Verteidigungsstrategien gegen Poisoning-Angriffe auf KI-Systeme

Praktikumsbericht

2. Januar - 27. März 2020

Lukas Schulth

27. März 2020

Inhaltsverzeichnis

1 Einführung

In diesem Bericht sind die Ergebnisse meines Praktikums beim Bundesamt für Sicherheit in der Informationstechnik zusammengefasst. Es werden verschiedene Poisoning-Angriffe auf Neuronale Netzwerke und entsprechende Verteidigungsstrategien untersucht.

Bei sogenannten Poisoning-Angriffen fügt der Angreifer im Datensatz einen Anteil von korrumpierten Daten ein, um das Verhalten des Netzes auf nicht korrumpierten Daten negativ zu beeinflussen. Diese Kategorie von Angriffen findet während des Trainingsprozesses eines Neuronalen Netzes statt. Die erste Art von Poisoning-Angriffen verfolgt das Ziel die Genauigkeit des Netzes negativ zu beeinflussen, d.h. die Genauigkeit auf dem Testdatensatz soll verringert werden. Die zweite Art von Poisoning-Angriffen zielt darauf ab, während des Trainings eine Hintertür im Netz zu implementieren. Ist die Hintertür erfolgreich implementiert so kann diese während der Verwendung des Neuronalen Netzes ausgenutzt werden: Bildeingaben, die ein spezielles Muster(Auslöser) aufweisen, sollen ein vom Angreifer gewünschtes Verhalten zeigen. Dieses besteht darin, dass Eingaben mit diesem Auslöser absichtlich falsch klassifiziert werden. Gleichzeitig soll das Netz unter Abwesenheit eines Auslösers eine hohe Genauigkeit auf den Testdaten aufweisen. Wir betrachten hier Angriffe auf KI-Systeme zur Straßenverkehrsschilderkennung. Diese Hintertür-Angriffe wurden in [1] dahingehend verbessert, dass das Label eine Datenpunktes auch zum entsprechenden Bild passt.

Beitrag
Im Rahmen des Praktikums wurden die in [1] und [2] vorgestellten Angriffe genauer untersucht. Andere Netzwerke wurden angegriffen und die Grenzen der vorgestellten Verteidigungen empirisch getestet.

2 Material und Methoden

2.1 Datensatz

Für die Poisoning-Angriffe auf verschiedene neuronale Netzwerke benutzen wir den Datensatz *German Traffic Sign Recognition Benchmark* [1]. Dieser besteht aus 52.001 Bildern von Verkehrsschildern aus 43 verschiedenen Kategorien der Pixelgröße 32x32. Etwa 75 Prozent der Bilder wird für das Training, die anderen 25 Prozent für das Testen benutzt. Der Datensatz wurde ursprünglich in einem Wettbewerb auf der International Joint Conference on Neural Networks (IJCNN) im Jahr 2011 benutzt. Die Bilder sind aus aus einer Videosequenz herausgeschnitten. Deshalb befinden sich in einer Klasse jeweils immer mehrere Bilder desselben Verkehrsschildes zu unterschiedlichen Zeitpunkten. Aufnahmen desselben Verkehrsschildes kommen nicht übergreifend in Training-, Validierung- oder Testdatensatz vor.

Verkehrsschild	Anzahl an Bildern
'Zulässige Höchstgeschwindigkeit: 20 km/h'	180
'Zulässige Höchstgeschwindigkeit: 30 km/h'	1980
'Zulässige Höchstgeschwindigkeit: 50 km/h'	2010
'Zulässige Höchstgeschwindigkeit: 60 km/h'	1260
'Zulässige Höchstgeschwindigkeit: 70 km/h'	1770
'Zulässige Höchstgeschwindigkeit: 80 km/h'	1650
'Halt! Vorfahrt gewähren'	690

Tabelle 1: Verteilung der Geschwindigkeitsbegrenzungen und Stoppschilder im Trainingsdatensatz

In Tabelle Tabelle 1 sind einige Klassen der Verkehrsschilder und deren Anzahl im Datensatz aufgelistet, die für einen Poisoning-Angriff interessant sein könnten. Die Anzahl der Schilder 'Halt! Vorfahrt gewähren'-Schilder im Trainingssatz beträgt etwa 690 Aufnahmen. Diese wurden von insgesamt nur 24 verschiedenen 'Halt! Vorfahrt gewähren'-Schildern aufgenommen. Da beim Erstellen der korrumpierten Daten auch immer das Bild aus der angegriffenen Klasse in die Zielklasse verschoben wird, wird die Anzahl der in der Ursprungsklasse verbleibenden Daten abhängig vom Anteil an korrumpierten Daten kleiner. Wir werden uns deshalb im Folgenden mit Angriffen auf die Klasse 'Zulässige Höchstgeschwindigkeit: 50 km/h' beschäftigen, da sie die höchste Anzahl an Daten aufweist.

2.2 Neuronales Netzwerk

Wir wollen die Bilder im Datensatz in die verschiedenen Klassen an Verkehrsschildern einordnen. Wir betrachten also ein Klassifikationsproblem, bei dem es

[1] http://benchmark.ini.rub.de/?section=gtsrb&subsection=dataset

das Ziel ist, zu einer Eingabe $x \in \mathcal{X}$ das entsprechende Label $y \in \{1, ..., k\}$ zu finden, wobei k die Anzahl an verschiedenen Klassen darstellt. Unser Neuronales Netzwerk ist eine Abbildung $f_\theta : \mathcal{X} \to \{1, ..., k\}$, die durch $\theta \in \mathbf{R}^d$ parametrisiert ist. Die Parameter θ des Netzwerks werden durch Minimierung der Fehlerfunktion $\mathcal{L}(x, y, \theta)$ bestimmt, die ein Maß dafür ist, wie gut das aktuelle Netzwerk auf den Eingabe-Label-Paaren (x, y) einer Menge $\mathcal{D} = \{(x_i, y_i)_{i=1}\}^n$ funktioniert:

$$\theta^* = \arg\min_\theta = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(x_i, y_i, \theta).$$

Diese Minimierung wird meistens mit Hilfe eines stochastischen Gradientenverfahrens erreicht. Für die verwendeten Netzwerke benutzen wir den Adam-Optimierer, eine Form des stochastischen Gradientenabstiegverfahrens.

2.3 Design der Neuronalen Netzwerke

Wir benutzen drei verschiedene Neuronale Netzwerke und vergleichen Angriffe und Verteidigungen auf diesen.

Das erste Netz ist ein kleinerer Nachbau eines Inception-Netzes [4] mit 3 Faltungs- und 3 linearen Schichten. Das zweite Netz ist ebenfalls ein Nachbau eines Inception Netzes. Es besteht aus 10 Faltungs- und 3 linearen Schichten. Der Aufbau ist im Anhang dargestellt Im dritten Fall benutzen wir das vortrainierte *inception_v3* aus der torchvision-Modellbibliothek [2].
In Tabelle Tabelle 2 sind die drei Netze mit den Genauigkeiten auf unkorrumpierten Testdaten und der Anzahl an Parametern angegeben.

