TECHNISCHE HOCHSCHULE NÜRNBERG
GEORG SIMON OHM

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

To be done

title: Fooling a TrafficSign-AI
authors: Leonhard Applis, Peter Bauer, Andreas Porada und Florian Stöckl

Kurzfassung

To be done

Titel:	Bilderzeugungsverfahren zur Überlistung einer Verkehrsschilder erkennenden KI
Authoren:	Leonhard Applis, Peter Bauer, Andreas Porada und Florian Stöckl

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

SQL	Structured Query Language
NN	Neuronales Netz
DNN	Deep Neural-Network
CNN	Convolutional Neural-Network
GTSRB	German Traffic Sign Recognition Benchmark

1 Einleitung

Der Traum von einem autonom gelenkten Automobil ist so alt wie das Automobil selbst [10]. 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[2] und Berichte von Automobilherstellern[3] und der Presse[5] zeigen. Die Forschung im Bereich **KI!** (**KI!**) hat mittlerweile einen Stand erreicht, der für die Automatisierung des Autos genutzt werden kann. Ein besonderer Fokus liegt hierbei auf dem Erkennen von Bilder aus der Umwelt mittels einer **KI!**. Im Straßenverkehr ist besonders die Erkennung von Straßenschildern von Bedeutung.

1.1 Problemstellung

Im Bereich der Bilderkennung erreichen neuronale Netze bahnbrechende Erfolge. Im Zuge der Forschung trat allerdings ein neues Phänomen auf, die sogenannten *Adversarial Attacks* [8].

Innerhalb dieser Angriffe werden gezielt Gewichte stimuliert, um gewünschtes Feedback des neuronalen Netzes zu erzielen. Die dabei erzeugten Fragmente haben selten etwas mit einem *echten* Bild zu tun - sie wirken entweder wie Rauschen oder moderne Kunst.

Da diese präparierten Bilder eben nicht aussehen, wie beispielsweise ein Verkehrsschild, kann ein Mensch schwer erkennen ob ein Angriff unternommen wird.

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 ohnehin geprägt ist durch die Debatte über *Vertrauen in Technik* [4], können erfolgreiche Angriffe zu einem forschungsschädlichen Misstrauen führen - und das gesamte Themengebiet frühzeitig begraben.

Um gegen Adversarial Attacks vorzugehen, werden zunächst einige dieser *Irrbilder* benötigt. Anschließend können, um das neuronale Netz zu härten, Tests durchgeführt werden und die Angriffe berücksichtigt werden.

1.2 Ziel der Arbeit

Ziel dieser Arbeit ist es, Methoden und Herangehensweisen vorzustellen, um die Aufgabenstellung des Informaticup 2019 zu erfüllen:

Hyperlink!

Hierbei soll ein neuronales Netz, welches sich hinter einer Webschnittstelle verbirgt und Verkehrsschilder erkennt, erfolgreich *überlistet* werden - es sollen absichtlich Bilder erzeugt werden, welche für den Menschen keine Verkehrsschilder sind aber mit einer Konfidenz von über 90% als solche erkannt werden.

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

Der erweiterte Rahmen dieser Arbeit umfasst eine Dokumentation der Methoden sowie Verbesserungen und Schlussfolgerungen aus den Implementierungen zu ziehen.

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 Informationen über die Webschnittstelle gesammelt und aufbereitet. Die Webschnittstelle stellt eine zentrale Rahmenbedingung der vorliegenden Arbeit dar.

Im Abschnitt 2.2 wird hierfür zunächst der German Traffic Sign Recognition Benchmark (GTSRB) vorgestellt, welcher für das Training der Webschnittstelle verwendet wurde. Dieses Datenset bildet ebenfalls einen zentralen Ausgangspunkt für einige der verfolgten Ansätze.

Anschließend werden in Abschnitt 3.2 die Eigenschaften des Modells zusammengefasst. Diese bestehen zum einen aus den offiziellen Angaben der Gesellschaft der Informatiker, zum anderen aus gewonnenen Erkenntnissen. Dieses Kapitel bildet die Grundlage, um die Schnittstelle einzuschätzen.

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 passender Bildveränderungen erzielen höhere Iterationen unkenntliche Ergebnisse.

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 Webschnittstelle 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 waren.

