TECHNISCHE HOCHSCHULE NÜRNBERG
GEORG SIMON OHM

Erstellung von Irrbildern 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 an TrafficSign-AI
authors: Leonhard Applis, Peter Bauer, Andreas Porada und Florian Stöckl

Kurzfassung

To be done

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

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Der Traum von einem autonom gelenkten Automobil ist so alt wie das Automobil selbst [**?**]. 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[**?**] und Berichte von Automobilherstellern[**?**] und der Presse[**?**] zeigen. Die Forschung im Bereich Künstliche Intelligenz hat mittlerweile einen Stand erreicht, der für die Automatisierung des Autos genutzt werden kann.

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* [**?**].

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* [**?**], 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.

Diese Irrbilder zu erzeugen stellt das Kernziel dieser Arbeit dar.

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 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 Verkehrschilder 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.

Im Abschnitt 2.1 werden hierfür zunächst die *German-Trafficsign-Recognition-Benchmark* (Kurz: GTSRB) vorgestellt, welche für das Training der Webschnittstelle verwendet wurden. Dieses Datenset bildet ebenfalls einen zentralen Ausgangspunkt für einige der verfolgten Ansätze.

Anschließend werden in Abschnitt 2.2 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 3.

Dieser Ansatz verändert iterativ ein Verkehrsschild, und behält die Änderungen bei, sollte der erzielte Score im akzeptablen Bereich liegen. Mit passender Bildver-

änderungen erzielen höhere Iterationen unkenntliche Ergebnisse.

Innerhalb des Kapitels wird zunächst im Abschnitt 3.1 die Idee anhand von Pseudocode weiter erläutert und anschließend in Abschnitt 3.2 die Implementierung für die Webschnittstelle gezeigt. Die Ergebnisse liegen gesondert im Abschnitt 3.3 vor.

Neben der Implementierung für die Webschnittstelle werden zum Abschluss des Kapitels in Abschnitt 3.4 noch weitere Verbesserungen für eine lokale Implementierung vorgestellt, welche allerdings nicht für die Webschnittstelle tauglich waren.

ANPE

Saliency Map Aufbau!

In Kapitel
Abschluss dieser Arbeit bildet im Kapitel 6 ein Fazit über die gefundenen Methoden, sowie ein Ausblick auf weiterführende Arbeiten.

1.4 Verwandte Werke und Primärquellen

1.5 Rahmenbedingungen des Informaticups

Die Umsetzung der Implementierung erfolgte innerhalb der webbasierten, interaktiven Entwicklungsumgebung Jupyter Notebook **??** (in der Version 5.7.4) zusammen mit der objektorientierten höheren Programmiersprache Python **??** (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 fünftbeleibtesten 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

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 1.1: Paketabhängigkeiten der implementierten Software

??.

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

Zur fehlerfreien Ausführung des Codes (Saliency Map Verfahren beziehungsweise Gradient Ascent Verfahren) müssen sowohl Python 3.6.5 als auch folgende Python-Bibliotheken in der wie folgt spezifizierten Version in der zur Laufzeit verwendeten Anaconda Environment vorliegen, wie in Tabelle **??** aufgeschlüsselt.

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 der Code zur Rubrik “Managing Anaconda Environment” zuerst ausgeführt werden. Andernfalls kann die korrekte Ausführung von weiteren Teilen des Codes in nachfolgenden Rubriken nicht gewährleistet werden.

2 Analyse der Webschnittstelle

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

2.1 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)

2.2 Transferierbarkeit von Angriffen auf ein Blackbox Modell

Verwandte arbeiten bestätigen die Transferierbarkeit von Angriffen, die auf einem ëigenen"neuronalen Netz erzeugt wurden und gegen ein unbekanntes Blackbox modell funktioniern

Einschränkungen? Probleme? Rechtfertigung der Implementation eines eigenen Modells

Transferability https://openreview.net/pdf?id=Sys6GJqxl

Blackbox Attacks: https://arxiv.org/pdf/1602.02697.pdf

2.3 Implementierung eines eigenen Modells zur Klassifizierung von Straßenschildern (Aphrodite)

Für die lokale Degeneration in Kapitel 3 wurde mithilfe Tensorflows ein eigenes Keras-Model erstellt zur Verkehrsschilderkennung.

at Peter: Bezug der Aphrodite in Gradient Ascent Fooling?