Netzwerk	Anzahl an Schichten	Anzahl an trainierbaren Parametern	Performance auf nicht korrumpierten Daten
(I) Net	12	363.227	94.43
(II) InceptionNet3	52	1.110.027	98.099
(III) $InceptionNet\_v3$	302	25.200.371	97.93

Tabelle 2: Überblick über die 3 verschiedenen Netze

[2] https://pytorch.org/docs/stable/torchvision/models.html

2.4 Einlesen der Daten

Beim Einlesen der Daten ins Netzwerk benutzen wir die folgenden Transformationen [3] auf den Trainingsdaten:

```
_train_transform = transforms.Compose([

transforms.RandomResizedCrop((im_size, im_size), scale=(0.6, 1.0)),
transforms.RandomRotation(degrees=15),
transforms.ColorJitter( brightness=0.1,
contrast=0.1, saturation=0.1, hue=0.1),
transforms.RandomAffine(15),
transforms.RandomGrayscale(),
transforms.ToTensor(),
])
```

Erklärung der einzelnen Transformationen: *RandomResizedCrop* wählt einen abhängig von der Eingabegröße des Bildes festgelegten Bildausschnitt und skaliert diesen wieder auf eine feste Bildgröße. *RandomRotation* rotiert das Bild um einen zufälligen Winkel zwischen -15 und 15 Grad. *ColorJitter* ändert zufällig die Helligkeit, den Kontrost sowie die Sättigung. *RandomAffine* führt eine affine Transformation mit maximal 15 Grad Rotation durch. *RandomGrayscale* erstellt aus der Eingabe mit einer Wahrscheinlichkeit von $p = 0.1$ ein Graustufenbild. *ToTensor* konvertiert das Bild in einen *torch.FloatTensor* der Form (C x H x W) im Intervall $[0.0, 1.0]$.

Mit Hilfe der angegebenen Transformation wird auf eine verbesserte Genauigkeit des Netzwerks abgezielt. Für das Einlesen der Validierungs- und Testdaten werden die Bilder lediglich in Tensoren umgewandelt, um an das Netzwerk weitergegeben werden zu können.

2.5 Training

Wir trainieren über 50 Epochen und benutzen das folgende Abbruchkriterium: Das Training wird dann abgebrochen, wenn entweder die 50 Epochen erreicht sind oder in den letzten 20 Epochen kein Rückgang des Fehlers auf den Validierungsdaten aufgetreten ist. Die batch size setzen wir auf 20, d.h., dass in einer einzelnen Trainingsiteration 20 Datenpunkte benutzt werden. Als Optimierungsverfahren benutzen wir den Adam(Adaptive Moment Estimation)-Algorithmus, eine Form des stochastischen Gradientenabstiegs. Die Lernrate setzen wir auf $lr = 10^{-3}$. Bei der Verwendung des vortrainierten Netzes trainieren wir für 10 Epochen.

3 Angriffe

3.1 Kenntnisse des Angreifers

Wir nehmen an, der Angreifer kennt das Netz, welches während des Trainings benutzt wird nicht. Ihm steht nur der Datensatz zur Verfügung, der gezielt

[3] https://pytorch.org/docs/stable/torchvision/transforms.html

verändert werden kann.

3.2 Ziel des Angreifers

Ziel des Angreifers ist es, während des Trainings eine Hintertür im Netzwerk zu implementieren, die dann bei der Verwendung des Netzwerkes im Realbetrieb ausgenutzt werden kann, um ein gewünschtes falsches Verhalten des Netzwerkes zu erreichen. Außerdem möchte der Angreifer möglichst wenig am Datensatz ändern, d.h. nur wenige Datenpunkte verändern oder neue Datenpunkte einfügen. Zudem soll das mit einer Hintertür versehene Netzwerk auf Eingabedaten, die keinen Auslöser besitzen trotzdem sehr gut funktionieren, sodass der Angriff im Normalbetrieb nicht sofort auffällt. Dies halten wir als **Genauigkeit auf unkorrumpierten Testdaten**(**GUD**) fest.

Der Erfolg eines Angriffs ist über die **Angriffserfolgsrate** (**AER**) definiert . Sie ist pro Klasse als das Verhältnis von Datenpunkten mit Auslöser, die vom Netzwerk gewollt falsch klassifiziert werden und Datenpunkten mit Auslöser, die trotz des vorhandenen Auslösers korrekt klassifiziert werden definiert.

3.3 Standard Poisoning-Angriffe

Bei dieser Art von Angriff [2] wird ein quadratischer Sticker mit einer Seitenlänge s zwischen 1 und 4 Pixeln in einem vorher abgesteckten Fenster auf dem Verkehrsschild der Ursprungsklasse eingefügt. Dieses korrumpierte Bild wird dann in die Zielklasse verschoben, d.h. das Label des Bildes wird abgeändert. Als Farbe des Stickers wurde ein gelb-grüner Farbton mit dem hexadezimalen Farbcode $\#f5ff00$ verwendet. Abhängig vom prozentualen Anteil der korrumpierten Bilder und der Größe des Auslösers verändert sich der Erfolg eines Angriffs und der Detektion. Für die zufällige Platzierung des Stickers auf dem Stoppschild wird ein festes Fenster $((x_{min}, x_{max}), (y_{min}, y_{max}))$ vorgegeben, in dem dann die linke obere Ecke des Stickers zufällig platziert wird.

Wir beschäftigen uns im Folgenden mit Angriffe auf die 50km/h-Klasse. Diese Schilder sollen in Anwesenheit eines Auslösers als 80km/h-Verkehrsschilder klassifiziert werden. Das Label dieses korrumpierten Bildes wird dann zusätzlich auf die falsche Klasse abgeändert. Ein Beispiel eines auf diese Weise korrumpierten Bildes ist in Abbildung 1 abgebildet.

