KOLLEKTIV 5

InformatiCup - Theoretische Ausarbeitung

Andreas Bremer, Jan Wolff, Mateusz Ryżewski und Mohibullah Obaidi

Einleitung

Dieses Projekt wurde im Rahmen des InformatiCup 2019 umgesetzt. Ziel der in diesem Projekt erstellten Software ist es, Bilder zu generieren, die ein neuronales Netz mit hoher Konfidenz in bestimmte Klassen einordnet. Trainiert ist das Netz auf das Erkennen von Verkehrsschildern. Es ist jedoch gewollt, dass es für Menschen unmöglich ist Verkehrsschilder in den generierten Bildern zu sehen.
Das betrachtete Netz wird über eine API angesprochen. Es handelt sich um eine Black Box, da lediglich bekannt ist, dass ein neuronales Netz verwendet wird. Der genaue Aufbau davon ist allerdings unbekannt.
Die entwickelte Lösung enthält mehrere Ansätze. Zum einen wird versucht das neuronale Netz nachzubilden, um so genauere Informationen über dessen Aufbau zu erhalten. Desweiteren wird ein evolutionärer Algorithmus zur Generation der Bilder implementiert. Dieser arbeitet direkt mit den Ausgaben der API und basiert auf den Ergebnissen einer Analyse des neuronalen Netzes.

Hintergrund der Lösung

Um eine Lösung zu erstellen, wurde zuerst das neuronale Netz hinter der API auf verschiedene Eigenschaften bezüglich der Farben, Kontraste und Formen innerhalb der Eingabebilder überprüft. Basierend darauf wurden verschiedene Ansätze entwickelt.
Im ersten Ansatz wird versucht ein neuronales Netz aufzubauen, das sich wie das neuronale Netz hinter der API verhält. So könnte das nachgebaute Netz ohne eine Limitierung benutzt werden. Desweiteren können, durch genaue Informationen über den inneren Aufbau, Irrbilder mit beispielsweise einem Gradientenanstieg[1] generiert werden.
In den weiteren zwei Ansätzen werden evolutionäre Algorithmen eingesetzt. Hierbei verwendet der zweite Ansatz verschiedene Polygone unter Betrachtung von Farb- und Konstrastbereichen. Der dritte Ansatz generiert Bilder, die eine Unterteilung in verschiedenfarbige Quadrate enthalten.
Ansatz Zwei liefert schlussendlich die besten Ergebnisse. Die dort entstehenden Bilder unterscheiden sich stark untereinander und können mit unter 60 API Anfragen eine hohe Konfidenz erreichen. Ansatz Drei kann zwar auch hohe Konfidenzen erreichen, allerdings ähneln sich die Bilder im Grundaufbau stark untereinander. Der erste Ansatz hat sich als unpraktikabel erwiesen, da der Aufbau des Netzes nicht approximiert werden konnte.

Hinter der gegebenen API (Application Programming Interface) befindet sich ein neuronales Netz, das jedem Bild, das an die Schnittstelle verschickt wird, fünf verschiedene Klassen mit einer bestimmten Konfidenz zuordnet. Um alle möglichen Klassen zu erhalten, die von der Schnittstelle zurückgegeben werden, wurden 12 630 Testbilder von Verkehrsschildern aus den GTSRB-Daten[2] an die Schnittstelle verschickt. Durch die Antworten konnten 35 von 43 Klassen gefunden werden (siehe Anhang).
Zudem wurde die Erkennungsrate (engl. Correct Classification Rate) des Neuronalen Netzes berechnet, dafür wurde der prozentuale Anteil der korrekt erkannten Testbilder ausgewertet. Von den für das Neuronale Netz bekannten 35 Klassen konnte eine Erkennungsrate von 87,10% berechnet werden. Für alle 43 Klassen aus GTSRB liegt sie bei 80,69%. Diese Werte dienen als Basis für den Ansatz 1.

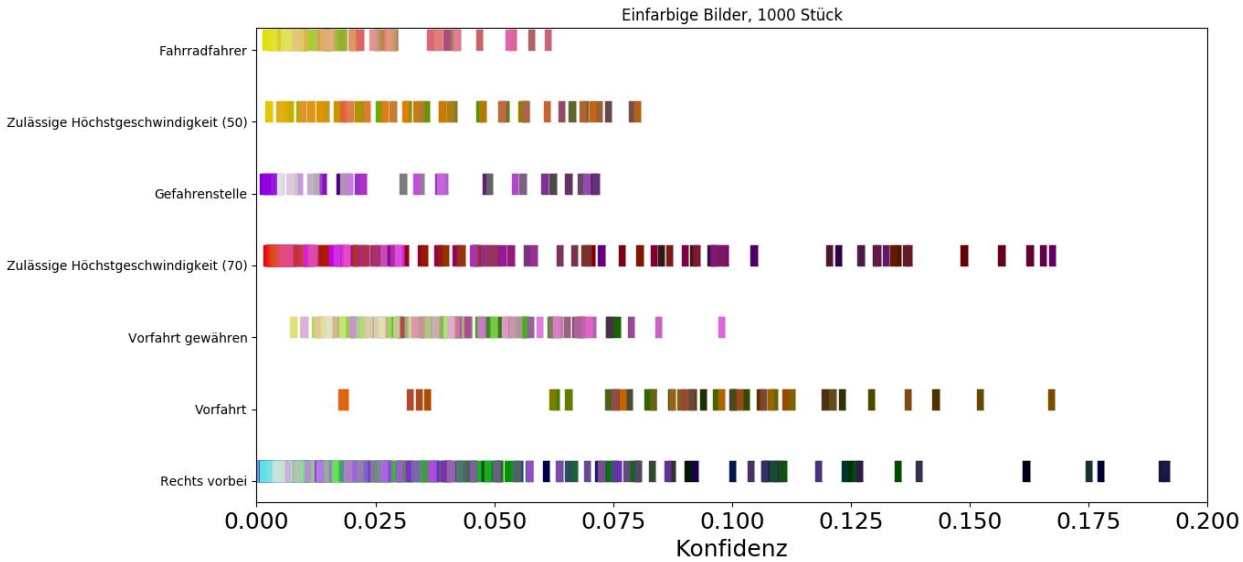
Im Folgenden werden die Antworten des neuronalen Netzes für Bilder mit bestimmten Eigenschaften wie Farbe, Kontrast und Formen untersucht.

Farben

Für die Untersuchung der Farben wurden einfarbige Bilder aus dem RGB-Farbraum generiert. Dafür wurden insgesamt 1000 Bilder erzeugt, die sich aus Farbwerten in 25er Schritten ergeben, also nach dem Muster RGB(0,0,0), RGB(0,0,25), …, RGB(225,225,225).

[1] Nguyen et al., Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images, 2015, http://www.evolvingai.org/files/DNNsEasilyFooled_cvpr15.pdf
[2] http://benchmark.ini.rub.de/?section=gtsrb&subsection=dataset

Die folgende Darstellung der Ergebnisse der API zeigt die Gegenüberstellung der Klassen, die für die Verkehrsschilder stehen und der Konfidenz.



