InformatiCup 2018: Generative Adversarial Examples

Julian Stastny
Freie Universität Berlin

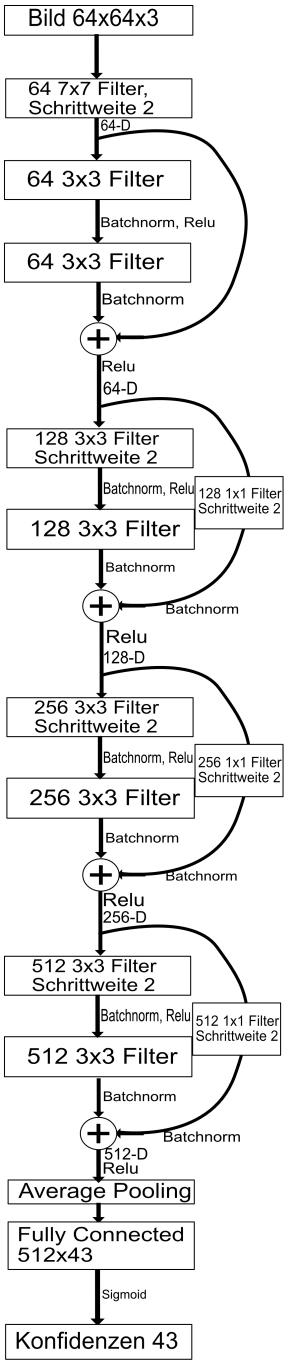
Florian Dorner
Freie Universität Berlin

2018
Januar

Neben einer allgemeinen Recherche im Themenkomplex "Adversarial Examples" sind auch die spezifischen Bedingungen durch die Aufgabenstellung und das gegebene neuronale Netz für unsere Strategie relevant.
Zwei Faktoren beeinflussen unseren Ansatz maßgeblich. Erstens, dass es sich bei den Adversarial Examples nicht um Verkehrsschilder handeln soll und zweitens, dass die Anzahl der Anfragen pro Minute auf 60 limitiert ist. Aufgrund der limitierten Anzahl an Anfragen pro Zeiteinheit basiert nur einer unserer Ansätze (Sticker-Attacke) darauf, die Beispiele direkt auf der Black Box zu optimieren, während unsere beiden weiteren Ansätze auf einem von uns rekonstruierten Modell optimieren. Der erste (FGSM) ist dabei auf die Berechnung der Gradienten angewiesen, während der zweite (ein Generative Adversarial Network (GAN)) keine Gradienten benötigt und damit theoretisch auch auf der Black Box operieren kann.
Um eine erste Approximation für die Black Box zu bekommen, wurde zunächst der gesamte GTSRB-Datensatz[1] an den Server geschickt, wobei wir die Vorhersagen der BB als Labels für unser destilliertes Modell verwenden. Die BB gibt immer für fünf Klassen eine Vorhersage zurück, wahrscheinlich handelt es sich hier um die mit der höchsten Konfidenz. Interessanterweise sind diese fünf Konfidenzen in der Summe manchmal so niedrig, dass mindestens eine der Konfidenzen für die übrigen 38 Klassen höher als die niedrigste ausgegebene Konfidenz sein müsste. Aufgrund dieser Beobachtung vermuten wir, dass die BB in der letzten Schicht eine Sigmoid- und nicht die sonst übliche Softmaxfunktion benutzt. Unsere rekonstruierte White Box (WB) nutzt daher auch eine Sigmoidfunktion in der letzten Schicht.
Bei unsererer WB handelt es sich um ein Residual Neural Network [2]. ResNets sind eine Erweiterung von Convolutional Neural Networks, bei der Schichten übersprungen werden können. Dadurch wird es Signalen ermöglicht, relativ ungestört durch viele Netz-

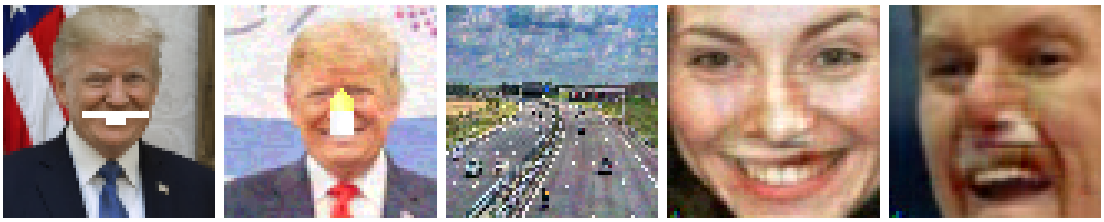
werkschichten zu gelangen. Die ResNet18-Architektur ist dabei ein guter Kompromiss zwischen hoher Genauigkeit und schneller Trainierbarkeit und war unsere erste Wahl für einen Prototyp für die WB.