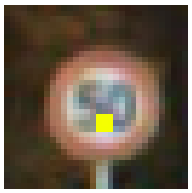

Abbildung 1: Korrumpiertes 50km/h-Verkehrsschild

3.4 Label-konsistente Poisoning-Angriffe

Im Unterschied zum Standardangriff, bei dem ein Mensch beim genauen Untersuchen des Datensatzes feststellen kann, dass ein Bild offensichtlich nicht das richtige Label aufweist, ist hier die Idee [1], das Bild so zu verändern, dass es für das menschliche Auge immer noch zur richtigen Klasse gehört, aber gleichzeitig vom neuronalen Netzwerk so schwierig zu klassifizieren ist, dass sich das Netz mehr auf den Auslöser verlässt. Die Datenpunkte werden zunächst so verändert, dass die Leistungsfähigkeit des Netzes beim Klassifizieren abnimmt. Anschließend wird auf dem entsprechenden Bild ein Auslöser angebracht. In [1] werden zwei Methoden vorgestellt, die dieses - aus Sicht des Angreifers - gewünschte Verhalten hervorrufen. Die erste Methode besteht darin, dass der Input x_1 der Zielklasse und der Input x_2 einer inkorrekten Klasse in einen niedrig-dimensionalen Raum, den Raum der latenten Variablen, eingebettet werden. Es wird über den latenten Raum optimiert, um Eingaben z_1,z_2 zu finden, die sich in der l_2-Norm minimal von den Eingaben x_1 und x_2 unterscheiden. Der schwer zu klassifizierende neue Datenpunkt wird nun durch Interpolation der beiden Punkte z_1 und z_2 erzeugt.
Beim zweiten Verfahren wird ein Bild mit einer Störung versehen, die die Leistungsfähigkeit des Netzes senken soll. Um die Genauigkeit des Netzes während des Trainings abzuschwächen, benutzen wir mit einer Störung versehene Datenpunkte im Trainingsdatensatz. Ein Datenpunkt wird mit einer Transformation gestört und anschließend ein Auslöser eingefügt. Die Störung des Netzwerkes sieht wie folgt aus: Sei f_θ unser Neuronales Netzwerk zusammen mit einer Fehlerfunktion $\mathcal{L}(x, y, \theta)$ und einem Eingabe-Label-Paar (x, y). Dann erzeugen wir eine Störung der Eingabe x als die Lösung des Optimierungsproblems:

$$x_{adv} = \underset{||x'-x||_p \leq \epsilon}{\arg\max}\ \mathcal{L}(x', y, \theta),$$

für eine Wahl einer l_p-Norm und einer Konstante $\epsilon > 0$. Dieses Optimierungsproblem lösen wir mit Hilfe eines Projizierten Gradientenverfahrens.

Im Anschluss daran fügen wir nun wieder einen unserer Auslöser ein: Dabei unterscheiden wir zwischen einem einfachen gelb-grünen Sticker (vgl. Unterabschnitt 3.3), einem 3x3 Pixel großen schwarz-weißen Sticker(vgl. [3]) bei verschiedenen Amplituden und einem solchen Amplitudensticker in allen 4 Ecken des Eingabebildes. Bei einem Amplitudensticker werden bei denjenigen Pixeln, die im ursprünglichen schwarz-weißen Auslöser schwarz sind, eine Amplitude $amp \in \{16, 32, 64, 255\}$ auf allen drei Farbkanälen addiert. Umgekehrt wird bei den entsprechenden weißen Pixelpositionen im schwarz-weißen Auslöser auf allen Farbkanälen amp subtrahiert. Für den Wert $amp = 255$ erhält man den ursprünglichen schwarz-weißen Sticker.

Eine Besonderheit dieses Angriffs ist, dass keine neuen Datenpunkte in den Datensatz eingefügt werden oder sich die Größe einzelner Klassen verändert wie beispielsweise bei dem oben beschriebenen Standardangriff.

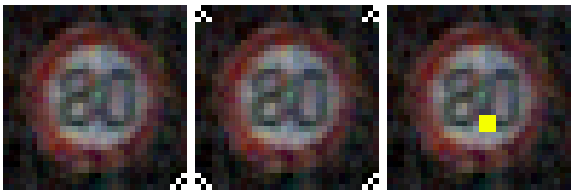

Abbildung 2: Angriff auf die Klasse 'Zulässige Höchstgeschwindigkeit: 80km/h' versehen mit drei verschiedenen Auslösern. Von links nach rechts: Muster in der rechten unteren Bildecke, Muster in jeder Bildecke, Sticker auf dem Verkehrsschild.

4 Verteidigungen

Bei den vorgestellten Verteidigungen interessiert uns die Qualität der Klassifikation unserer Verteidigungsstrategie in verdächtige und unverdächtige Datenpunkte. Um dies zu bewerten. Benutzen wir die folgende Terminologie.
Die Genauigkeit der Vorhersage ist definiert durch den Quotienten $\frac{TP+TN}{TP+FP+TN+FN}$, wobei TP(true positives) für die Anzahl an positiven Testergebnis bei positiver Grundwahrheit steht. Analog steht TN(true negatives) für die Anzahl an negativen Testergebnissen bei negativer Grundwahrheit. Stimmen Testergebnis und Grundwahrheit nicht überein, so beschreibt FP(false positives) die Anzahl an positiven Testergebnissen bei negativer Grundwahrheit und FN(false negatives) umgekehrt die Anzahl an negativen Testergebnissen bei positiver Grundwahrheit. Wir werden diese Genauigkeit im weiteren als Detektionsrate bezeichnen. Aus den obigen Definitionen lassen sich diese weiteren Größen ableiten: Die Falsch-Negativ-Rate $FNR = \frac{FN}{FN+TP}$ beschreibt das Verhältnis an getesteten Datenpunkten, die vom Test als negative deklariert werden, obwohl sie eine positive Grundwahrheit besitzen. In unserem Fall ist die Falsch-Negativ-Rate die wichtigste Größe. Ist sie minimal sein, so werden die meisten korrumpierten Datenpunkte aussortiert. Umgekehrt können wir die Falsch-Positiv-Rate $FPR = \frac{FP}{FP+TN}$ definieren, die anteilsmäßig beschreibt, wie viele Datenpunkte mit negativer Grundwahrheit fälschlicherweise als positiv erklärt werden. Die Richtig-Positiv-Rate $TPR = 1-FNR = \frac{TP}{TP+FN}$ gibt an, wie viele Datenpunkte mit positiver Grundwahrheit als positiv erkannt werden. Umgekehrt beschreibt die Richtig-Negativ-Rate $TNR = 1 - FPR$, wie viele der Datenpunkte mit negativer Grundwahrheit als negativ erkannt werden.

4.1 Clustering auf den Bilddaten

Eine erste Idee, korrumpierte Datenpunkte in einem Datensatz zu finden könnte sein, ein Clustering auf den rohen Bilddaten durchzuführen. Bevor wir die Bilder mit Hilfe des kMeans-Algrithmus clustern, führen wir eine Dimensionsreduktion durch.