Hierbei wurde untersucht, inwiefern die Farben ein Verkehrsschild repräsentieren. Es ist zu sehen, dass die einfarbigen Bilder mit einem hohen Rot- und Schwarzanteil das Verkehrsschild “Zulässige Höchstgeschwindigkeit (70)” darstellen. Des Weiteren taucht bei vielen Klassen die grüne Farbe auf. Dies liegt wahrscheinlich an dem Hintergrund der Trainingsbilder, wie beispielsweise auf folgenden Bildern:

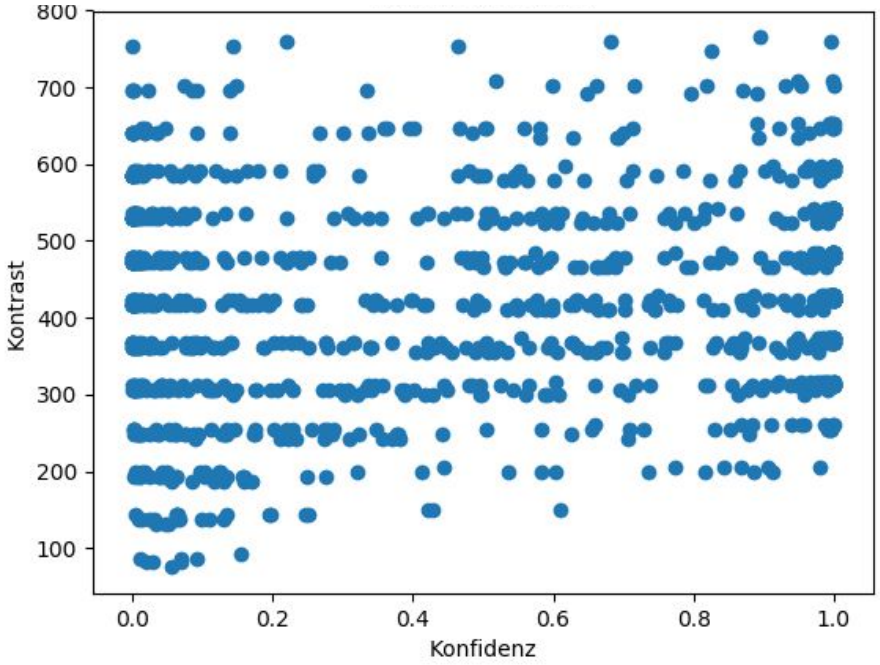
Zudem ist aus dem Diagramm zu entnehmen, dass Bilder mit dunkleren Farben höhere Konfidenzen geliefert haben, jedoch nur für bestimmte Klassen und bei wenigen Farbwerten. Dies ist im Bereich der Konfidenz ab 10% zu sehen.

Kontrast

Der Kontrast ist ein Maß, um den Unterschied von Farben zu beschreiben. Es existieren verschiedene Definitionen des Kontrastes, die sich jeweils nach bestimmten Formeln berechnen lassen. Dabei steht ein höherer Wert des Kontrastes für einen größeren Unterschied zwischen den Farben. Hierbei wird meistens der Helligkeitsunterschied der Farben als wesentliches Unterscheidungsmerkmal verwendet.
Bei der Bilderkennung von Verkehrszeichen kann aber auch der Unterschied zweier gleich heller Farben von Bedeutung sein. Beispiel: Rot(255, 0, 0) und Blau(0, 0, 255) haben die gleiche Summe der RGB-Anteile, können aber als verschiedene Farben wahrgenommen werden.

Deshalb wurde der Kontrast zweier Farben als die Summe (der Absolutwerte) der Differenzen der RGB Anteile der Farben definiert.
Also Kontrast = |R1 - R2| + |G1 - G2| + |B1 - B2|.

Es wurde die Auswirkung dieses Kontrastes auf die Bilderkennung überprüft. Dazu wurden zuerst 1000 zweifarbige Bilder (Hintergrundfarbe und Form in der Mitte) mit Farbkombinationen aus dem gesamten Farbspektrum (von jeweils 0 bis 252 für RGB Anteile) für Vorder- und Hintergrundfarbe generiert und ihrer von der API gelieferten Konfidenz zugeordnet:



Hierbei stellte sich heraus, dass ein Kontrast von mindestens ca. 250 benötigt wird, um Bilder mit einer Konfidenz über 90% zu generieren, sowie dass ein Kontrast von 300 bis 600 zu mehr Bildern mit hoher Konfidenz führt.

Formen

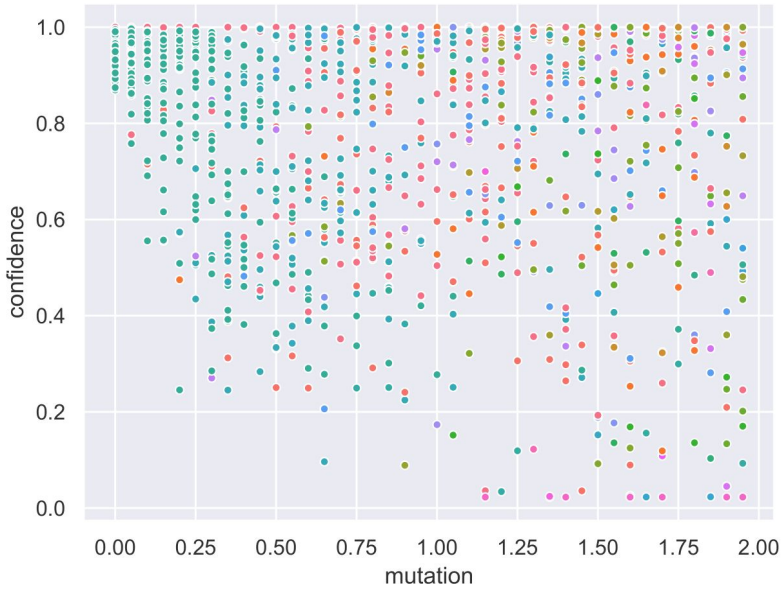
Es ist davon auszugehen, dass die Klassifikation des neuronalen Netzes auf einem sog. Convolutional Neural Network aufbaut. Solch ein Netzwerk generiert selbständig Filtermatrizen und wendet diese zur automatischen Feature Selektion auf das Eingabebild an. Die so entstehenden Filter können selbständig Formen erkennen und zuordnen. Daher wird das Ergebnis der Klassifikation nebst den Farben auch von der Form der Flächen beeinflusst.
Es galt also zu Analysieren, wie das Netz der API auf verschiedene Formen reagiert. Hierbei ist aus dem Datensatz abzuleiten, dass in allen Bildern das Verkehrsschild eine zentrale Position einnimmt. Weiterhin ist stets nur ein einzelnes Schild sichtbar, sodass in dieser Betrachtung auch jeweils nur ein einzelnes zentriertes Polygon generiert wird. Natürlich beeinflussen auch die Farben des Polygons und Hintergrundes die Klassifikation. Hier wurde jedoch nur die Kombination aus schwarzem Hintergrund und weißem Polygon evaluiert, um den Parameterraum möglichst klein zu halten. Mit dieser Farbwahl ist der Kontrast maximiert und

vorangegangene Evaluationen haben gezeigt, dass der Hintergrund des Bildes möglichst dunkel gehalten sein sollte.
Zwei Aspekte wurden hier evaluiert: Zum einen die Anzahl der Kanten des Polygons und zum anderen die Positionierung dieser. Im ersten Durchlauf wird ein Polygon, dessen Durchmesser etwa 90% der Breite des Bildes entspricht mit n-Ecken erzeugt. Dabei werden die Ecken kreisförmig, mit einheitlichem Abstand zueinander um den Mittelpunkt herum angeordnet. Somit entspricht das Polygon mit n=3 Ecken einem gleichseitigen Dreieck, welches sich mit steigendem n mehr und mehr einem Kreis annähert. Hier lässt sich folgendes beobachten: Polygone mit n=3 werden der Klasse „Verbot für Fahrzeuge aller Art“ zugeordnet. N=4 ist nicht Rotationsinvariant und wird, je nach Grad der Rotation, entweder ebenfalls als „Verbot für Fahrzeuge aller Art“ oder als „Vorfahrt“ klassifiziert. Alle Polygone mit n>=5 fallen in die Klasse „Ende aller Streckenverbote“ und scheinen somit einheitlich als kreisförmig erkannt zu werden.
Im zweiten Durchlauf werden zusätzlich die Positionen der einzelnen Kanten zufällig variiert, wodurch die Formen noch weiter verfremden. Mit steigender Versatz-rate werden die Eckpunkte der Polygone in einem größer werdenden Radius randomisiert verschoben. Bei einer Rate von genau 1.0 entspricht der Radius mit 32 Pixeln der halben Breite des Bildes.