Da die BB keine besonders hohe Genauigkeit auf dem GTSRB-Datensatz aufweist, stellt sich eine an ResNet18 angelehnte Architektur mit halb so vielen Schichten und leichten anderweitigen Änderungen (Kernel-Größe 7 in der ersten Schicht und Bias im ersten Convolutional Layer) als ausreichend zur Rekonstruktion heraus (Mean Squared Error von 0.00018 auf dem GTSRB-Test Datensatz und von 0.00210 auf dem Labeled Faces in the Wild[3] Datensatz, auf dem nicht trainiert wurde). Das in der fertigen Software verwendete Netzwerk wurde auf dem zu $64 * 64$-Bildern konvertierten GTSRB-Datensatz und 100.000 durch Rotation, Verschiebung, Zoom, Scherung und Helligkeit augmentierten Daten daraus, sowie 50000 Bildern mit zufälligem Rauschen trainiert. Als Labels dienten die Vorhersagen der BB für die jeweilgen Bilder.



Eine plausible Erklärung dafür, warum sich moderne neuronale Netze durch winzige Änderungen im Bildsignal massiv stören lassen, und dafür dass oftmals die selbe Störung eine Vielzahl verschiedener Netze zu Klassifikationsfehlern bringen kann, liefern Goodfellow et al. in [4]: Formal gesehen handelt es sich bei neuronalen Netzen um die abwechselnde Verkettung (affin) linearer Funktionen und nichtlinearer Aktivierungsfunktionen. Affin lineare Funktionen haben die Eigenschaft, dass die selbe Störung des Arguments unabhängig von ebendiesem Argument zur selben (absoluten) Störung des Funktionswertes führt. Ohne die nichtlinearen Aktivierungsfunktionen würde also eine Bildstörung, die für ein Bild dazu führt, dass das neuronale Netz einer Klasse einen höhere Wert zuordnet, auch bei jedem anderen Bild den ausgegebene Wert für diese Klasse erhöhen, und zwar um genau so viel.

Qualitativ dürfte sich daran in den meisten Fällen auch durch die für Klassifikation normalerweise am Ende des neuronalen Netzes stehende Softmaxfunktion, die gegebene Werte in eine Wahrscheinlichkeitsverteilung umwandelt nichts ändern: die Softmaxfunktion nimmt das Exponential jedes Eingangswerts und normiert den resultierenden Vektor zu einer Wahrscheinlichkeitsverteilung. Aufgrund der Monotonie der Exponentialfunktion sorgt eine höhere Eingabe für ein Argument normalerweise auch für eine höhere zugehörige Wahrscheinlichkeit, auch wenn es gelegentlich durch Verschiebung der relativen Größen der anderen Eingabewerte anders kommen kann. Für weitestgehend lineare Netze erwarten wir also, dass die selbe Bildstörung für eine Vielzahl von Bildern sehr ähnliche Effekte hat.



Abgabebilder: Von links nach rechts: Stickerangriff, Stickerangriff mit anschließendem FGSM (Bound 10), FGSM (Bound 10), GAN mit anschließendem FGSM (Bound 5), GAN. Die Sticker wurden aus ästhetischen Gründen minimal auf manuelle Weise verkleinert und sind auch in dieser Form in der Software enthalten.
Klassifikationen der Bilder (Unmodifiziert): Verbot der Einfahrt: 0.99 (0), Baustelle: 0.99 (0.04), Zulässige Höchstgeschwindigkeit 30: 0.97 (0.26), Baustelle: 0.98 (0), Baustelle: 1 (0)
Bildquellen: Weißes Haus, Weisßes Haus,
https://commons.wikimedia.org/wiki/File:Garching_Bundesautobahn_9 .jpg, Faces in thw Wild[3]

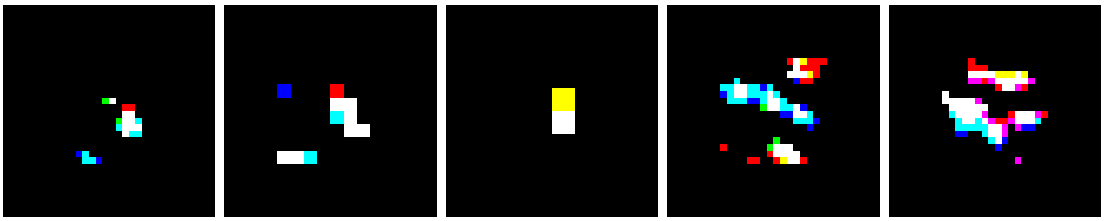
Moderne neuronale Netze sind nun aber oftmals, um das Training zu erleichtern genau darauf ausgelegt, sich möglichst linear zu verhalten: Bei Netzen mit Sigmoidfunktionen, wird meist versucht, die Eingabewerte größtenteils auf den quasi-linearen Bereich der Aktivierungsfunktion zu Beschränken und diesen nur wenn es für das Lernen einer Funktion wirklich nötig ist zu verlassen (Siehe auch /citeGlorot10). Die populären ReLU-Funktionen [5] verhalten sich stets linear oder geben Null aus. Auch wenn sich theoretisch jede stetige Funktion durch ein Netz mit ausreichend vielen ReLU-Neuronen approximieren lässt, wofür es essentiell ist, dass die ReLU-Funktion eben nicht immer linear ist, ist ein neuronales Netz, das ausschließlich ReLU-Neuronen besitzt nur in der Lage, stückweise lineare Funktionen exakt darzustellen. Die Anzahl der "Stücke" wächst

mit der Anzahl und Größe der Schichten des Netzes aufgrund der damit verbundenen häufigeren Anwendung der Aktivierungsfunktion, weshalb sich gerade kleinere Netze mit ReLU in großen Bereichen des Eingaberaums linear verhalten. Zwei Netze, die in der selben Region linear sind und die selbe Zielfunktion approximieren sollen, werden sich weiterhin in dieser Region meistens ähnlich verhalten, da die optimale Approximation einer Funktion durch lineare Funktionen für die meisten gängigen Maße für Approximationsgüte (wie zum Beispiel die $L2$-Norm) eindeutig ist.

