervor, dass nur 33 Klassen unterschieden werden, keine 43 wie im original datensatz
4. Softmax-Ausgabefunktion
5. Interpolationsfunktion (vllt mit einem Bild in 3 Interpolationsversionen und jeweiligen Score)
6. Overfitting bei Trainingsdaten
7. unzuverlässigkeit bei nicht-Schildern (z.B. OhmLogo)

# 3 Technisches Konzept

## 3.1 Verwendete Technologien

Die Umsetzung der Implementierung erfolgt innerhalb der webbasierten, interaktiven Entwicklungsumgebung Jupyter Notebook <sup>1</sup> (in der Version 5.7.4) zusammen mit der objektorientierten höheren Programmiersprache Python <sup>2</sup> (in der Version 3.6.5).

Jupyter Notebook bietet aufgrund seiner plattformübergreifenden Einsatzmöglichkeit und Kompatibilität zu allen gängigen Webbrowsern eine hohe Flexibilität, was die Darstellung und Ausführung von Python-Code angeht. Darüber hinaus bietet Python eine hohe Verfügbarkeit von Open-Source-Repositories im Bereich Datenverarbeitung, Machine Learning und Deep Learning ???. Die Programmiersprache wurde ferner im Rahmen der StackOverflow Befragung 2017 von den befragten Softwareentwicklern zur fünftbeliebtesten Technologie des Jahres 2017 gewählt ???. Nicht zuletzt ist Python und die inbegriffenen umfangreichen Standardbibliotheken auf allen gängigen Plattformen, wie beispielsweise Linux, Apple MacOS und Microsoft Windows, kostenlos und in Quell- oder Binärform verfügbar ??.

Als Paketmanager wird die frei verfügbare Anaconda Distribution in der derzeit aktuellsten Version 2018.12 gewählt, da sie eine vereinfachte Paketinstallation und -verwaltung ermöglicht. Darüber hinaus bietet Anaconda die Möglichkeit Jupyter Notebooks sowie Python und dessen verfügbare Pakete in verschiedenen Entwicklungs- und Testumgebungen isoliert voneinander zu verwalten und zu betreiben ???. Schließlich erlaubt "Anaconda Accelerate" den programmatischen Zugriff auf numerische Softwarebibliotheken zur beschleunigten Codeausführung auf Intel Prozessoren sowie NVIDIA Grafikkarten ??.

---

<sup>1</sup><https://jupyter.org/>

<sup>2</sup><https://www.python.org/>

Name	Version	Beschreibung
Keras	2.2.4	Enthält Funktionen für Deep-Learning Anwendungen [7]
Torchvision	0.2.1	Enthält Datensätze, Modellarchitekturen und gängige Bildtransaktionsoperationen für Computer-Vision Anwendungen [8]
OpenCV	3.4.2	Enthält Funktionen für echtzeit Computer-Vision Anwendungen [9]
NumPy	1.15.3	Enthält Funktionen zur effizienten Durchführung von Vektor- oder Matrizenberechnungen [10]
Requests	2.18.4	Enthält Funktionen zur Vereinfachung von HTTP Requests [11]
Pillow	5.2.0	Enthält Funktionen zum laden, modifizieren und speichern von verschiedenen Bilddateiformaten [12]
Matplotlib	2.2.3	Enthält Funktionen zum Plotten von Graphen oder Bildern [13]
SciPy	1.1.0	Enthält wissenschaftliche und technische Funktionen zur Datenverarbeitung [14]

Tabelle 3.1: Paketabhängigkeiten der implementierten Software

Zur fehlerfreien Ausführung des Codes der Implementierungen in den nachfolgenden Kapiteln muss Python in der Version 3.6.5 verwendet werden. Weiterhin bestehen Abhängigkeiten zu den Bibliotheken, welche in Tabelle 3.1 mit den dazugehörigen Versionen angegeben sind. Durch den Einsatz von Anaconda kann eine Environment erstellt werden, in der die passenden Versionen installiert werden.

Um die Voraussetzungen zur benötigten Python Version respektive der erforderlichen Python-Bibliotheken zu erfüllen, muss beim ersten Öffnen des Jupyter Notebooks zum Saliency Map Verfahren beziehungsweise zum Gradient Ascent Verfahren immer zuerst der Code unter der Rubrik “Managing Anaconda Environment” ausgeführt werden. Andernfalls kann die korrekte Ausführung von weiteren Teilen des Codes in nachfolgenden Rubriken nicht gewährleistet werden.

## 3.2 Transferierbarkeit von Adversarial Attacks

Eine Herausforderung des Wettbewerbs ist der Umgang mit dem neuronalen Netz, welches getäuscht werden soll. Die Analyse der Web-Schnittstelle in Abschnitt 2.3



liefert keine genauen Informationen über das zugrunde liegende Modell, was deren Architektur oder weitere Eigenschaften angeht. Dennoch wurde bereits bestätigt, dass es selbst für „Black Box“ Modelle möglich ist, Irrbeispiele und Bilder zu erzeugen.

Papernot et al.[16] bestätigten, dass Täuschungen auf ein bekanntes Netz mit hoher Wahrscheinlichkeit auch auf fremden Modellen fehlklassifiziert werden, also, dass die Ergebnisse nicht nur ein zufälliges Ereignis aufgrund von Overfitting des spezifischen Neuronalen Netzes sind. Die Ergebnisse wurde an verschiedenen Modelltypen getestet (bspw. Support-Vector-Machine ([SVM](#)), Deep Neural Network ([DNN](#))) und zeigten immer eine in ihrer Ausprägung schwankende Korrelation zwischen Fehlklassifikation auf einem fremden Modell im Vergleich zu bekannten Modellen.

Diese Ergebnisse führten zu dem Entschluss, ein eigenes neuronales Netz zu modellieren, welches als Substitute (dt. Ersatz) dient.

### 3.3 Implementierung eines eigenen Modells zur Klassifizierung von Straßenschildern (Aphrodite)

Für einige der nachfolgenden Umsetzungen wird ein selbst modelliertes [NN](#) als Substitute verwendet. Das Modell wird im Projekt unter dem Namen *Aphrodite* geführt. Bei dem [NN](#) handelt es sich um ein Keras-Modell, welches mithilfe von Tensorflow für die Erkennung von Verkehrsschildern trainiert wurde. Das [NN](#) wird für die lokale Degeneration in Kapitel 4 und im Saliency Maps Verfahren eingesetzt.

Das Modell *Aphrodite* umfasst vier Convolutional-Layer, drei Dense-Layer und zuletzt im Ausgabelayer eine Softmax-Funktion für die 43 Klassen. Ein detaillierter Aufbau des Netzes befindet sich im Anhang.

Für das Training wurden die [GTSRB](#)-Trainings- und Test-Daten verwendet. Diese wurden um die richtige Auflösung zu erreichen auf 64x64 interpoliert.

Da die verwendete Interpolationsfunktion des Black Box Modells des Wettbe-

Netzzusammenfassung  
als Tabelle in  
den Anhang

Link in den  
Anhang

werbs unbekannt ist, wurde für das Training jedes Bild mehrfach interpoliert und ebenfalls mehrfach für das Training verwendet. Für die Testdaten wurde eine zufällige Interpolationsfunktion ausgesucht.

*Aphrodite* erreicht eine Genauigkeit von 96.5% bei der Erkennung der Trainingsdaten. Eine Übersicht über die Trainingsparameter findet sich im Repository unter `/DegenerationCode/Training.py`.

Der Name Aphrodite wurde gewählt, um dem ersten Modell (Model A) innerhalb des Projektes einen sprechenden Namen zu geben.

Sollte man  
das anders  
schreiben?  
Oder packen  
wir das file in  
den Anhang?

## 4 Degeneration

Innerhalb dieses Kapitels wird der Ansatz der *Degeneration* vorgestellt.

Die Benennung schöpft sich aus der Nähe zu genetischen Algorithmen [10], allerdings aus einer invertierten Perspektive: Um auf unbekannte Modelle einzugehen, wird hierbei von einem korrekt erkannten Bild *weggearbeitet*. Anstatt allerdings *gute Gene* zu kombinieren [19], wird eine Genmutation erzeugt und überprüft, ob diese den Anforderungen der Klassifizierung noch genügt.

Zunächst wird das Konzept anhand von Pseudocode genauer erläutert. Anschließend wird die Implementierung eines Algorithmus vorgestellt, der das unbekannte [NN](#) des Wettbewerbs verwendet. Den Abschluss dieses Kapitels bildet eine lokale Implementierung inklusive einiger Verbesserungen, welche sich aufgrund der Limitierungen des Zugriffs auf die *remote-AI* des Wettbewerbs nicht angeboten haben.