Hierbei lässt sich beobachten, dass eine Verfremdung der Polygone die Erkennung weiterer Klassen zulässt. In der Tat ist es möglich mit relativ wenigen Eckpunkten eine Einteilung in viele der möglichen Klassen zu erzeugen. Eine hohe Versatz-rate der Eckpunkte kann Polygone mit überschneidenden Kanten erzeugen. Somit lässt sich die deutlich höhere Varianz der Klassenzuordnungen erklären, die bei hohen Versatz-raten auftritt:

In dieser Darstellung entspricht jeder Bildpunkt einem evaluierten Polygon. Die Farbe der Bildpunkte spiegelt die Klasse mit der höchsten Konfidenz wieder. Die Konfidenz selbst ist durch die Größe der Punkte dargestellt. Der Test wurde mehrmals wiederholt. Einige Punkte

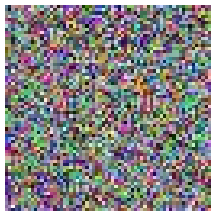
enthalten innerhalb noch weitere Punkte. In diesen Fällen führte eine ähnliche Konfiguration (gleiche Mutationsrate und erkannte Klasse) auf eine unterschiedliche Konfidenz.
Während die Klasseneinteilung mit zunehmender Verfremdung stärker variiert, nimmt hingegen die Konfidenz der Klassifizierungen im Mittel ab:



Dennoch ist die hohe Verfremdung von Vorteil, da somit eine größere Auswahl an Klassen erzeugt werden kann. Zumal die geringe Verfremdung beinahe ein Nachvollziehen der Klassifikation zulässt, da es beispielsweise nahe liegt ein einzelnes Dreieck als Warnschild zu identifizieren.
Für die weitere Anwendung ist also eine Verfremdung von jeweils einem einzelnen Polygon sinnvoll.

Ansatz 1: NN nachbauen

Eine Idee war es, mithilfe von Keras[3] und dem frei zugänglichen GTSRB Datensatz[4] ein neuronales Netz zu trainieren. Dieses sollte das statt der Web API genutzt werden. Hierfür wurden verschiedene Aufbauten aus dem Paper “Man vs. Computer: Benchmarking Machine Learning Algorithms for Traffic Sign Recognition”[5] betrachtet, in dem mehrere Ansätze miteinander verglichen werden. Ein Mehrheitsentscheid aus mehreren Convolutional Neural Networks, eingereicht von der IDSIA Gruppe, erreicht hier mit 99,46% die größte Erkennungsrate auf dem Datensatz.
Diese Ergebnisse konnten repliziert werden, sodass nun ein Neuronales Netz vorliegt, mit dem Bilddaten lokal klassifiziert werden können.
Somit ergeben sich zwei entscheidende Vorteile. Zum einen ist die verfügbare Rechenleistung nicht mehr durch das Limit der API gedrosselt. Da das Netz nun eine White-Box ist, besteht außerdem die Möglichkeit per Gradientenanstieg eine Eingabe zu erzeugen, die eine vordefinierte Ausgabe des Netzes erzwingt. Die so entstandenen Eingabedaten können dann wiederum als Bilder abgespeichert werden, um diese über die API zu Klassifizieren.



Ein so entstandenes Bild sieht stark verrauscht aus und ähnelt keiner erkennbaren Klasse, wird jedoch von dem lokalen Netz mit einer Konfidenz von etwa 99,9% als Vorfahrtsschild erkannt. Jedoch lässt sich diese Klassifikation nicht auf das Netz der API übertragen. Hier wird das Bild mit gerade einmal 3,51% in die Klasse “Vorfahrt gewähren” eingeordnet.
Insgesamt wurden 16 Anläufe vorgenommen um den Aufbau der IDSIA Gruppe zu trainieren, in der Hoffnung eine vergleichbare Konfiguration zu erzeugen. Keiner der entstandenen Aufbauten kann allerdings Ergebnisse erzeugen, die sich mit dem API Netz decken. Somit lässt sich entweder darauf schließen, dass der Zufall beim Trainieren eine viel zu große Rolle spielt oder die API einen anderen Aufbau nutzt. In beiden Fällen ist davon auszugehen, dass es nicht möglich ist ein etwa äquivalentes Netz zu dem der API nachzubilden. Somit ist dieser Ansatz nicht umsetzbar, da das Erzeugen von Irrbildern in direktem Zusammenhang mit den Eigenschaften des Netzes steht. Eingabedaten die weit von dem ursprünglichen Datensatz abweichen, werden je nach verwendeter Klassifizierung in stark unterschiedliche Klassen eingeteilt.

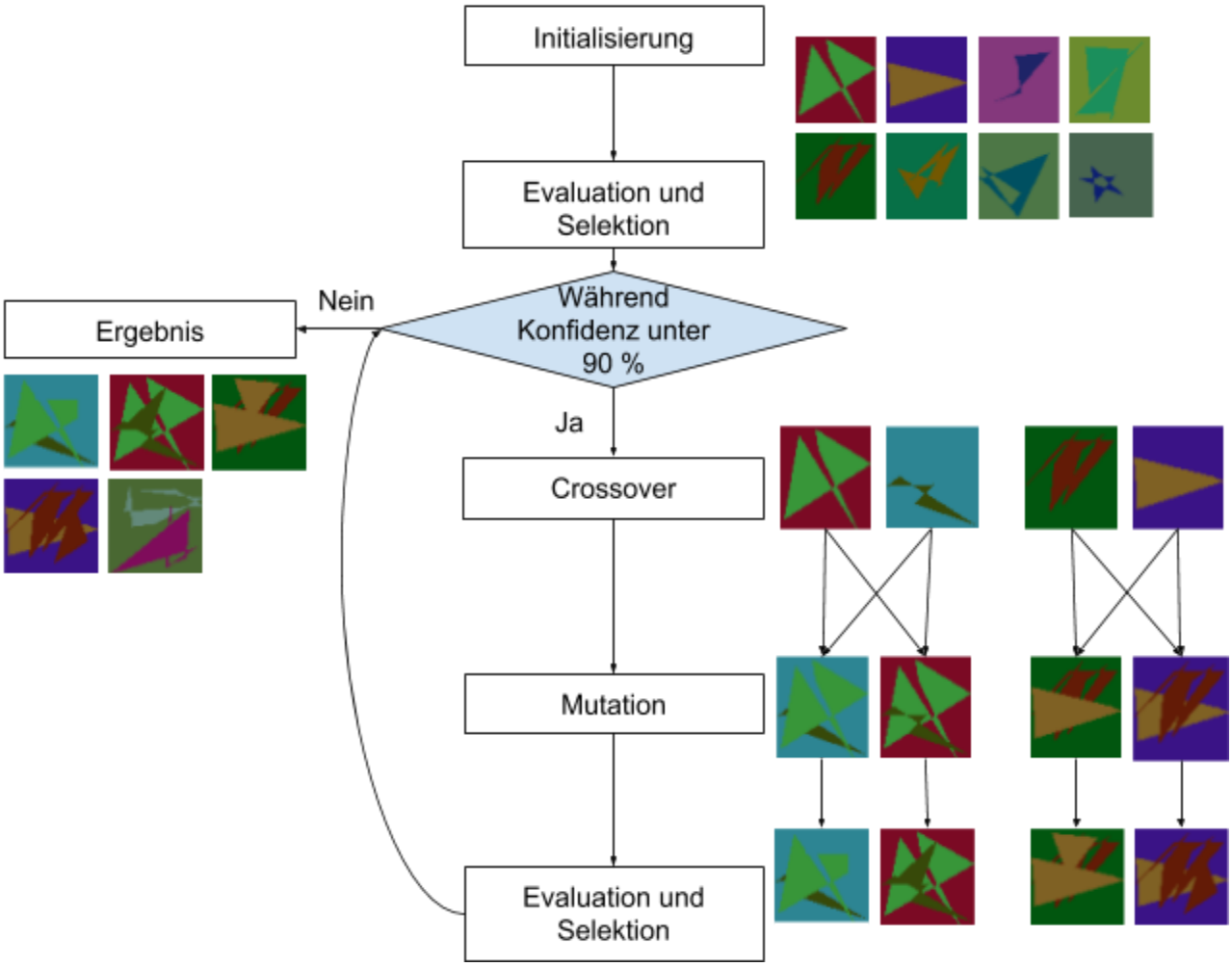
---

[3] https://keras.io/, Letzter Zugriff am 13.1.19
[4] http://benchmark.ini.rub.de/?section=gtsrb&subsection=dataset, Letzter Zugriff am 13.1.19
[5] Man vs. Computer: Benchmarking Machine Learning Algorithms for Traffic Sign Recognition von Stallkamp et al. Quelle: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.713.656&rep=rep1&type=pdf