Die Ähnlichkeit neuronaler Netze zu linearen Funktionen erklärt nicht nur, warum Täuschungsversuche oftmals generalisieren, sondern kann auch motivieren, warum oftmals kleine Störungen ausreichen um einen großen Effekt auf die Ausgabe eines Netzes zu haben: Linearität und eine hohe Dimension des Eingaberaums erlauben, dass sich die einzeln betrachtet kleinen Effekte kleiner Störungen einzelner Dimensionen aufaddieren, was bei zufälligen Störungen kein Problem ist, da sich die Veränderungen in der Ausgabe in Erwartung aufheben, aber bei gezielten Attacken zu massiven Fehlern in der Ausgabe durch winzige Störungen in der Eingabe führen kann.

Unser erster und einfachster Ansatz basiert vollständig auf der angenommenen Ähnlichkeit des zu täuschenden neuronalen Netzes zu einer linearen Funktion: Ausgehend von einem Schwarzen Bild, wird für jede Quadratische Pixelgruppe auf einem Raster einzeln jeder Farbkanal separat von 0 auf 255 (den Maximalwert) gestellt und die Ausgabe des zu täuschenden neuronalen Netzes für das resultierende Bild gespeichert (ein analoger Ansatz für weißen Hintergrund ist natürlich auch möglich). Unter Annahme von Linearität lässt sich so die maximal mögliche Störung der Ausgabe durch die Veränderung der Pixelgruppe auf diesem Kanal bestimmen. Im linearen Fall addieren sich die Effekte der Störung zudem auf, sowohl bezüglich der Farbkanäle als auch bezüglich der Pixelgruppen. Um nun das Netz dazu zu bringen, ein Bild als gegebenes Verkehrszeichen zu klassifizieren, beginnen wir mit einem Schwarzen Bild und aktivieren einen Farbkanal immer dann für eine Pixelgruppe, wenn dessen Aktivierung auf dem Schwarzen Bild zu einer Vergrößerung der Konfidenz des Netzwerkes in das gegeben Verkehrszeichen über einem Schwellenwert geführt hat. Für ein beliebiges Ausgangsbild können wir nun das veränderte schwarze Bild als "Sticker" verwenden: Pixelgruppen des Ausgangsbildes, bei denen der Sticker nicht schwarz ist werden durch die Pixelgruppe auf dem Sticker ersetzt (alternativ kann das ganze auch kanalweise "transparent" gemacht werden). Die Linearität sorgt nun für eine ähnliche Vergrößerung der Konfidenz wie bei dem Schwarzen Bild. Da wir nicht auf allen Kanälen mit 0 starten, ist der Effekt etwas schwächer als bei

dem schwarzen Ausgangsbild. Trotz der wahrscheinlich nichttrivialen Abweichung von der Linearität des Netzes lassen sich mit dieser Methode bereits gute Resultate erzielen:



Sticker für Pixelblockgröße 7, 4 und 2 und die Klasse "Baustelle", sowie Sticker der Größe 2 für die Klassen "Zulässige Höchstgeschwindigkeit 60", sowie -"70". Die Konfidenzen liegen für den 7-Pixel Sticker bei 98% *und sonst stets über* 99.9%*. Die Gesamtgröße des Rasters betrug für den ersten Sticker* 14×14 *und für die anderen jeweils* 32×32 *Pixel. Für die Generierung der Sticker wurden für die verschiedenen Blockgrößen 30, 256 und 812 Anfragen an das neuronale Netz benötigt (Abfrage der Konfidenz für einzelne Blöcke sowie Probe ob der Sticker auf schwarzem Hintergrund zu mehr als* 95% *Konfidenz führt). Dadurch konnten mit einem Schwellenwert von* 1% *für die Aufnahme eines Blockes Sticker für 3, 16 und 9 verschiedene Klassen generiert werden.*

Konfidenz	kein Sticker	2-Block (32×32)	4-Block (32×32)	7-Block (14×14)
Geschwindigkeit 120	0.02013	0.84139	0.898033	0.13658
Verbot der Einfahrt	0.13153	0.64134	0.73279	0.63858
Baustelle	0.10344	0.62279	0.74832	0.41303