### 4.1 Konzept

Die grundlegende Idee des Algorithmus bezieht sich darauf, ein Urbild  $i$  zu einem Abbild  $\hat{i}$  zu manipulieren, welches von dem unbekannten Klassifizierungsalgorithmus weiterhin korrekt erkannt wird.

Abhängig von der Stärke der Manipulation soll eine *Tiefe* gewählt werden, ab welcher der Algorithmus beendet wird. Als Beispiele der Manipulation seien insbesondere Rauschen und Glätten genannt, allerdings auch Kantenschärfung und Veränderungen der Helligkeit und anderer Metaparameter.

Mit fortschreitender Tiefe wird nahezu jedes Bild für den Menschen unkenntlich.

Zusätzlich sollten allerdings weitere Parameter als Abbruchkriterien aufgenommen werden, konkret eine Anzahl an Gesamt-Iterationen der Manipulationsfunktionen und eine Abbruchbedingung, beispielsweise wenn keine weiteren Fortschritte erreicht werden.

**Pseudocode**

Folgende Parameter erwartet die (generische) Implementierung des Degeneration-Algorithmus:

- Einen Eingabewert  $i$
- Eine Manipulations-Funktion  $a : i \rightarrow \hat{i}$
- Eine Klassifizierungsfunktion  $p : i \rightarrow \mathbb{R}$
- Eine gewünschte Tiefe  $d$  (empfohlen, nicht notwendig)
- Eine Iterationszahl  $its$  (empfohlen, nicht notwendig)
- Ein Schwellwert  $t$ , um wie viel % die Vorhersage schlechter sein darf, als das vorhergegangene Bild

Auf einige der Punkte wird in den Anmerkungen gesondert eingegangen.

```
input :  $i, a, p, d, its, t$   
output:  $\hat{i}, \text{score}$   
 $depth \leftarrow 0, loop \leftarrow 0$  ;  
 $s \leftarrow p(i)$  ;  
 $ii \leftarrow i, is \leftarrow s$  ;  
while  $depth < d \parallel loop < its$  do  
     $ai \leftarrow a(i)$  ;  
     $as \leftarrow p(ai)$  ;  
    if  $as \geq is - t$  then  
         $is \leftarrow as$  ;  
         $ii \leftarrow ai$  ;  
         $depth++$  ;  
    end  
     $loop++$  ;  
end  
return  $ii, is$  ;
```

**Algorithm 1:** Degeneration

**Anmerkungen**

Die Manipulationsfunktionen müssen genau ein Bild der Größe  $(x,y)$  erhalten, genau ein Bild der Größe  $(x,y)$  wiedergeben und (für die generischen Implementierungen) keine weiteren Parameter erhalten.

Zusätzlich sollte die Manipulationsfunktion zufällige Elemente erhalten. Sollte eine einfache, idempotente Glättungsfunktion den Schwellwert nicht erfüllen, so wird niemals eine größere Tiefe erreicht.

Tiefe, Schwellwert und Manipulationsfunktion müssen aufeinander abgestimmt werden. Es gibt einige Funktionen, welche eine starke Veränderung hervorrufen, und für welche eine geringe Tiefe bereits ausreicht. Auf der anderen Seite dieses Spektrums können Funktionen, welche lediglich minimale Änderungen vornehmen, schnell große Tiefen erreichen, ohne ein merklich verändertes Bild hervorzurufen zu haben.

Diese Parameter auszubalancieren obliegt dem Nutzer.

Bei der Auswahl der Parameter sollte zusätzlich berechnet werden, wie groß die letztendliche Konfidenz ist, falls die maximale Tiefe erreicht wird.

Innerhalb der Implementierungen sollte zusätzlich eine *verbose*-Funktion eingebaut werden. Hiermit kann zum einen ein ergebnisloser Versuch frühzeitig erkannt werden und zusätzlich, ob der Algorithmus sich verklemmt hat. Üblicherweise kann man erkennen, wenn die Manipulationsfunktion *zu stark*, beziehungsweise der Schwellwert zu niedrig gewählt ist.

## 4.2 Implementierung Remote

Im Rahmen des Wettbewerbs wird mit einer Rest-API gearbeitet, welche in Abschnitt 2.3 analysiert wird und besondere Herausforderungen mit sich bringt:

- Anfragen können fehlschlagen
- zwischen Anfragen muss ein Timeout liegen
- Mehrere Nutzer, welche die API mit dem gleichen API Key beanspruchen, blockieren sich

Zusätzlich wird der Grundalgorithmus um die *Verbose*-Funktion und eine *History* erweitert. Mithilfe der *History* können nach der Ausführung des Algorithmus hilfreiche Plots erstellt werden.

Diese ist in dem nachfolgenden Code in Listing 4.1 nicht enthalten.

Ebenso ist anzumerken, dass ignoriert wurde, welche Klasse zuerst erzeugt wird. Solange irgendeine Klasse mit einer passenden Konfidenz gefunden wird, gilt das Ergebnis als hinreichend. Im Normalfall bleibt es allerdings bei derselben Klasse.

Die Klassifizierungsfunktion wird innerhalb der Remote-Degeneration durch einige Hilfsfunktionen umgesetzt. Diese bereiten ein als *Bytearray* vorliegendes Bild auf und senden es an das Trasi-Webinterface. Dies geschieht in der Methode *Scorer.Send\_ppm\_image(img)*. In der Antwort des Trasi-Webinterfaces befindet sich ein JSON-Array mit den Scores einiger Klassen des bewerteten Bildes.

Die Hilfsmethode *Scorer.get\_best\_score(response)* gibt den höchsten gefundenen Score wieder.

```

1 # Parameters :
2 #   An image ( as 64 x64x3 Uint8 Array ) ,
3 #   a function to alter the image ,
4 #   a threshold how much the image can be worse by every
   step
5 #   The # of Iterations i want to ( successfully ) alter my
   image
6 #   The # of loops which i want to do max
7 def remoteDegenerate(image, alternationfn = _noise, decay =
   0.01, iterations = 10, maxloops=2000, verbose=True,
   history=True):
8     # First: Check if the credentials are correct and the
       image is detected
9     initialResp = Scorer.send_ppm_image(image)
10    if(initialResp.status_code!=200):
11        return
12    totalLoops = 0 # Counts all loops
13    depth = 0 # Counts successfull loops
14    lastImage = image
15    lastScore = Scorer.get_best_score(initialResp.text)
16    # To check if we put garbage in
17    print("StartConfidence:",lastScore)
18    # We stop if we either reach our depth , or we exceed the
       maxloops
19    while(depth<iterations and totalLoops<maxloops):
20        totalLoops+=1
21        # Alter the last image and score it
22        degenerated = alternationfn(lastImage.copy())
23        degeneratedResp = Scorer.send_ppm_image(degenerated)
24        if (degeneratedResp.status_code==200):
25            degeneratedScore= Scorer.get_best_score(
                degeneratedResp.text)
26        else:
27            print("Error, status code was: ",
                degeneratedResp.status_code)
28        # If our score is acceptable ( better than the set
           decay ) we keep the new image and score
29        if(degeneratedScore>=lastScore-decay):
30            lastImage=degenerated
31            lastScore=degeneratedScore
32            depth+=1
33        # We are working remote , we need to take a short
           break
34        time.sleep(1.1)
35    return lastScore,lastImage

```

Listing 4.1: Quellcode der Remote Degeneration



### 4.3 Ergebnisse Remote

In diesem Abschnitt werden die mit der Degeneration erzielten Ergebnisse in Bezug auf die Trasi-Schnittstelle vorgestellt. Zunächst werden einige positive Beispiele (Erfolge) gezeigt, anschließend wir auf einige Probleme, die aufgetreten sind, eingegangen und zuletzt wird ein kurzes Zwischenfazit gezogen.

#### Positive Ergebnisse

Die zuverlässigsten Ergebnisse werden mit einfachem Rauschen erzeugt. Die Abbildung 4.1 zeigt, dass zunächst vorallem die Pixel außerhalb des eigentlichen Schildes verändert werden. Dieses Verhalten wird erwartet. Die farbstarken bunten Pixel sind hierbei entstanden, da Werte welche die gültige Reichweite  $[0,255]$  verlassen, wieder zyklisch zurück in den Farbbereich geholt werden. Sollte ein Farbwert durch das Rauschen einen Wert  $-2$  erreichen, wird er auf  $253$  gesetzt.