Desweiteren werden in Kapitel 5 verschiedene Methoden zur Erzeugung von sog. *Saliency Maps* (dt. Ausprägungskarte) vorgestellt. Unveränderte Bilder mit einer hohen Konfidenz werden ausgewählt, um die einzelnen Pixel hervorzuheben, welche für die Klassifikation den höchsten Einfluss hatten.

In Kapitel 6 wird das Verfahren des *Gradient Ascent* beschrieben und evaluiert. Dabei wird mithilfe einer *targeted Backpropagation* für jede Klasse ein Bild erzeugt, der die Funktion in Richtung der Zielklasse maximiert, bis die enthaltenen Merkmale eine hohe Konfidenz in der gewünschten Klasse ermöglichen.

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

2 Anforderungsanalyse und Rahmenbedingungen

2.1 Rahmenbedingungen des Informaticups

In das Kapitel kommen die Dinge die wir über die Trasi-AI wissen

2.2 Eigenschaften des bereitgestellten neuronalen Netz

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 orginal 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 [1] (in der Version 5.7.4) zusammen mit der objektorientierten höheren Programmiersprache Python [2] (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 Softwarebibliothe zur beschleunigten Codeausführung auf Intel Prozessoren sowie NVIDIA Grafikkarten **??**.

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

[1] https://jupyter.org/
[2] 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 Bildtransformationsoperationen 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

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 Angriffen auf ein Blackbox Modell

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

in Abschnitt 2.2 ermöglicht keine genauen Informationen über die verwendete Architektur oder anderen Details. Dennoch wurde bereits bestätigt, dass es selbst für "Blackbox" Modelle möglich ist, Irrbeispiele und Bilder zu erzeugen.

Papernot et al.[13] 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. **SVM!** (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 die Umsetzung der verschiedenen Ansätze wird ein selbst modelliertes Neuronales Netz (NN) als Substitute verwendet.

Das (am meisten verwendete) 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.

Netzzusammenfassu als Tabelle in den Anhang

Link in den Anhang

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

Da nicht sicher war, welche Interpolationsfunktion innerhalb der Remote-Schnittstelle verwendet wurde, wurde für das Training jedes Bild mehrfach interpoliert und ebenfalls mehrfach für das Training verwendet. Für die Testdaten wurde eine zufällige Interpolationsfunktion ausgesucht.

Aphrodite erreichte eine Genauigkeit von 96.5 % auf die Trainingsdaten. Eine Übersicht

über die Trainingsparameter findet sich im Repository unter /DegenerationCode/Training.py.

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

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

4 Degeneration

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

Die Benennung schöpft sich aus der Nähe zu genetischen Algorithmen [7], allerdings aus einer invertierten Perspektive: Um auf unbekannte Modelle einzugehen, wird hierbei von einem korrekt erkannten Bild *weggearbeitet*. Anstatt allerdings *gute Gene* zu kombinieren [16], wird ein Genmutation erzeugt, und überprüft ob es den Anforderungen der Klassifizierung noch genügt.

Zunächst wird das Konzept anhand von Pseudocode genauer erläutert. Anschließend wird die Implementierung des Algorithmus für die Verwendung der unbekannten Trasi-AI vorgestellt, und den Abschluss dieses Kapitels bildet eine lokale Implementierung zuzüglich einiger Verbesserungen, welche sich aufgrund der Limitierungen des Zugriffes auf die *remote-AI* 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 unkenntlich. Zusätzlich sollten allerdings weitere Parameter als Abbruchkriterien aufgenommen werden, konkret

eine Anzahl an Gesamt-Iterationen und ein Abbruch, sollten 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 der 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 überschlagen 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 festgefahren hat. Üblicherweise kann man erkennen, wenn die Manipulationsfunktion *zu stark* ist, beziehungsweise der Schwellwert zu niedrig gewählt wurde.

Der oben genannte Algorithmus lässt sich auch für text- oder sprachbasierte Klassifikationen adaptieren.

Hierfür müssen lediglich andere Manipulations- und Klassifizierungsfunktionen gewählt werden.

4.2 Implementierung Remote

Im Rahmen des Wettbewerbs wird mit einer Rest-API gearbeitet, welche in Abschnitt 3.2 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 wurde 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(im*
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 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.