Ansätze mit Evolutionärem Algorithmus (EA)

Die Idee hinter diesem Ansatz besteht darin zufällige Bilder zu erzeugen und diese solange zu verändern bis sie die gewünschte Konfidenz von mindestens 90% erreichen. Dabei sollen fünf Bilder mit verschiedenen Klassen generiert werden.
Für die Umsetzung bietet sich ein evolutionärer Algorithmus an. Dieser wird anhand der folgenden Grafik dargestellt. Dabei wird am Anfang eine Generation von Individuen erzeugt. Diese werden rekombiniert, mutiert und danach selektiert. Das wird so lange wiederholt, bis das Abbruchkriterium erfüllt wird.
Die Selektion ist für die Auswahl der Individuen mit der besten Konfidenz zuständig. Bei der Rekombination (Crossover) werden die besten Eigenschaften von mehreren Individuen zusammengefügt, um daraus ein neues Individuum zu erzeugen. Die Mutation hingegen dient zum Verändern der einzelnen Individuen. Bei der Überprüfung des Abbruchkriteriums werden auch lokale Maxima innerhalb der Population vermieden, indem neue Individuen zu der Population hinzugefügt werden, wenn sich das Ergebnis nicht verbessert hat.

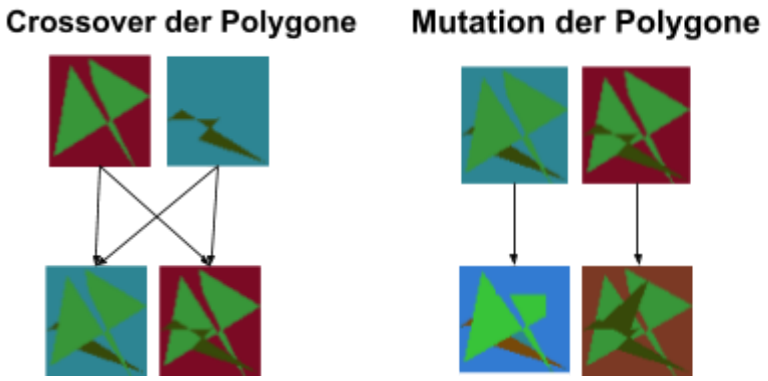
Abbildung zum Evolutionären Algorithmus



Bei diesem Ansatz werden die fünf Bilder mit den verschiedenen Klassen gleichzeitig erzeugt. Auf diese Weise können alle Antworten der verschickten Bilder an die API benutzt werden um weitere Bilder mit anderen Klassen zu generieren. Dies führt zu weniger API-Anfragen und somit zu einer schnelleren Lösung.
Dieser Ansatz kann ebenfalls für eine gezielte Generierung von bestimmten Klassen benutzt werden.

Ansatz 2: Polygone mit EA

Bei dieser Variante des Evolutionären Algorithmus werden die Individuen mit Polygonen in verschiedenen Farben und Formen generiert. Dabei wird für die initiale Population jeweils ein zufälliges Polygon pro Bild erzeugt.
Danach werden aus der Population jeweils zwei Bilder mit der besten Konfidenz für die gleiche Klasse ausgewählt und für das Crossover benutzt. Dabei werden die Polygone der zwei Bilder zusammengefügt und daraus zwei Variationen erzeugt (siehe Abbildung). Die beiden Ausgangsbilder für den Crossover werden nicht entfernt, da diese wiederverwendet werden können, falls die erzeugten Bilder durch den Crossover keine bessere Konfidenz bringen. Der Crossover führt im Durchschnitt zu einer Verbesserung mit 63,73%, basierend auf 91 durchgeführten Crossovern.



Jedes bei dem Crossover erzeugte Bild wird danach mutiert. Als erstes werden die Farbwerte der Polygone sowie des Hintergrunds mit einer Mutationsrate aus einem definierten Bereich verändert. Dann werden den Polygonen weitere Punkte zufällig hinzugefügt oder entfernt. Zusätzlich werden die Positionen der einzelnen Punkte ebenfalls mit einer bestimmten Mutationsrate verändert (siehe Abbildung).

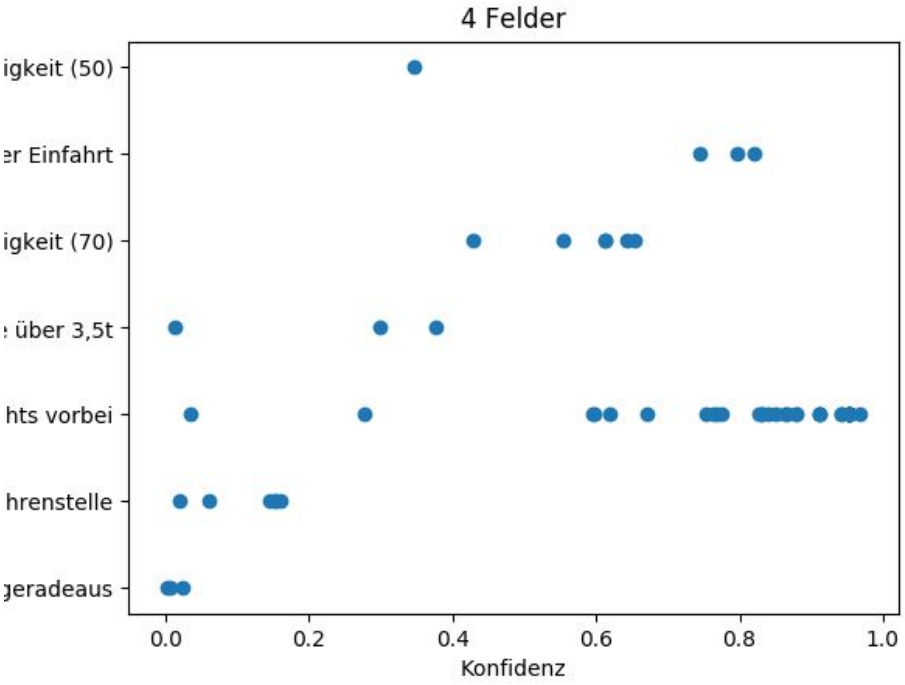
Ansatz 3: Felder mit EA

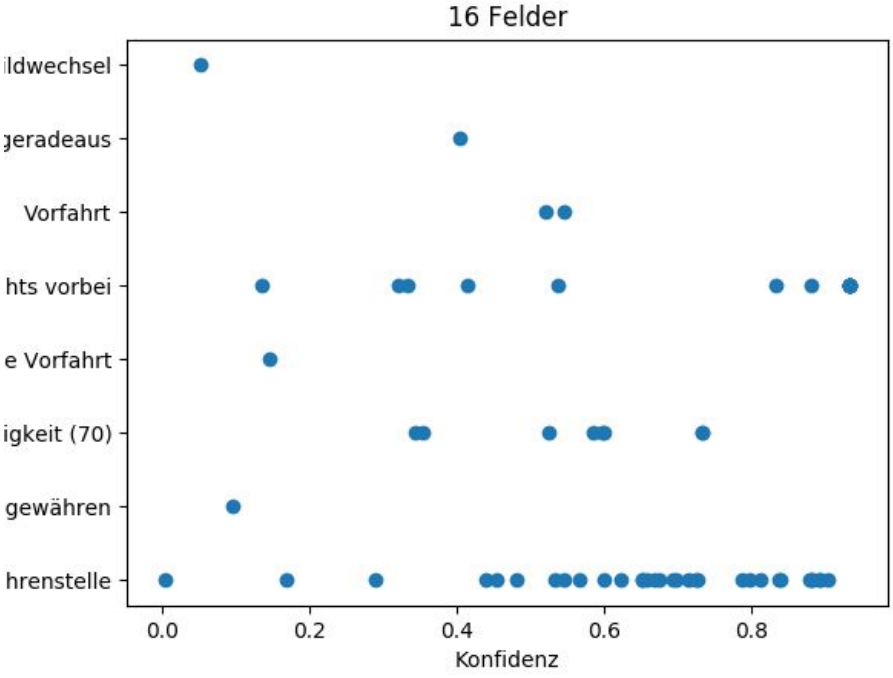
Die Idee für diesen Ansatz entstand durch folgende Bilder der GI:

