TECHNISCHE HOCHSCHULE NÜRNBERG
GEORG SIMON OHM

Bilderzeugende Verfahren zum Angriff einer Verkehrsschilder erkennenden KI

IT-Projekt Bericht

Master-Studiengang *Informatik*

Technische Hochschule Nürnberg Georg Simon Ohm

von

Leonhard Applis, Peter Bauer, Andreas Porada und Florian Stöckl

Abgabedatum:	14.03.2019
Gutachter der Hochschule:	Prof. Dr. Gallwitz

Eidesstattliche Erklärung

Wir versichern hiermit, dass der vorliegende IT-Projekt Bericht mit dem Titel

Bilderzeugende Verfahren zum Angriff einer Verkehrsschilder erkennenden KI

selbständig verfasst wurde und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt wurden. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Wir versichern zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Nürnberg, den 14. März 2019

Leonhard Applis, Peter Bauer, Andreas Porada und Florian Stöckl

Abstract

This paper uses several scientific approaches to show how convolutional neural networks can be tricked into recognizing and classifying false images as 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 blackbox-attacks), attacks by false images generated in another way 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 a similiar known neural network which serves as *substitute*. Additionally the failed approach of *Greyboxing* is shortly presented and its flaws are analyzed.
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.
In conclusion the successful methods are compared and evaluated against each other.

title: Fooling a TrafficSign-AI
authors: Leonhard Applis, Peter Bauer, Andreas Porada und Florian Stöckl
reviewer TH: Prof. Dr. Gallwitz

Kurzfassung

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

Titel:	Bilderzeugende Verfahren zum Angriff einer Verkehrsschilder erkennenden KI
Autoren:	Leonhard Applis, Peter Bauer, Andreas Porada und Florian Stöckl
Prüfer der Hochschule:	Prof. Dr. Gallwitz

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

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

1. Einleitung

Der Traum von einem autonom gelenkten Automobil ist so alt wie das Fahrzeug selbst [27]. 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 Auto 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 jedoch weiterhin an der Entwicklung eines autonomen Fahrzeugs geforscht, wie die Vergabe von Forschungsgeldern [16] und Berichte von Automobilherstellern [17] und der Presse [19] 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 und Passanten 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 [23].

Der Erfolg von *Adversarial Attacks* liegt in der gezielten Stimulation von Gewichten, welche die KI aus ihrem Trainingssatz erarbeitet hat, beziehungsweise die vom Entwickler definiert wurden. Auf diese Weise ist es möglich ein gewünschtes Feedback eines NNs zu erhalten, obwohl das Bild 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 Bildern 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 er von seiner KI erwartet.

Durch den steigenden Einsatz von Machine Learning in verschiedenen sensiblen Bereichen des täglichen Lebens, wie selbstfahrenden Autos, Terrorismusbekämpfung oder Betrugserkennung können Angriffe verheerende Schäden anrichten und stellen ein attraktives Ziel für Angreifer dar. Vor allem im Bereich des autonomen Fahrens, welcher aktuell geprägt ist durch eine allgemeine Debatte über das *Vertrauen in Technik* [18], können erfolgreiche Angriffe zu einem forschungsschädlichen Misstrauen führen und die Nutzung sowie Entwicklung 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. Erst nach der Erzeugung dieser Beispiele kann ein Model gegen *Adversarial Attacks gehärtet* 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[1] der 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 [20].

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 dieser 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 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 Rahmenbedingungen 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)

[1] https://gi.de/informaticup/

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 dem (gescheiterten) *Greyboxing* in Kapitel 4. Es wird zunächst das geplante Konzept in Abschnitt 4.1 vorgestellt, die grundlegende Implementierung in Abschnitt 4.2 und zuletzt eine Problem und Fehleranalyse des Ansatzes in Abschnitt 4.3.

Daraufhin wird im Kapitel 5 die *Degeneration* vorgestellt. 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 in Abschnitt 5.1 die Idee anhand von Pseudocode genauer erläutert und anschließend in Abschnitt 5.2 die Implementierung für die Kommunikation mit der Schnittstelle des Wettbewerbs gezeigt. Die Ergebnisse liegen gesondert in Abschnitt 5.3 vor. Neben der Implementierung für die Webschnittstelle werden zum Abschluss des Kapitels in Abschnitt 5.4 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 6 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 7 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.

Den Abschluss dieser Arbeit bildet im Kapitel 8 ein Fazit über die vorgestellten 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 bilderzeugenden Verfahren zum Angriff einer Verkehrsschilder erkennenden KI analysiert. Die Anforderungen untergliedern sich dabei in funktionale Anforderungen und nicht funktionale Anforderungen. Die analysierten Anforderungen werden durch die Rahmenbedingung des Wettbewerbs ergänzt, die im Wesentlichen aus Auflagen zu dem bereitgestellten neuronalen Netz bestehen.

2.1. Funktionale Anforderungen

Die Kernaufgabe der implementierten Lösungen der nachfolgenden Kapitel 4 bis 7 besteht darin Bilder zu genieren. Als Eingangsparameter dürfen dafür Bilder verwendet werden, die anschließend durch Algorithmen modifiziert werden.
Neben der Generierung von Bildern zur Täuschung des NN des GI-Wettbewerbs sollen Funktionen bereitgestellt werden, mit denen die erzeugten Bilder bewertet werden. Dazu sollen die Bilder an die Webschnittstelle des Wettbewerbs gesendet und das Ergebnis aufbereitet werden.

2.2. Nicht funktionale Anforderungen

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

das NN des GI-Wettbewerbs während des Wettbewerbzeitraums nicht verändert wird.

Weiterhin ergibt sich aus der Aufgabenstellung des GI-Wettbewerbs eine qualitative Anforderung an die erzeugten Bilder zur Täuschung der Blackbox-KI. Die erzeugten Bilder müssen mindestens mit einer Konfidenz von 90% von der Blackbox-KI als Verkehrszeichen erkannt werden. [20]
Um diese Anforderung zu erfüllen, muss eine weitere indirekte Anforderung erfüllt sein, die sich aus den Hinweisen und der nachfolgenden Untersuchung des NN des GI-Wettbewerbs ergibt. Da die Blackbox-KI mit dem GTSRB Datensatz trainiert wurde, müssen die generierten Bilder der in Kapitel 4 - 7 folgenden Implementierungen den Klassen zugeordnet werden können, die der GTSRB enthält, beziehungsweise, die vom NN des Wettbewerbs erkannt werden.

Abschließend legt die Aufgabenstellung des GI-Wettbewerbs fest, dass die erzeugten Bilder zur Täuschung der KI des Wettbewerbs nicht von einem Menschen als Straßenschild erkennbar sein dürfen. Die Beurteilung dieses Faktes wird durch ein persönliches Urteil der Autoren vorgenommen.