4.2 Activation Clustering

Zur Verteidigung des Poisoning Angriffs benutzen wir die Idee des Activation Clusterings [2]. Diese beruht darauf, dass die Aktivierungen der letzten verdeckten Schicht die Entscheidungen des Netzwerkes sehr gut codieren. Deshalb kann man anhand dieser Aktivierungen mögliche Hintertüren im Datensatz aufdecken.
Ein Angriff ist erfolgreich, wenn eine große Anzahl an Datenpunkten der Ursprungsklasse, versehen mit einem Auslöser, der Zielklasse zugeordnet werden. Im Falle eines erfolgreichen Angriffs werden korrumpierte und nicht korrumpierte Datenpunkte im Trainingsdatensatz derselben Klasse zugeordnet. Der Grund weshalb diese derselben Klasse zugeordnet werden, unterscheidet sich jedoch. Beim Activation Clustering wird nun angenommen, dass pro Klasse entweder korrumpierte und nicht korrumpierte Datenpunkte oder nur nicht korrumpierte Datenpunkte existieren. Deshalb werden die Aktivierungen der letzten verdeckten Schicht des Netzwerkes aus dem Netz extrahiert, nach ihren zugehörigen Klassen der Labels segmentiert, auf 10 Dimensionen reduziert und anschließend mit Hilfe des kMeans-Algorithmus geclustert. Das kleinere Cluster wird immer als der Anteil an verdächtigen Datenpunkten betrachtet. Die Idee ist es, dass die korrumpierten Datenpunkte, sofern welche existieren, alle in die eine und die nicht korrumpierten Datenpunkte in das andere Cluster aufgeteilt werden. Sind keine korrumpierten Datenpunkte vorhanden, so sollen beide Cluster ungefähr dieselbe Anzahl an Datenpunkten erhalten.
Wir werten die Qualität des Clusterings anschließend aus. Als Detektionsrate beschreiben wir die Genauigkeit des Clusterings auf den Trainingsdaten. Zur Bestimmung, ob eine Klasse korrumpierte Daten enthält, kann das Ergebnis des Clusterings mit den folgenden Methoden untersucht werden:
Vergleich der relativen Größe: Eine Möglichkeit, korrumpierte Datenpunkte zu erkennen, ist der Vergleich der relativen Größen der beiden Cluster. Laut [2] ist die relative Größe bei nicht korrumpierten Klassen ca. 50 Prozent, bei korrumpierten Daten und einem erfolgreichen Clustering würde die relative Größe dann dem prozentualen Anteil an korrumpierten Datenpunkten entsprechen.
Silhouette-Koeffizient [4] Eine weitere Möglichkeit besteht darin, die Qualität des Clusterings mit Hilfe des Silhouette-Koeffizienten zu beschreiben. Dieser gibt an, wie gut ein Clustering zu den gegebenen Datenpunkten mit den entsprechenden Labeln passt und ist wie folgt definiert: Sei das Ergebnis eines Clustering-Algorithmus mit verschiedenen Clustern gegeben. Zu einer Beobachtung im Cluster A, $x \in A$ wir die Silhouette $s(x) = \frac{dist(B,o)-dist(A,o)}{max\{dist(A,o),dist(B,o)\}}$ festgelegt. Dabei beschreibt $dist(B, o)$ die Distanz zum nächstgelegenen Cluster B. $dist(A, o)$ wird berechnet als $dist(A, o) = \frac{1}{n_A-1} \sum_{a \in A, a \neq o} dist(a, o)$, d.h. als der Mittelwert der Distanz zwischen allen anderen Beobachtungen im Cluster A und der Beobachtung o. Dabei steht n_A für die Anzahl der Beobachtungen im Cluster A. Das nächstgelegene Cluster B ist definiert als: $dist(B, o) = \min_{C \neq A} dist(C, o)$.
Exklusives Retraining: Beim exklusiven Retraining wird das neuronale Netz

[4] `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.silhouette_score.html`

von Grund auf neu trainiert. Das oder die verdächtigen Cluster werden beim erneuten Training nicht benutzt. Mit Hilfe des neu trainierten Netzes werden dann anschließend die vorenthaltenen, verdächtigen Cluster klassifiziert. Falls das Cluster Aktivierungen von Datenpunkten enthält, die zum Label des Datenpunktes gehören, erwarten wir, dass die Vorhersage des Netzwerks mit dem Label übereinstimmen. Gehören die Aktivierungen eines Datenpunktes im verdächtigen Cluster jedoch zu einer anderen Klasse als die durch das Label angedeutete Klasse, so sollte das Netzwerk den Datenpunkt einer anderen Klasse zuordnen. Um nun zu entscheiden, ob ein verdächtiges Cluster korrumpiert oder nicht korrumpiert ist, wird wie folgt vorgegangen. Sei l die Anzahl an Vorhersagen, die zum Label des Datenpunktes passen. Sei p die größte Anzahl an Vorhersagen, die für eine weitere Klasse C sprechen, wobei C nicht die Klasse mit den Labeln des zu untersuchenden Clusters ist. Der Quotient $\frac{l}{p}$ gibt dann an, ob das Cluster korrumpiert ist oder nicht: Es wird ein Schwellenwert $T > 0$ gesetzt. Gilt $\frac{l}{p} < T$, wurde mehr Datenpunkte einer anderen Klasse zugeordnet und das Cluster wird als korrumpiert deklariert. Umgekehrt wird das verdächtige Cluster im Fall von $\frac{l}{p} > T$ als nicht korrumpiert/sauber eingestuft.

5 Ergebnisse

In diesem Abschnitt fassen wir die Ergebnisse der verschiedenen Angriffe und Verteidigungen kurz zusammen.

5.1 Angriffe

Wir werten zunächst die Angriffe auf Net, InceptionNet3 und $Inception\_v3(pretrained)$ aus. Dabei versehen wir die Klasse mit den 50km/h-Schildern einen Sticker ein und ändern das Label auf 80km/h ab.

5.1.1 Net

Trainieren wir das Netz auf nicht korrumpierten Trainingsdaten erhalten wir eine Genauigkeit von 94.172 Prozent auf den unkorrumpierten Trainingsdaten. Wir trainieren das Netzwerk Net auf den korrumpierten Trainingsdaten bei unterschiedlichen Stickergrößen und prozentualen Anteilen an korrumpierten Daten und erhalten in Abhängigkeit davon diese Angriffe:

5.1.2 InceptionNet3

Wir führen einen Angriff mit 15 Prozent korrumpierten Daten durch, wobei die Stickergröße 3x3 Pixel beträgt. Die Angriffsrate beträgt 100 Prozent, bei einer Performance von 97.73 Prozent Genauigkeit des korrumpierten Netzes auf den unkorrumpierten Trainingsdaten.

Seitenlänge des Stickers	Prozentualer Anteil an korrumpierten Datenpunkten	AER	Genauigkeit auf korrumpierten Testdaten	GUD
2	0.1	0.48	0.935	0.930
2	0.2	0.84	0.935	0.925
2	0.15	0.79	0.924	0.916
2	0.33	0.96	0.916	0.905
3	0.1	0.78	0.912	0.904
3	0.2	0.98	0.941	0.920
3	0.15	0.953	0.926	0.914
3	0.33	1.0	0.935	0.924
4	0.1	0.907	0.938	0.928
4	0.2	0.973	0.900	0.889
4	0.15	1.0	0.936	0.925
4	0.33	1.0	0.937	0.926

Tabelle 3: Qualität des Angriffs auf Net in Abhängigkeit von der Stickergröße und des Anteils an korrumpierten Datenpunkten

5.1.3 $Inception\_v3(pretrained = True)$

Wir greifen das vortrainierte Netz während des Trainings über 10 Epochen ebenfalls mit 15 Prozent korrumpierten Daten, versehen mit Stickern der Größe 3x3 Pixel an. Dabei erhalten wir eine Testgenauigkeit von 97.73 Prozent des korrumpierten Netzes auf den unkorrumpierten Daten, während die Performance eines unkorrumpierten Netzes auf unkorrumpierten Daten bei knapp über 98 Prozent liegt.