Zuerst wurden einfarbige Bilder generiert. Diese führten hier zu einer schlechteren Konfidenz, wenn auch hier die dunkleren Farbtöne eine leicht bessere Konfidenz lieferten. Bei einem Bild mit zwei symmetrischen Rechtecken und tendenziell dunkleren Farben entstand ein höherer Konfidenzbereich, jedoch wurden hier auch oftmals nicht die geforderten +90% bei der Erkennung erreicht. Da, wie unter “Farben” erläutert, festgestellt wurde, dass gewisse Farbkombinationen tendenziell auf spezielle Verkehrsschilder abbilden, wurde der Ansatz verfolgt, das Bild in vier gleichgroße Quadrate (Felder) aufzuteilen. Zuerst wurde das Bild komplett Schwarz (RGB 0,0,0) initialisiert.
Des Weiteren wurde bei diesem Ansatz nach jeder Iteration, eine Mutation und ein Crossover (wenn die Konfidenz unter 90% lag) durchgeführt. Die Mutation hat immer bei einem zufälligen Feld die Farbe geändert. Zusätzlich hat die Crossover Funktion zufällig bei einem von zwei Feldern die Farben vertauscht.
Der selbe Ansatz wurde dann verwendet, um zu untersuchen, wie sich die Erkennungsrate bei 16 Felder verhält.

Bilder mit vier Rechtecken haben mitunter die besten Ergebnisse geliefert, gemessen an den benötigten API-Anfragen und der benötigten Anzahl an Versuchen um fünf Bilder zu generieren, welche eine Konfidenz von mindestens 0,9 haben.

Die folgenden Grafiken verdeutlichen, welche Verkehrsschilder bei den Varianten in jeweils einer Iteration erzeugt wurden:


4 Felder
igkeit (50)
er Einfahrt
igkeit (70)
e über 3,5t
hts vorbei
hrenstelle
geradeaus
0.0
0.2
0.4
0.6
0.8
1.0
Konfidenz


16 Felder
ildwechsel
geradeaus
Vorfahrt
hts vorbei
e Vorfahrt
igkeit (70)
gewähren
hrenstelle
0.0
0.2
0.4
0.6
0.8
Konfidenz

Auswertung

Um die Qualität der Lösungen untereinander vergleichen und evaluieren zu können, wurden die folgenden Metriken herausgearbeitet.

Effizienz

Um eine geeignete Metrik über die Zeit zu definieren, wurde die benötigte Anzahl der API-Anfragen gewählt. So kann verglichen werden, wie effizient der Algorithmus funktioniert. Bei weniger benötigten API-Anfragen schafft es der Algorithmus das neuronale Netz zu täuschen, ohne das API-Anfragen-Limit von 60 Anfragen pro Minute zu erreichen. So werden bei weniger API-Anfragen die fünf Bilder generiert, was sowohl auf einen effizienteren Algorithmus, als auch auf ein insgesamt effizienteres Verfahren deutet. Auch werden für das Ergebnis ebenfalls weniger Rechenoperationen benötigt.

Kreativität

Um zu prüfen wie kreativ unsere Lösung ist und wie sich unsere Ansätze in ihrer Kreativität unterscheiden, wurde entschieden, die Anzahl der Farben und der Formen auf den generierten Bildern als Metrik zu wählen. Bei einer höheren Anzahl von Formen und einer höheren Anzahl an Farben sind die Bilder kreativer als bei einem generierten Bild mit weniger verschiedenen Farben und Formen.
Auch fließt die Diversität der Formen mit in die Bewertung ein. Dieses Bewertungskriterium ist jedoch relativ subjektiv. Kreativität und Kunst wird in der Moderne meist am Preis erkannt.

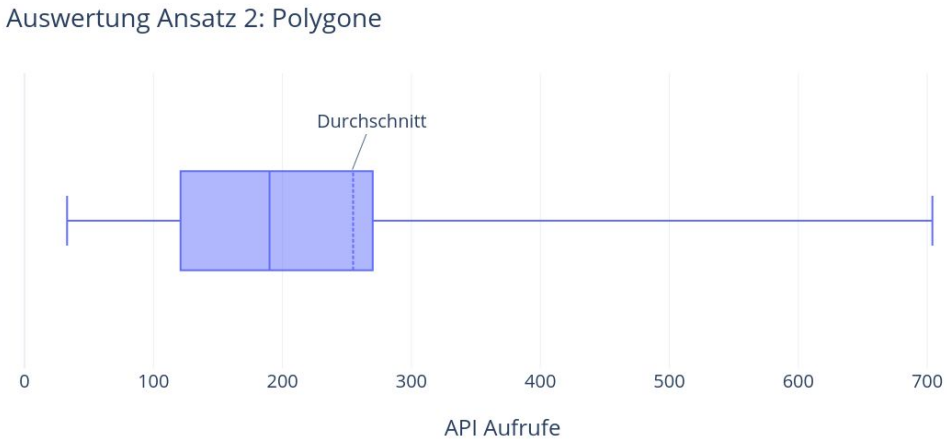
Ansatz 2: Polygone mit EA

Im Folgenden wird der Ansatz zur Generierung von Bildern mithilfe von Polygonen ausgewertet. Für den evolutionären Algorithmus können verschiedene Parameter, wie die Mutationsrate, die Größe der initialen Population, die Anzahl der Punkte für die Polygone, die Farbwerte sowie auch der Kontrast, zwischen den Polygonen angepasst werden. Aufgrund verschiedener Kombinationen dieser Parameter konnten mehrere Presets (Voreinstellungen) erstellt werden, die dementsprechend auch Ergebnisse mit unterschiedlichen Eigenschaften in Bezug auf die Effizienz und Kreativität liefern. Für die Auswertung der folgenden Presets wurden jeweils 10 Iterationen durchgeführt um die fünf Bilder zu generieren.

Preset 1 - The creative one

Bei dieser Konfiguration der Parameter wurde der Kontrastbereich auf 100 bis 756 eingestellt. Damit wird garantiert, dass zumindest ein sichtbarer Kontrast zwischen dem Hinter- und Vordergrund existiert. Trotzdem können die Farben sehr zufällig gewählt werden, da der

Kontrastbereich sehr groß ist. Die Größe der initialen Population liegt in diesem Preset bei 20. Damit enthält die initiale Bildermenge die selben Klassen mehrmals, so dass ein Crossover gefolgt von einer Mutation durchgeführt werden muss bis das Ziel erreicht wurde.
Bei dieser Konfiguration der Parameter wurden ca. 20 API-Anfragen benötigt, um ab dann das Crossover zu nutzen. Das bedeutet, dass in diesem Ansatz eine hohe Anzahl an API-Anfragen benötigt werden könnte, verglichen mit den anderen Ansätzen. Dies lässt dahingehend recht kreative Bilder entstehen. Durch das Crossover werden verschiedene Farben und sehr diverse Formen erzeugt.
Eine Auswertung der durchgeführten Iterationen wird im folgenden Boxplot dargestellt:



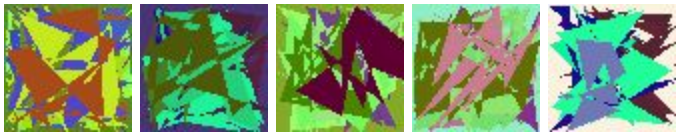
Dabei lag das arithmetische Mittel (Durchschnitt) bei 254,8 API-Anfragen um das Ziel der Generierung zu erreichen. Die entspricht bei der Limitierung der API etwa drei Minuten. Die Anzahl der API-Anfragen unterscheidet sich innerhalb der zehn Iterationen sehr stark, so dass das Maximum 704 API-Anfragen betrug, während das Minimum bei nur 33 Anfragen lag. Die folgende Bilderreihe beinhaltet Ergebnisse mit einer Konfidenz von 96 bis 99 %:

(Erkannte Klassen)


(33 API-Anfragen)