2.3. Eigenschaften des Neuronales Netzes des GI-Wettbewerbs

Die Rahmenbedingung der vorliegenden Arbeit ergibt sich aus der Aufgabenstellung und den Ressourcen des GI-Wettbewerbs, die den Teilnehmern zur Verfügung gestellt werden. Bei den Ressourcen handelt es sich um ein NN, an welches über eine Webschnittstelle Anfragen gestellt werden können, und Informationen über dieses Netz.
Um ausführlichere Informationen über das Blackbox-NN des Wettbewerbs zu sammeln, werden zu Beginn und während des Wettbewerbs verschiedene Tests an der Webschnittstelle durchgeführt. Die gewonnenen Erkenntnisse befinden sich in der folgenden Aufzählung:

1. Aus der Aufgabenstellung ist abzulesen, dass das NN des Wettbewerbs PNG-Bilder in der Auflösung 64x64 verarbeitet. Vermutlich wurde das Netz ausschließlich auch mit derartigen Bildern trainiert

2. Die Bilder, mit denen die Blackbox-KI trainiert wurde, stammen aus dem GTSRB[1] Datensatz
   a) Datensatz enthält 43 verschiedene Klassen, die 43 verschiedenen deutschen Straßenschildern entsprechen
   b) Datensatz enthält insgesamt mehr als 50.000 Bilder
   c) Datensatz ist gegliedert in Trainings- und Testbilder
   d) Bei den Bildern des Datensatzes handelt es sich um reale Bilder (Kameraaufnahmen)
   e) Die Bilder sind einzigartig, kommen also im Datensatz nur jeweils einmal vor. Einige der Bilder sind allerdings Fotografien vom gleichen Straßenschild.
3. Die Blackbox-KI liefert Konfidenz-Scores zu 33 verschiedenen Klassenlabels, zehn Klassen des GTSRB Datensatzes werden nicht genutzt
4. Die Webschnittstelle liefert als Antwort auf eine Anfrage mit einem zu bewertenden Bild ein JSON-Objekt mit Konfidenzwerten verschiedener Klassen zurück
5. Bei der Rückgabe der Konfidenz-Scores wird voraussichtlich eine Softmax-Ausgabefunktion genutzt, da die Summe der einzelnen Scores eines bewerteten Bildes teilweise unter 100% liegt
6. Das NN des Wettbewerbs besitzt keine Klasse „Kein Straßenschild“ und liefert für jedes Bild Konfidenz Werte für Straßenschilder zurück
7. Beim Training der Blackbox-KI wurde vermutlich eine Interpolationsfunktion genutzt, da die ursprüngliche Größe eines Bildes keinen Einfluss auf die Scores der Schnittstelle hat
8. Die Blackbox-KI enthält ein starkes Overfitting bezüglich der Trainingsdaten
9. Die Blackbox-KI ist sehr unzuverlässig bei Eingabebildern, welche keinem Verkehrszeichen ähneln. So wird beispielsweise das unveränderte Logobild

[1] http://benchmark.ini.rub.de/?section=gtsrb&subsection=dataset

der Technischen Hochschule Nürnberg - Georg Simon Ohm von der Blackbox-KI mit einer Konfidenz von 99,99% als Verkehrszeichen vom Typ „Ausschließlich geradeaus“ erkannt.

3. Technisches Konzept

3.1. Verwendete Technologien

Die Umsetzung der Implementierung erfolgt innerhalb der webbasierten, interaktiven Entwicklungsumgebung Jupyter Notebook [8] (in der Version 5.7.4) zusammen mit der Programmiersprache Python [13] (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 [24]. Die Programmiersprache wurde ferner im Rahmen der StackOverflow Befragung 2017 von den befragten Softwareentwicklern zur fünftbeliebtesten Technologie des Jahres 2017 gewählt [11]. 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 [13].

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 [3]. Schließlich erlaubt „Anaconda Accelerate“ den programmatischen Zugriff auf numerische Softwarebibliotheken zur beschleunigten Codeausführung auf Intel Prozessoren sowie NVIDIA Grafikkarten [1].

Zur fehlerfreien Ausführung des Codes der Implementierungen in den nachfolgenden Kapiteln muss Python in der Version 3.6.5 verwendet werden. Weiterhin

Name	Version	Beschreibung
Keras	2.2.4	Enthält Funktionen für Deep-Learning Anwendungen [2]
Torchvision	0.2.1	Enthält Datensätze, Modellarchitekturen und gängige Bildtransformationsoperationen für Computer-Vision Anwendungen [38]
OpenCV	3.4.2	Enthält Funktionen für Echtzeit Computer-Vision Anwendungen [6]
NumPy	1.15.3	Enthält Funktionen zur effizienten Durchführung von Vektor- oder Matrizenrechnungen [5]
Requests	2.18.4	Enthält Funktionen zur Vereinfachung von HTTP-Requests [9]
Pillow	5.2.0	Enthält Funktionen zum Laden, Modifizieren und Speichern von verschiedenen Bilddateiformaten [7]
Matplotlib	2.2.3	Enthält Funktionen zum Plotten von Graphen oder Bildern [4]
SciPy	1.1.0	Enthält wissenschaftliche und technische Funktionen zur Datenverarbeitung [10]

Tabelle 3.1.: Paketabhängigkeiten der implementierten Software

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 Ausführen 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 Webschnittstelle in Abschnitt 2.3 liefert keine genauen Informationen über das zugrunde liegende Modell, dessen Architektur oder weitere Eigenschaften.

Papernot et al. [30] kamen zu dem Ergebnis, dass Deep Neural Networks, welche mit demselben Datensatz trainiert wurden, sich jedoch in ihrer Architektur unterscheiden, annähernd gleich verwundbar sind gegen feindliche Angriffe. Diese Ergebnisse führten zu dem Entschluss, eigenständige Convolutional Neural Networks (*Aphrodite* und *AlexNet*) zu trainieren und als Basis für verschiedene Angriffsszenarien auf das NN des GI-Wettbewerbs zu nutzen.

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

Für einige der nachfolgenden Umsetzungen wird ein selbst erstelltes 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[1] für die Erkennung von Verkehrsschildern trainiert wurde.
Das NN wird für die lokale Degeneration in Kapitel 5 und im *Saliency Map*s 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 A.

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 Blackbox-Modells des Wettbewerbs 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 (Von Bild zu Bild unterschiedlich ausgewählt).

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

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

[1] https://www.tensorflow.org/

4. AI-Greyboxing

Der erste Ansatz, der verfolgt wird, wird im Folgenden *Greyboxing* genannt.

4.1. Konzept

Das Konzept ist hierbei, dass über einen Generator zufällige Nicht-Straßenbilder erzeugt werden, diese von der Schnittstelle bewertet werden, und eine zweite, lokale AI auf die Scores der Schnittstelle trainiert werden.
Mit fortschreitendem Experiment soll die lokale KI ihr Verhalten an das der Schnittstelle angleichen und als Greybox dienen: Die lokal erzeugten Bilder und die lokalen Scores werden ähnlich zu den Scores der Schnittstelle. Ist dieser Zustand erreicht, kann man lokal Bilder generieren und sobald die Greybox dieses akzeptiert an die Schnittstelle schicken. Diese doppelt-bewerteten Ergebnisse können erneut ins Training einfließen, um die Greybox weiter zu verbessern.

Ausgehend hiervon können ebenfalls weitere Verbesserung am Generator vorgenommen und lokal getestet werden. Somit kann der Generator ebenfalls *trainiert* werden um gegebenenfalls Hyperparamter wie Farbanteile oder Helligkeit anzupassen.

Das Konzept umfasst im erweiterten Sinn ebenfalls alle Maßnahmen, um die stetige, iterative Verbesserung der beteiligten Komponenten zu vollziehen. Als Generator wird für die Implementierung einfaches Rauschen gewählt. Andere mögliche Generatoren sind z.B. Webcrawler, welche Bilder nach gewissen Parametern aussuchen oder Generatoren, welche ein Bild lediglich *verzerren* (wie beim Verfahren der Degeneration, welches in Kapitel 5 erläutert wird).