Es ist auffällig, dass die Angriffe mit einer Stickergröße der Seitenlänge 3 bei den größeren Netzwerken InceptionNet3 und dem vortrainierten $Inception\_v3$ auch bei einem kleineren Anteil an korrumpierten Daten deutlich besser funktionieren als bei dem kleineren Netzwerk Net. Sowohl die Angriffe auf InceptionNet3 als auch auf $Inception\_v3$ funktionieren unter den Voraussetzungen von 15 Prozent korrumpierten Daten und einem 3x3 Pixel großen Auslöser mit einer Angriffserfolgsrate von 100 Prozent.

5.2 Verteidigungen

Für die im vorherigen Abschnitt vorgestellten Angriffe betrachten wir verschiedene Verteidigungsstrategien, um die Angriffe aufzudecken.

5.2.1 Referenzwert: kMeans(k=2)

Wir setzen das Poison-Label eines korrumpierten Bildes auf 1 und sehen dies als positiven Fall in der Detektion. Umgekehrt ist das Poison-Label eines nicht korrumpierten Bildes 0 und wird als negativer Fall betrachtet. Als Referenz benutzen wir einen Clusteralgorithmus auf den Bilddaten selbst. Wir betrachten

den Fall, dass so viele korrumpierte Bildern aus der Ursprungsklasse in die Zielklasse verschoben werden, dass 15% der Bilder in der veränderten $target_class$ korrumpiert sind. Wir führen zunächst eine Komponentenanalyse zur Dimensionsreduktion auf den Bilddaten durch und benutzen anschließend den kMeans-Algorithmus zur Clusteranalyse für 2 Klassen.
Zur Reduktion benutzen wir einerseits die FastICA[5], eine Version der Unabhängigkeitsanalyse und andererseits eine klassische Hauptkomponentenanalyse[6] (PCA). Wir reduzieren die Bilddaten auf 10 verbleibende Dimensionen. Damit ergeben sich die Werte in Tabelle Tabelle 4 auf dem Trainingsdatensatz. Betrachtet man nur die Genauigkeit, ist der Schluss zulässig, dass das Clustering im Anschluss an eine PCA leicht besser funktioniert. Betrachtet man jedoch auch die anderen Zeilen, wird deutlich, dass die FNR und damit die Anzahl an korrumpierten Datenpunkten, die im Datensatz verbleiben, mit 17.87% deutlich geringer gegenüber 51.54% ist. Bei beiden Herangehensweisen liegt die Anzahl an unveränderten Datenpunkten, die fälschlicherweise entfernt werden, über 70%. Hierbei beschreibt die zweite Zeile in Tabelle **??** die Genauigkeit der

	FastICA	PCA
Genauigkeit	66.15	67.49
FNR	17.87	51.54
TPR	82.13	48.45
TNR	25.33	29.70
FPR	74.67	70.30

Tabelle 4: Ergebnisse des Clusterings auf den Rohdaten in Prozent bei FastICA und PCA zur Dimensionsreduktion

Vorhersage.

Im Folgenden werten wir die Verteidigung auf den drei verschiedenen Netzwerken mit Hilfe des Activation Clusterings aus.

5.2.2 Net

Nach 20 Trainingsepochen erhalten wir bei 15 Prozent korrumpierten Daten eine Angriffserfolgsrate von 89.2 Prozent.
Die Genauigkeit auf dem sauberen Testdatensatz beträgt 89.2 %. Der Trainingsdatensatz wird nach dem Retraining nicht als korrumpiert erkannt. Es ergibt sich folgende Verteilung der Vorhersagen auf den verdächtigen Daten:

[0 2 59 25 3 669 0 20 8 0 0 0 1 2 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 1 0 0].
Wir sehen, dass 669 Datenpunkte unseres verdächtigen Clusters als Klasse 5 klassifiziert werden. 59 Datenpunkte werden der Klasse 3 zugeordnet. Mit 11.34

[5]https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.FastICA.html
[6]https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

liegt der Trainingsdatensatz deutlich über dem Schwellenwert $T = 1$ und wird nicht als korrumpiert erkannt.

Auch für 33, 10 und 7.5 Prozent an korrumpierten Daten wird der Angriff nicht erkannt.

5.2.3 InceptionNet3

Für die Verteidigung der Angriffe auf InceptionNet3 erhalten wir auf den Trainingsdaten die in Tabelle Tabelle 5 dargestellten Ergebnisse und Entscheidungen, ob der Datensatz korrumpiert ist.

Anteil korrumpierter Daten in Prozent	AER	GUD	Detektionsrate Training	$\frac{l}{p}$	GUD nach Retraining
7.5	100	97.91	99.49	5.65	97.69
10.0	100	97.56	99.89	0.13	98.36
15.0	100	97.66	100.00	0.0	97.75
33.0	100	97.87	100.00	0.0	97.34

Tabelle 5: Ergebnis des Activation Clusterings auf InceptionNet3 für verschiedene prozentuale Anteile an korrumpierten Daten. Mögliche Stickerposition: ((16,17),(16,19)).

Es ist auffällig, dass trotz einer hundertprozentigen Angriffserfolgsrate der Angriff mit 7.5 Prozent korrumpierten Datenpunkten nicht mehr erkannt wird. Wir können zudem für jede Klasse gleichzeitig mit Hilfe eines exklusiven Retrainings überprüfen, ob die verdächtigen Cluster einer Klasse als korrumpiert eingestuft werden. Wir verwenden wieder 15 Prozent an korrumpierten Daten. Das Exklusive Retraining liefert die richtige Aussage, dass alle Klassen außer Klasse 5:80 km/h nicht korrumpiert sind.

Untersuchen wir nicht nur eine Klasse auf einen durchgeführten Poisoning-Angriff, sondern alle Klassen gleichzeitig mit Hilfe des exklusiven Retrainings, so wird nur die tatsächlich korrumpierte Klasse als korrumpiert deklariert. Die Angriffserfolgsrate nach dem Retraining fällt auf 0.13 Prozent ab. Die Genauigkeit nach dem Retraining auf unkorrumpierten Daten liegt bei 96.46 Prozent. Beobachtung: Training Accuracy wird nicht wirklich besser: Backdoor ist aber entfernt und alle anderen nicht korrumpierten Klassen werden auch nicht als korrumpiert deklariert.

Im Fall von 10 Prozent korrumpierten Daten bleibt im Datensatz ein Bild mit Sticker vorhanden. Dadurch entsteht der Angriff mit 14.67 Prozent Erfolgsrate. Wieso wird bei 15 Prozent trotzdem ein Bild mit Sticker falsch erkannt wird. Das ist vermutlich einfach ein Bild, das generell schwer zu klassifizieren ist.