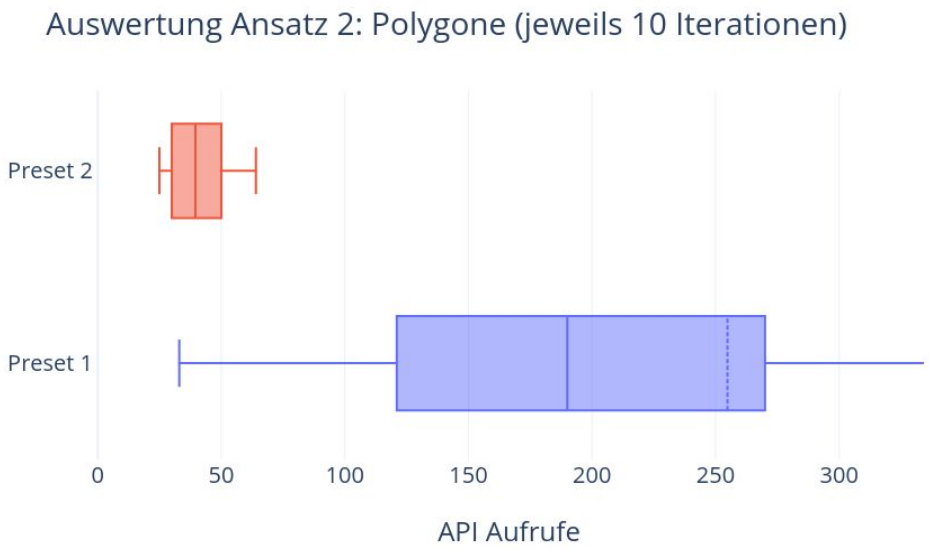
Die beiden folgenden Bilderreihen enthalten Ergebnisse mit höherer Anzahl an API-Anfragen:


(ca. 300 API-Anfragen)


(704 API-Anfragen)

Preset 2 - Fast Initial Generation

Diese Konfiguration ermöglicht die Generation der fünf Bilder mit vergleichsweise wenig API-Anfragen. Dafür wurde die Größe der initialen Population auf 60 gesetzt und der Kontrastbereich auf den Bereich von 380 bis 765 begrenzt.
Bei diesem Ansatz kann das Crossover erst nach 60 API-Anfragen eingesetzt werden. Die fünf Zielbilder werden jedoch schon innerhalb der initialen Population erzeugt. Dies war bei 9 von 10 Iterationen der Fall. Das arithmetische Mittel für die API-Anfragen liegt bei 41,1. Ein Vergleich zu dem Preset 1 ist im folgendem Boxplot zu sehen:



Die erzeugten Bilder sehen im Vergleich zu dem Preset 1 auch etwas weniger kreativ aus, da sie nur aus zwei Farben mit einem Polygon bestehen, wie es in folgender Bilderreihe zu sehen ist.

-  (Erkannte Klassen)
-  (25 API-Anfragen)

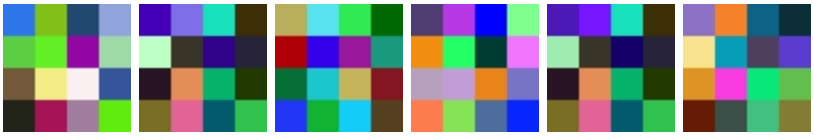
Insgesamt punktet dieses Preset mit der Effizienz, da mit wenigen API-Anfragen das Ziel der Generierung erreicht wird.

Ansatz 3: Felder mit EA

Insgesamt hat sich hier gezeigt, dass die Variante mit den vier Quadraten der bessere Ansatz ist. Die Erkennungsrate mit einer Konfidenz von >90% liegt hier im Mittel bei 43 API-Anfragen. Am häufigsten wurde das Verkehrsschild “Rechts vorbei” generiert. Folgend sind 6 generierte Bilder zu “Rechts vorbei”:

Als Vergleich würde ein Bild für dasselbe Verkehrsschild bei 16 Quadraten wie folgt aussehen:

Bei der Variante mit den 16 Quadraten liegt die Erkennungsrate bei der gleichen Anzahl an Iterationen bei 49 API-Anfragen im Mittel. Das lässt darauf schließen, dass die vielen verschiedenen Farbtöne die Erkennung des Bildes als Verkehrsschild erschweren. Hier wurde am häufigsten das Verkehrsschild “Gefahrenstelle” generiert. Folgend sind sechs generierte Bilder zu “Gefahrenstelle”:



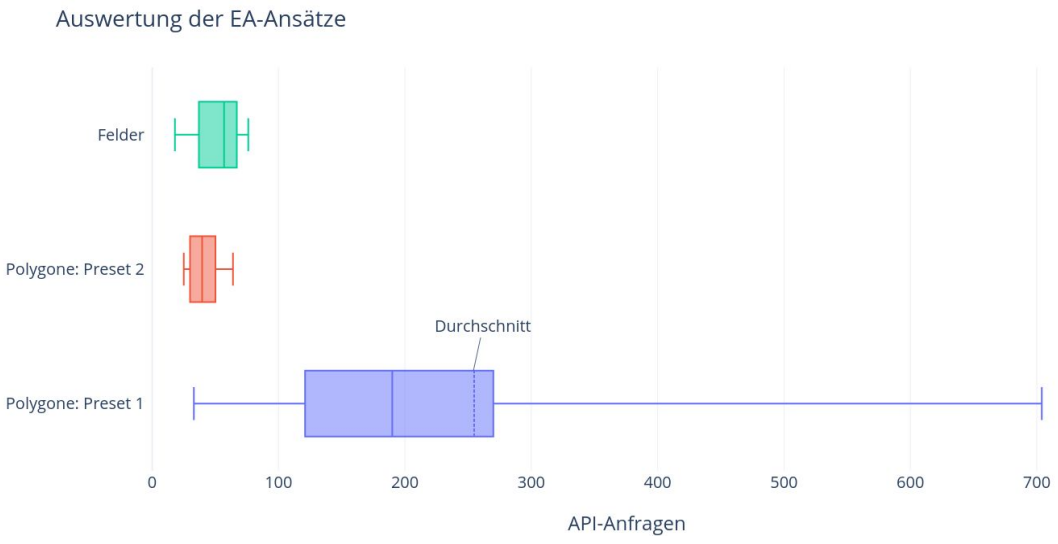
Als Vergleich würde ein Bild für das selber Verkehrsschild bei 4 Quadraten wie folgt aussehen:

Wie auf den Bildern zu erkennen ist, sind gewisse Farbtöne bzgl. der jeweiligen Verkehrsschilder immer vertreten. Beispielsweise ist bei dem generierten Verkehrsschild “Rechts vorbei” auf jedem Bild ein Grünton und ein Blauton vorhanden. Ein ähnliches Muster ist bei den generierten Bildern mit 16 Quadraten bzgl. des Verkehrsschildes “Gefahrenstelle” zu beobachten. Hier sind besonders dunklere Farbtöne dominierend.
Das lässt darauf deuten, dass die Farben und die Farbgemische von der API als Referenz genommen werden, um zu schauen, welches Verkehrsschild repräsentiert werden soll.
Die Schilderkennung kann recht zuverlässig überlistet werden. Jedoch besteht bei beiden Varianten die Problematik, dass in den meisten Fällen nicht fünf unterschiedliche

Verkehrsschilder generiert werden. Somit bietet der Ansatz zwar eine akzeptable Lösung, allerdings auf Kosten der Diversität bzgl. der verschiedenen Verkehrsschilder.
Der Algorithmus wurde testweise angepasst, um zu schauen, wie es sich mit der Erkennungsrate und den benötigten API-Anfragen verhält. Er wurde dahingehend verändert, dass bei jeder Iteration fünf verschiedene Verkehrsschilder generiert werden.
Jedoch hat sich gezeigt, dass sich die Anzahl benötigter API-Anfragen sich sehr erhöht hat. Bei zehn Iterationen lag die durchschnittliche Anzahl an benötigten API-Anfragen bei 53. Dies liegt weit über den benötigten durchschnittlichen API-Anfragen bei dem Algorithmus, bei dem nur 43 benötigt wurden.
Daher wurde dieser Ansatz verworfen, um weiterhin eine möglichst effiziente Lösung anbieten zu können. Denn die Lösung mit den Feldern ist nicht kreativ bzgl. unserer Metrik. Daher hat er mit der höheren durchschnittlichen API-Anfragen in unserer Auswertung die qualitativ schlechteren Ergebnisse geliefert.