4.2. Implementierung und erste Ergebnisse

Die Abbildung 4.1 zeigt einen Überblick über die *Setup-Phase* des Greyboxing:

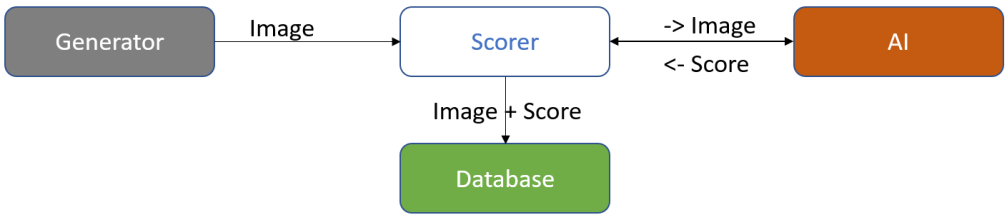


Abbildung 4.1.: Komponenten Setup-Phase

- **Generator:** Produziert zufällige Bilder auf Abruf. Innerhalb des ersten Ansatzes wird der Generator als einfaches Pythonfile mit simplen Methoden umgesetzt.
- **Scorer:** Erhält Bilder vom Generator, lässt diese von der Schnittstelle bewerten und speichert die Ergebnisse samt Bild in einer Datenbank. Die Hauptlogik findet somit im Scorer statt. Er wurde als Jupyternotebook umgesetzt.
- **AI! (AI!):** Die Trasi-Schnittstelle, wie in Abschnitt 2.3 beschrieben.
- **Database:** Eine Datenbank, die Bilder und Scores abspeichert. In der Implementierung wird eine MongoDB gewählt, welche sich gut eignet um die JSON-Antworten der Schnittstelle direkt aufzunehmen.

Mit dieser Architektur werden über einen längeren Zeitraum knapp 100 000 Bilder gesammelt und abgespeichert. Nach der Sammlung der Daten wird eine KI trainiert, welche die Scores der Schnittstelle imitiert. Hierfür wird das Framework Tensorflow gewählt und verschiedene Modellkonfigurationen sowie Hyperparameter durchgespielt.

Bei der praktischen Umsetzung zeigt sich, dass kein zufriedenstellendes Modell erzeugt werden kann. Eine detaillierte Ausführung dazu findet sich im nachfolgenden Abschnitt 4.3.

Unabhängig davon soll noch einmal das Konzept nach dem Training vorgestellt werden, dargestellt in Abbildung 4.2: Als neue Komponente ist die **lokale KI** hinzu-

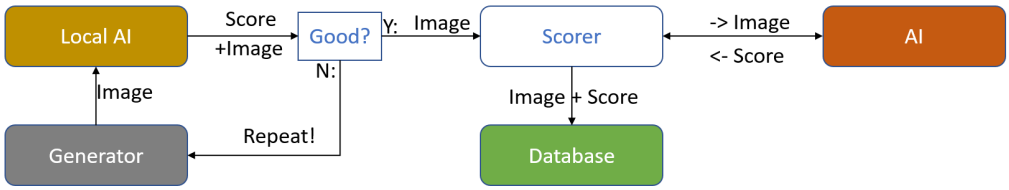


Abbildung 4.2.: Komponenten Betriebs-Phase

gekommen, welche zunächst die Bilder des Generators prüft, und lediglich solche weitergibt, welche bereits von ihr als *gut* eingestuft werden.

Vorausgesetzt die lokale KI erfüllt ihren Zweck, hilft sie iterativ mit jedem Bild eine bessere Datenmenge zu sammeln, bzw. innerhalb der Datenbank höhere Scores zu hinterlegen. Mit steigender Dauer und mehr Training sollte die lokale KI somit immer bessere Ergebnisse erzielen. Weiterhin ist es denkbar, ein *Ensemble* aus verschiedenen Klassifizieren zu etablieren, wobei der erste zwischen *schlecht* und *medium* entscheidet, der zweite zwischen *medium* und *gut* und die Kette nach diesem Muster fortgesetzt wird.

Ebenso hilft eine funktionierende lokale KI direkt an dem Generator *zu experimentieren*, um dort bessere Parameter zu erarbeiten.

Als weiterführender Gedanke kann eine zusätzliche Komponente *Generator-Tuner* kreiert werden, welche den Generator für die lokale KI mit jedem erzeugten Bild *optimiert*. Die Schwierigkeit dabei ist, dennoch von einander verschiedene Bilder zu erzeugen.

Die Ähnlichkeit der lokalen KI zur Schnittstelle kann direkt in der Datenbank gemessen werden: Jedes hinterlegte Bild eines Testsets wird ebenfalls von der lokalen KI bewertet und der Score in der Datenbank erfasst. Dieses Tripel aus <*Bild*, *RemoteScores*, *LocalScores*> ermöglicht eine ganze Bandbreite von Ähnlichkeitsmaßen direkt in der Datenbank, von welcher aus sie performant abrufbar sind.

4.3. Fehler- und Problemanalyse

Der Ansatz des *Greyboxing* ist bereits in seiner Setup-Phase gescheitert. Zu betonen ist allerdings, dass er vor allem aus zeitlichen Aspekten verworfen wurde, um im Rahmen des Projekts andere, vielversprechende Verfahren auszuprobieren.
Grund für das frühe Ende ist das Training des lokalen neuronalen Netzes: Dieses kann bereits auf Basis der separierten Testdaten keine hinreichenden Scores erzielen.

Im ersten Ansatz werden die Schnittstellen-Scores nach ihrem Prozentsatz mit einem Label versehen: *Schwach* für einen Score <20%, *Medium* für einen Score >20% und <50% und *Stark* für >50%. Zunächst wird mit dem *rohen Verhältnis* von 80:18:2 gearbeitet (Trainings- und Testsets mit entsprechendem Verhältnis), wobei die erzeugten Netze lediglich das Verhältnis wiedergegeben und in jedem Fall *schwach* vorhergesagt haben.

In einem zweiten Ansatz werden lediglich *schwache* und *medium* Bilder ausgewählt für das Training in einem 1:1 Verhältnis. Dieses neuronale Netz hat ebenfalls nur *geraten* und erreichte entsprechend eine Accuracy von 50%. Ein möglicher und wahrscheinlicher Grund für diesen Wert ist die Zufälligkeit der Bilder, bzw. die Verschiedenheit der einzelnen Bilder zueinander. Diese ist ebenfalls zufällig. Für das nicht trainierte Netz besitzen die Bilder keinen Zusammenhang zu ihrem Label. Diese Zuordnung erscheint dem Netz ebenfalls zufällig.

Innerhalb der Back-Propagation [39] werden nun die Gewichte angezogen, damit das zufällige Bild bei einem erneuten Durchlauf einen erhöhten Score erzielt. Da das zugrundeliegende Bild allerdings nur aus Rauschen besteht, wird im Wesentlichen nur ein (verringertes) Rauschen auf die Gewichte des Netzes angewandt. De facto hat das Netz also nichts gelernt, sondern sich nur etwas (zufällig) verformt.

In genauerer Betrachtung *schrumpfen* die Gewichte des Netzes und lediglich der Bias der Ausgabeschicht ist ausschlaggebend für das Ergebnis der Klassifikation. Der Bias entspricht mit fortschreitendem Training dem höchsten Verhältnis innerhalb der Trainingsmenge.

Mögliche Lösungsansätze

Zwei mögliche Lösungsansätze sind die Verwendung eines nicht-zufällig initiali-

sierten Netzes via Transfer Learning [29] oder die Verwendung eines weniger zufälligen Generators. Ein transferiertes Netz kann hierbei seine ersten Layer, welche Features extrahieren und z.B. Kanten erkennen, behalten, und lediglich die hinteren Schichten dafür nutzen das Verhalten der Schnittstelle nachzuahmen. Innerhalb dieses Ansatzes ist es unabdingbar, dass die ersten Schichten des Netzes **nicht** trainiert bzw. verändert werden, sondern ihre Funktionalitäten behalten.

Als alternative Generatoren bieten sich solche an, welche *Formen* generieren: Streifen, Kugeln oder *echte* Bilder. Die Verwendung eines solchen Generators hat zur Folge, dass die Bilder im Trainingsset nicht vollständig zufällig sind, sondern im Wesentlichen durch ihre Formen definiert sind. Das Netz ist somit im Stande auf jeden Fall Formen zu erkennen,diese zu gewichten und zu bewerten. Als Variante hiervon eignen sich Webcrawler-Generatoren, welche z.B. geeignete Katzenbilder aus dem Internet suchen.