Wir sehen hier anhand des Beispiels des InceptionNet3 mit einem Sticker der Größe $s = 3$ und 15 Prozent korrumpierten Daten, dass der Vergleich der relativen Größe kein guter Hinweis dafür ist, ob eine Klasse korrumpierte Datenpunkte beinhaltet oder nicht. Bei einer Angriffserfolgsrate von 100 Prozent

Anteil korrumpierter Daten in Prozent	AER	GUD	Detektionsrate Training	$\frac{l}{p}$	AER nach Retraining
0.15	56.8	97.28	50.30	∞	62.93
0.2	94.8	97.10	52.69	388.5	69.46
0.25	66.53	98.05	53.20	∞	49.6
0.4	74.93	97.81	65.91	8.41	15.6
0.75	99.33	97.87	74.84	∞	100
7.5	100	97.5	99.83	0.72	31.73
10.0	100	97.48	99.95	0.224	14.67
15.0	100	97.30	100.00	0.005	0.13
33.0	100.00	97.42	99.92	0.005	0.13

Tabelle 6: Ergebnis des Activation Clusterings auf InceptionNet3 für verschiedene prozentuale Anteile an korrumpierten Daten. Mögliche Stickerposition: ((13,17),(16,19)). Schwellenwert $\frac{l}{p}$ auf Trainingsdaten berechnet

erhalten wir eine Genauigkeit des kMeans-Algorithmus von 99.95 Prozent und eine relative Größe von 14.94 Prozent (1651:290). Ein einziger korrumpierter Datenpunkt wird fälschlicherweise als negativ klassifiziert. Bei den relativen Größen auf den anderen Klassen ergibt sich folgendes Bild: Auf Klasse 32 erhalten wir eine relative Größe von 8.84 Prozent. Auf Klasse 2 ergibt sich eine relative Größe von 49.5 Prozent. Auf den unkorrumpierten ergibt sich ein arithmetisches Mittel von 36.01 Prozent. Mit einem lp-Wert von 0.127 wird das Netzwerk als korrumpiert erkannt, wobei 30 Datenpunkte als 50 km/h-Schild und 236 Datenpunkte als 80km/h-Schild klassifiziert werden. Im Vergleich zu [2] schaffen wir es auch, für 10 Prozent und weniger einen Angriff mit 100 Prozent Erfolgsrate zu implementieren. Diese Angriffe werden anhand des exklusiven Retrainings auch erkannt. Das Clustering funktioniert jedoch nicht zu exakt 100 Prozent, sondern liegt zwischen 99 und 100 Prozent, sodass die Hintertür nicht vollständig entfernt wird. Der Hintertürangriff funktioniert dann immer noch, auch wenn die AER mit 31.73 und 14.67 Prozent deutlich geringer als die ursprünglichen 100 Prozent sind.

5.2.4 $Inception\_v3(pretrained = True)$

Das Netz lässt sich bei 15 Prozent korrumpierten Daten mit Hilfe des Activation Clusterings perfekt verteidigen. Wir erhalten sowohl auf den Trainings- als auch den Validierungsdaten eine Genauigkeit beim Clustern von 100.00 Prozent. Die Quotienten $\frac{l}{p}$ betragen $2/288 = 0.0069$ und $212/240 = 0.8833$ für Trainings- bzw. Validierunsgdaten. Im Fall von 33 Prozent korrumpierten Daten erhalten wir 100 Prozent Genauigkeit auf dem Trainingsdatensatz, während die Genauigkeit auf dem Validierungsdatensatz 83.07 Prozent, bei einer Falsch-Negativ-Rate von 51.46 Prozent und einer FPR von 0.0 Prozent, entspricht.

5.3 Label-konsistente Poisoning-Angriffe

Wir greifen in diesem Fall die Klasse 'Zulässige Höchstgeschwindigkeit: 80 km/h' an, d.h. ein prozentualer Anteil an Bildern mit diesem Label wird gestört und mit einem Auslöser versehen. Hierbei unterscheidet sich die Bewertung eines erfolgreichen Angriffs. Anstatt der Angriffserfolgsrate auf der Ursprungsklasse betrachten wir eine arithmetisch **gemittelte Angriffserfolgsrate** (**mAER**) aller Klassen exklusive der Zielklasse.

5.3.1 Referenzwert

Der einfachste Fall ist es, einen Auslöser in der entsprechenden Klasse einzufügen und das Label nicht zu verändern. Wir fügen also einen 3x3 Sticker in die 80 km/h Klasse im Trainingsdatensatz ein, sodass 15 Prozent der Klasse korrumpiert sind, und hoffen, dass das Netz den Trigger erkennt und lernt. Dies funktioniert jedoch nicht. Die mittlere Angriffserfolgsrate beträgt 0.0 Prozent. Ein Clustering auf den Trainingsdaten liefert eine Detektionsrate von 54.77 Prozent bei einer FNR von 72.45 Prozent, während wir auf den Validierungsdaten eine Genauigkeit von 86.72 Prozent bei einer FNR von 100 Prozent erhalten, da aufgrund des nicht funktionierenden Angriffs beim Clustering beinahe alles in eine Klasse geschoben wird. Deshalb müssen wir die Inputs des Netzes so stören, dass die Performance des Netzes abnimmt und es sich auf die Auslöser fokussiert.

5.3.2 Einfluss der Augmentierung

Im entsprechenden Paper [1] werden im Standardmodell keine Transformationen beim Einlesen der Daten zur Augmentation benutzt. Dadurch funktionieren die Angriffe deutlich besser als im Fall der Verwendung unserer bisherigen Transformationen. Wir erhalten die folgenden mehr oder weniger erfolgreichen Angriffe auf das InceptionNet3:

Wir greifen das Netzwerk zunächst mit 100 Prozent korrumpierten Daten an und variieren die Transformationen zum Einlesen der Daten während des Trainings:
Wir benutzen die folgenden Transformationen (vgl. Unterabschnitt 2.4): (1)RandomRotation(15), (2)RandomSizedCrop(32,32), (3)ColorJitter, (4)RandomAffine(15), (5)RandomGrayScale, (6)ToTensor.

Tabelle 7 zeigt, dass die Wahl der Augmentierungen bei dem Angriff mit dem schwarz-weißen Auslöser in der unteren Ecke einen entscheidenden Einfluss hat. Wird beim Einlesen der Daten *RandomRotation* benutzt, funktioniert der Angriff nicht. Die mAER liegt bei 0.0 Prozent.