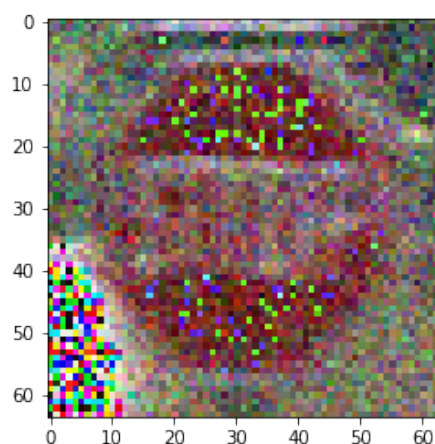


Abbildung 4.1: Rausch - Degeneration mit 600 Iterationen

Während die Abbildung 4.1 noch als Verkehrsschild zu erkennen ist, führt ein längeres Ausführen der Degeneration zu einem Ergebnis wie in Abbildung 4.2. Um dieses Ergebnis zu erzielen wurden 4400 Sekunden benötigt, also ca. 73 Minuten.

Die Plots in Abbildung 4.3 stellen den Verlauf des Algorithmus dar: Das erste Diagramm zeigt einen Verlauf der aktuellen *Tiefe* über die Iterationen, der zweite die jeweils produzierte Genauigkeit der jeweiligen Iteration (nicht nur die der akzeptierten), und der letzte Plot visualisiert diejenigen Iterationen, an welchen eine

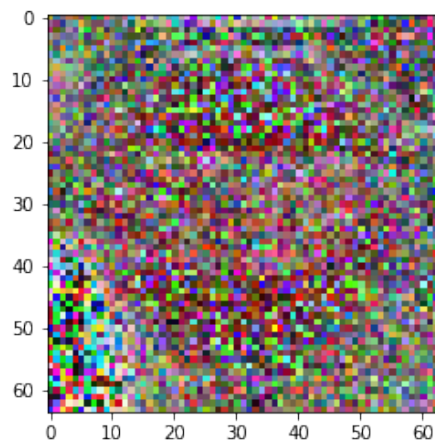


Abbildung 4.2: Rausch - Degeneration mit 4000 Iterationen

Änderung stattgefunden hat (weißer Strich) oder keine (schwarzer Strich). Innerhalb der Implementierung werden ebenfalls standardmäßig diese Plots erzeugt.

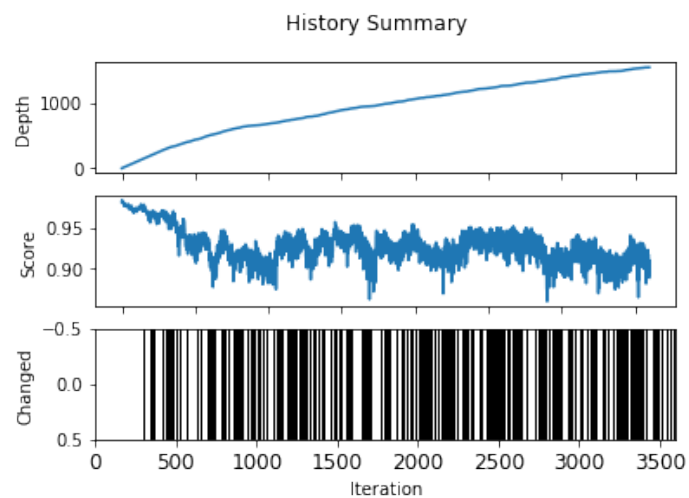


Abbildung 4.3: Plot der Rausch-Degeneration

Es wurden ebenfalls einige sehr positive Ergebnisse mit einer Mischung aus starkem Rauschen und Glätten erzeugt. Allerdings waren diese nicht zuverlässig reproduzierbar.

### Negative Ergebnisse

Es gibt zwei primäre Fehlerquellen in Bezug auf die Remote-Degeneration: Die Auswahl von Bildern, welche im GTSRB-*Training*-Set waren, sowie die Auswahl ungeeigneter Manipulationsfunktionen.

Das NN des Wettbewerbs scheint sich die Bilder aus dem Trainingsset *gemerkt* zu haben. Bereits minimale, unwichtige Änderungen des Schildes (z.B. Einfügen einiger blauer Punkte im Hintergrund) führen zu einer drastischen Verschlechterung des Ergebnisses. Dieses starke *Ausschlagen* des Scores macht die Benutzung der Degeneration unbrauchbar.

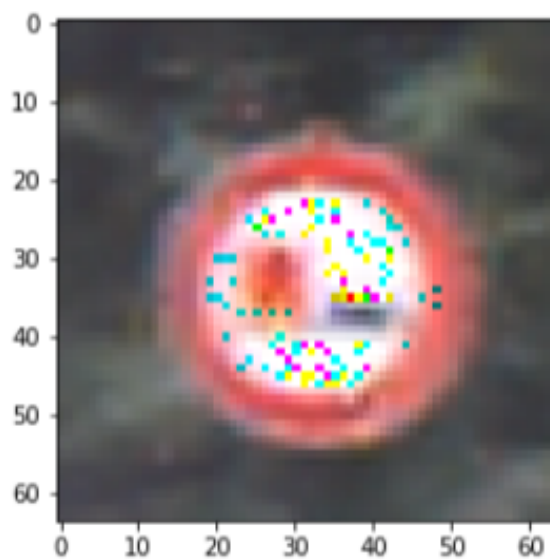


Abbildung 4.4: Rausch-Degeneration auf Trainingsbild - 36000 Iterationen

Dieses Problem hat sich herauskristallisiert, als über einen längeren Zeitraum (~10 Stunden) kein Bild erzeugt wurde, welches auch nur leicht verändert wurde. Die einzigen Änderungen, welche erzielt werden, befinden sich innerhalb des weißen Bereiches des Überholverbotsschildes, wie in Abbildung 4.4. Es werden aber keine Änderungen außerhalb des Schildes vorgenommen, wo diese zu erwarten wären und bei vorhergehenden Versuchen auch zu beobachten sind (vgl. Abbildung 4.1). Dieses Problem tritt ausschließlich, allerdings zuverlässig, bei der Verwendung von Bildern aus dem Trainingsset auf. Es tritt nicht auf, sobald man Bilder aus dem Test-Set oder *GTSRB-fremde* Bilder verwendet. Voraussetzung dabei ist, dass sie eine akzeptable Startkonfidenz besitzen.

Innerhalb der lokalen Implementierung ist dieses Problem ebenfalls aufgetreten, konnte allerdings behoben werden, sobald man das Overfitting erkannt hatte.

### Fazit

Innerhalb dieses kurzen Zwischenfazits sollen noch einmal die Vor- und Nachteile der Degeneration zusammengefasst werden:

Vorteile	Nachteile
<b>Model-Agnostic:</b> Der Algorithmus funktioniert unabhängig und ohne Wissen über das zugrundeliegende Modell	<b>Zeitintensiv:</b> v.A. die Remote-Variante benötigt größere Zeitspannen
<b>Kontext-Unabhängig:</b> Die Herangehensweise ist nicht auf Bilderkennungen limitiert	<b>Vorwissen benötigt:</b> Das Anwendungsfeld des zugrundeliegenden Modells muss bekannt sein und ein geeignetes Startbild muss ausgewählt werden
<b>Erweiterbar:</b> Die Manipulationsfunktionen können weiter ausgebaut werden und haben noch großes Potenzial	Die Degeneration erzielt bei empfindlichen Modellen schlechtere Ergebnisse - gerade sorgfältig trainierte Modelle sollten lange brauchen, um so überlistet zu werden
<b>Simplel:</b> Der Algorithmus ist einfach implementiert und erläutert, er benötigt keine höhere Mathematik oder Vorwissen zur Thematik <i>Machine Learning</i> und der Modelle/Verfahren im Speziellen	Im Remote-Umfeld kann die Degeneration als DDoS wahrgenommen werden und entsprechend frühzeitig unterbunden werden.

Tabelle 4.1: Zwischenfazit Degeneration

Als besonderen Fall sind solche Modelle zu nennen, die mit jeder Anfrage *hinzu-lernen*:

Diese sind entweder besonders anfällig gegenüber der Degeneration, weil sie die bereits veränderten Bilder als *korrekt* klassifizieren und somit den Entstehungsprozess der Degeneration verinnerlichen oder sie *härten* sich mit jedem Versuch gegen die neuen Änderungen und sind de facto immun gegen diesen Angriff.

Solche permanent trainierenden Modelle sind in der Praxis allerdings selten im Einsatz, da sie für eine Vielzahl verschiedener Angriffe anfällig sind. Als einfaches

Beispiel ist der Chatbot *Tay* von Microsoft zu nennen: Dieser sollte angenehme Unterhaltungen mit Twitternutzern führen und aus den entstandenen Konversationen kontinuierlich weiterentwickelt werden [18]. Innerhalb weniger Stunden gelang es einigen böswilligen Nutzern, dass Tay rassistische Äußerungen von sich gab [17]. Microsoft hat den Service von Tay am zweiten Tag eingestellt.