5. Degeneration

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

Die Benennung schöpft sich aus der Nähe zu genetischen Algorithmen [22], allerdings aus einer invertierten Perspektive: Um auf unbekannte Modelle einzugehen, wird hierbei von einem korrekt erkannten Bild *weg gearbeitet*. Anstatt allerdings *gute Gene* zu kombinieren [33], wird 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 Zugriffes auf die *remote-AI* des Wettbewerbs nicht angeboten haben.

5.1. 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 $p : i \rightarrow \mathrm{R}$
- 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 dieser Punkte wird in den Anmerkungen gesondert eingegangen.

input : i,a,p,d,its,t
output: $\hat{i}$, score

$depth \leftarrow 0, loop \leftarrow 0$;
$s \leftarrow p(i)$;
$ii \leftarrow i, is \leftarrow s$;
while $depth < d \;||\; loop < its$ **do**
    $ai \leftarrow a(i)$;
    $as \leftarrow p(ai)$;
    if $as >= is - t$ **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 hervorgerufen 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 ebenfalls eine *verbose*-Funktion eingebaut werden. Hiermit kann zum einen ein ergebnisloser Versuch frühzeitig erkannt werden sowie zusätzlich, ob der Algorithmus sich *verklemmt* hat. Üblicherweise kann man erkennen, wenn die Manipulationsfunktion *zu stark*, beziehungsweise der Schwellwert zu niedrig gewählt ist.

5.2. 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 5.1 nicht enthalten.