Durchschnittliche Konfidenzen für den Stickerangriff auf drei verschiedene Klasse, jeweils evaluiert an 25 Testbildern. Mehr tatsächlich von den Stickern ausgefüllte Fläche führt zu höheren Konfidenzen. Mehr Details (kleinere Blöcke) sollten dies theoretisch auch. Hier lässt sich der gegenteilige Effekt beobachten, was wohl an dem selben verwendeten Schwellenwert für die Aufnahme einer Pixelgruppe in den Sticker liegt. Da kleinere Pixelgruppen diesen nur seltener erreichen, nehmen die 2-Block Sticker deutlich weniger Platz ein (Bei der ersten Klasse 92% *der Fläche des 4-Block-Stickers und für den Rest jeweils weniger als die Hälfte).*

Im Gegensatz zu den folgenden Ansätzen muss für den Stickerangriff kein eigenes neuronales Netz trainiert werden, wodurch die resultierende Attacke deutlich einfacher und billiger zu reproduzieren und auf andere neuronale Netze auszuweiten ist. Die vergleichbar starke Störung des Ausgangsbildes sorgt zwar einerseits dafür, dass manipulierte Bilder deutlich als nicht natürlich wahrnehmbar sind, andererseits lassen sich eine Vielzahl von Ausgangsbildern ohne jeglichen Mehraufwand (wie das Auswerten eines anderen neuronale Netzes) stören. Weiterhin stellt die "Unnatürlichkeit" der ausgegebenen Bilder für einen potentiellen Angreifer nur dann ein Problem dar, wenn diese auch als Gefahr erkannt werden. Einige der generierten Bilder sehen aus wie Sticker, die auch plausibel aus ästhetischen oder symbolischen Gründen an einem Ort, wie zum Beispiel dem Rückfenster eines PKWs kleben könnten. Weiterhin führen die generierten Sticker oftmals so robust zu Fehlklassifikationen, dass das Entfernen und Addieren einzelner klei-

nerer Pixelgruppen den Täuschungseffekt nicht beeinflusst, was es Angreifern ermöglicht, die Sticker noch unverdächtiger aussehen zu lassen. Immerhin sollte die Verteidigung gegen diesen Angriff vergleichsweise einfach sein: Da die Effektivität der Attacke stark von der Nähe des verwendeten Netzes zur Linearität abhängt, ist zu erwarten, dass die Attacke für tiefere neuronale Netze mit vielen Neuronen pro Schicht ein deutlich kleineres Problem darstellt. Auch wenn solche Netze deutlich mehr Daten benötigen um eine Überanpassung an den Trainingsdatensatz zu verhindern und zu generalisieren, ist doch hoffentlich davon auszugehen, dass für sicherheitsrelevante Anwendungen entsprechend mächtige Modelle genutzt werden, vor allem da diese auch in Szenarien ohne einen Angreifer bessere Klassifikationen liefern dürften.

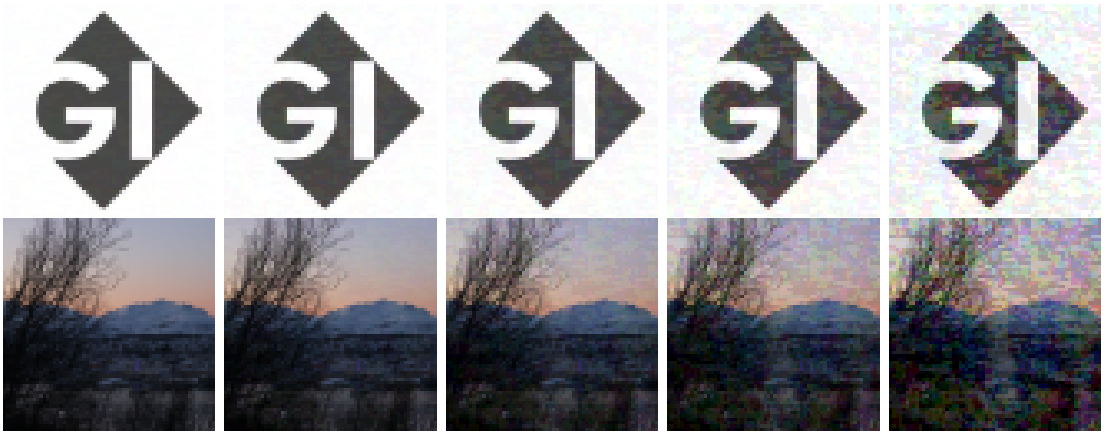
Der nächste Angriff basiert auf der „fast gradient sign method“ oder FGSM[4] beziehungsweise der Iterierten Version davon aus Kurakin et al. [7]. Hier verabschieden wir uns von dem Ansatz das anzugreifende Netz approximativ als global linear zu betrachten und schwächen die Annahme zu der von approximativer lokaler Linearität ab. Die lokal beste lineare Approximation für eine Funktion an einer Stelle ist durch die Ableitung (hier Gradient bzw. Jacobimatrix) gegeben. Um nun die Konfidenz, die das neuronale Netz einem Bild bezüglich einer Klasse zuordnet zu erhöhen, bewegen wir das Bild in die "Richtung", in der die lineare Approximation am schnellsten ansteigt, also in Richtung des Gradienten. Ausgehend vom Bild I können wir ein Netz N, das Bildern eine Konfidenz für eine Klasse zuordnet also durch die Störung $\widetilde{I} = I + \alpha * \nabla N(I)$ dazu bringen, eine höhere Konfidenz auszugeben. Da Bilder meistens als ganze Zahlen kodiert werden, wenden wir vor dem Schritt den Vorzeichenoperator auf den Gradienten an, um wieder ein Bild zu erhalten, also: $\widetilde{I} = I + \alpha * \text{sign}(\nabla N(I))$.
Um die Genauigkeit des Verfahrens zu verbessern benutzen wir $\alpha = 1$ und iterieren ausgehend vom Startbild I_0: $I_{k+1} = I_k + \text{sign}(\nabla N(I_k))$: Anstatt einen großen Schritt in die Anfangs richtige Richtung zu gehen, machen wir also viele kleine und berechnen die neue Richtung nach jedem Schritt. Dies kann nun für eine fixe Zahl an Schritten, bis sich die Konfidenz nicht mehr erhöht, oder bis eine Wunschkonfidenz erreicht ist wiederholt werden. Je nach Ausgangsbild kann es aber bei den letzten zwei Ansätzen dazu kommen, dass das Ursprungsbild bis zur Unkenntlichkeit verformt wird und beim ersten je nach Schrittzahl zu dem selben Problem, oder aber dazu, dass sich die Konfidenz nur minimal verändert, gerade bei sehr nichtlinearen Netzen, wo es passieren kann, dass viele Pixel weitestgehend mit den Iterationen oszillieren. Um die Vorteile beider Ansätze zu kombinieren, benutzen wir eine hohe Maximalzahl von Schritten, um die visuelle Ähnlichkeit zu bewahren Projizieren wir das Bild jedoch nach jedem Schritt auf eine Umgebung des