## 4.4 Implementierung Lokal

### Anpassungen und Verbesserungen

Innerhalb dieses Abschnittes werden zunächst die Änderungen bei der lokalen Verwendung des Algorithmus kurz behandelt und anschließend zwei konzeptionelle Verbesserungen vorgestellt: Parallel- und Batch-Varianten des Algorithmus.

Auf weitere Code-Beispiele wird im Rahmen des Umfangs verzichtet - sie befinden sich im Anhang.

#### Anpassungen

Für die lokale Implementierung wurde zunächst von Grund auf ein eigenes Modell mithilfe der [GTSRB](#)-Daten trainiert (siehe dazu Abschnitt 3.3). Das *Scoring* der Remote-Implementierung wird durch die *predict()*-Funktion des Models ersetzt.

Als zusätzliche Erweiterung wird für die lokale Implementierung umgesetzt, dass sich der Nutzer für eine bestimmte Klasse entscheiden kann, auf welche die Degeneration ausgelegt ist. Es wird also zuverlässig bspw. ein Stoppschild erzeugt, und kein beliebiges Schild mit hohem Score.

Des Weiteren entfällt die Wartezeit, welche zwischen Anfragen an die Schnittstelle benötigt wird. Auf diese Weise erhöht sich die Geschwindigkeit des Algorithmus maßgeblich.

Eine zusätzliche, passive Verbesserung wird erzielt, indem die GPU-Acceleration Funktionen von Tensorflow verwendet werden. Diese beschleunigen nicht nur das Training des lokalen Models maßgeblich, sondern auch die Vorhersagen. Insbesondere die Batch-Variante konnte im Zusammenhang mit der eingesetzten NVIDIA

Grafikkarte GTX 1070<sup>1</sup> um den Faktor 20 beschleunigt werden.

### Fazit

Das wichtigste Fazit, welches im Umgang mit der lokalen Implementierung gezogen werden kann, ist die Nichtverwendbarkeit der lokalen Bilder für die Schnittstelle. Während dies ursprünglich die Motivation war, schnell lokal Bilder zur Täuschung der „Black Box“ des GI-Wettbewerbs zu erzeugen und remote zu verwenden, stellte sich heraus, dass die lokalen Bilder keine guten Scores an der Schnittstelle erzielten und vice versa.

Es ist anzunehmen, dass die Modelle dieselben Stärken haben bei der korrekten Erkennung von Verkehrsschildern, allerdings unterschiedliche *Schwächen*. Die erzeugten Bilder zur Täuschung scheinen im Model selbst zu fußen und sind somit hochgradig spezifisch.

Die meisten stark veränderten Bilder, welche i.A. nicht mehr vom Menschen als Verkehrsschilder erkannt werden, erzeugen bei dem NN, welches im Algorithmus verwendet wird, Werte >90%, und beim anderen Modell zuverlässig einen Score von  $\approx 30\%$ .

Für ein Bild, welches an sich nichts mehr mit einem Verkehrsschild zu tun hat, sind dies immer noch unwahrscheinlich hohe Werte und die Zuverlässigkeit mit der dieser Zusammenhang auftritt lässt einen leichten, inhaltlichen Zusammenhang der Bilder erahnen.

---

<sup>1</sup><https://www.nvidia.com/en-gb/geforce/products/10series/geforce-gtx-1070/>

### 4.4.1 Batch-Degeneration

Innerhalb des Batch-Variante wird anstatt eines einzelnen Bildes ein Array aus  $n$  veränderten Bildern erzeugt.

Diese werden alle bewertet und falls das beste Bild des Batches den Schwellwert erfüllt wird mit dem besten Bild weiter gearbeitet. Dieses Verfahren entspricht deutlich näher der Genoptimierung von genetischen Algorithmen [9]: Es wird aus  $n$ -Genen das beste ausgewählt.

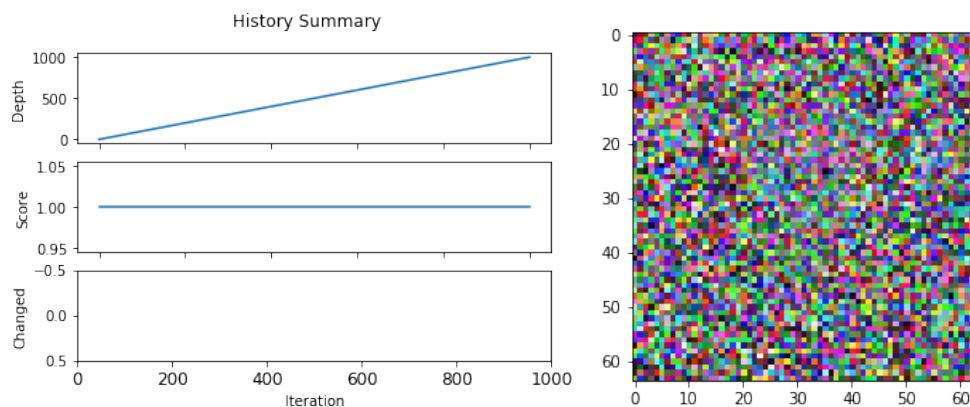


Abbildung 4.5: Verlauf der Batch-Degeneration mit Batchsize=10 und Tiefe=1000

Abbildung 4.5 zeigt den deutlich besseren Verlauf der Batch-Degeneration gegenüber der unveränderten Implementierung in Abbildung 4.3. Auffallend ist die durchgängige Veränderung und die gleichbleibend hohe Konfidenz von mehr als 99.9% <sup>2</sup>.

Dieses Verhalten für die Webschnittstelle des [GI-Wettbewerbs](#) einzusetzen ist möglich, allerdings wurde aufgrund der Wartezeit zwischen den Anfragen davon abgesehen.

Diese Variante profitiert maßgeblich von der *GPU-Acceleration* innerhalb Tensorflows.

Selbst ohne Verwendung des CUDA-Frameworks ist ein Tensorflow-Model auf Batch-Verarbeitung ausgelegt.

<sup>2</sup>Innerhalb des Plots wird es auf 1 gerundet

Die optimale Batchgröße zu finden ist systemabhängig und sollte kurz getestet werden. Insgesamt benötigt die Batch-Degeneration trotzdem maßgeblich mehr Zeit: Für das Beispiel in Abbildung 4.5 wurden ca. 15 Minuten benötigt, was knapp 5 mal so lange ist wie die ursprüngliche Implementierung.

**Anmerkung:** Ein prinzipielles Problem der Batch-Degeneration liegt in der Zufälligkeit der Manipulationsfunktion. Als Beispiel sei einfaches Rauschen gewählt.

Ein naheliegendes Verhalten für den Algorithmus ist von den 100 erzeugten Bildern dieses auszuwählen, welches das geringste Rauschen aufweist und als solches am wenigsten verändert wurde. Im Normalfall weist das am wenigsten veränderte Bild den nächsten Score auf.

Glücklicherweise ist dies ein hypothetisches Problem und in der tatsächlichen Implementierung nicht aufgetreten. Dennoch sollte es vor allem für die Manipulationsfunktion berücksichtigt werden. Im Falle einer Manipulationsfunktion, welche konstante Elemente beinhaltet (zum Beispiel Glätten oder statische Veränderungen der Helligkeit) fördert die Batch-Degeneration den selektiven Ansatz.

#### 4.4.2 Parallel-Degeneration

Die Parallel-Variante stützt sich auf die Idee, mehrere Threads zu starten, welche gleichzeitig eine Degeneration durchführen.

Sobald ein einzelner Thread die gewünschte Tiefe erreicht hat, wird der Prozess beendet.

Die Implementierung der Parallel-Degeneration ist aufgrund mehrerer technischer Gründe gescheitert:

- **Modelgröße:** Jeder Thread braucht ein eigenes Model, welches allerdings zu groß war. Naive Benutzung eines gemeinsamen Models führen zu Race-Conditions, *geschickte* Benutzung des Models führen zu einem Verhalten wie innerhalb der Batch-Variante
- **Numpy-Arrays:** Die Bilder für die lokale Degeneration lagen als Numpy-Arrays vor, welche ein besonderes Verhalten und eine besondere Benutzung



innerhalb der Parallelverarbeitung benötigen<sup>3</sup>.