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

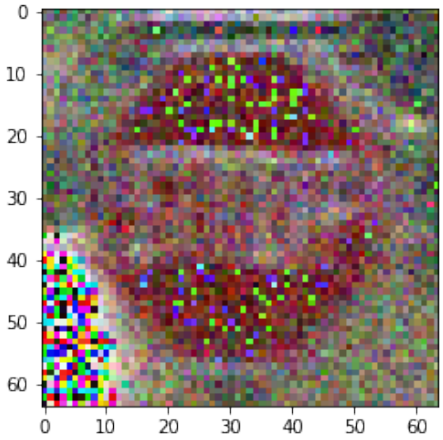


Abbildung 4.1: Rausch - Degeneration mit 600 Iterationen

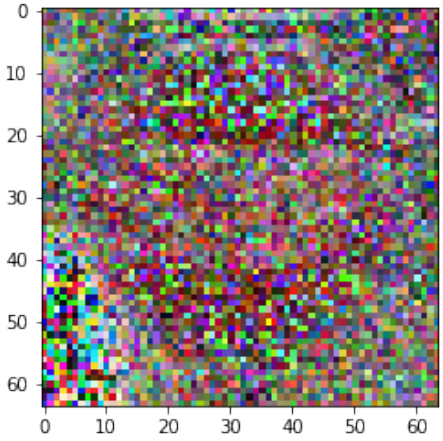


Abbildung 4.2: Rausch - Degeneration mit 4000 Iterationen

die jeweils produzierte Genauigkeit der jeweiligen Iteration (nicht nur die der akzeptierten), und der letzte Plot visualisiert diejenigen Iterationen, an welchen eine Änderung stattgefunden hat (weißer Strich) oder keine (schwarzer Strich). Innerhalb der Implementierung werden ebenfalls standartmäßig diese Plots erzeugt.

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

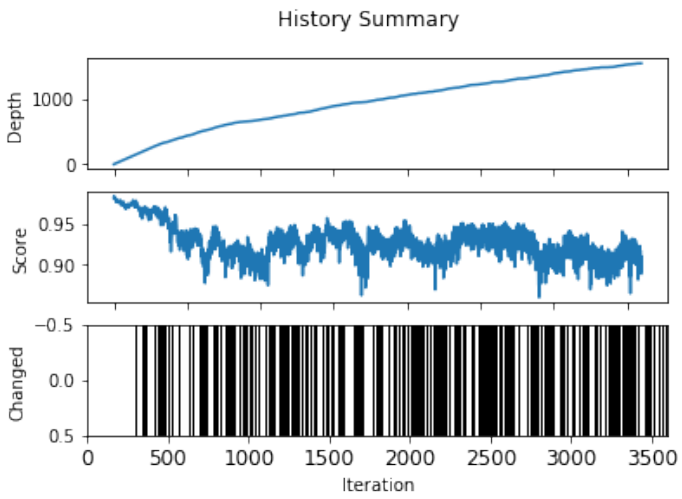


Abbildung 4.3: Plot der Rausch-Degeneration

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.
Die AI innerhalb der Schnittstelle 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.
Dieses Problem hat sich herauskristallisiert, als über einen längeren Zeitraum ($\tilde{1}0$ 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 verwendet oder *GTSRB-fremde* Bilder (vorausgesetzt, sie besitzen eine akzeptable Startkonfidenz).

Innerhalb der lokalen Implementierung ist dieses Problem ebenfalls aufgetreten,

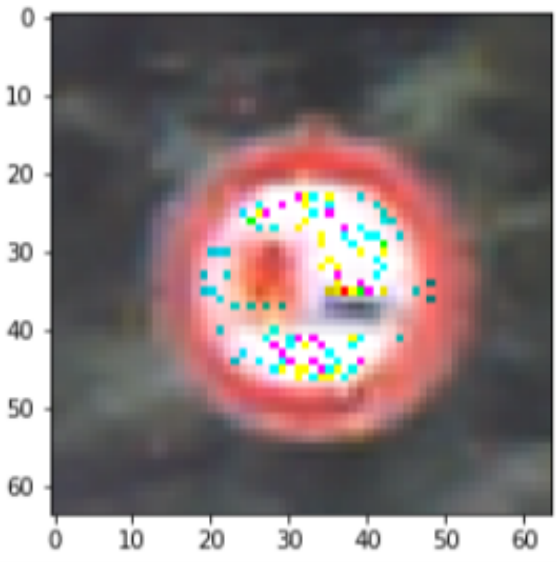


Abbildung 4.4: Rausch-Degeneration auf Trainingsbild - 36000 Iterationen

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: 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 *korrekte* 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 ganze Bandbreite an Angriffen anfällig sind. Als einfaches Beispiel ist der Chatbot *Tay* von Microsoft zu nennen: Dieser sollte angenehme Unterhaltungen mit Twitternutzern führen und aus den enstandenen Konversationen kontinuierlich weiterentwickelt werden [15]. Innerhalb weniger Stunden gelang es einigen böswilligen Nutzern, dass Tay rassistische Äußerungen von sich gab [14]. Microsoft hat den Service von Tay am zweiten Tag eingestellt.