Vergleich der Ansätze

Es hat sich gezeigt, dass bzgl. unserer Ansätze die Auswertung wie folgt gestaltet:
Ansatz 2 mit Preset 1 generiert die kreativeren Bilder im Gegensatz zu den anderen Verfahren. Die Bilder beinhalten mehr Farben als bei dem gleichen Ansatz mit Preset 2 und auch sind die Formen diverser. Jedoch geht die Kreativität zu Lasten der Effizienz. Ansatz 2 mit Preset 2 hat gezeigt, dass mit weniger Kreativität eine höhere Effizienz erreicht wird. Dieser Ansatz benötigt weniger API-Anfragen, hat jedoch weniger Formen und auch nur 2 verschiedene Farben in seinen produzierten Bildern.
Im Vergleich zu Ansatz 3 benötigt dieser im Durchschnitt ähnlich wenige API-Anfragen wie Ansatz 2 mit Preset 2. Jedoch ist er von der Kreativität her eher der unkreativste Ansatz von allen. Die Bilder haben zwar unterschiedliche Farben, jedoch ist der Aufbau der Bilder immer exakt gleich.
Um eine Auswertung innerhalb der Ansätze 2 und 3 durchzuführen, wurden Ansatz 2 mit Preset 1 bzw. 2 und Ansatz 3 mit dem Generieren der fünf verschiedenen Bildklassen ausgewählt.



Diskussion

Das Besondere an den Lösungsansätzen ist, dass sie sich auf andere neuronale Netze (NNs) übertragen lassen. Der Lösungsweg kann jedoch beibehalten werden. Durch die Polygone haben die Bilder zufällige Farben und Formen. Durch mehrmaliges Anwenden des Crossovers kann eine Vielfalt von verschiedenen Bildern generiert werden und auf verschiedene Problematiken angewendet werden.
Auch lassen sich die generierten Bilder als Trainings- und Test-Sets verwenden, um so ein neuronales Netz zu optimieren.

Problematisch ist jedoch, dass mit einer geringen Wahrscheinlichkeit die produzierten Bilder aus dem Ansatz 2 dem ursprünglichen Bild ähneln könnten, da dort zufällige Polygone kreiert werden. Gegebenenfalls könnte es dort zu unangemessenen Bildern kommen, beispielsweise zu verfassungswidrigen Symbolen.
Weiterhin existiert keine Kontrolle über die Dauer für die Generierung der jeweiligen Bilder, da die Bilder immer zufällig produziert werden. Durch die Auswertung der Generierung konnte jedoch ein Mittelwert für die geschätzte Dauer ermittelt werden.

Praktisch einsetzen ließe sich diese Lösung für das zukünftige autonome Fahren. Hierfür könnten die generierten Bilder als echte Verkehrsschilder implementiert werden. Die Bilder sind für den Menschen nicht als Verkehrsschild erkennbar und so könnte eine Manipulation von außen durch den Menschen reduziert werden. Es wäre dann nicht mehr ersichtlich, wie sich die Veränderung des generierten Verkehrsschildes auf die Erkennung des Schildes auswirkt. Denkbar wäre, dass es sich höchstens auf die Konfidenz der Erkennung auswirkt, aber sich nicht ein anderes Verkehrsschild durch die Manipulation ergeben würde.

Softwarearchitektur

Um die Generation von Bildern nutzerfreundlich zu gestalten, bietet die Software eine GUI. In dieser kann der Datensatz eingesehen werden, sowie einige Einstellungen vorgenommen werden.

Grundaufbau

Die grundsätzliche Architektur der Software orientiert sich an dem Model-View-Controller (MVC) Pattern. Python wird dabei, bis auf wenige Ausnahmen, rein objektorientiert eingesetzt. Die Ausnahmen sind hierbei einige Module, die in vergleichbaren Sprachen als statische Klassen implementiert worden wären.
Während der Entwicklung lag der Fokus prinzipiell auf zwei Komponenten. Im Vordergrund stehen die Bildgeneratoren, in denen die verschiedenen Ansätze umgesetzt werden. Diese umgesetzten Features müssen dann durch die GUI anwenderfreundlich zur Verfügung gestellt werden. Nach dem MVC Pattern bietet die GUI Komponente die View auf die generierten Bilder und den Datensatz. Den Controller bilden hier die Bild-Generatoren. Unter den Bereich des Modells lassen sich der Datensatz, aber auch der Zustand der Generatoren zusammenfassen.
Um die GUI umzusetzen wird Qt5 über den PyQt5[6] Wrapper verwendet. Es existieren insgesamt zwei Fenster, die fest zum Programm gehören, sowie ein Einstellungsfenster, welches zum Bildgenerator gehört.
Die Berechnungen der Bildgeneratoren werden in einem separaten Thread ausgeführt. Hauptsächlich, da für die API Anfragen die Library Python-Requests[7] verwendet wird. In deren Implementation blockieren HTTP Anfragen, wodurch die GUI in regelmäßigen Abständen nicht auf Interaktionen reagieren würde. Damit Informationen aus dem Generator-Thread zurück an die GUI gegeben werden können, werden sogenannte Signals aus PyQt5 verwendet.

Modularität

Die Software soll leicht zu erweitern sein. Gerade um das Testen von verschiedenen Algorithmen zu ermöglichen können beliebig viele Bildgeneratoren eingepflegt werden. Diese erben allesamt von der Klasse *AbstractGenerator*.
Nach je einer Iteration des Algorithmus wird der momentane Stand in der GUI angezeigt. Hierbei kann es sich, je nach Arbeitsweise, um die bisher am besten bewerteten Bilder handeln. Ob der Bildgenerator über Optionen verfügt und wie diese umgesetzt werden, ist der jeweiligen Klasse überlassen. Der evolutionäre Algorithmus mit Polygonen nutzt beispielsweise ein separates Qt Fenster.

[6] https://www.riverbankcomputing.com/software/pyqt/intro (Abgerufen am 13.01.2019)
[7] https://python-requests.org (Abgerufen am 13.01.2019)

Der Evolutionäre Algorithmus

Kernstück der Software ist der Bildgenerator, der einen evolutionären Algorithmus einsetzt. Dieser ist als Klasse implementiert, die von dem vorher beschriebenen *AbstractGenerator* erbt. Es werden je 5 Bilder simultan über mehrere Iterationen generiert. Ein Aufruf der *step()* Methode des Generators entspricht einer Generation des evolutionären Algorithmus. Zum Zeichnen der Bilder wird die Python Image Library[8] (PIL) eingesetzt.
Die betrachtete Population wird hierbei als Instanzenparameter über die Generationen hinweg gespeichert. Sämtliche Individuen der Population sind Tupel der Form *(image, confidence, colors, shapes, class, lastCrossover)*. Der Wert *Image* hält eine Referenz auf das zum Individuum gehörende Bild. *Confidence* und *Class* entstammen den Werten die von der API geliefert werden und stellen die Fitness dar. Haben diese den Wert *None* wurde das Individuum noch nicht bewertet. *Confidence* entspricht hierbei direkt der Fitness da es das Ziel ist, diesen Wert zu maximieren. Ist die erkannte Klasse nicht in dem Array *self._targetClasses* vorhanden, fällt die Fitness des Individuums auf Null. Werden mehrere Klassen in dem Bild erkannt, wird lediglich diejenige gespeichert, die die höchste Konfidenz aufweist und Teil der gewählten Klassen ist.
Colors und *Shapes* beinhalten schließlich die Eigenschaften des Individuums, also dessen Farben und Polygone. *lastCrossover* wird lediglich verwendet um zu verhindern, dass ein durch Crossover entstandenes Individuum in der selben Generation mutiert.
Zusätzlich bietet der Bildgenerator die Möglichkeit Wertebereiche und Zielgröße der Population vom Anwender konfigurieren zu lassen. Hierfür wird ein separates Fenster eingesetzt, welches Zugriff auf die Werte der Generator Instanz besitzt. Die eingestellten Werte können in Form von mehreren Presets gespeichert und wieder abgerufen werden.