- **Grafikkarteneinbindung:** Sobald die GPU-Acceleration innerhalb Tensorflows eingerichtet ist, werden (nahezu alle) Anfragen an die Grafikkarte weitergeleitet. Diese unterstützt das parallele Verhalten der einzelnen Threads nicht.

Die Probleme sind hardware- oder frameworkbezogen. Je nach Umfeld können diese somit entfallen. Race-Conditions entfallen beispielsweise, wenn man in der Cloud arbeitet.

Diese Variante war für die Remote-Implementierung nicht umsetzbar, da gleichzeitige Anfragen (mit dem selben API-Key) fehlschlagen. Ein internes Scheduling der Anfragen führt nicht zu schnelleren Ergebnissen.

#### 4.4.3 Tree-Degeneration

Diese Variante führt eine Merkstruktur ein, welche die bisherigen Ergebnisse und Schritte zwischenspeichert.

Das bisherige Verhalten entspricht dem einer Liste, bei welcher lediglich der letzte Knoten verwendet wird. Mit dem jeweils letzten Bild wird weitergearbeitet, bis entweder ein neues korrektes Bild erzeugt wird oder der Algorithmus endet.

Die Variation beinhaltet das führen eines *Retry-Counters*, welcher bei jedem Versuch von einem Knoten erhöht wird. Sollte eine gewisse Anzahl an Versuchen ergebnislos bleiben, wird der aktuelle Knoten verworfen und der Vorgänger benutzt.

Dieses Verhalten führt, je nach gewählter Maximalanzahl der Kinder eines Knotens, zu einem (Binär-)Baum. Das Abbruchkriterium der Tiefe kann weiterhin beibehalten werden und entspricht der Tiefe des Baumes. Im Falle eines versuchs- oder zeitbedingten Abbruchs wird das Bild mit der bisher größten Tiefe ausgegeben.

---

<sup>3</sup>Dieses Problem ist sicherlich lösbar, allerdings ein Problem aus dem Bereich der Parallelverarbeitung, was im Kontext dieser Arbeit nicht weiter verfolgt wird.

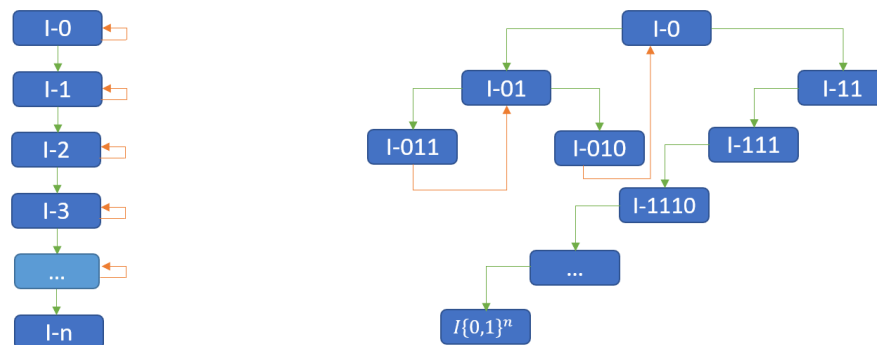


Abbildung 4.6: Derzeitige Implementierung und eine Baum-basierte Implementierung

Die Entwicklung dieser Variante entstand durch die Beobachtung, dass die Geschwindigkeit der Degeneration stark abhängig sind vom Ausgangsbild. Es kann ein Bild erreicht werden, welches sehr *sensibel* wahrgenommen wird und deutlich schwerer Änderungen *toleriert*.

Die Batch-Degeneration ist mit dieser Variante frei kombinierbar.

## 5 Saliency Maps

Das Saliency Map Verfahren wurde erstmals von den Neurowissenschaftlern Itti et al. [12] in ihrer Studie vorgeschlagen. Die Studie beschreibt eine Methode zur Extraktion von einzigartigen Merkmalen, wie beispielsweise Farben, Farbintensitäten und Strukturen, sogenannten High-Level Features, aus Bildern, die wiederum in sogenannten Saliency Maps topografisch visualisiert werden. Diese High-Level Features sind nichts anderes als spezifische Pixel im Bild auf Mikroebene (Low-Level Features), welche die optisch ansprechendsten respektive bedeutendsten Stellen in einem Bild repräsentieren [12].

Um also spezifischen Bildmerkmalen eine semantische Bedeutung zuzuordnen, werden beispielsweise Farbbilder anhand des Saliency Map Verfahrens in Schwarzweißbilder umgewandelt, um die stärksten darin vorhandenen Farben zu analysieren und zu extrahieren.

### 5.1 Konzept

Bei Convolutional Neural-Network (CNN) werden Merkmale von Eingabebildern extrahiert, indem zunächst im Input-Layer Low-Level Features und mit jeder weiteren Schicht des Netzwerks immer komplexere Merkmale (High-Level Features), wie etwa Kanten, Rundungen oder Strukturen erlernt werden [1].

Diese Kenntnis um CNN und Saliency Maps machten sich Simonyan et. al erstmalig im Kontext von Deep Learning zunutze, um die gelernten Merkmale des zugrunde liegenden CNN anhand von Saliency Maps zu visualisieren [20]. Die in der Arbeit von Simonyan et. al erzeugten Bilder sind hierbei nicht oder nur ansatzweise für den Menschen erkennbar, wurden jedoch vom zugrunde liegenden CNN mit einer hohen Konfidenz richtig klassifiziert werden.

Aus dieser Kenntnis ergeben sich folgende Hypothesen:

- Ein Convolutional Neural Network mit beliebiger Architektur, jedoch trainiert mit demselben Datensatz ([GTSRB](#)) wie das Black Box Modell (aus der Aufgabenstellung), lässt sich ähnlich gut angreifen wie das Black Box Modell selbst. Zu dieser Erkenntnis kamen Papernot et al. [16] im Rahmen ihrer Arbeit. Auf diesen Sachverhalt wird in Kapitel [3.2](#) näher eingegangen.
- Saliency Maps repräsentieren die wesentlichen Merkmale, die das Convolutional Neural Network zu den Eingabebildern gelernt hat. Die erzeugten Bilder mit dem Saliency Map Verfahren werden wiederum vom [CNN](#) mit einer hohen Konfidenz richtig klassifiziert. Das heißt das jeweils erzeugte Bild wird vom [CNN](#) als Verkehrszeichen erkannt und mit hoher Konfidenz richtig klassifiziert, ist jedoch für den Menschen nicht als solches erkennbar.

## 5.2 Implementierung

Als vorverarbeitenden Schritt werden zunächst alle 12.630 Bilder aus dem GTSRB-Testdatensatz, welche in 24-Bit Farbtiefe und im Portable Pixmap Image ([PPM](#)) Dateiformat vorliegen, in das Portable Network Graphic ([PNG](#)) Dateiformat konvertiert und im Dateisystem abgespeichert.

Die vorliegenden PNG-Bilder werden anhand dem „Aphrodite“-Modell, auf dessen Architektur und Training in Kapitel [3.3](#) eingegangen wird, klassifiziert:

Nur diejenigen Bilder, welche vom „Aphrodite“-Modell mit einer Konfidenz von 100% korrekt klassifiziert wurden (3.063/12.630) werden im Dateisystem in einem eigenständigen Verzeichnis abgespeichert.

Gemäß einer auf die Anforderungen an die Aufgabenstellung abgewandelten Variante zu Anh [3], erfolgt nun die Implementierung der verschiedenen Saliency Map Verfahren.

Hierzu werden zunächst die drei Basismethoden „Vanilla“ [20], „Guided Backpropagation“ [22] und „Integrated Gradient“ [23] implementiert. Unter Hinzufügung von Zufallsrauschen, gemäß [21], werden die Basismethoden verbessert, wie die

Gegenüberstellung der Ergebnisse zu den eingesetzten Verfahren in Kapitel 5.3 verdeutlicht. Diese optimierten Varianten werden als „Smoothed Vanilla“, „Smoothed Guided Backpropagation“ und „Smoothed Integrated Gradient“ bezeichnet und implementiert.

Jedes der Saliency Map Verfahren wird auf die mit dem „Aphrodite“-Modell klassifizierten Bilder angewendet. Diese Bilder werden nacheinander vom Dateisystem geladen und anschließend auf eine Zielbildgröße von  $64 \times 64$  Pixel skaliert. Das Saliency Map Verfahren extrahiert nun die vom zugrunde liegenden Modell (Aphrodite) gelernten Merkmale. Hierzu wird das jeweilige Eingabebild und das Aphrodite Modell verwendet. Die damit erzeugten Bilder werden anschließend in einem nach dem verwendeten Saliency Map Verfahren bezeichneten Verzeichnis in der Größe  $64 \times 64$  Pixel im Dateisystem abgespeichert.