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:** Der Sinn 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 schlechter Ergebnisse - gerade *professionelle* 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 4.1: Zwischenfazit Degeneration

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 Umfanges verzichtet - sie befinden sich im Anhang.

Anpassungen

Für die lokale Implementierung wurde zunächst ein eigenes Modell (von Grund auf) mithilfe der GTSRB-Daten trainiert. 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 für die Schnittstelle benötigt wird, sowie die Wartezeit. Letzteres erhöht die Geschwindigkeit des Algorithmus maßgeblich.

Eine zusätzliche, passive Verbesserung wird erzielt, indem die Verwendung der *GPU-Acceleration* von Tensorflow eingebunden wurde. Diese beschleunigte nicht nur das Training des lokalen Models maßgeblich, sondern auch die Vorhersagen, insbesondere für die Batch-Variante, wurden um ein vielfaches (≈Faktor 20) schneller.

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 Irrbilder zu erzeugen und Remote zu verwenden, stellte sich heraus, dass die lokalen Irrbilder 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 Irrbilder 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 der jeweilig erstellen AI Werte >90%, und bei der anderen Implementierung zuverlässig einen Score von ≈30%.

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

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 Threshold erfüllt wird mit dem besten Bild weitergearbeitet. Dieses Verfahren entspricht deutlich näher der Genoptimierung von genetischen Algorithmen [6]: Es wird aus n-Genen das beste ausgewählt.

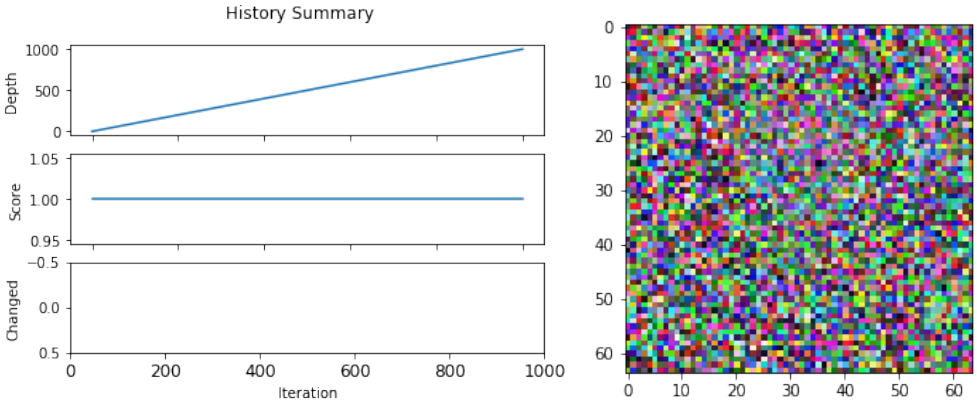


Abbildung 4.5: Verlauf der Batch-Degeneration mit Batchsize=10 und Tiefe=1000

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% [1].

Dieses Verhalten für Remote 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 die 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 4.5 wurden ca. 15 Minuten benötigt, was knapp

[1] Innerhalb des Plots wird es auf 1 gerundet

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 weißt das am wenigsten veränderte Bild den nähesten 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 [2].

[2] Dieses Problem ist sicherlich lösbar - allerdings tief verankert im Bereich der Parallelverarbeitung und somit nicht im Scope dieser Arbeit

- 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.

Schaubild: Liste vs. Tree

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.

5 Saliency Maps

5.1 Konzept

Ein verbreitetes Verfahren aus dem Bereich Computer Vision zur Visualisierung relevanter Pixel ist die Erstellung von *Saliency Maps* (dt. Ausprägungskarten). Diese können verwendet werden, um die Qualität, sprich Aussagekraft, jedes einzelnen Bildpunkts ersichtlich zu machen. Simonyan et al. [17] führt die grundlegende Methodik entsprechend Saliency Maps weiter aus und unterscheidet zwischen einer allgemeinen "'Klassen-Definierenden"' Saliency Map sowie einer Saliency Map zu einem gegebenen Eingabebild entsprechend der Zielklassen.