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 /Degeneration-Code/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.

ANPE: Das ALEXNET-Model hier auch listen?

3 Degeneration

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

Die Benennung schöpft sich aus der Nähe zu genetischen Algorithmen , allerdings aus einer invertierten Perspektive: Um auf unbekannte Modelle einzugehen, wird hierbei von einem korrekt erkannten Bild *weggearbeitet*.

genauer, was das ist?

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.

3.1 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 unsere (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 und 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 (bzw. der Schwellwert zu niedrig gewählt wurde).

Der oben genannte Algorithmus lässt sich auch für Text- oder Sprach-basierte Klassifikationen adaptieren.

Hierfür müssen lediglich andere Manipulations- und Klassifizierungs-Funktionen gewählt werden.

3.2 Implementierung Remote

Im Rahmen des Wettbewerbs wurde mit einer Rest-API gearbeitet, welche besondere Herausforderungen mit sich bringt:

Marker nach oben

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

Zusätzlich wurde der Grundalgorithmus um die *Verbose*-Funktion und eine History erweitert. Mithilfe der *History* können anschließend hilfreiche Plots erstellt werden.

Diese wurden für den untenstehenden Code weggelassen.

Ebenso ist anzumerken, das ignoriert wurde, welche Klasse zuerst erzeugt wurde. Solange irgendeine Klasse mit einer passenden Konfidenz gefunden wurde, wird dies als hinreichend erachtet. 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 entsprechend auf und senden es an das Trasi-Webinterface (Die Methode *Scorer.Send_ppm_image(i* und gibt ein JSON-Array der Scores wieder.

Die Hilfsmethode *Scorer.get_best_score(response)* gibt den höchsten gefunden Score wieder.

Zeilennummern, Caption

```
# Parameters:
#   An image (as 64x64x3 Uint8 Array),
#   a function to alter the image,
#   a threshold how much the image can be worse by every step
#   The # of Iterations i want to (successfully) alter my image
#   The # of loops which i want to do max
def remoteDegenerate(image, alternationfn = _noise, decay = 0.01, itera
    # First: Check if the credentials are correct and the image is dete
    initialResp = Scorer.send_ppm_image(image)
    if(initialResp.status_code!=200):
        return
    totalLoops = 0 # Counts all loops
    depth = 0 # Counts successfull loops
    lastImage = image
    lastScore = Scorer.get_best_score(initialResp.text)
    # To check if we put garbage in
    print("StartConfidence:",lastScore)
    #We stop if we either reach our depth, or we exceed the maxloops
    while(depth<iterations and totalLoops<maxloops):
        totalLoops+=1
        # Alter the last image and score it
        degenerated = alternationfn(lastImage.copy())
        degeneratedResp = Scorer.send_ppm_image(degenerated)
        if (degeneratedResp.status_code==200):
            degeneratedScore= Scorer.get_best_score(degeneratedResp.tex
        else:
            print("Error, status code was: ", degeneratedResp.status_co
        # If our score is acceptable (better than the set decay) we kee
        if(degeneratedScore>=lastScore-decay):
            lastImage=degenerated
            lastScore=degeneratedScore
            depth+=1
        #We are working remote, we need to take a short break
        time.sleep(1.1)
    return lastScore,lastImage
```

3.3 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 einige Probleme die aufgetreten sind und zuletzt noch ein kurzes Zwischenfazit gezogen.

Positive Ergebnisse

Die zuverlässigsten Ergebnisse wurden mit einfachem Rauschen erzeugt. Die Abbildung 3.1 zeigt, dass zunächst vorallem die Pixel außerhalb des eigentlichen Schildes verändert werden - ein erwartetes verhalten. 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.