Zur abschließenden Evaluierung der erzeugten Bilder am Black Box Modell, werden alle Bilder zu jedem eingesetzten Saliency Map Verfahren im Zyklus von 60 Bilder pro Minute (Übertragungslimit pro Minute) an das Webinterface des Wettbewerbs gesendet. Die Informationen, das heißt die vom Black Box Modell erkannte Zielklasse und Konfidenz, werden anschließend aus dem HTTP-Responsecode zu jedem übermittelten Bild in zwei verschiedenen Logdateien gespeichert: Die erste Logdatei speichert alle Ergebnisse aus dem jeweiligen HTTP-Responsecode, wohingegen die zweite Logdatei nur gefilterte Ergebnisse, das heißt Konfidenzen von mehr als 90%, zu den übermittelten Bildern enthält.

## 5.3 Ergebnisse

Mit den verschiedenen Saliency Map Verfahren wurden Graustufenbilder erzeugt, in denen die relevanten Bildmerkmale durch helle Pixeleinfärbungen gekennzeichnet sind. Die erzeugten Bilder sind hierbei vom Menschen nicht als Verkehrszeichen wahrnehmbar.

Die mit den Basismethoden „Vanilla“, „Guided Backpropagation“ und „Integrated Gradient“ erzeugten Bilder werden vom Black Box Modell mit jeweils 45,834911% als „Baustelle“ klassifiziert. Dieses Ergebnis lässt sich möglicherweise auf die be-


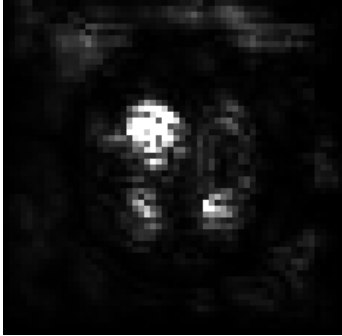
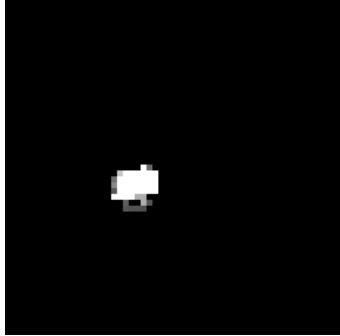
		
5.1a: Ursprungsbild Höchstgeschwindigkeit (30)	5.1b: Guided Backprop. Überholverbot 0.9155	5.1c: Vanilla Saliency Höchstgeschwindigkeit (30) 0.9283

Tabelle 5.1: Ergebnisse zu den verschiedenen Saliency Map Basisverfahren

sonderen „Faltungseigenschaften“ von Convolutional Neural Networks zurückführen, dass wesentliche bildcharakterisierende Merkmale und die damit verbundene semantische Information aus den erzeugten Bildern verloren ging.

Dahingegen konnten mit den optimierten Saliency Map Verfahren („Smoothed Vanilla“, „Smoothed Guided Backpropagation“ und „Smoothed Integrated Gradient“) Bilder erzeugt werden, die Zielkonfidenzen von mehr als 90% am Black Box Modell erreichten.

Tabelle 5.1 zeigt exemplarisch, wie aus dem Ursprungsbild (5.1a „Höchstgeschwindigkeit (30)“), unter Verwendung des „Smoothed Guided Backpropagation“ Verfahrens, ein Bild erzeugt wurde, das vom Black Box Modell mit 91,55% als „Überholverbot“ klassifiziert wurde (5.1b). Das „Smoothed Vanilla Saliency“ Verfahren erzeugte hingegen aus demselben Ursprungsbild ein Bild, das mit einer Zielkonfidenz von 92,83% vom Black Box Modell als „Höchstgeschwindigkeit (30)“ erkannt wurde (5.1c).

Tabelle 5.2 veranschaulicht weitere Beispiele für erfolgreich erzeugte Bilder, unter Verwendung der optimierten Saliency Map Verfahren. Wieder kann beobachtet werden, dass die erzeugten Bilder zwar mit einer Zielkonfidenz von mehr als 90% vom Black Box Modell klassifiziert wurden, die ursprüngliche Klasse jedoch abweicht.


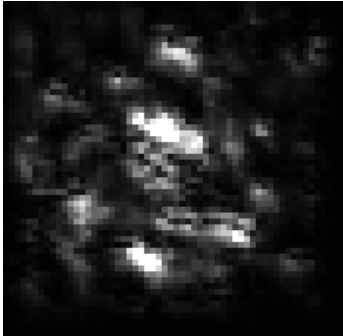



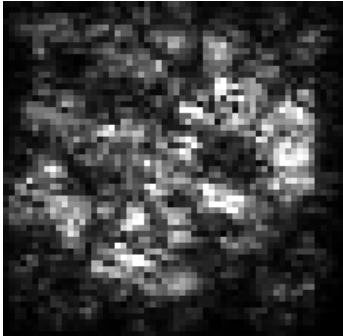
	
Rechts Vorbei	Guided Backprop. Einmalige Vorfahrt 0.9678
	
Einfahrt Verboten	Integrated Grad. Überholverbot 0.9571
	
Kreisverkehr	Integrated Grad. Baustelle 0.9999

Tabelle 5.2: Ergebnisse zu den verschiedenen optimierten Saliency Map Verfahren

Zusammenfassend konnte hiermit gezeigt werden, dass die in Kapitel 5.1 formulierten Hypothesen zutreffen und sich Bilder – unter Verwendung eines eigens trainierten CNN Modells („Aphrodite“) sowie verschiedener Saliency Map Verfahren – erzeugen lassen, die zwar keine für den Menschen sinnvolle Bedeutung haben, jedoch beim Black Box Modell Zielkonfidenzen von mehr als 90% erreichten.



## 6 Gradient Ascent

Beim Training von künstlichen neuronalen Netzwerken, wie etwa CNN, werden die Gewichte  $w_i$  und der Schwellwerte (Bias) so lange verändert und im Modell des Netzwerks gespeichert, bis die Ausgabe aus dem Netzwerk die Eingabedaten (Trainingsbilder) annähern. Um zu quantifizieren, wie gut dieses Ziel erreicht wird, wird eine sogenannte Kostenfunktion (engl. cost function) definiert. Die Kostenfunktion geht also über jedes einzelne Bild aus dem Trainingsdatensatz und berechnet den Unterschied zwischen der gewünschten Ausgabe und dem Ausgabevektor  $a$  des Netzwerks. Der Ausgabevektor  $a$  ist hierbei abhängig vom Eingabebild, dem Gewicht  $w_i$  und dem Schwellwert. Die Kostenfunktion hat den Wert 0 in etwa dann angenähert, wenn das Eingabebild annähernd dem Ausgabevektor  $a$  entspricht, was das Ziel des Trainings eines künstlichen neuronalen Netzwerks darstellt. Denn dann wurde das Eingabebild korrekt erkannt. Im Rahmen des Trainings sollen also eine Reihe von Gewichten  $w_i$  und Schwellwerte gefunden werden, welche die Kostenfunktion möglichst klein halten. Dies wird erreicht mit dem Algorithmus, der als Gradient Descent bezeichnet wird [24]. Dieser Algorithmus berechnet wiederkehrend den Gradienten.

Das Gradient Ascent Verfahren ist als Pendant zum Gradient Descent Verfahren zu verstehen. Hier wird auf die berechneten Gradienten eines trainierten Convolutional Neural Networks zugegriffen und das Eingabebild so lange verändert, bis dies dem gewünschten Ergebnisbild entspricht.

### 6.1 Konzept

Um gegebene CNN gezielt anzugreifen, das heißt Bilder zu erzeugen, die einer bestimmten Zielklasse entsprechen und gleichzeitig vom Menschen nicht als solche erkannt werden, eignet sich die Methode „Targeted Backpropagation“, welche ei-

ne Variante des Gradient Ascent Verfahrens darstellt [13]. Diese Methode setzt ein bereits trainiertes CNN voraus, greift auf die Gradienten des Modells zu und verändert das Eingabebild – wie etwa ein zufallsgeneriertes Bild – so lange, bis es der gewünschten Zielklasse entspricht oder diese annähert. Bei diesem Verfahren kann man beobachten, dass relevante Pixel, welche die bedeutendsten Stellen im Bild repräsentieren, stärker mutiert werden als unwichtige Pixel, die weitgehend unverändert bleiben.