Gebräuchliche Anwendung dieses Verfahrens im Machine Learning Bereiches betreffen der Darstellung von High Level Features einzelner Neuronen [9] im Bezug auf eine gezielte Klasse. Diesen Ansatz weiterverfolgend, entsprechen die erzeugten Saliency Maps im letzten Layer einer starken Neuronenaktivierung, die einer "High Level"' Darstellung der gezielten Klasse entspricht, die vom NN für das Bild errechnet wurde.

Diese Saliency Map sollte also im Rückschluss mit einer hohen Konfidenz bezüglich der entsprechenden Klasse klassifiziert werden, wenn das erzeugte Bild als Eingabe verwendet wird. Daraus ergibt sich die Hypothese, mit Hilfe dieser Visualisierung für den Menschen semantisch nicht erkennbare Verkehrszeichen Bilder zu erzeugen, die hohe Zielkonfidenzen von 0.9 oder mehr erreichen. Die beschriebene Methode wird in der Implementierung als "'Vanilla Saliency"' bezeichnet.

5.2 Implementierung

Die Verfahren zur Erzeugung der Saliency Maps setzen eine identische Datenvorverarbeitung voraus. Die Trainingsbilder des GTSRB Datensatz wurden konvertiert und entsprechend

der Eingabeparameter des Aphrodite-Modells als PNG mit Größe 64×64 im Dateisystem abgespeichert.

Nachfolgend wird der vollständige Datensatz lokal am eigenen Modell klassifiziert und alle Bilder mit einer Konfidenz von 100% separat gesammelt.

Die vorgestellten Verfahren Vanilla Saliency[17], Guided Backpropagation[18] und Integrated Gradient[19] werden jeweils in einer ungefilterten und und geglätteten Variante verwendet entsprechend einer auf die Aufgabenstellung angepasster Implementation auf der Basis von [1].

Ausgegeben wird ein Graustufenbild in der die relevanten Pixelbereiche durch hellere Einfärbung gekennzeichnet werden.

Diese Bilder wurden danach an das Trasi Webinterface geschickt und das Resultat der Klassifikation abgespeichert. Zusätzlich wird pro Verfahren ein Logfile erstellt, welches alle Bilder mit einer Konfidenz >0.9 sowie die geschätzte Klasse und das Ursprungsbild festhält.

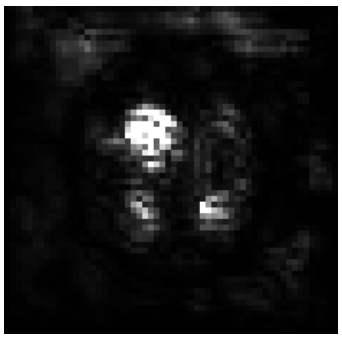
5.3 Ergebnisse

Aus der Implementation konnten folgende Ergebnisse zu den 6 vorgestellten Variationen gesammelt werden.

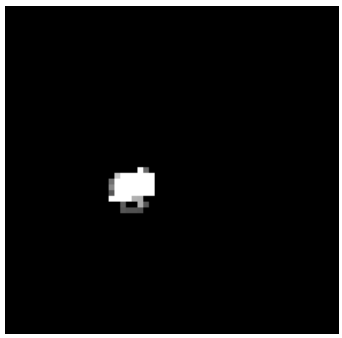
Alle ungeglätteten Verfahren konnten keine positiven Ergebnisse bezüglich der Bilderzeugung mit einer Konfidenz > 0.9 liefern. Dies lässt sich möglicherweise auf die besonderen Einschaften von Convolutional Neural-Networks (CNNs) zurückführen, dass ohne Glättung in dem - ohne farbkanäle - bereits dimensionsreduziertem Bild keine ausgeprägten Kanten vorhanden sind und nach dem “Falten“ Informationen verloren gehen. Desweiteren konnte am Blackbox Modell beobachtet werden, dass alle erzeugten ungegeglätteten Saliency Maps mit einer Konfidenz von 0.45834911 als Klasse “Baustelle“ erkannt werden.