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

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 gefunden Score wieder.

```
1  #  Parameters:
2  #     An image (as 64x64x3 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 5.1: Quellcode der Remote Degeneration

5.3. 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 5.1 zeigt, dass zunächst vor allem 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.

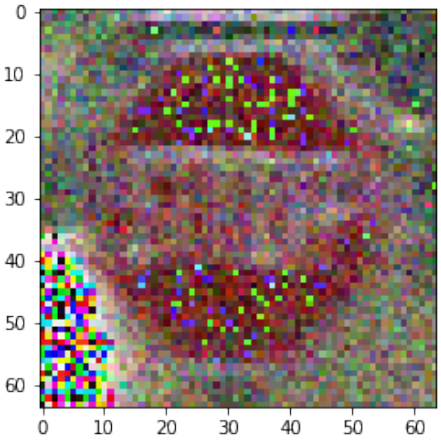


Abbildung 5.1.: Rausch - Degeneration mit 600 Iterationen

Während die Abbildung 5.1 noch als Verkehrsschild zu erkennen ist, führt ein längeres Ausführen der Degeneration zu einem Ergebnis wie in Abbildung 5.2. Um dieses Ergebnis zu erzielen wurden 4400 Sekunden benötigt, also ca. 73 Minuten.

Die Plots in Abbildung 5.3 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

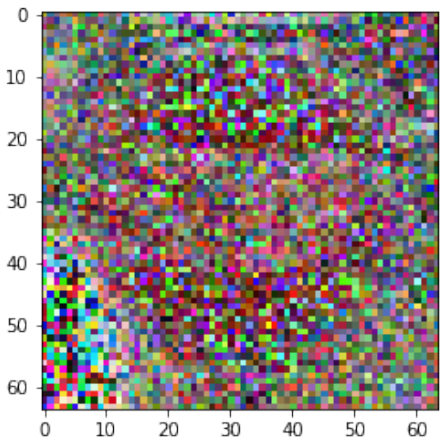


Abbildung 5.2.: Rausch - Degeneration mit 4000 Iterationen

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

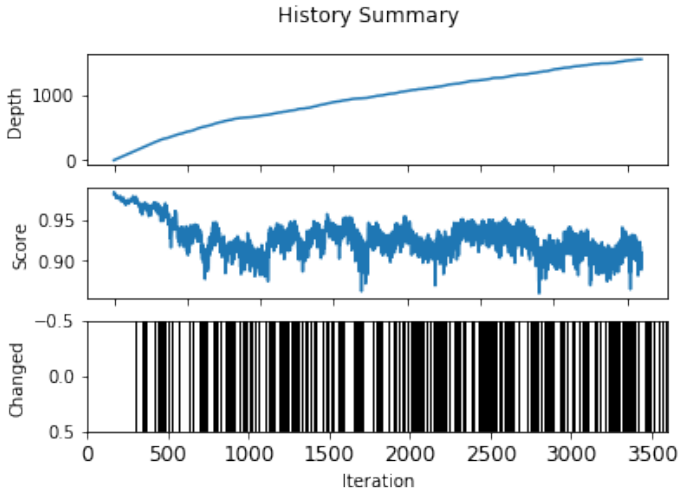


Abbildung 5.3.: Plot der Rausch-Degeneration

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.

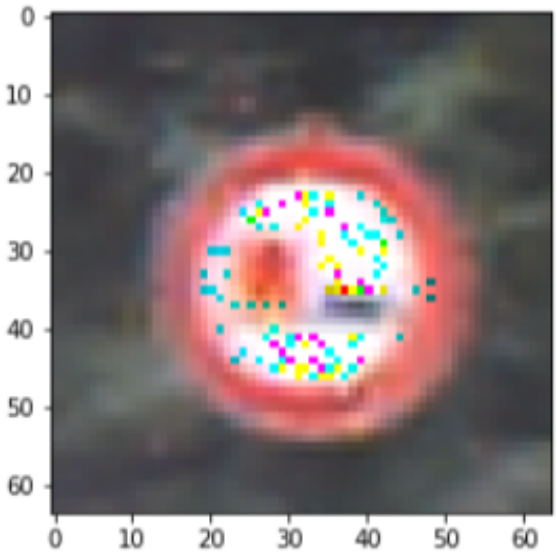


Abbildung 5.4.: Rausch-Degeneration auf Trainingsbild - 36000 Iterationen

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 5.4. Es werden aber keine Änderungen außerhalb des Schildes vorgenommen, wo diese zu erwarten wären und bei vorhergehenden Versuchen auch zu beobachten sind (vgl. Abbildung 5.1). 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 für diese *fremden* Bilder ist, dass eine akzeptable Startkonfidenz vorliegt).

Bei 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**
Model-Agnostic: Der Algorithmus funktioniert unabhängig und ohne Wissen über das zugrundeliegende Modell	**Zeitintensiv:** v.A. die Remote-Variante benötigt größere Zeitspannen
Kontext-Unabhängig: Die Herangehensweise ist nicht auf Bilderkennungen limitiert	**Vorwissen benötigt:** Das Anwendungsfeld des zugrundeliegenden Models muss bekannt sein und ein geeignetes Startbild muss ausgewählt werden
Erweiterbar: 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
Simpel: Der Algorithmus ist einfach implementiert und erläutert, er benötigt keine höhere Mathematik oder Vorwissen zur Thematik *Machine Learning* und der Modelle/Verfahren im Speziellen	Im Remote-Umfeld kann die Degeneration als DDoS wahrgenommen werden und entsprechend frühzeitig unterbunden werden.

Tabelle 5.1.: Zwischenfazit Degeneration

Als besonderen Fall sind solche Modelle zu nennen, die mit jeder Anfrage *hinzulernen*:
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 somit *immun* gegen diesen Angriff.

Solche lebendig 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 [32]. Innerhalb weniger Stunden gelang es

einigen böswilligen Nutzern, dass Tay rassistische Äußerungen von sich gab [31]. Microsoft hat den Service von Tay am zweiten Tag eingestellt.

5.4. 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.

Anpassungen

Für die lokale Implementierung wird 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 deutlich.

Eine zusätzliche, passive Verbesserung wird erzielt, indem die GPU-Acceleration Funktionen von Tensorflow verwendet werden. Diese beschleunigen nicht nur das Training des lokalen Models spürbar, sondern auch die Vorhersagen. Insbesondere die Batch-Variante konnte im Zusammenhang mit der eingesetzten NVIDIA Grafikkarte GTX 1070[1] 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 „Blackbox“ des GI-Wettbewerbs zu erzeugen und remote zu verwen-

[1] https://www.nvidia.com/en-gb/geforce/products/10series/geforce-gtx-1070/

den, 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 ≈30%. Für ein Bild, welches an sich nichts mehr mit einem Verkehrsschild zu tun hat, sind dies immer noch hohe Werte und die Zuverlässigkeit mit der dieser Zusammenhang auftritt lässt einen leichten, inhaltlichen Zusammenhang der Bilder und Modelle erahnen.

5.4.1. 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 [21]: Es wird aus n-Genen das beste ausgewählt.

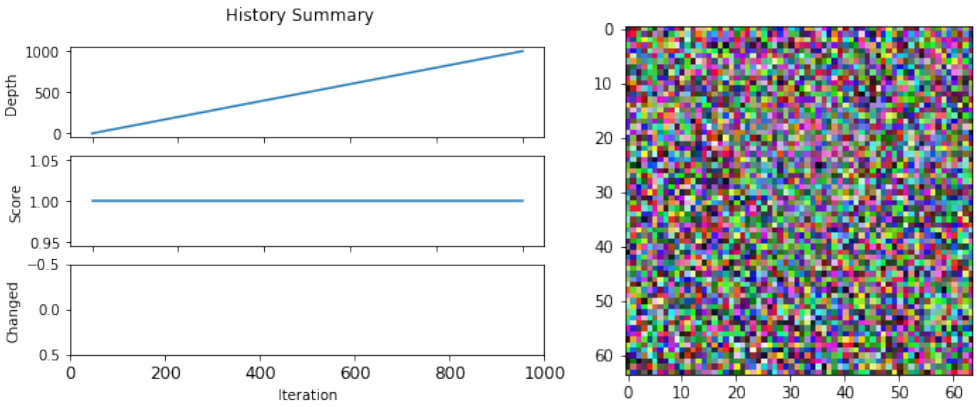


Abbildung 5.5.: Verlauf der Batch-Degeneration mit Batchsize=10 und Tiefe=1000

Abbildung 5.5 zeigt den besseren Verlauf der Batch-Degeneration gegenüber der unveränderten Implementierung in Abbildung 5.3. Auffallend ist die durchgängige Veränderung und die gleichbleibend hohe Konfidenz von mehr als 99.9% [2]. Dieses Verhalten für die Webschnittstelle des GI-Wettbewerbs einzusetzen ist möglich, allerdings wird aufgrund der Wartezeit zwischen den Anfragen davon abgesehen.

Diese Variante profitiert ebenfalls von der *GPU-Acceleration* innerhalb Tensorflows. Selbst ohne Verwendung des CUDA-Frameworks ist ein Tensorflow-Model auf Batch-Verarbeitung ausgelegt. 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 5.5 werden 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 tritt überraschenderweise in der tatsächlichen Implementierung nicht auf. 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 genetisch-selektiven Ansatz.

[2] Innerhalb des Plots wird es auf 1 gerundet

5.4.2. 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 [3].
- 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.

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

5.4.3. 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.

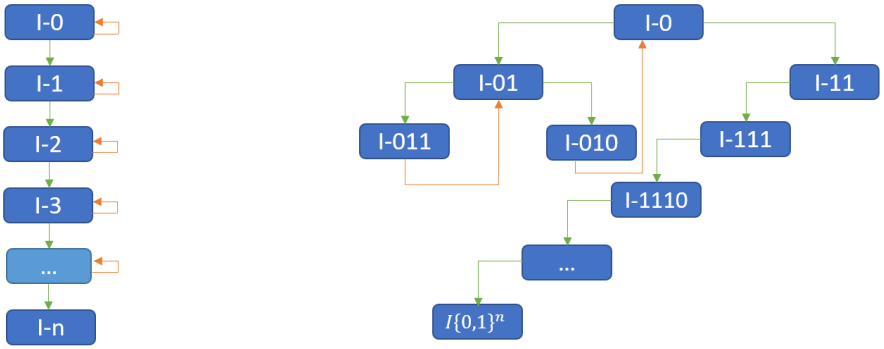


Abbildung 5.6.: Derzeitige Implementierung und eine baum-basierte Implementierung

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.

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.

6. Saliency Maps

Das *Saliency Map* Verfahren wurde erstmals von den Neurowissenschaftlern Itti et al. [25] 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 [25].

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 extrahieren.

6.1. 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 [12].

Diese Kenntnis um CNN und *Saliency Maps* machten sich Simonyan et. al erstmalig im Kontext von Deep Learning zunutze, um die gelernten Merkmale eines CNN anhand von *Saliency Maps* zu visualisieren [34]. 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.

Aus dieser Kenntnis ergeben sich folgende Hypothesen:

- Ein Convolutional Neural Network mit beliebiger Architektur, jedoch trainiert mit demselben Datensatz (GTSRB) wie das NN des GI-Wettbewerbs, lässt sich ähnlich gut angreifen wie das NN des GI-Wettbewerbs selbst. Zu dieser Erkenntnis kamen Papernot et al. [30] 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, dass jeweils erzeugte Bild wird vom CNN als Verkehrszeichen erkannt und mit hoher Konfidenz richtig klassifiziert, ist jedoch für den Menschen nicht als solches erkennbar.