## 6.2 Implementierung

Zunächst wurde als vorverarbeitender Schritt ein CNN unter Verwendung der AlexNet Architektur in PyTorch trainiert. Die hierbei verwendete Architektur des AlexNet entstammt [2] und wurde dahingehend modifiziert, dass das AlexNet zu Eingabebildern der Größe  $64 \times 64$  Pixel, drei Farbkanälen (RGB) sowie zu den 43 Klassen aus dem GTSRB Datensatz kompatibel ist. Die Bilder aus dem GTSRB Trainingsdatensatz, welche unterschiedliche Bildgrößen aufweisen, wurden im Zuge des Trainings auf eine Bildgröße von ebenfalls  $64 \times 64$  Pixel normiert. Das Training des AlexNet erfolgte 50 Epochen lang mit dem GTSRB Trainingsdatensatz und erzielte mit dem Stochastic-Gradient-Descent (SGD) Optimierer eine Genauigkeit von annähernd 89% (validiert mit dem GTSRB Testdatensatz). Die Implementierung des Gradient Ascent Verfahrens, unter Verwendung der „Targeted Backpropagation“ Methode und dem trainierten AlexNet, erfolgte in einer modifizierten Variante zu [15].

Unter Angabe der Zielklasse sowie der minimalen Zielkonfidenz erzeugt der Algorithmus zunächst ein Zufallsbild, welches als Eingabeparameter für das trainierte AlexNet verwendet wird. Unter Anwendung des SGD-Optimierers auf das Eingabebild werden die Gradienten entsprechend der spezifizierten Zielklasse berechnet. Diese Gradienten werden nicht verwendet, um die Parameter des AlexNet-Modells zu verändern, wie es beim Training der Fall war, sondern um das eingegebene Bild dahingehend zu modifizieren, dass es der gewünschten Zielklasse entspricht. Die entsprechenden Pixel der aktivierten Neuronen werden hierzu (je nach Aktivierungsstärke) mit den Bildpixelwerten addiert. Dieser Prozess wird so

lange wiederholt, bis das veränderte Bild die gewünschte minimale Zielkonfidenz zu der angegebenen Zielklasse erreicht hat.

Der Algorithmus wurde für jede der 43 [GTSRB](#)-Klassen so lange wiederholt, bis das veränderte Bild die gewünschte Mindest-Konfidenz von 100% erreicht. Zur abschließenden Evaluierung der erzeugten Bilder am Black Box Model, werden alle Bilder an das Webinterface des Wettbewerbs gesendet. Die Informationen, das heißt die vom Black Box Modell erkannte Zielklasse und Konfidenz, werden anschließend aus dem HTTP-Responsecode zu jedem übermittelten Bild in zwei verschiedenen Logdateien gespeichert: Die erste Logdatei speichert alle Ergebnisse aus dem jeweiligen HTTP-Responsecode, wohingegen die zweite Logdatei nur gefilterte Ergebnisse, das heißt Konfidenzen von mehr als 90%, zu den übermittelten Bildern enthält.

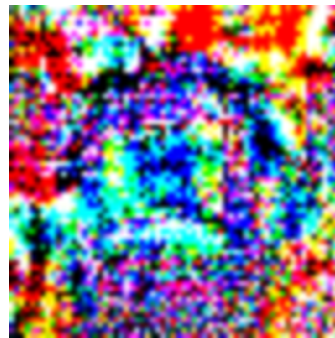
## 6.3 Ergebnisse

Mit dem Gradient Ascent Verfahren, unter Verwendung der „Targeted Backpropagation“ Methode, wurden 43 Farbbilder – also je ein Farbbild zu jeder [GTSRB](#)-Klasse – erzeugt. Die erzeugten Bilder sind hierbei vom Menschen nicht als Verkehrszeichen wahrnehmbar. Jedoch klassifizierte das Black Box Modell 20 der 43 erzeugten Bilder mit einer Konfidenz von über 90% als Verkehrszeichen. Unter diesen 20 erzeugten Bildern stimmte lediglich bei 10 Bildern die im Algorithmus angegebene Zielklasse mit der vom Black Box Modell ausgegebenen Klasse überein.

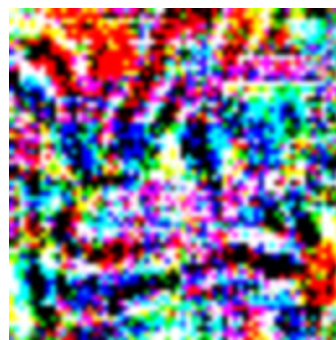
Tabelle [6.1](#) veranschaulicht vier repräsentative Ergebnisse, welche mit dem Gradient Ascent Verfahren erzeugt und vom Black Box Modell mit einer Konfidenz von über 90% als Verkehrszeichen klassifiziert werden. Die Abbildungen oben links und unten rechts werden vom Black Box Modell als „Einfahrt Verboten“ mit 99,99% Konfidenz beziehungsweise als „Kreisverkehr“ mit 98,68% Konfidenz korrekt klassifiziert. Das heißt, die im Algorithmus angegebene Zielklasse entsprechen der tatsächlichen Zielklasse. Hingegen weicht bei den beiden anderen Bildern (oben rechts und unten links) die im Algorithmus angegebene Zielklasse von der



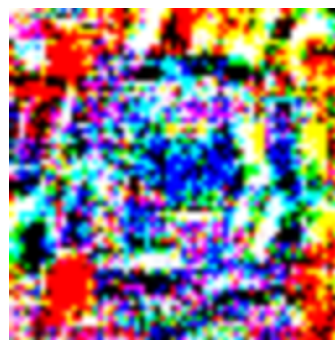
6.1a: Einfahrt Verboten  
0.9999



6.1b: Kreisverkehr  
0.9535



6.1c: Rechts Vorbei  
0.9950



6.1d: Kreisverkehr  
0.9868

Tabelle 6.1: Beispiele für Irrbilder mit Konfidenz  $>0.9$

tatsächlich erkannten Zielklasse ab. Bei Abbildung oben rechts wird im Algorithmus „Ausschließlich links“ als Zielklasse angegeben. Das erzeugte Bild wird mit einer Konfidenz von 95,35% vom Black Box Modell als „Kreisverkehr“ erkannt. Bei Abbildung unten links wird im Algorithmus „Links vorbei“ als Zielklasse angegeben. Das erzeugte Bild wird mit einer Konfidenz von 99,50% als „Rechts vorbei“ vom Black Box Modell erkannt.

## 7 Fazit

Nach einer abschließenden Zusammenfassung, werden in diesem Kapitel zusätzlich die verschiedenen Ansätze verglichen und bewertet. Aus dieser Diskussion werden die Einschränkungen und Möglichkeiten für weitere Arbeiten angesprochen.

### 7.1 Zusammenfassung

Mit der Degeneration konnte ein simpler Algorithmus erstellt werden, welcher zuverlässig gute Ergebnisse liefert. Der wesentliche Vorteil der Degeneration ist die Unabhängigkeit vom Model - man muss nicht zuerst ein Transfermodel erstellen, um Angriffe zu erzeugen, sondern kann diese anhand der vorliegenden *Black Box* generieren.

Der primäre Nachteil der Degeneration liegt in dem Zeitaufwand: Für einen *guten* Angriff musste über eine Stunde gearbeitet werden. *Bessere* Netze können zusätzlich empfindlicher auf Rauschen reagieren, und dementsprechend die Zeiten weiter erhöhen.

Dennoch ist die Effektivität der Degeneration überraschend - immerhin ist es im wesentlichen ein Brute-Force Angriff.

Des Weiteren konnten mit den optimierten Saliency Map Verfahren Erfolge in den geglätteten Varianten verbucht werden. Somit konnte bestätigt werden, dass die Visualisierung der relevanten Pixel für ein Bild mit hoher Konfidenz geeignet sind, um als *minimale Beispiele* für das NN verwendet werden können. Die entsprechenden Verfahren erzeugten Täuschungen mit Konfidenzen  $>0.9$ , allerdings lassen sich keine Aussagen über die allgemeine Verlässlichkeit und Zielgerichtetheit treffen.

Zuletzt können auch mit dem Gradient Ascent Verfahren sehr gute Ergebnisse erzielt werden. Für 10 von 43 Klassen können Täuschungen mit hohen Konfidenzen am [NN](#) des [GI](#)-Wettbewerbs erzielt werden. Es wird vermutet, dass die Ausbeute mit weiterer Optimierung vergrößert werden kann oder auch echte Bilder für die Erzeugung der Täuschungen verwendet werden können.

## 7.2 Diskussion

Die Herausforderungen des Wettbewerbs wirken sich auf die Erfolge der Verfahren aus. Es wird vermutet das beide Verfahren *Saliency Map* und *Gradient Ascent* bessere Ergebnisse liefern könnten, wenn das verwendete Bild größer als  $64 \times 64$  wäre. Des Weiteren kann nichts über die Validierungsgenauigkeit des [NN](#) des Wettbewerbs gesagt werden, weshalb auch die nicht zielgerichteten Täuschungsbilder als gutes Ergebnis betitelt werden.