5.3.3 Angriffe ohne Rotationen: IncNet3 auf IncNet3

Wir nehmen an, der Angreifer kennt das verwendete Modell, d.h die korrumpierten Datenpunkte können auf demselben Netz erstellt werden. Wir erzeugen die Samples nun auf InceptionNet3 und greifen auch dieses Netzwerk an. Wir

Transformationen	mAER
(5,6)	86.90
(3,5,6)	96.50
(3,4,5,6)	85.61
(2,3,4,5,6)	93.66
(1,2,3,4,5,6)	0.0

Tabelle 7: Verschiedene Angriffe mit unterschiedlichen Augmentierungen bei jeweils 100 Prozent korrumpierten Daten in der angegriffenen Klasse

greifen im ersten Fall mit dem Amplitudensticker aus dem Paper [3] an und variieren die Amplituden sowie den Abstand zum Rand. Im zweiten Fall führen wir wieder einen Angriff mit unserem ursprünglichen grün-gelben Sticker auf dem Verkehrsschild durch.
Verwenden wir den einfachen schwarz-weißen Auslöser mit einer Amplitude von $amp = 255$, so erhalten wir weder bei 15 noch bei 33 Prozent korrumpierten Daten einen erfolgreichen Angriff. Die mAER liegt bei 0 Prozent.
Benutzen wir den ursprünglichen grün-gelben Sticker mit einer Pixelgröße von 3x3 und einer Fenstergröße von $((16, 17), (16, 19))$ ohne jegliche Rotationen, so ergeben sich die in Tabelle 8

Anteil korrumpierter Daten in Prozent	mAER	GUD	Detektionsrate
15.0	48.02	97.85	70.24
33.0	85.50	98.19	100.00

Tabelle 8: Erfolg der Angriffe bei Verwendung des ursprünglichen grün-gelben Stickers, platziert in einem zufälligen Fenster mit den Pixelpositionen ((16,17),(16,19)) bei verschiedenen prozentualen Anteilen an korrumpierten Daten

Wir vergleichen den Angriff mit einem zufällig platzierten Sticker mit dem Angriff mit Hilfe eines Amplitudenstickers in der rechten unteren Ecke mit einer Amplitude von 255.
In Tabelle 9 wird deutlich, dass die Angriffe mit einem gelb-grünen Sticker auch bei einem kleineren prozentualen Anteil an korrumpierten Datenpunkten besser funktionieren als bei dem Amplitudensticker in der rechten unteren Ecke des Bildes, sofern keine Augmentation beim Einlesen der Daten benutzt wird.

5.3.4 Angriffe mit Rotationen: InceptionNet3 auf InceptionNet3

Aus Unterunterabschnitt 5.3.3 und Tabelle 7 entsteht die Idee, dass der Angriff mit Hilfe eines Amplitudenstickers besser funktionieren könnte, wenn sich der Auslöser weiter in Richtung der Bildmitte verschoben wird, da durch die Rotationen und das Schneiden des Bildes die Ecken und damit auch die Sticker

Anteil korrumpierter Daten in Prozent	Verwendeter Auslöser	mAER	Detektionsrate	Relative Größe	$\frac{l}{p}$
25.00	sticker	76.63	57.31	43.08	100.5
33.00	sticker	74.43	21.52	46.33	6.80
25.00	amp	0.25	60.83	37.61	8.56
33.00	amp	74.42	21.53	46.33	6.80

Tabelle 9: Label-konsistente Poisoning-Angriffe auf InceptionNet3 mit 25 und 33 Prozent an korrumpierten Daten bei verschiedenen Stickern ohne Augmentierung

abgeschnitten werden können.
Wir variieren den Abstand der inneren Ecke des Stickers zu dem Pixel, das der Nullzustand ist, sofern sich der quadratische Sticker ganz in der Bildecke befindet.

Dabei ergibt sich bei 20 Epochen, 33 Prozent korrumpierten Daten, $amp = 255$ und allen Rotationen folgende Abhängigkeit der mittleren Angriffserfolgsrate, der Genauigkeit beim Clustern und der Falsch-Negativ-Rate auf dem Trainingsdatensatz von dem Abstand:
Wir wählen als Abstand vom Rand $dist = 10$. Vergrößern wir den Abstand

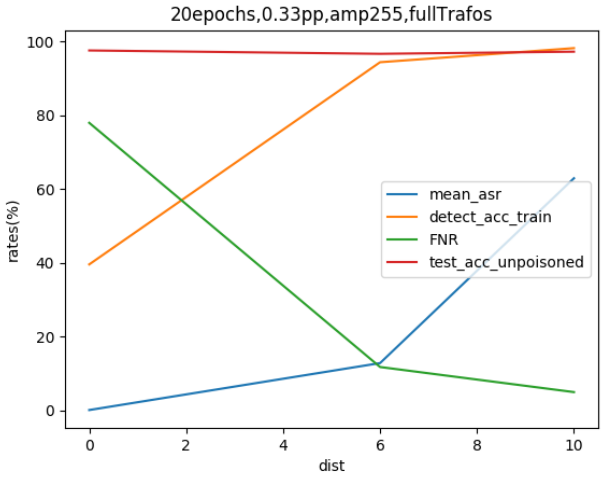

Abbildung 3: Performance in Abhängigkeit des Stickerabstandes $dist = 0, 6, 10$ vom Rand

zum Rand noch weiter, erhalten wir aufgrund von $amp = 255$ einen riesigen Sticker in der Bildmitte.

Der Angriff ist erfolgreich mit einer mittleren Angriffserfolgsrate von 57.43 Prozent Wir erhalten beim Clustern eine Genauigkeit von 98.85 Prozent auf dem Trainingsdatensatz bei einer Fehlerrate von 2.57 Prozent.

Das exklusive Retraining kann hier nicht funktionieren, da sowohl korrumpierte als auch unkorrumpierte Daten dasselbe Label besitzen. Das Clustering könnte vielleicht trotzdem die verschiedenen Entscheidungsstrategien finden, die innerhalb der angegriffenen Klasse existieren. Bei Betrachtung der relativen Clustergröße erhalten wir das Verhältnis 1115:535, was einem Anteil von 32.42 Prozent entspricht. Anhand der relativen Größe wird dieser Angriff also trotzdem aufgedeckt, sofern der Schwellenwert auf 0.33 gesetzt wird.
Fixieren wir $dist = 10$, benutzen alle Rotationen und 20 Epochen zum Training, so erhalten wir abhängig von amp die folgenden Ergebnisse:

amp	255	64	32	16
mAER	56.42	14.61	2.44	0.44
Detektionsrate	98.97	39.27	34.44	27.40
FNR	2.00	76.47	79.60	86.40

Tabelle 10: Angriffs- und Detektionsrate in Abhängigkeit von der Amplitude

Über verschiedene Epochen hinweg ergibt sich das in Abbildung 4 gezeigte Verhalten.