Werden diese Hypothesen erfüllt, so eignet sich das Verfahren der *Saliency Maps* ebenfalls zur erzeugung von *Adversarial Attacks*

6.2. 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 des *Aphrodite*-Modells, 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 [15], erfolgt nun die Implementierung der verschiedenen *Saliency Map*-Verfahren.

Hierzu werden zunächst die Basismethoden *Vanilla* [34], *Guided Backpropagation* [36] und *Integrated Gradient* [37] implementiert.

Unter Verrauschung, gemäß [35], werden die Basismethoden verbessert, wie die Gegenüberstellung der Ergebnisse zu den eingesetzten Verfahren in Kapitel 6.3 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 64×64 Pixel interpoliert. 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 gesonderten Verzeichnis in der Größe 64×64 Pixel im Dateisystem abgespeichert.

Zur abschließenden Evaluierung der erzeugten Bilder am NN des GI-Wettbewerbs, werden alle Bilder zu jedem eingesetzten *Saliency Map* Verfahren im Zyklus von 60 Bildern pro Minute an das Webinterface des Wettbewerbs gesendet. Die Informationen, das heißt die vom NN des GI-Wettbewerbs 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 Ergebnissemit über 90% Konfidenz, zu den übermittelten Bildern enthält.

6.3. Ergebnisse

Mit den verschiedenen *Saliency Map* Verfahren wurden Graustufenbilder erzeugt, in denen die relevanten Bildmerkmale durch helle Pixeleinfärbungen gekennzeichnet sind.

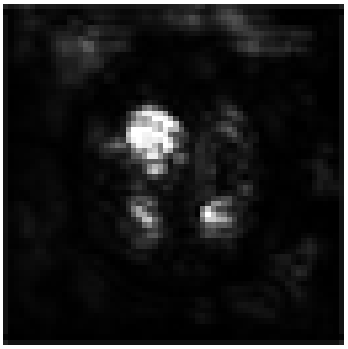
Die mit den Basismethoden *Vanilla*, *Guided Backpropagation* und *Integrated Gradient* erzeugten Bilder werden vom NN des GI-Wettbewerbs mit jeweils 45,83% als „Baustelle“ klassifiziert.

Dieses Ergebnis lässt sich möglicherweise auf die besonderen „Faltungseigenschaften“ von Convolutional Neural Networks zurückführen, dass bildcharakteri-

5.1a: Ursprungsbild

30ger Zone

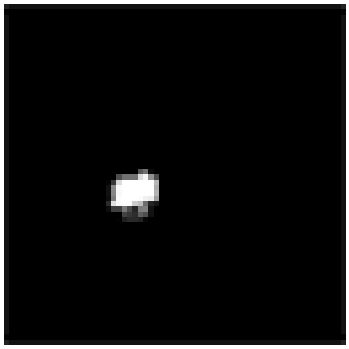
99,39% Konfidenz



5.1b: Smoothed Guided Backprop.

Allgemeines Überholverbot

91,55% Konfidenz



5.1c: Smoothed Vanilla Saliency

30ger Zone

92,83% Konfidenz

Tabelle 6.1.: Ergebnisse der Verfahren Smoothed Guided Backpropagation und Smoothed Vanilla Saliency

sierende Merkmale und die verbundene semantische Information aus den erzeugten Bildern verloren ging.

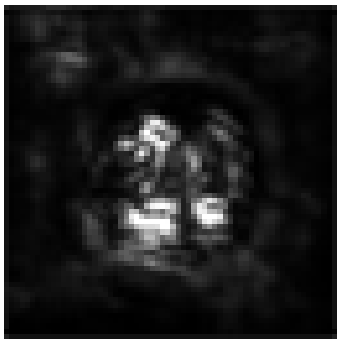
Mit den optimierten *Saliency Map* Verfahren (*Smoothed Vanilla*, *Smoothed Guided Backpropagation* und *Smoothed Integrated Gradient*) Bilder erzeugt werden, die Zielkonfidenzen von mehr als 90% am NN des GI-Wettbewerbs erreichten.

Tabelle 6.1 zeigt exemplarisch, wie aus dem Ursprungsbild (5.1a "30ger Zone“), unter Verwendung des *Smoothed Guided Backpropagation* Verfahrens, ein Bild erzeugt wurde, das vom NN des GI-Wettbewerbs 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 NN des GI-Wettbewerbs als „30ger Zone“ erkannt wurde (5.1c).

Tabelle 6.2 zeigt weitere Beispiele für erfolgreiche Bilder, unter Verwendung der optimierten *Saliency Map*-Verfahren. Wieder kann beobachtet werden, dass die erzeugten Bilder zwar mit einer Zielkonfidenz von über 90% vom NN des GI-Wettbewerbs klassifiziert wurden, die ursprüngliche Klasse jedoch abweicht.

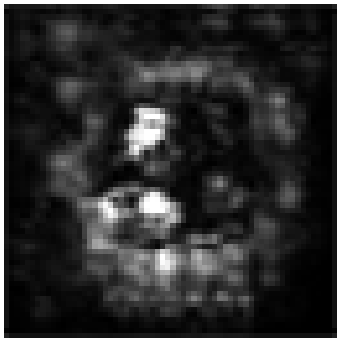
Zusammenfassend konnte hiermit gezeigt werden, dass die in Kapitel 6.1 formulierten Hypothesen zutreffen und sich Bilder – unter Verwendung eines eigens trainierten CNN Modells (*Aphrodite*) sowie verschiedener *Saliency Map* Verfahren – erzeugen lassen, die zwar keine für den Menschen sinnvolle Bedeutung haben,

Ursprungsbild:
20ger Zone
97,54% Konfidenz



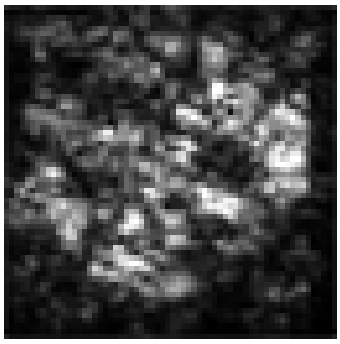
Smoothed Guided Backpropagation:
Allgemeines Überholverbot
97,63% Konfidenz

Ursprungsbild:
30ger Zone
66,07% Konfidenz



Smoothed Guided Backpropagation:
50ger Zone
99,94% Konfidenz

Ursprungsbild:
Kreisverkehr
93,92% Konfidenz



Smoothed Integrated Gradient:
Baustelle
99,99% Konfidenz

Tabelle 6.2.: Ergebnisse Smoothed Guided Backpropagation und Smoothed Integrated Gradient

jedoch beim NN des GI-Wettbewerbs Zielkonfidenzen von mehr als 90% erreichten.

7. 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 $\vec{a}$ des Netzwerks. Der Ausgabevektor $\vec{a}$ 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 $\vec{a}$ 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 Schwellwerten gefunden werden, welche zu einer minimalen Kostenfunktion führen. Dies wird erreicht mit dem Gradient Descent-Algorithmus[39]. Dieser Algorithmus berechnet wiederholt den Gradienten und verbessert iterativ die Gewichte.

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.

7.1. 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 eine Va-

riante des *Gradient Ascent*-Verfahrens darstellt [26]. 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.

7.2. Implementierung

Zunächst wird als vorverarbeitender Schritt ein CNN unter Verwendung der AlexNet Architektur in PyTorch trainiert. Die hierbei verwendete Architektur des AlexNet entstammt [14] und wurde dahingehend modifiziert, dass das AlexNet zu Eingabebildern der Größe 64×64 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 ebenfalls auf eine Bildgröße von 64 x 64 Pixel interpoliert. 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 [28].

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 Konfidenz zu der angegebenen Zielklasse erreicht hat.

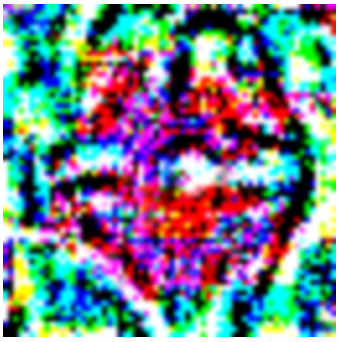
Der Algorithmus wurde für jede der 43 GTSRB-Klassen so lange wiederholt, bis das veränderte Bild die gewünschte Mindest-Konfidenz von 100% erreicht. Zur abschließenden Evaluierung der erzeugten Bilder am NN des GI-Wettbewerbs, werden alle Bilder an das Webinterface des Wettbewerbs gesendet. Die Informationen, das heißt die vom NN des GI-Wettbewerbs 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.

7.3. Ergebnisse

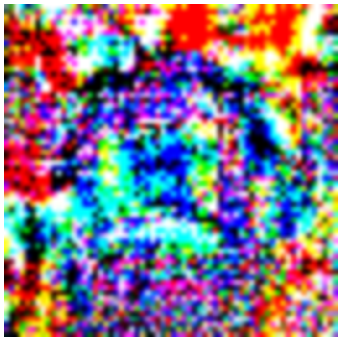
Mit dem *Gradient Ascent*-Verfahren, unter Verwendung der *Targeted Backpropagation*-Methode, wurden 43 Farbbilder – also je ein Bild zu jeder GTSRB-Klasse – erzeugt. Die erzeugten Bilder sind hierbei vom Menschen nicht als Verkehrszeichen wahrnehmbar. Jedoch klassifizierte das NN des GI-Wettbewerbs nur 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 NN des GI-Wettbewerbs ausgegebenen Klasse überein.

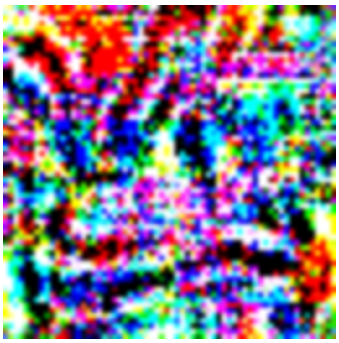
Tabelle 7.1 veranschaulicht vier repräsentative Ergebnisse, welche mit dem *Gradient Ascent*-Verfahren erzeugt und vom NN des GI-Wettbewerbs mit einer Konfidenz von über 90% als Verkehrszeichen klassifiziert werden. Die Abbildungen oben links und unten rechts wurden vom NN des GI-Wettbewerbs als „Einfahrt 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 tatsächlich erkannten Zielklasse ab. Bei Abbildung oben rechts wird im Algorithmus „Ausschließlich links“ als Zielklasse angegeben. Das erzeugte Bild