Konfiguration und weitere externe Dateien

Einige Werte des Programms können dauerhaft vom Nutzer verändert werden. Dazu gehören die URL und der verwendete Key der API und die Presets des evolutionären Algorithmus. Um diese Informationen zu speichern wird eine Datei namens *.kollektiv5.ini* im Heimverzeichnis des Benutzers erstellt. Um vordefinierte Werte für den ersten Start der Software zur Verfügung zu stellen, ist eine weitere solche Datei Teil des Programmcodes. Dafür existiert ein besonderer Ordner namens *resources* in dem Python Modul *kollektiv5*.
In diesem Ordner befinden sich neben den Standardwerten noch weitere Dateien. Zum Einen die Vorschaubilder für die Klassen des Datensatzes, sowie die Definition des Datensatzes selbst. Dieser ist in der JavaScript Option Notation (JSON) gegeben und enthält ein Array aller Klassen. Pro Klasse ist der textuelle Name, der numerische Identifikator und der Pfad zu dem Vorschaubild hinterlegt. Zusätzlich ist gespeichert, ob die Klasse von der API erkannt wird.

[8] https://python-pillow.org/ (Abgerufen am 13.01.2019). Bei Pillow handelt es sich um einen PIL kompatiblen Fork, da PIL nicht mehr aktiv entwickelt wird.

Testen

Beim Entwickeln eines Testkonzeptes kam die Zweiteilung in Algorithmus und GUI zu Gute. Schließlich lässt sich die Arbeitsweise des Algorithmus deutlich einfacher automatisiert testen als das Interface. Daher wurden für diesen einige Unit Tests angelegt. In diesen wird geprüft ob die Spezifikation erfüllt wird, also ob fünf verschiedene Klassen mit einer Konfidenz von über 90% generiert werden können. Ebenso werden die Teilabschnitte des evolutionären Algorithmus getestet. Die initial erstellte Population wird hinsichtlich ihrer Größe und Fitness bewertet.
Einige erforderliche Eigenschaften für den Crossover Schritt werden gesondert geprüft. Hier müssen mindestens zwei Bilder gleicher Klasse vorliegen, da nur diese Bilder ein Elternpaar bilden. Dazu wird geprüft, ob die verwendete Crossover Methode im Mittel eine Verbesserung liefert. Somit konnten in der Anfangsphase sehr schnell verschiedene Ansätze verglichen werden.

Die GUI automatisiert zu testen hat sich als schwierig erwiesen. Die Funktionalität wurde hier während der Entwicklung manuell überprüft. Da die GUI jedoch im Gesamtbild keine große Menge an Funktionen aufweist, ist diese Art und Weise noch umsetzbar.
Weitaus wichtiger war es, die GUI möglichst bedienerfreundlich zu gestalten. Hierbei kam bereits die Aufgabenteilung innerhalb der Gruppe zuGute, da so direkt Feedback von anderen Gruppenmitgliedern eingeholt werden konnte, die nicht aktiv an der GUI entwickelt haben.

Konventionen

Als Python Konvention wird PEP 8[9] verwendet. Die darin enthaltenen Regeln verbessern die Lesbarkeit des Python Codes. Beispielsweise wird verlangt, dass bzgl. der Variablen passende Namen gewählt werden.

```
# Not recommended
x = 'John Smith'
# Recommended
name = 'John Smith'
```

Deutlich grundsätzlicher ist allerdings die Wichtigkeit einen einheitlichen Regelsatz innerhalb der Gruppe festzulegen. Da jeder Teilnehmer persönliche Präferenzen besitzt, würden sich Codeabschnitte stark voneinander unterscheiden. Ein späteres Lesen des Codes wäre somit erschwert.

Wartbarkeit

Durch den modularen Grundaufbau ist die Software ohne viel Aufwand erweiterbar. Neue Bildgeneratoren können leicht eingepflegt werden und der vorhandene Algorithmus kann durch

[9] https://www.python.org/dev/peps/pep-0008/#introduction (Abgerufen am 13.01.2019)

Anpassung der Individuen, sowie Mutations- und Crossoverfunktion, um neue Eigenschaften erweitert werden.
Die GUI wird hauptsächlich per Code erzeugt und kann so auch angepasst werden. Eine Ausnahme bildet das Einstellungsfenster des evolutionären Algorithmus. Dieses ist mit dem graphischen Qt Designer erzeugt. Über das Modul *PyQt5.uic.pyuic* kann aus dem Design eine Python Klasse erzeugt werden. Der genaue Befehl dafür ist der Readme des Programmes zu entnehmen.
Soll das System auf einer anderen API eingesetzt werden, so muss unter Umständen der Aufbau der API Anfragen verändert werden. Hierfür existiert ein Modul Names *api* im Unterordner *utils* der Software.

Fazit

Von den drei vorgestellten Ansätzen wurden die beiden auf dem Evolutionären Algorithmus basierenden Ansätze weiterverfolgt und implementiert. Beide erfüllen die Anforderung, fünf Bilder zu generieren, die mit einer Konfidenz von über 90% erkannt werden. Jedoch wird der Ansatz 2 (Polygone) favorisiert. Er produziert, verglichen mit dem Ansatz 3 (Felder), Bilder, die sich noch weniger ähneln. Ansatz 3 verwendet zwar verschiedene Farben, aber Ansatz 2 kann sowohl eine höhere Anzahl von Farben als auch verschiedene Formen implementieren. Des Weiteren generiert der Ansatz 2 die fünf Bilder verschiedener Klassen mit einem arithmetischen Mittel von nur 41,1 API-Anfragen, was zu einer schnelleren Lösung führt.
Als zusätzliches Feature wurde zu den beiden Ansätzen eine grafische Oberfläche entwickelt. Über diese lassen sich die Ansätze auswählen und die Konfiguration der Parameter für die Generierung der Bilder justieren. Außerdem können in der graphischen Oberfläche die Bilder während der Generierung angezeigt und gespeichert werden.

Anhang

Erkannte und nicht erkannte Klassen mit IDs

Erkannte Klassen:

- Zulässige Höchstgeschwindigkeit (20), 0
- Vorfahrt, 12
- Kreisverkehr, 40
- Ende des Überholverbotes für Kraftfahrzeuge mit einer zulässigen Gesamtmasse über 3,5t, 42
- Fußgänger, 27
- Gefahrenstelle, 18
- Einmalige Vorfahrt, 11
- Zulässige Höchstgeschwindigkeit (30), 1
- Fahrradfahrer, 29
- Verbot für Kraftfahrzeuge mit einer zulässigen Gesamtmasse von 3,5t, 16
- Zulässige Höchstgeschwindigkeit (50), 2
- Zulässige Höchstgeschwindigkeit (120), 8
- Ausschließlich rechts, 33
- Baustelle, 25
- Ende der Geschwindigkeitsbegrenzung (80), 6
- Stoppschild, 14
- Unebene Fahrbahn, 22
- Kurve (links), 19
- Ende aller Streckenverbote, 32
- Kurve (rechts), 20
- Doppelkurve (zunächst links), 21
- Zulässige Höchstgeschwindigkeit (70), 4
- Wildwechsel, 31
- Ausschließlich geradeaus, 35
- Überholverbot für Kraftfahrzeuge mit einer zulässigen Gesamtmasse über 3,5t, 10
- Verbot der Einfahrt, 17
- Vorfahrt gewähren, 13
- Überholverbot für Kraftfahrzeuge aller Art, 9
- Schleudergefahr bei Nässe oder Schmutz, 23
- Verbot für Fahrzeuge aller Art, 15
- Zulässige Höchstgeschwindigkeit (60), 3
- Zulässige Höchstgeschwindigkeit (100), 7
- Rechts vorbei, 38
- Zulässige Höchstgeschwindigkeit (80), 5

Nicht erkannte Klassen:

- Ampel, 26
- Kinder, 28
- Schneefall, 30
- Ausschließlich links, 34
- Geradeaus oder rechts, 36
- Geradeaus oder links, 37
- Links vorbei, 39
- Ende des Überholverbotes für Kraftfahrzeuge aller Art, 41

Alle 43 Verkehrszeichen von GTSRB Dataset


20
30
50
60
70
80
80
100
120
STOP