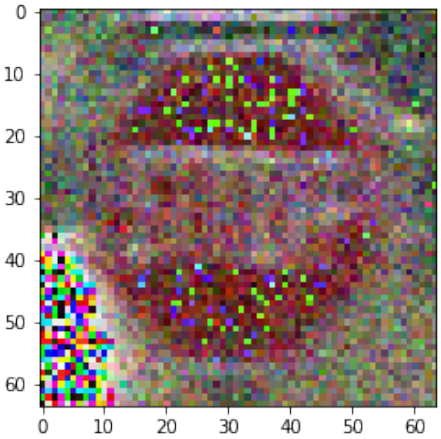


Abbildung 3.1: Rausch - Degeneration mit 600 Iterationen

Während die Abbildung 3.1 noch als Verkehrsschild zu erkennen ist, führt ein längeres ausführen der Degeneration zu einem Ergebniss wie in Abbildung 3.2. Um dieses Ergebnis zu erzielen wurden 4400 sekunden benötigt, also ca. 73 Minuten.

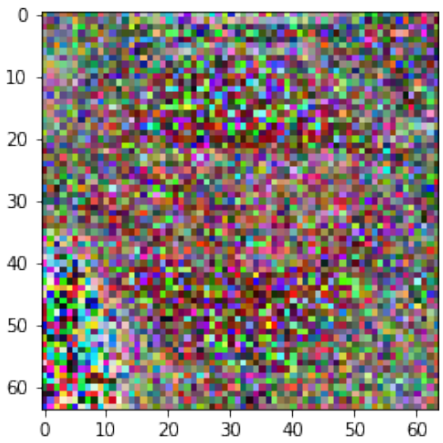


Abbildung 3.2: Rausch - Degeneration mit 4000 Iterationen

Die Plots in Abbildung 3.3 stellen den Verlauf des Algorithmus dar: Das erste Diagramm zeigt einen Verlauf der aktuellen *Tiefe* über die Iterationen dar, der zweite die jeweils produzierten Genauigkeit der jeweiligen Iteration (nicht nur die der akkzeptierten), 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.

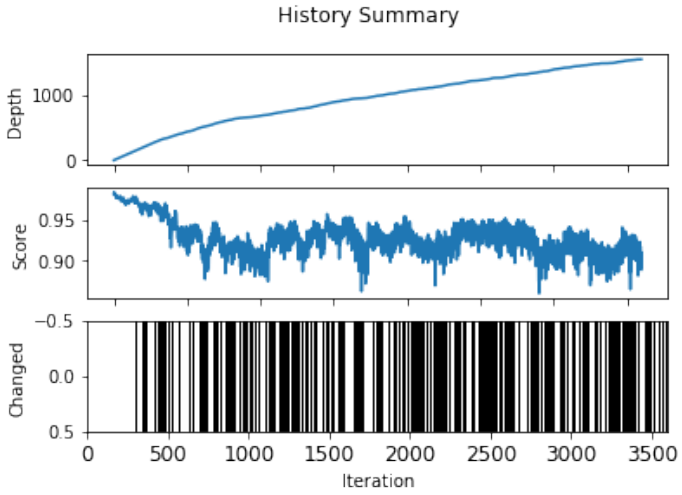


Abbildung 3.3: Plot der Rausch-Degeneration

Es wurden ebenfalls einige sehr positive Ergebnisse mit einer Mischung aus star-

kem Rauschen und Glätten erzeugt, allerdings waren diese nicht zuverlässig reproduzierbar.

Negative Ergebnisse

Es gab 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 machte die Benutzung der Degeneration unbrauchbar.

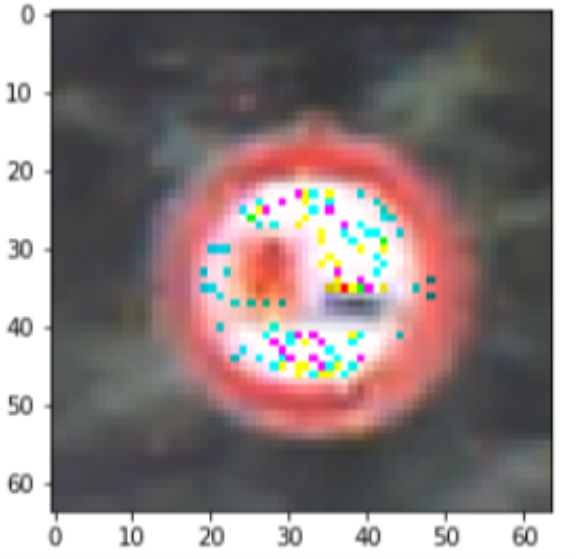


Abbildung 3.4: Rausch-Degeneration auf Trainingsbild - 36000 Iterationen

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 wurden, waren innerhalb des weißen Bereiches des Überholverbotsschildes wie in Abbildung 3.4. Es wurden aber ebenfalls keine Änderungen vorgenommen außerhalb des Schildes, wo diese zu erwarten wären und bei vorhergehenden Versuchen auch zu beobachten waren (vgl. Abbildung 3.1).