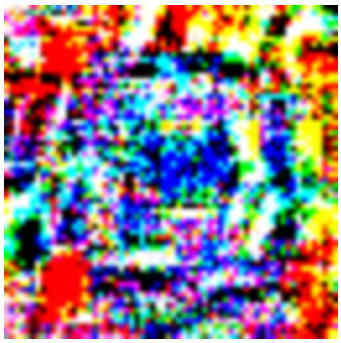
6.1a: Einfahrt verboten
99,99% Konfidenz



6.1b: Kreisverkehr
95,35% Konfidenz



6.1c: Rechts Vorbei
99,50% Konfidenz



6.1d: Kreisverkehr
98,68% Konfidenz

Tabelle 7.1.: Ergebnisse mit dem *Gradient Ascent*-Verfahren (*Targeted Backpropagation*)

wird mit einer Konfidenz von 95,35% vom NN des GI-Wettbewerbs 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 NN des GI-Wettbewerbs erkannt.

8. Fazit

Die vorangehende Arbeit stellt verschiedene bilderzeugende Verfahren zum Angriff einer Verkehrsschilder erkennenden KI vor.

Die Motivation für die Arbeit ist das Forschungsfeld des autonomen Fahrens, in welchem KIs für die optische Erkennung der Umwelt, im Besonderen für das Erkennen von Straßenschildern, verwendet werden sollen. Vorangehende Arbeiten mit Neuronalen Netzen zeigten, dass diese zum aktuellen Zeitpunkt anfällig für Manipulationen sind und fehlerhaft reagieren können. Das gezielte Ausnutzen dieser Fehleranfälligkeit wird in der Forschung als *Adversarial Attack* bezeichnet. Das Ziel der Arbeit ist es verschiedene Verfahren vorzustellen und zu untersuchen, mit denen *Adversarial Attacks* gegen ein NN zur Erkennung von Straßenschildern durchgeführt werden können. Die Aufgabenstellung und das NN werden von der Gesellschaft für Informatik im Rahmen des Wettbewerbs *InformatiCup 2019* gestellt.

Im ersten Schritt der Arbeit werden die Anforderungen an die bilderzeugenden Verfahren zum Angriff der KI des GI-Wettbewerbs analysiert und die Rahmenbedingungen ermittelt. Die Hauptquelle dafür ist die Aufgabenstellung des InformatiCups 2019, sowie die zur Verfügung stehende KI des Wettbewerbs, welche über eine Webschnittstelle erreichbar ist.

In Kapitel 3 werden die verwendeten Technologien der Arbeit vorgestellt. Diese dienen als Grundlage für die verschiedenen Verfahren in den Kapiteln 5 - 7. Anschließend wird erläutert, warum sich *Adversarial Attacks* für die Zielerreichung der Arbeit eignen. Der letzte Abschnitt des Kapitels behandelt die Implementierung eines eigenen Modells zur Klassifizierung von Straßenschildern, welches im Projekt *Aphrodite* genannt wird. Das Modell wird in dem Verfahren lokale Degeneration in Kapitel 5 und im *Saliency Maps* Verfahren in Kapitel 6 eingesetzt. Das eigene Modell dient als Ersatz für das NN des GI-Wettbewerbs, da Anfragen an letzteres Beschränkungen wie einer Anzahl an Requests pro Minute unterliegen. Zusätzlich bietet ein lokales Modell Möglichkeiten zur detaillierten Untersuchung

der Verfahren, da es sich nicht um eine Blackbox handelt.

Zunächst wird der nicht erfolgreiche Ansatz *Greyboxing* vorgestellt und seine Schwächen analysiert.

Das erste vorgestellte erfolgreiche bilderzeugende Verfahren zum Angriff einer Verkehrsschilder erkennenden KI ist die Degeneration. Bei diesem Verfahren wird iterativ ein Bild manipuliert, welches ein Objekt enthält, dass vom Menschen und dem NN des GI-Wettbewerbs als Straßenschild klassifiziert wird. Durch verschiedene Bildbearbeitungsfunktionen wird das Bild solange verändert, bis ein Mensch kein Straßenschild mehr erkennt. Bei jeder Iteration wird darauf geachtet, dass das veränderte Bild von der KI weiterhin mit hoher Konfidenz einer Straßenschild-Klasse zugeordnet wird. Die Degeneration ist ein vergleichsweise simpler Algorithmus, welcher zuverlässig gute Ergebnisse liefert. Der wesentliche Vorteil der Degeneration ist die Unabhängigkeit vom Modell, welches angegriffen werden soll. Es muss kein Transfermodell erstellt werden, um angepasste Angriffe zu erzeugen, sondern der Algorithmus erzeugt Angriffsbilder anhand der Antworten der Blackbox-KI des GI-Wettbewerbs. Diese Tatsache stellt gleichzeitig einen Nachteil der Degeneration dar, da Anfragen an die Webschnittstelle des GI-Wettbewerbs mehr Zeit benötigen als Anfragen an ein lokales NN. Für einen Angriff, bei dem ein hoher Konfidenzwert für eine Straßenschild-Klasse erreicht werden soll, muss über eine Stunde gerechnet werden. Die Zeit erhöht sich weiter, wenn das angegriffene NN dynamisch auf Rauschen reagiert und zugesendete Bilder für ein kontinuierliches Training nutzt. Da eine Grundvoraussetzung des Wettbewerbs jedoch ein unverändertes trainiertes Netz ist, gehört dieses Problem nicht zum Rahmen der Arbeit. Trotz des Zeitaufwands ist die Effektivität der Degeneration vergleichsweise hoch, obwohl es sich im Wesentlichen um einen Brute-Force Angriff handelt.

Im weiteren Verlauf der Arbeit wird gezeigt, dass das NN des GI-Wettbewerbs nicht immer direkt angegriffen werden muss, um hohe Angriffserfolge zu erzielen. So wird bei den verschiedenen *Saliency Map* und dem *Gradient Ascent* Verfahren die Kenntnis über die Möglichkeit zur Transferierbarkeit von Modellen genutzt. Das heißt, dass jeweilige Verfahren setzt zur Erzeugung von schädlichen Bildern jeweils ein lokal trainiertes CNN (*Aphrodite* beziehungsweise *AlexNet*) ein. Die Bilderzeugnisse sind bei den beiden soeben genannten Verfahren in keinem Fall vom Menschen als Verkehrszeichen wahrnehmbar. Jedoch werden einige dieser erzeugten Schadbilder vom NN des GI-Wettbewerbs mit Konfidenzen von zum Teil mehr

als 90% als Verkehrszeichen erkannt. Ein Nachteil an den *Saliency Map* Verfahren ist, dass sie nicht erlauben GTSRB-Klassen gezielt anzugreifen, wohingegen mit dem *Gradient Ascent Verfahren* in 10 von 20 Fällen erfolgreich Schadbilder zur gewünschten GTSRB-Zielklasse erzeugt wurden.

8.1. Diskussion

Die Herausforderungen des Wettbewerbs wirken sich auf die Erfolge der Verfahren aus. Es wird vermutet, dass die Verfahren *Saliency Map* und *Gradient Ascent* bessere Ergebnisse liefern könnten, wenn das verwendete Bild größer als 64×64 Pixel wäre. Der Grund dafür liegt in der zusätzlichen Menge an Manipulationsmöglichkeiten, die eine höhere Auflösung bietet. Des Weiteren kann nichts über die Validierungsgenauigkeit des NN des Wettbewerbs gesagt werden, weshalb auch die nicht zielgerichteten Täuschungsbilder als gutes Ergebnis betitelt werden. Die Aufgabe des Wettbewerbs bestand nicht darin manipulierte Bilder gewisser Klassen zu generieren, sondern Bilder zu erzeugen, die für die KI des Wettbewerb mit hoher Konfidenz einer beliebigen Klasse zugeordnet werden. Gleichzeitig soll das Bild für einen Menschen nicht als Straßenschild erkennbar sein.

Orientiert man sich beim Vergleich des *Saliency Map* und *Gradient Ascent* Verfahrens trotzdem am Maßstab der Klassengenauigkeit, ist das zuletzt genannte zu bevorzugen. Denn bei diesem Verfahren ist es möglich das zu generierende Bild vorab genau einer Klasse zuzuordnen.