Dahingegen konnten bei allen geglätteten Varianten Bilder erzeugt werden, die vom Blackboxmodell mit hoher Sicherheit eine Klasse zugeordnet wurden. Es konnten einige unerwarteten Anomalien bei der Klassifizierung am Fremdmodell beobachtet werden. Tabelle 5.1 zeigt wie aus Bild 5.1a (Höchstgeschwindigkeit (30)) mit den Verfahren “Smoothed Guided Backpropagation“ (vgl. 5.1b) und “Smoothed Vanilla Saliency“ (vgl. 5.1c) ein Bild erzeugt, welche mit einer Konfidenz >0.9 vom Webinterface

5.1a Ursprungsbild
Höchstgeschwindigkeit (30)



5.1b Guided Backprop.
Überholverbot
0.9155



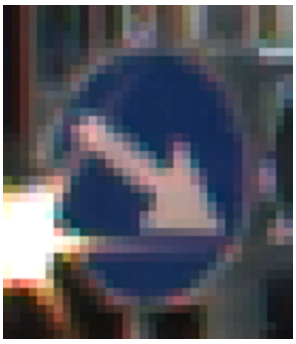
5.1c Vanilla Saliency
Höchstgeschwindigkeit (30)
0.9283

Tabelle 5.1: Ergebnisse von verschiedenen Verfahren am selben Bild.

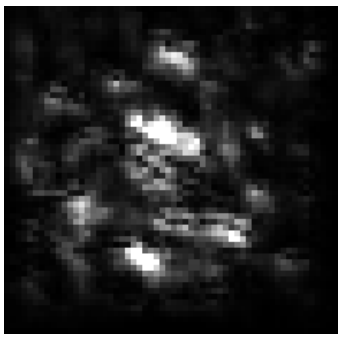
erkannt wurden. Dabei muss beachtet werden, dass die Bilder als unterschiedliche Klassen eingestuft wurden. "Vanilla Saliency" stimmt überein (Höchstgeschwindigkeit (30), Konfidenz 0.9283), "Guided Backpropagation" wurde vom Fremdmodell einer anderen Klasse zugeordnet (Überholverbot, Konfidenz 0.9155).

Tabelle 5.2 zeigt weitere Beispiele für erfolgreiche Saliency Maps. Wieder kann beobachtet werden, dass Bilder nicht mehr der ursprünglichen Klasse zugeordnet werden, aber trotzdem Konfidenzen >0.9 erreicht wurden.

Zusammenfassend konnte gezeigt werden, dass es mithilfe der Erstellung von Saliency Maps an einem eigenen Modell Bilder erzeugt werden können, welche von einem unbekanntem NN mit hohen Konfidenzen verschiedenen Klassen zugeordnet werden. Die Erfolgsrate in Relation zur Anzahl erzeugter Bilder und dem Rechenaufwand ist dabei jedoch gering und es herrscht aktuell keine verlässliche Aussagekraft darüber, welcher Klasse das erzeugte Bild zugeordnet wird.

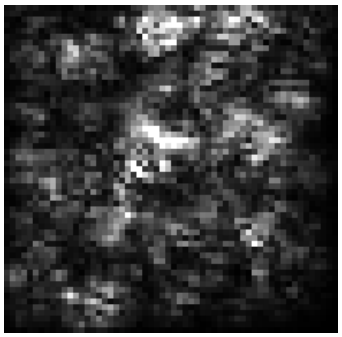


Rechts Vorbei



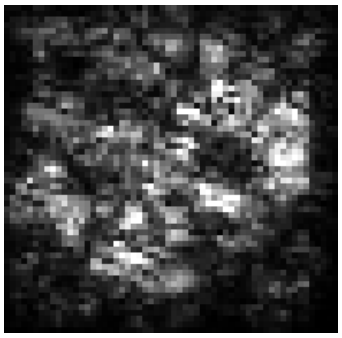
Einmalige Vorfahrt
0.9678

Einfahrt Verboten



Überholverbot
0.9571

Kreisverkehr



Baustelle
0.9999

Tabelle 5.2: Weitere Beispiele für Irrbilder mit Konfidenz >0.9 inklusive entsprechendem Ursprungsbild.