Beim Vergleich des *Saliency Map* und *Gradient Ascent* Verfahrens kann das zuletzt genannte bevorzugt werden.

Diese Methoden lassen sich schwer mit der Degeneration vergleichen - die schnelleren Erfolge werden voraussichtlich mithilfe der Degeneration erzeugt, da sie deutlich weniger Vorlauf benötigt. Nachdem allerdings das entsprechende Umfeld (eigenes Netz, Bibliotheken und Code) erzeugt wurde, können innerhalb des Gradient Ascent Foolings deutlich schneller zuverlässige Bilder erzeugt werden.

## 7.3 Weiterführende Arbeiten

Die Ergebnisse dieser Arbeit liefern weitere Ansätze für zukünftige Aufgaben. Zum einen können die verwendeten Ansätze individuell weiter optimiert werden, bezüglich des selbsterstellten lokalen Neuronales Netz und der Algorithmik bzw. deren Parameter. Zum anderen können Verfahren entwickelt werden, welche sich aller Methoden gezielt bedienen.

Eine insgesamt spannende Arbeit wäre die Anwendung von Adversarial Attacks

auf Sprachassistenten. Vor allem die Degeneration kann bereits in ihrem jetzigen Zustand genutzt werden, um Störgeräusche zu erzeugen, welche dennoch als Schlüsselwörter erkannt werden und ein *Smarthome* hacken.

Auch die anderen Verfahren sind insbesondere geeignet, sollte das Modell offenliegen. Der Sprachassistent-Hersteller Mycroft setzt auf Open-Source und stellt dementsprechend auch das (allgemeine) Modell bereit. Zu bemerken ist hierbei noch, dass der Nutzer innerhalb der ersten Aktionen den Sprachassistenten auf seine Aussprache konfiguriert.

An der Degeneration können ebenfalls großflächige Weiterentwicklungen vorgenommen werden:

Zum einen die Verwendung der Tree-Degeneration, zum anderen können Beschleunigung und Verfall der einzelnen Alternations eingebaut werden. Ebenso sollte ein kleines Script erstellt werden, welche die verschiedenen Manipulationsfunktionen kurz auslötet und dem Nutzer vorstellt.

Als komplexere Weiterentwicklung können mithilfe der Degeneration *Manipulationsvektoren* erstellt werden und in einem Raum abgebildet werden. Hierbei stellt jedes *Rauschen* (bzw. Bilddifferenz) und die Score-Differenz einen Vektor dar, welcher in einem Raum abgebildet werden kann.

Mithilfe dieser Vektoren könnte über statistische bzw. numerische Verfahren solch ein Vektor gefunden werden, welcher die größte Länge hat allerdings die geringste Score-Differenz aufweist. Angewandt auf das Urbild sollte dieser Vektor ein optimales Ergebnis erzielen.

Einen weiteren Blick sollte man der Überführbarkeit der Angriffe von einem lokalen Modell auf ein unbekanntes Modell widmen, wie es in Abschnitt 3.2 thematisiert wird:

Diese Eigenschaft werden von den Saliency-Maps und dem Gradient Ascent Fooling hinreichend erfüllt, wohingegen solche Versuche beim Degeneration-Verfahren scheitern.

Im Rahmen dieses Ergebnisses könnte man einen gezielten Test der Verfahren auf zwei bekannte Modelle durchführen, um hier transparentere Werte zu erhalten und Gründe für dieses Verhalten auszumachen.

# Literaturverzeichnis

- [1] Unsupervised Feature Learning and Deep Learning Tutorial. <http://ufldl.stanford.edu/tutorial/supervised/FeatureExtractionUsingConvolution/>.
- [2] Datasets, Transforms and Models specific to Computer Vision: pytorch/vision. <https://github.com/pytorch/vision>, January 2019. original-date: 2016-11-09T23:11:43Z.
- [3] Huynh Ngoc Anh. Implementations of some popular Saliency Maps in Keras: experiencor/deep-viz-keras. <https://github.com/experiencor/deep-viz-keras>, January 2019. original-date: 2017-06-23T08:16:07Z.
- [4] BMBF-Internetredaktion. Das Auto von morgen: autonom, sicher, effizient - BMBF. <https://www.bmbf.de/de/automatisiertes-fahren-4158.html>.
- [5] BMW. Autonomes Fahren - Die 5 Stufen zum selbstfahrenden Auto. <https://www.bmw.com/de/automotive-life/autonomes-fahren.html>.
- [6] ZDF dpa. Ein unfall, der am branchen-versprechen nagt. <https://www.zdf.de/nachrichten/heute/uber-unfall-schlecht-fuer-vertrauen-in-technik-100.html>.
- [7] Marcus Efler. Autonomes Fahren: Das Ende des Lenkrads. *Die Zeit*, January 2018.
- [8] Gesellschaft für Informatik e.V. InformatiCup2019-Irrbilder.pdf. <https://gi.de/fileadmin/GI/Hauptseite/Aktuelles/Wettbewerbe/InformatiCup/InformatiCup2019-Irrbilder.pdf>, October 2018.



- [9] Ingrid Gerdes, Frank Klawonn, and Rudolf Kruse. *Evolutionäre Algorithmen: Genetische Algorithmen—Strategien und Optimierungsverfahren—Beispielanwendungen*. Springer-Verlag, 2013.
- [10] Jochen Heistermann. *Genetische Algorithmen: Theorie und Praxis evolutionärer Optimierung*, volume 9. Springer-Verlag, 2013.
- [11] Sandy H. Huang, Nicolas Papernot, Ian J. Goodfellow, Yan Duan, and Pieter Abbeel. Adversarial attacks on neural network policies. *CoRR*, abs/1702.02284, 2017.
- [12] L. Itti, C. Koch, and E. Niebur. A model of saliency-based visual attention for rapid scene analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(11):1254–1259, November 1998.
- [13] Yanpei Liu, Xinyun Chen, Chang Liu, and Dawn Song. Delving into Transferable Adversarial Examples and Black-box Attacks. *arXiv:1611.02770 [cs]*, November 2016. arXiv: 1611.02770.
- [14] Markus Maurer, J. Christian Gerdes, Barbara Lenz, and Hermann Winner, editors. *Autonomes Fahren: Technische, rechtliche und gesellschaftliche Aspekte*. Springer Vieweg, Berlin, 2015.
- [15] Utku Ozbulak. Pytorch implementation of convolutional neural network adversarial attack techniques : utkuozbulak/pytorch-cnn-adversarial-attacks. <https://github.com/utkuozbulak/pytorch-cnn-adversarial-attacks>, January 2019. original-date: 2017-12-15T01:39:12Z.
- [16] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. (+) Practical Black-box Attacks Against Machine Learning.pdf. *arXiv:1602.02697 [cs]*, February 2016. arXiv: 1602.02697.
- [17] Sarah Perez. microsoft-silences-its-new-a-i-bot-tay-after-twitter-users-teach-it-racism/. <https://techcrunch.com/2016/03/24/microsoft-silences-its-new-a-i-bot-tay-after-twitter-users-teach-it-racism>.

- [18] Sarah Perez. microsofts-new-ai-powered-bot-tay-answers-your-tweets-and-chats-on-groupme-and-kik. <https://techcrunch.com/2016/03/23/microsofts-new-ai-powered-bot-tay-answers-your-tweets-and-chats-on-groupme-and-kik>.
- [19] Eberhard Schöneburg, Frank Heinzmann, Sven Feddersen, et al. Genetische algorithmen und evolutionsstrategien. *Bonn: Addison-Wesley*, pages 185–218, 1994.
- [20] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps. *arXiv:1312.6034 [cs]*, December 2013. arXiv: 1312.6034.
- [21] Daniel Smilkov, Nikhil Thorat, Been Kim, Fernanda Viégas, and Martin Wattenberg. SmoothGrad: removing noise by adding noise. *arXiv:1706.03825 [cs, stat]*, June 2017. arXiv: 1706.03825.
- [22] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for Simplicity: The All Convolutional Net. *arXiv:1412.6806 [cs]*, December 2014. arXiv: 1412.6806.
- [23] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic Attribution for Deep Networks. *arXiv:1703.01365 [cs]*, March 2017. arXiv: 1703.01365.
- [24] XueFei Zhou. Understanding the Convolutional Neural Networks with Gradient Descent and Backpropagation. *Journal of Physics: Conference Series*, 1004(1):012028, 2018.