Ursprungsbildes: I_0: $I_{k+1} = P_\theta(I_k + \text{sign}(\nabla N(I_k)))$. In unserem Fall ist P_θ die Projektion auf die $\theta - l_\infty$ Kugel um das Ursprungsbild $B_\theta = \{x : \max(x_i - I_0) < \theta\}$, welche hier schlichtweg durch das "Stutzen" von zu großen Pixelstörungen gegeben ist. Neben der Projektionen auf Kugeln anderer Normen, wie der euklidischen Distanz sind auch komplett andere Ansätze denkbar. Zum Beispiel könnte der Angriff auf einen bestimmten Teil eines Bildes oder jeden 2. Pixel beschränkt werden.

Da moderne neuronale Netze quasi ausschließlich über gradientenbasierte Verfahren trainiert werden, sind auf neuronale Netze ausgelegte Machine Learning Bibliotheken wie Tensorflow oder das von uns verwendete Torch darauf optimiert, Gradienten analytisch und schnell und zu berechnen. Dementsprechend lässt sich die beschriebene Attacke schnell und einfach implementieren, wenn man denn Zugriff auf die Parameter des Netzes hat. In unserem Fall ist das anzugreifende Netz jedoch wie bereits erwähnt eine Black Box und wir können zwar Anfragen zur Klassifikation an es senden, können jedoch keine Gradienten abfragen und haben keinerlei Einblick in die verwendete Architektur und die gelernten Parameter. Die numerische Approximation der Gradienten ist unrealistisch, da wir hier für jede Iteration $64 * 64 * 3$, also mehr als 10000 Anfragen an das Netz senden müssten. Stattdessen trainieren wir unser eigenes Netz und hoffen, dass aus den Anfangs beschriebenen Gründen Angriffe auf diese WB auch auf der BB erfolgreich sind. Durch das Trainieren auf die von der BB ausgegebenen Klassen anstatt der tatsächlichen, werden auch Eigenarten der BB erfasst. Weiterhin trainieren wir nach der Generation eines Bildes die WB dann kurz darauf, die Regression für das Ausgabebild besser durchzuführen. Anschließend versuchen wir die verbesserte WB ausgehend vom letzten Ausgabebild erneut zu täuschen und wiederholen den Gesamtvorgang bis sich die Ausgaben von WB und BB ausreichend ähnlich sind.

Zur Verteidigung gegen solche gradientenbasierte Angriffe auf eine Black Box gibt es im wesentliche zwei Ansätze: Einerseits kann versucht werden es dem Angreifer zu erschweren die Black Box ausreichend gut zu rekonstruieren und andererseits kann die Black Box selbst robuster gegen die Angriffe gestaltet werden. Mögliche Ansätze der ersten Art sind das Runden der ausgegebenen Konfidenzen, beziehungsweise das ausschließliche Bereitstellen der Klasse mit der höchsten Wahrscheinlichkeit [8], das Geheimhalten des Trainingsdatensatzes, sowie die Ausgabe falscher oder widersprüchlicher Information. Beispielsweise könnten gezielt falsche Werte für mit hoher Wahrscheinlichkeit nicht Entscheidungsrelevante Informationen, wie die genaue Konfidenz für die Klasse mit der fünfthöchsten Konfidenz ausgegeben werden. Je nach Verwendungszweck entstünde mit