6 Gradient Ascent

6.1 Konzept

Mit Zugriff auf die berechneten Gradienten des Modells, können diese verwendet werden, um gezielt Bilder zu verändern. Dieses Verfahren wird als *Gradient Ascent* bezeichnet, welches als Pendant zum *Gradient Descent*, einem Verfahren zum Training von neuronalen Netzen, verstanden werden kann. Demnach bedeutet der Ascent (dt. Aufstieg), dass die Eingabe verändert wird, um die Funktion und Neuronenaktivierung zu maximieren. Mithilfe einer *targeted Backpropagation* werden demnach die Gradienten berechnet, die zum Erreichen einer Zielausgabe (sprich Klasse) am relevantesten sind. Die Gradienten werden dann von dem ursprünglichen Bild (üblicherweise zufallsgeneriert) mit einem Skalierungsfaktor mathematisch subtrahiert. Dabei kann man beobachten, dass relevante Pixel stärker mutiert werden und unwichtige Pixel weitgehend unverändert bleiben.

Die Methode sollte nach aussage von Ngunyen et al. [11] auf für selbst ausgewählte "echte" Bilder funktioniere. Um jedoch die Erkennbarkeit der ursprünglichen Bildes zu erhalten, sollten kleinere Skalierungsfaktoren verwendet werden. So wird ein "über das Ziel hinausschießen" verhindert und nur so gering wie nötig abgeändert, benötigt dann allerdings auch mehr Iterationsdurchläufe. Diese Implementation wurde allerdings nicht weiter verfolgt, da das Konzept auch mit Zufallsbildern bestätigt werden konnte.

6.2 Implementierung

Der verwendete Code zur Erzeugung der Schadbilder mithilfe des Gradient Ascent Verfahrens basiert auf einer modifizierten Version von [12]. Desweiteren wird ein angepasstes Modell basierend auf der AlexNet Architektur verwendet, mit Eingabedimensionen

$64 \times 64 \times 3$ und einer Softmax Ausgabe von den 43 Klassen des GTSRB Datensatz. Dieser Datensatz wurde für ein Training des NN verwendet. Die final verwendete Version unseres AlexNet erreichte in der Validierung eine Genauigkeit von 0.89

Es wird ein zufälliges Bild erzeugt, welches als Eingabeparameter für das trainierte NN verwendet wird. Daraus werden durch Anwendung der *categorical cross entropy* die Gradienten entsprechend für eine Zielklasse berechnet. Diese Gradienten werden nicht verwendet, um die Parameter des NN zu ändern wie es beim Training der Fall wäre, sondern um das eingegebene Bild gezielt zu modifizieren, indem die entsprechenden Pixel der aktivierten Neuronen je nach Aktivierungsstärke den Pixelwerten addiert werden. Dieser Prozess wird iterativ wiederholt, bis das veränderte Bild die gewünschte Mindestkonfidenz einer ausgewählten Klasse erreicht hat.

Die verwendete Implementation modifiziert Bilder, bis im lokalen NN eine Konfidenz von 1 erreicht wurde. Diese Bilder werden im darauffolgenden Schritt an das Blackbox Modell gesendet und klassifiert.

Ergebnisbilder mit einer Konfidenz >0.9 für irgendeine Klasse, wurden in eine gesonderte Textdatei mit Referenz auf den Speicherort, die erkannte Klasse und zugehörige Konfidenz abgespeichert.

6.3 Ergebnisse

Als Ergebnis des Algorithmus wurden 43 Bilder respektive der 43 Klassen des letzten Layers erzeugt. Die Evaluation am zu überlistenden NN ergab Konfidenzen >0.9 für 20 der 43 Bilder. Durch die Ähnlichkeit zwischen verschiedenen Klassen bzw. Straßenschildern, beispielsweise Höchstgeschwindigkeit 30 und 50, wurden nur 10 verschiedene Klassen in den 20 "erfolgreichen" Bildern erkannt. Nachfolgende Grafiken 6.1 zeigen verschieden Beispiele für die erzeugten Bilder mit initialen Zufallswerten. Wieder kam es zu Ergebnissen, welche eine andere Klasse lieferten, als für die Erzeugung verwendet wurde. Das für die Klasse *Links Abbiegen* erzeugte Bild wurde vom Trasi-NN beispielsweise als *Kreisverkehr* erkannt.

Abschließend kann bestätigt werden, dass das Gradient Ascent verfahren die erzeugung von Täuschungsbildern ermöglicht. Einschränkungen des Verfahrens belaufen sich auf die Architektur und Ähnlichkeit des NN. Es steht die Annahme, dass das verfahren mit größeren Bildern bessere Ergebnisse liefert.