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, 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:** 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.

Caption und Label

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 Modelle sind selten im Einsatz, weil zuviel Schabernack damit getrieben werden kann. Hierfür brauche ich noch eine Quelle

3.4 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) trainiert mithilfe der GTSRB-Daten. Das *Scoring* der Remote-Implementierung wird durch die *predict()*-Funktion des Models ersetzt.

Als zusätzliche Erweiterung wurde 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 wurde, sowie die Wartezeit. Letzteres erhöht die Geschwindigkeit des Algorithmus maßgeblich.

Eine zusätzliche, passive Verbesserung wurde 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.

Quelle? Oder Ausarbeiten?

Fazit

Das wichtigste Fazit, welches im Umgang mit der lokalen Implementierung gezogen werden konnte, ist die nicht-verwendbarkeit der lokalen Bilder für die Schnittstelle. Während dies ursprünglich die Motivation war, schnell lokal Irrbilder zu erzeugen und Remote zu verwenden, stellte sich heraus das die lokalen Irrbilder keine guten Scores an der Schnittstelle erzielten und vice versa.

Es ist anzunehmen, das die Modelle dieselben stärken haben (Verkehrsschilder korrekt zu erkennen), 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.

Beispielbild, Remote 90 % und Lokal 30% oder vice versa

3.4.1 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.

Nähe zu genetischen Algorithmen herausbringen, Quelle!

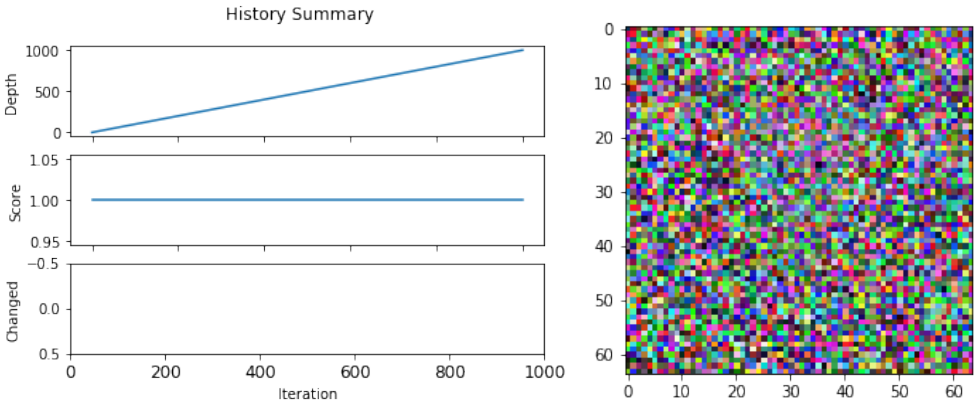


Abbildung 3.5: Verlauf der Batch-Degeneration mit Batchsize=10 und Tiefe=1000

Abbildung 3.5 zeigt den deutlich besseren Verlauf der Batch-Degeneration gegenüber der unveränderten Implementierung in Abbildung 3.3. 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 3.5 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 worden.

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.

3.4.2 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, sollte man in der Cloud arbeiten.

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.

3.4.3 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 wurde, 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 Kinder eines Knotens, zu einem (Binär-)Baum. Das Abbruchkriterium der Tiefe kann weiterhin beibehalten werden und entspricht der Tiefe des Baumes. Im Falle eines Versuch- oder Zeitbedingten Abbruchs wird das Bild mit der bisher größten Tiefe ausgegeben.

Die Entwicklung dieser Variante entstand durch 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.