diesem Ansatz eine Abwägung zwischen der Qualität des Modells im Vakuum und der Verteidigung gegen potentielle Angreifer. Ohne gezielte Täuschung kommt der folgende Ansatz aus: Anstatt eines einzelnen Modells wird ein Ensemble trainiert und nur die Vorhersage eines der Modelle, deren Vorhersage nah an der durchschnittlichen Vorhersage des Ensembles liegt wird ausgegeben. Dadurch sind die Vorhersagen zwar fast genau so korrekt wie die des Ensembles, es entsteht jedoch Varianz, die die Rekonstruktion des Modells deutlich erschweren dürfte.
Für den zweiten Ansatz ist zunächst eine größere Anzahl von Neuronen und Schichten nötig, damit das Netz überhaupt in der Lage ist, für die Resistenz gegen den Angriff ausreichend nichtlineare Funktionen zu repräsentieren. Dann muss das Modell noch dazu gebracht werden, zu eine solcher Repräsentation anstatt einer oftmals einfacher zu lernenden, lineareren Repräsentation zu konvergieren. Dies kann durch so genanntes „Adversarial Training“ [4] erreicht werden: Anstatt eine Kostenfunktion $J_\psi(I, y)$ (Wie zum Beispiel die Kreuzentropie zwischen den echten Klassen y und den Vorhersagen des Netzes mit Parametern ψ) direkt zu minimieren, wird für ein $\beta \in [0, 1]$ die Funktion $\beta J_\psi(I, y) + (1 - \beta)J_\psi(I + \alpha * \operatorname{sign} \nabla_I J_\psi(I, y), y)$ minimiert. Dadurch wird das Netz direkt während des Trainings „bestraft“, wenn sich die eigentliche Kostenfunktion durch FGSM-Attacken zu leicht erhöhen lässt und lernt dementsprechend eine gegen diesen Angriff robustere Repräsentation. Der Parameter β charakterisiert hierbei den Grad der Abwägung zwischen der Minimierung der Kostenfunktion auf dem Trainingsdatensatz und der Robustheit gegen Angriffe. Durch das Abschwächen der direkten Minimierung der Kostenfunktion eignet sich Adverarial Trainig zudem zur Regularisierung, also als Methode um die Überanpassung zu mächtiger Modelle zu verringern. Alternativ lässt sich die Black Box natürlich auch einfach auf Bildern, die generiert wurden um ein anderes Netzwerk zu täuschen, trainieren, solange die echten Klassen zu Verfügung stehen.



Originalbild, Rauschen mit Schwellenwert 2,5,10,20. Beim Schwellenwert 2 ist quasi kein Unterschied erkennbar. Bei 5 erkennt man leichtes Rauschen. 10 rauscht stark, das Rauschen sieht aber noch natürlich aus und könnte bei Fotos auch als ISO-Rauschen wahrgenommen werden. Das Rauschen beim Schwellenwert 20 sieht bereits sehr unnatürlich aus.

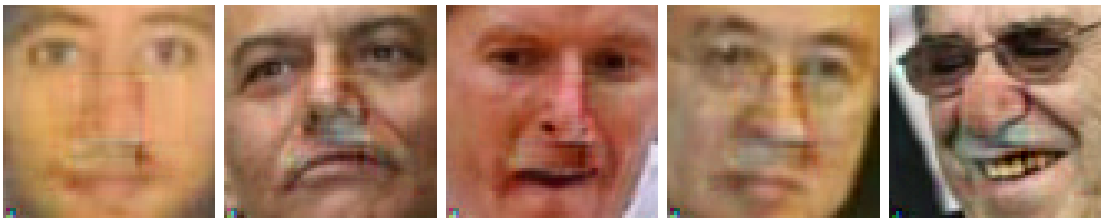
Konfidenz	keine Störung	Schwelle (θ) 2	Schwelle 5	Schwelle 10	Schwelle 20
	0.58316	0.687491	0.80196	0.88016	0.93279

Durschnittliche Konfidenz für FGSM-Angriff auf 25 Testbilder und die Klasse mit der höchsten Konfidenz auf dem Originalbild für l_∞-Schwellenwerte von 2,5,10,20. Maximale Anzahl an FGSM-Iterationen vor Korrektur der White Box: 25, Maximale Korrekturen: 10 Abbruch bei Black Box-Konfidenz über 99%. Höhere Schwellen führen deutlich erkennbar zu mehr Konfidenz. Die hohen Werte ohne Störung sind etwas verwunderlich, obwohl der Testdatensatz zwei Verkehrsschildern sehr ähnliche Verbotsschilder enthält.