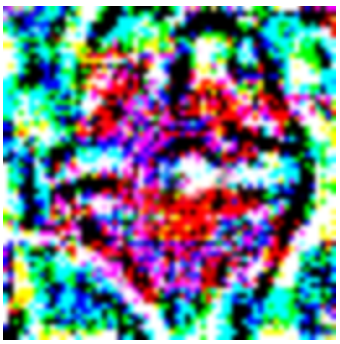
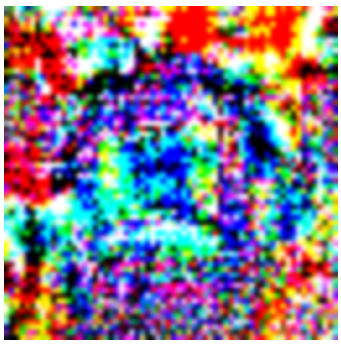
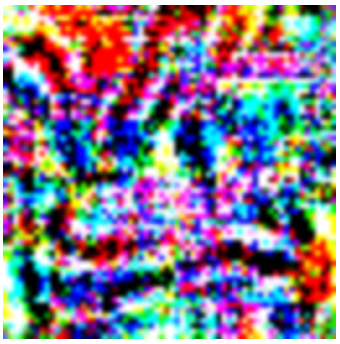
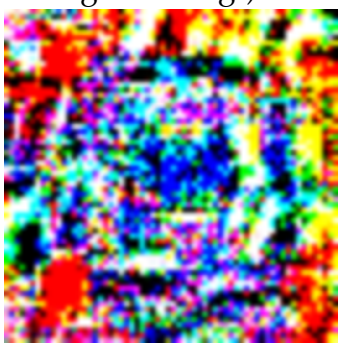
	
Einfahrt Verboten 0.9999	Kreisverkehr 0.9535 (Ursprünglich als Links abbiegen erzeugt)
	
Rechts Vorbei 0.9950	Kreisverkehr 0.9868

Tabelle 6.1: Beispiele für Irrbilder mit Konfidenz >0.9

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

Ergebnisse Degen

Desweiteren konnten Erfolge mit der Erzeugung von Saliency Maps 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 konnte auch mit dem *Gradient Ascent* Verfahren sehr gute Ergebnisse erzielt werden. Für 10 von 43 Klassen konnten Täuschungen mit hohen Konfidenzen am Remote-NN 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 64×64 wäre. Desweiteren kann nichts über die Validierungsgenauigkeit des Trasi-NN gesagt

werden, weshalb auch die nicht zielgerichteten Täuschungsbilder als gutes Ergebnis betitelt werden.

Im Vergleich der Methoden *Saliency Map* und *Gradient Ascent*, kann das letztere Verfahren bevorzugt 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 neuronalem Netz und der Algorithmik bzw. deren Parameter. Desweiteren können verfahren entwickelt werden, die sich aller Methoden gezielt bedienen.

Literaturverzeichnis

[1] Huynh Ngoc Anh. Implementations of some popular Saliency Maps in Keras: experiencor/deep-viz-keras, January 2019. original-date: 2017-06-23T08:16:07Z.

[2] BMBF-Internetredaktion. Das Auto von morgen: autonom, sicher, effizient - BMBF.

[3] BMW. Autonomes Fahren - Die 5 Stufen zum selbstfahrenden Auto.

[4] ZDF dpa. Ein unfall, der am branchen-versprechen nagt.

[5] Marcus Efler. Autonomes Fahren: Das Ende des Lenkrads. *Die Zeit*, January 2018.

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

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

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

[9] 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.

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

[11] Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep Neural Networks Are Easily Fooled: High Confidence Predictions For Unrecognizable Images.pdf. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 427–436, Boston, MA, USA, June 2015. IEEE.

[12] Utku Ozbulak. Pytorch implementation of convolutional neural network adversarial attack techniques : utkuozbulak/pytorch-cnn-adversarial-attacks, January 2019. original-date: 2017-12-15T01:39:12Z.

[13] 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.

[14] Sarah Perez. microsoft-silences-its-new-a-i-bot-tay-after-twitter-users-teach-it-racism/.

[15] Sarah Perez. microsofts-new-ai-powered-bot-tay-answers-your-tweets-and-chats-on-groupme-and-kik.

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

[17] 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.

[18] 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.

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