4 Saliency Maps

4.1 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. [3] 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 Eingabe(bild) entsprechend der Zielklassen.

Gebräuchliche anwendung dieses Verfahrens im Machine Learning bereiches betreffen der Darstellung von High level Features einzelner Neuronen [Beispiel Quelle] 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 Koinzidenz bezüglich der entsprechende Klasse klassifiziert werden, wenn das erzeugte Bild als Eingabe verwendet wird. Daraus ergibt sich die Hypothese, mithilfe dieser Visualisierung für den menschen semantisch nicht erkennbare Verkehrszeichen Bilder zu erzeugen, die hohe Zielkonfidenzen mit 90% oder mehr erreichen. Die beschriebene Methode wird in der Implementierung als "Vanilla Saliency" bezeichnet.

4.2 Implementierung

Die Verfahren zur Erzeugung der Saliency Maps setzen eine identische Datenvorverarbeitung voraus. Die Trainingsbilder des GTSRB Datensatz wurden konver-

tiert 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[], Guided Backpropagation[] und Integrated Gradient[] werden jeweils in einer ungefilterten und und geglätteten Variante verwendet entsprechend einer auf die Aufgabenstellung angepasster Implementation auf der Basis von [https://github.com/experiencor/deep-viz-keras].

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 prediction abgespeichert. Zusätzlich wird pro Verfahren ein Logfile erstellt, welches alle Bilder mit einer Konfidenz >0.9 sowie die geschätzte Klasse und das Ursprungsbild festhält.

4.3 Ergebnisse

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

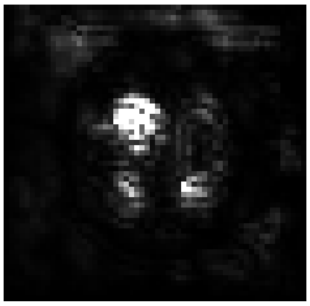
Alle ungeglätteten Verfahren konnten keine positiven Ergebnisse bezüglich der Bilderzeugung mit einer Konfidenz > 0.9 liefern. Dies lässt sich möglicherweise auf die besonderen Einschaften von CNNs zurückführen, dass ohne Glättung in dem - ohne farbchannel - bereits dimensionsreduziertem Bild keine ausgeprägten Kanten vorhanden sind und nach dem "FaltenİInformationen verloren gehen. Desweiteren konnte am Blackbox Modell beobachtet werden, dass alle erzeugten ungegeglätteten Saliency Maps mit einer Konfidenz von 0.45834911 als Klasse "Baustelleërkannt 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 [] zeigt wie aus Bild [] (Höchstgeschwindigkeit (30)) mit den Verfahren SSmoothed Guided Backpropagation"(vgl. []) und SSmoothed Vanilla Saliency"(vgl. []) ein Bild erzeugt, welche mit einer Konfidenz >0.9 vom We-

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

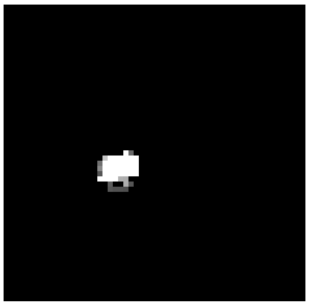
wird noch aligned

Ursprungsbild
Höchstgeschwindigkeit
(30)



Guided Backprop.
Überholverbot

0.9155

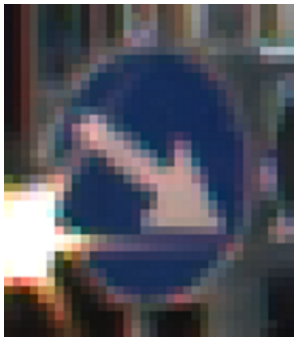


Vanilla Saliency
Höchstgeschwindigkeit
(30)

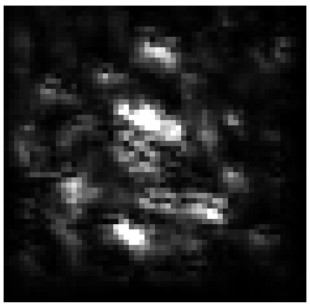
0.9283

Tabelle [] 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.

