TECHNISCHE HOCHSCHULE NÜRNBERG
GEORG SIMON OHM

Erstellung von Irrbildern zur Überlistung einer Verkehrsschilder erkennenden KI

IT-Projekt Bericht

Studiengang *Informatik*

Technische Hochschule Georg Simon Ohm

von

Leonhard Applis, Peter Bauer, Andreas Porada und Florian Stöckl

Abgabedatum:	15.03.2019
Gutachter der Hochschule:	Prof. Dr. Gallwitz

Eidesstattliche Erklärung

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

Erstellung von Irrbildern zur Überlistung einer Verkehrsschilder erkennenden KI

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

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

Nürnberg, den 24. Dezember 2018

Leonhard Applis, Peter Bauer, Andreas Porada und Florian Stöckl

Abstract

To be done

title:	Fooling an TrafficSign-AI
author:	Leonhard Applis, Peter Bauer, Andreas Porada und Florian Stöckl
reviewer DHBW:	Prof. Dr. Gallwitz

Kurzfassung

To be done

Titel:	Erstellung von Irrbildern zur Überlistung einer Verkehrsschilder erkennenden KI
Author:	Leonhard Applis, Peter Bauer, Andreas Porada und Florian Stöckl
Prüfer der Hochschule:	Prof. Dr. Gallwitz

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

1.1 Motivation

Optische Täuschungen, Irrbilder, Rahmenbedingungen

1.2 Ziel der Arbeit

1.3 Aufbau der Arbeit

1.4 Verwandte Werke und Primärquellen

1.5 Rahmenbedingungen des Informaticups

2 Gegneranalyse

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

Anderer Chapter-Title

2.1 Ursprungsdaten

Hier gehen wir kurz auf die Trainingsdaten ein die wir haben, zeigen ein paar Bilder und wie fürchterlich hässlich die sind in 64x64

2.2 Modellschätzungen

Hier kommen unsere Erfahrungen, die wir mit dem Modell gemacht haben

1. Gekürzte Klassen: Aus unserer MongoDB können wir ziemlich sicher sagen, dass wir nur 33 Klassen von Trasi haben, keine 43. Wir können nachsehen welche fehlen
2. Softmax-Ausgabefunktion
3. Interpolationsfunktion (vllt mit einem Bild in 3 Interpolationsversionen und jeweiligen Score)
4. Overfitting bei Trainingsdaten
5. unzuverlässigkeit bei nicht-Schildern (z.B. OhmLogo)

3 AI-Greyboxing

3.1 Konzept

Wichtigste Inhalte:

1. hübsches Diagramm was für Komponenten
2. Stichpunktartige Beschreibung der Komponenten
3. Workflow durch Setup (Gen - Score - DB - Training - AI)
4. Workflow Ansatz in betrieb (Gen - AI - Scorer)

3.2 Implementierung und erste Ergebnisse

Code zeigen? MongoDB sagen?

3.3 Fehler- und Problemanalyse

Schätzen, das man aus Pixelbrei nichts lernen kann.

4 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.

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 unsere (generische) Implementierung des Degenerationalgorithmus:

- 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.

4.2 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.

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.

```
# 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)
```

```
if history:
    h = []
#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
        #Attempts do not count
        totalLoops-=1
    # if we verbose, we want console output (then we see directly
    if verbose:
        print("Score:",degeneratedScore,"Depth:",depth, "Loop:" , t
    # if we have history=True, we collect the same data as in verbo
    if history:
        h.append((degeneratedScore,depth,totalLoops))
    # 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)
print("FinalConfidence:",lastScore)
if h!=[] :
    plotHistory(h)
    return lastScore,lastImage,h
else:
    return lastScore,lastImage
```

4.3 Ergebnisse Remote

Hier kommen ein paar Beispiele und Plots.

Auch hierhin kommt ein Fazit welche AlternationFunktionen wie gut waren und das die Trainingsbilder nicht geeignet waren

Sehr wichtig sind die benötigten Zeiten.

4.4 Implementierung Lokal

Ich weiß nicht ob das ein extra Punkt ist, aber an sich würde ich hier das Model bei uns kurz vorstellen

4.5 Anpassung und Verbesserung Lokal

Hier kommen zunächst so Dinge wie "wait"rauszunehmen aber GPU-Acceleration würde ich auch hernehmen.

4.5.1 Batch-Degeneration

Ist noch ein To-Do: Degenerieren von 100 Bildern, Wahl des "besten". Mach ich noch.

4.5.2 Parallel-Degeneration

Nur Ansatz: Hat nicht geklappt das zu basteln weil numpy Arrays echt nervig sind bei Parallelverarbeitung.

Vorstellen kann man das allerdings kurz. Konflikt mit GPUAcceleration.

5 Saliency Maps ANPE

Hier kommt Andreas und Peters Teil.

Das File zu Evolutionären Algorithmen ist noch vorhanden, nur nicht eingebunden.

6 Fazit

6.1 Zusammenfassung

6.2 Weiterführende Arbeiten

Literaturverzeichnis