Der dritte Ansatz nutzt ein Generative Adversarial Network (GAN) [9]. GANs bestehen in der Grundidee aus zwei neuronalen Netzen, dem Generator (G) und dem Diskriminator (D) . Dabei soll G letztlich Bilder erzeugen, die möglichst realistisch sind. Um dies zu erreichen, arbeiten G und D gegeneinander: D soll lernen, echte von G-erzeugten Bildern zu unterscheiden, während G lernt, möglichst realistische Bilder zu erzeugen. G kann parallel auch andere Ziele verfolgen bzw. Fehlerfunktionen minimieren. Für den Zweck des Erzeugens von Adversarial Examples verwenden wir ein GAN, bei dem G zusätzlich lernt, einen Klassifizierer C dazu zu bringen, für ein bestimmtes Label hohe Konfidenzen auszugeben. Da realistische Bilder mit hohen Konfidenzen seitens C und nicht die Generation völlig neuer Bilder unser eigentliches Ziel sind, trainieren wir G nur darauf, gegebene Bilder zu stören. Wir vermuten allerdings, dass mit mehr Rechenressourcen auch die freie Erzeugung von Bildern möglich sein sollte. Orientierend an Xiao et al.[10] minimiert G den Fehler $L_G = \epsilon * J(t, C(\hat{I})) + \alpha * D(\hat{I}) + \beta * ||I - \hat{I}||_2)$, wobei t das Label, das C ausgeben soll, J eine Fehlerfunktion, I das Ausgangsbild und $\hat{I}$ das von G

erzeugte Bild ist. α, β, ϵ sind Hyperparameter. Für J stellte sich der Carlini-Wagner-Loss [11] $\max\{\max\{C(\hat{I}_{i\neq t})\} - C(\hat{I}_t), -\kappa)\}$ mit $\kappa = 0.75$ als effektiv heraus. Der Generator soll also versuchen, t als das Label mit maximaler Konfidenz zu etablieren und den Abstand zu den anderen Labels bis zum Schwellenwert κ zu bringen. Ferner nutzen wir $\epsilon = 0.00005, \alpha = 1$ und $\beta = 1$ Als Diskriminator-Fehler verwenden wir den quadratische Fehler zwischen der Ausgabe (ein Wert zwischen 0 und 1) und dem richtigen Label (1 für „Kommt vom Generator“ und 0 für „Ist ein authentisches Bild“), der sich für das Training als deutlich stabiler als die Kreuzentropie herausstellte. Mit dem Ziel, beliebige Bilder von Gesichtern durch für Menschen nicht wahrnehmbare Veränderungen von neuronalen Netzen als Verkehrszeichen erkannt werden zu lassen, trainieren wir das GAN auf dem Faces in the Wild Datensatz[3] darauf, unsere White Box zur Klassifikation der Gesichter als Baustellen-Warnschild zu manipulieren. Die Architektur ist für D ein ResNet mit halb so vielen Schichten wie das ResNet18 und LeakyRelu-Aktivierungen. G hat auch Residual Blocks und orientiert sich an Johnson et al[12]. Anders als Johnson et al. verwenden wir aber auch hier LeakyRelu, damit G sich nicht zu stark an einen suboptimalen D gewöhnt. Wir verwenden in G außerdem anstatt Transposed Convolutions eine Kombination aus Upsampling und Convolutions, inspiriert durch [13]. Der letzte Unterschied zu Xiao et al.[10] ist, dass wir die Störung nach der letzten Schicht von G in den Bereich $[-0.3, 0.3]$ quetschen, um in Anbetracht beschränkter Ressourcen den Trainingsprozess zu beschleunigen.

Sobald man über ein trainiertes GAN verfügt, lassen sich Adversarial Examples extrem schnell erzeugen (Im Gegensatz zum iterierten FGSM Verfahren ist nur ein Forward Pass durch ein neuronales Netz nötig), was dabei hilftdas eigene Netz durch Versionen von Adversarial Training effizient robuster gestalten zu können. Der große Nachteil ist, dass das GAN extra trainiert werden muss. Dieser Nachteil wiegt schwer, da das Training von GANs sich aufgrund der nötigen Balance zwischen den beiden Netzwerken als deutlich schwieriger als das Training von herkömmlichen neuronalen Netzen erweist. Während Xiao et al. [10] es schaffen, hochauflösende Bilder mit unsichtbaren Störungen zu produzieren, waren wir nicht in der Lage, diesen Grad an Fehlerfreiheit zu erreichen. Wir vermuten, dass dies einerseits an den Hyperparametern liegen könnte, die wir mit unseren Ressourcen nicht besonders stark optimieren konnten, oder an der niedrigen 64*64-Auflösung, die G weniger Pixel zur Verfügung stellt, die es stören könnte und das Netzwerk damit zwingt, größere (und damit besser sichtbare) Störungen pro Pixel zu erzeugen.



Durch das GAN generierte Bilder aus dem Faces in the Wild Datensatz. Konfidenzen der Black Box für das Label Baustelle: 0.77, 1, 0.86, 0.96, 0.98: Auf unserer White Box erreichten die generierten Bilder auf dem aus 20% des Datensatzes bestehenden Testset eine durchschnittliche Konfidenz von 0.89. Ohne die Störung lag der Durschnitt bei 4% Auf dem Cifar-10 Datensatz, der keine Gesichter enthält konnte das GAN die Konfidenz für die Klasse Baustelle immerhin von durchschnittlich 4% auf 18% erhöhen.