Mit dem dritten Verfahren, der Degeneration, lassen sich die *Saliency Map*- und das *Gradient Ascent*-Verfahren nur schwer vergleichen. Die Degeneration erreicht voraussichtlich schneller Erfolge bei einem Angriff auf ein unbekanntes NN. Sie benötigt weniger Vorlauf und geringere Kenntnisse über eine anzugreifende KI. Dafür benötigt die Degeneration bei der Erzeugung von Bildern, mit denen fälschlicherweise hohe Konfidenzen erreicht werden sollen, mehr Zeit. Denn nachdem das entsprechende Umfeld für *Saliency Map*-Verfahren oder das *Gradient Ascent*-Verfahren mit einem eigenen lokalen Netz, Bibliotheken und Code erzeugt wurde, können innerhalb des Foolings deutlich schneller zuverlässige Bilder erzeugt werden.

8.2. Weiterführende Arbeiten

Die Ergebnisse dieser Arbeit liefern Ansätze für weitere zukünftige Forschungen. 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.

Im nächsten Schritt kann das Anwendungsfeld der Attacken erweitert werden. Eine spannende Arbeit wäre zum Beispiel 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* manipulieren.

Auch die anderen Verfahren sind insbesondere dann geeignet, wenn ein Modell offenliegt. Der Sprachassistent-Hersteller Mycroft beispielsweise setzt auf Open-Source und stellt dementsprechend auch das (allgemeine) trainierte Modell bereit. Zu bemerken ist hierbei noch, dass der Nutzer bei der Durchführung von ersten Aktionen den Sprachassistenten auf seine Aussprache konfiguriert und das Training des Modells damit weiterführt.

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 auslotet 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 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 Model auf ein unbekanntes Model 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 der Degeneration 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] Anaconda Accelerate — Anaconda 2.0 documentation. https://docs.continuum.io/accelerate/2.2/.

[2] Home - Keras Documentation. https://keras.io/.

[3] Managing environments — Conda documentation. https://conda.io/docs/user-guide/tasks/manage-environments.html.

[4] Matplotlib: Python plotting — Matplotlib 3.0.2 documentation. https://matplotlib.org/.

[5] NumPy — NumPy. http://www.numpy.org/.

[6] OpenCV library. https://opencv.org/.

[7] Pillow — Pillow (PIL Fork) 5.3.0 documentation. https://pillow.readthedocs.io/en/5.3.x/.

[8] Project Jupyter. https://www.jupyter.org.

[9] Requests: HTTP for Humans™ — Requests 2.21.0 documentation. http://docs.python-requests.org/en/master/.

[10] SciPy.org — SciPy.org. https://www.scipy.org/.

[11] Stack Overflow Developer Survey 2017. https://stackoverflow.com/insights/survey/2017/?utm_source=so-owned&utm_medium=social&utm_campaign=dev-survey-2017&utm_content=social-share.

[12] Unsupervised Feature Learning and Deep Learning Tutorial. http://ufldl.stanford.edu/tutorial/supervised/FeatureExtractionUsingConvolution/.

[13] What is Python? Executive Summary. https://www.python.org/doc/essays/blurb/.

[14] Datasets, Transforms and Models specific to Computer Vision: pytorch/vision. https://github.com/pytorch/vision, January 2019. original-date: 2016-11-09T23:11:43Z.

[15] 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.

[16] BMBF-Internetredaktion. Das Auto von morgen: autonom, sicher, effizient - BMBF. https://www.bmbf.de/de/automatisiertes-fahren-4158.html.

[17] BMW. Autonomes Fahren - Die 5 Stufen zum selbstfahrenden Auto. https://www.bmw.com/de/automotive-life/autonomes-fahren.html.

[18] ZDF dpa. Ein Unfall, der am Branchen-Versprechen nagt. https://www.zdf.de/nachrichten/heute/uber-unfall-schlecht-fuer-vertrauen-in-technik-100.html.

[19] Marcus Efler. Autonomes Fahren: Das Ende des Lenkrads. https://www.zeit.de/mobilitaet/2018-01/autonomes-fahren-ces-2018-deutsche-autos-volkswagen, January 2018.

[20] Gesellschaft für Informatik e.V. InformatiCup2019-Irrbilder.pdf. https://gi.de/fileadmin/GI/Hauptseite/Aktuelles/Wettbewerbe/InformatiCup/InformatiCup2019-Irrbilder.pdf, October 2018.

[21] Ingrid Gerdes, Frank Klawonn, and Rudolf Kruse. *Evolutionäre Algorithmen: Genetische Algorithmen—Strategien und Optimierungsverfahren—Beispielanwendungen*. Springer-Verlag, 2013.

[22] Jochen Heistermann. *Genetische Algorithmen: Theorie und Praxis evolutionärer Optimierung*, volume 9. Springer-Verlag, 2013.

[23] Sandy H. Huang, Nicolas Papernot, Ian J. Goodfellow, Yan Duan, and Pieter Abbeel. Adversarial attacks on neural network policies. *CoRR*, abs/1702.02284, 2017.

[24] Udacity India. Why Python is the most popular language used for Machine Learning. https://medium.com/@UdacityINDIA/why-use-python-for-machine-learning-e4b0b4457a77, March 2018.

[25] 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.

[26] 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.

[27] Markus Maurer, J. Christian Gerdes, Barbara Lenz, and Hermann Winner, editors. *Autonomes Fahren: Technische, rechtliche und gesellschaftliche Aspekte*. Springer Vieweg, Berlin, 2015.

[28] 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.

[29] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, Oct 2010.

[30] 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.

[31] 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.

[32] 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.

[33] Eberhard Schöneburg, Frank Heinzmann, Sven Feddersen, et al. Genetische algorithmen und evolutionsstrategien. *Bonn: Addison-Wesley*, pages 185–218, 1994.

[34] 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.

[35] 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.

[36] 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.

[37] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic Attribution for Deep Networks. *arXiv:1703.01365 [cs]*, March 2017. arXiv: 1703.01365.

[38] PyTorch Core Team. torchvision: image and video datasets and models for torch deep learning. https://github.com/pytorch/vision.

[39] XueFei Zhou. Understanding the Convolutional Neural Networks with Gradient Descent and Backpropagation. *Journal of Physics: Conference Series*, 1004(1):012028, 2018.

A. Anhang

Aphrodite Summary

Layer (type)	Output Shape	Param #
conv2d_30 (Conv2D)	(None, 64, 64, 32)	896
conv2d_31 (Conv2D)	(None, 62, 62, 32)	9248
max_pooling2d_15 (MaxPooling	(None, 31, 31, 32)	0
dropout_15 (Dropout)	(None, 31, 31, 32)	0
conv2d_32 (Conv2D)	(None, 31, 31, 64)	18496
conv2d_33 (Conv2D)	(None, 29, 29, 128)	73856
max_pooling2d_16 (MaxPooling	(None, 14, 14, 128)	0
dropout_16 (Dropout)	(None, 14, 14, 128)	0
flatten_6 (Flatten)	(None, 25088)	0
dense_18 (Dense)	(None, 128)	3211392
dense_19 (Dense)	(None, 128)	16512
dense_20 (Dense)	(None, 43)	5547

Total params: 3,335,947
Trainable params: 3,335,947
Non-trainable params: 0