centering

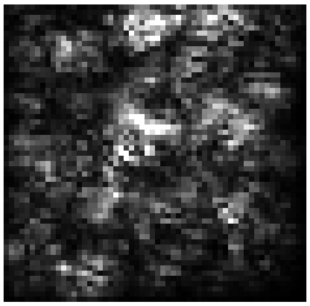


Rechts Vorbei



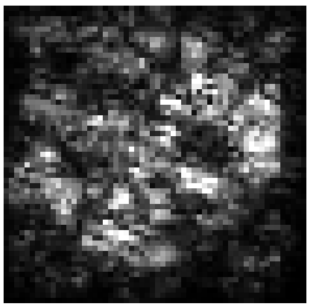
Einmalige Vorfahrt
0.9678

Einfahrt Verboten



Überholverbot
0.9571

Kreisverkehr



Baustelle
0.9999

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 Aussagekraft

darüber, welcher Klasse das erzeugte Bild zugeordnet wird.

5 Gradient Ascent

5.1 Konzept

Unsere Implementation ist eine art von DIRECT encoding. Aber anstelle von lediglich Der Confidenz und einer Klasse, werden die Gradienten mittels einer Kreuz-Entropie zu einer Zielklasse berechnet, welche dann als „orientierungshilfe“ für die weiter manipulation der bilder dienen. -> Das wird als Targeted Backpropagation bezeichnet.

The method keeps the same, whether a random “noisy” image or a “real” image is used initially. Only the modification weights need do be smaller, as the goal is to keep the “original image” as much as recognizable as possible.

5.2 Implementierung

Code modifiziert auf Basis von https://github.com/utkuozbulak/pytorch-cnn-adversarial-attacks

Training und Evaluierung des AlexNet Modells

Quelle Des AlexNet hinzufügen. https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

WICHTIG, aber eigentlich ist können wir das genauso wichtig behandeln wie die Aphrodite? Unterschied zur Aphrodite? Eigenes modell vs Alexnet architektur die auf die 43-Klassen angepasst wurde. Training und Evaluierung des AlexNet mit dem GTSRB Datensatz

Abschluss des Training mit genauigkeit von 89

Bilderzeugung anhand des Gradient Ascent Fooling Verfahrens REF NGUYEN

WICHTIG Anwendung des Gradient Ascent Fooling Verfahrens auf das trainierte AlexNet

Generierung eines zufallsbildes (random noise) iterativer ablauf: prediction gradienten berechnung durch kreuzentropie zur gewünschten klasse modifikation des bildes durch anteil der gradienten

Der vorgestellte ansatz ändert sich nicht bei eingabe eines ausgewählten Bildes, die veränderungsrate solle aber gering gehalten werden, um das bild so wenig wie nötig zu verändern

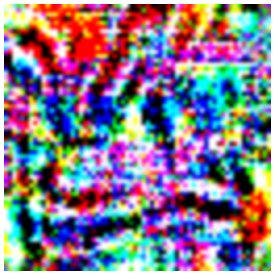
5.3 Ergebnisse

43 bilder Erzeugt respektive 1 pro vorhandene Klasse (im GTSRB Datensatz) 20 Bilder für mit 10 verschiednen klassen erreichten konfidenz höher 0.9

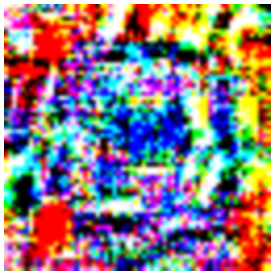
Einfahrt Verboten

Kreisverkehr

(Ursprünglich als Links abbiegen erzeugt)



Rechts Vorbei



Kreisverkehr

6 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.

6.1 Zusammenfassung

zusammentragen und knappe reflexion der ergebnisse
noch ohne wertung

6.2 Diskussion

hier kommen rein vergleich der ergebnisse (reproduzierbarkeit, gezielt/random, qualität der bilder) - einschränkungen probleme unserer ansätze,
METRIK ZUR BEWERTUNG DER BILDER:
Rechnerischer Aufwand: WENIG / VIEL
Verlässlichkeit(bei erzeugung)
Erkennbarkeit: Garnicht / Hoch
... vorschläge?!

6.3 Weiterführende Arbeiten