Wir nutzen für die Software ausschließlich Python und darauf aufbauende Bibliotheken. Dies liegt vor allem daran, dass die üblichen Deep Learning Bibliotheken auf Python basieren. Wir entschieden uns hierbei für PyTorch. Die Benutzung der Attacken erfolgt über ein einfaches Graphical User Interface, bei dem verschiedene Parameter, die Angriffsmethode und das zu benutzende Bild eingestellt werden kann. Aufgerufen wird es nach der Installation von Python und der entsprechenden Bibliotheken auf dem Ubuntu 18.04 LTS AMD64 Referenzsystem mit der Eingabe "python main.py" in der Kommandozeile.

Bezüglich des Stils, in dem unser Programm geschrieben ist, haben wir uns nach PEP 8 und den Clean-Code-Richtlinien gerichtet. Letztere besagen vor allem, dass Funktionen- und Variablennamen so benannt sein sollen, dass Kommentare nicht nötig werden. Ausnahmen, wie z.B. das großgeschriebene X für eine Datenmatrix, entstehen aufgrund gängiger Konventionen im Machine Learning. Um die Software zu strukturieren, wurden folgende Maßnahmen getroffen: Erstens, befinden sich alle Hyperparameter und anwenderabhängige Variablen (wie z.B. Pfade) in einer zentralen Konfigurationsdatei. Möchte man die Software für einen anderen Datensatz oder eine andere Black Box verwenden, können die ensprechenden Einstellungen daher in *config.py* geändert werden. Die drei verschiedenen Ansätze sind getrennt in drei verschiedenen Skripten (*fgsm.py*, *sticker.py* und *gan.py*) zu finden. Hilfsfunktionen, die von mehr als einem Ansatz genutzt werden, sind in *utilities.py* definiert. Die WB wird auch von allen Ansätzen benutzt, wird aber in *whitebox.py* gekapselt, um die Software leichter mit einem anderen Klassifizierer anpassen zu können. Bezüglich Software Testing haben wir uns auf typische Fehlerquellen in Machine Learning konzentriert. Diese sind meistens still, lösen also keine Fehlermeldung

aus, sondern sind nur indirekt in beispielsweise eigenartigen Bildern oder Vorhersagen zu beobachten. Aus diesem Grund nutzen wir vor den ressourcenintensiven Trainings Sanity Tests. Diese sehen z.B. so aus, dass die WB auf den GTSRB-Datensatz getestet wird und den Loss zurückgibt oder dass die Zahl der Datenpunkte ausgegeben wird. Durch die Sanity Checks wurden wir auf einige Probleme beim Einladen der Daten aufmerksam und konnten diese dementsprechend lösen, bevor der Schaden zu groß wurde.

Literatur

[1] Stallkamp, J., Schlipsing, M., Salmen, J., Igel, C.: *The German Traffic Sign Recognition Benchmark: A multi-class classification competition*, IJCNN 2011 https://www.researchgate.net/publication/224260296

[2] He, K., Zhang, X., Ren, S., Sun, R.: *Deep Residual Learning for Image Recognition*, 2015 https://arxiv.org/abs/a1512.03385

[3] Huang, G., Ramesh, M., Berg, T., Learned-Miller, E.: *Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments*, University of Massachusetts, Amherst, Technical Report 07-49 2007. http://vis-www.cs.umass.edu/lfw/lfw.pdf

[4] Goodfellow, I., Shlens, J., Szegedy, C.: *Explaining and Harnessing Adversarial Examples*, ICLR 2015 https://arxiv.org/abs/1412.6572

[5] Glorot, X., Bordes, A., Bengio, Y.: *Deep Sparse Rectifier Neural Networks*, AISTATS 2011 http://proceedings.mlr.press/v15/glorot11a/glorot11a.pdf

[6] Glorot,X.,Bengio, Y.: *Understanding the difficulty of training deep feedforward neural networks*, AISTATS 2010
http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf

[7] Kurakin, A., Goodfellow, I., Bengio, S.: *Adversarial Machine Learning at Scale*, ICLR 2017 https://arxiv.org/pdf/1611.01236.pdf

[8] Tramèr, F., Zhang, F., Juels, A., Reiter, M., Riestenpart,T.: *Stealing Machine Learning Models via Prediction APIs*, Usenix Security 2016 https://arxiv.org/abs/1609.02943

[9] Goodfellow, I., Pouget-Abadie J., Mirza M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio,Y. : *Generative Adversarial Networks*, NeurIPS 2014

[10] Xiao, X., Li, B., Zhu, J., He, W., Liu, M., Song, D.: *Generating Adversarial Examples with Adversarial Network*, IJCAI 2018 https://www.ijcai.org/proceedings/2018/0543.pdf

[11] Carlini, N., Wagner,D.: *Towards Evaluating the Robustness of Neural Networks*, 2016 https://arxiv.org/abs/1608.04644

[12] Johnson, J., Alahi,A., Fei-Fei, L.: *Perceptual losses for real-time style transfer and super-resolution*, European Conference on Computer Vision 2016 https://cs.stanford.edu/people/jcjohns/eccv16/

[13] Oden, A., Dumoulin, V., Olah, C.: *Deconvolution and Checkerboard Artifacts* Distill 2016 http://distill.pub/2016/deconv-checkerboard