Führen wir einen Angriff mit dem zufällig platzierten gelb-grünen Sticker mit 15 Prozent korrumpierten Datenpunkten und allen Rotationen durch, ergibt sich folgender Sachverhalt: Die Performance auf den unkorrumpierten Testdaten liegt bei 97.88 Prozent. Wir erhalten eine mittlere Angriffserfolgsrate von 68.93 Prozent. Auf den verschiedenen Klassen ergeben sich die folgenden Angriffserfolgsraten:
[0.83 0.78 0.78 1.0 0.96 1.0 1.0 0.98 0.99 0.91 0.97 0.28 0.57 0.71 0.97 0.91 0.93 0.50 0.68 0.55 0.66 0.2 0.33 0.68666667 0.83 0.49 0.65 0.55 0.4 0.41 0.78 0.91 0.93 0.45 0.25 0.68205128 0.31 0.62 0.50 0.33 0.98 0.65 1.0].
Nach dem exklusiven Retraining auf den nicht verdächtigen Daten und einer Klassifikation der verdächtigen Datenpunkte ergeben sich folgende Anzahlen für die verschiedenen Klassen: [0 5 111 18 8 240 0 31 10 0 2 28 76 1 2 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 12 0 0 0 0] D.h., wir haben 240 Datenpunkte des verdächtigen Cluster, die Ordner 5: 80km /h zugeordnet werden, und 111 Datenpunkte des verdächtigen Clusters, die Ordner 2: 50 km/h zugeordent werden. Damit ergibt sich ein $lp - score > 1$ und das Cluster wird nicht als korrumpiert erkannt. [TODO/Fazit]: Clustering funktioniert ganz gut, aber Retraining erkennt den Angriff nicht. Was heißt das als Konsequenz? Ein Mensch müsste pro Klasse die verdächtigen und nicht verdächtigen Daten inspizieren und anhand seiner Beobachtung entscheiden, ob die Daten korrumpiert sind. Der Algorithmus kann das nicht, da der das Label zum Datenpunkt passt.

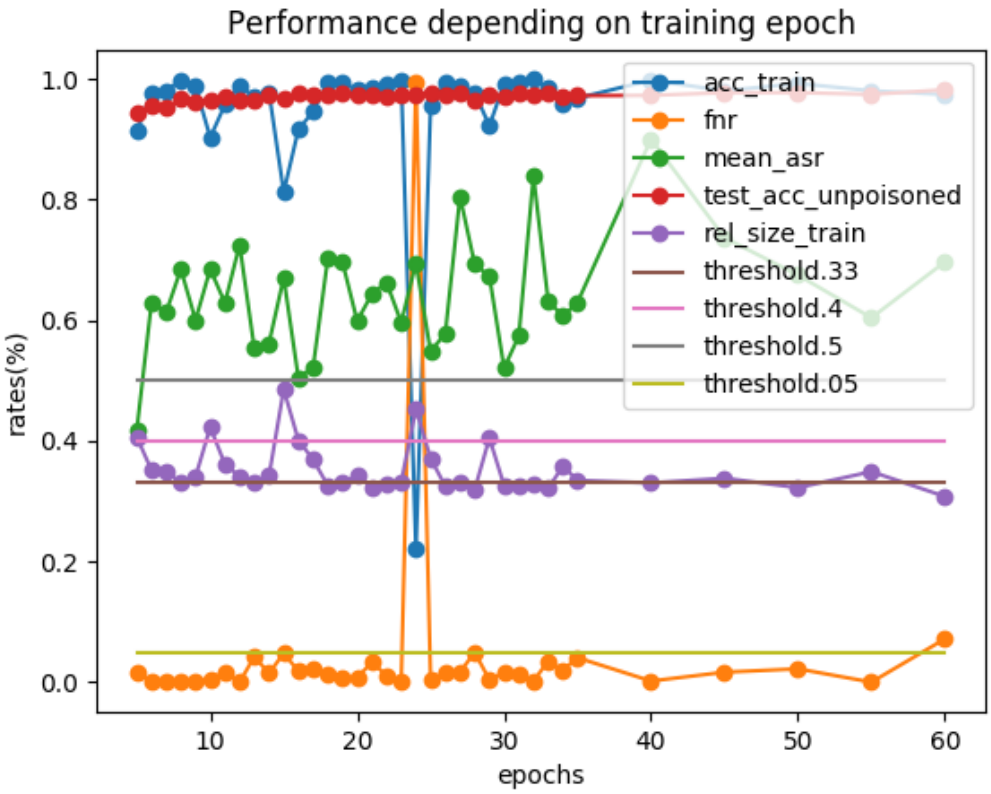

Abbildung 4: Performance in Abhängigkeit von der Epochenanzahl mit 4 Amplitudensticker mit Amplitude 255 und Abstand dist=10

5.3.5 Angriffe mit Rotationen: InceptionNet3 auf Net

Erzeugen wir nun korrumpierte Bilder mit Hilfe des InceptionNet3 und greifen Net an, so erhalten wir bei 20 Epochen und einem Anteil von 33 Prozent korrumpierten Daten eine mittlere Angriffsrate von 61.88 Prozent. Auf dem Trainingsdatensatz erhalten wir eine Genauigkeit des Clusterings von 84 Prozent, was im Vergleich zu den bisherigen Genauigkeiten auf Net recht hoch ist. Zudem ist die FNR mit 4.6 Prozent recht niedrig. Die FPR von 21.25 Prozent führt dann zu zwei Clustern die mit 896 und 754 Datenpunkten ungefähr gleich groß sind. Der Anteil des verdächtigen Trainingssets beträgt 45.70 Prozent.

5.3.6 Angriffe ohne Rotationen: IncNet3 auf Net

Wir vergleichen wieder Angriff und Verteidigung bei verschiedenen prozentualen Anteilen an korrumpierten Daten und verschiedenen Stickern. Wir betrach-

ten Angriffe auf Net, wobei die korrumpierten Datenpunkte mit Hilfe des unabhängigen Netzes IncNet3 generiert wurden. Auch hier funktionieren diejenigen Angriffe besser, die einen gelb-grünen-Sticker benutzen. Die Detektionsgenauigkeit auf dem Trainingsdatensatz ist mit Werten zwischen 58 und 65 Prozent eher gering. Keiner der Angriffe wird anhand des exklusiven Retrainings erkannt. Dies ist in Tabelle Tabelle 11 dargestellt.

Anteil korrumpierter Daten in Prozent	Verwendeter Auslöser	mAER	Detektionsrate	Relative Größe	$\frac{l}{p}$
25.00	sticker	41.39	60.98	41.16	221.0
33.00	sticker	39.24	58.51	39.09	643.05
25.00	amp	18.81	58.39	45.82	66.909
33.00	amp	20.72	64.11	43.76	238.66

Tabelle 11: Label-konsistente Poisoning-Angriffe auf InceptionNet3 mit 25 und 33 Prozent an korrumpierten Daten bei verschiedenen Stickern ohne Augmentierung

6 Fazit

Wir haben gesehen, dass die Standard-Hintertür-Angriffe auf allen drei Netzwerken sehr gut funktionieren. Selbst für sehr kleine prozentuale Anteile an korrumpierten Daten lässt sich noch immer erfolgreich eine Hintertür im Netzwerk implementieren. Die Angriffserfolgsrate liegt jedoch nicht immer bei 100 Prozent. Für größere Anteile funktioniert das Activation Clustering sehr gut. Der Vergleich von relativer Größe oder die Berechnung des Silhouette-Koeffizienten funktioniert jedoch nicht. Dies liegt vermutlich auch daran, dass unser Datensatz im Vergleich zum gewählten Datensatz aus [2] aus deutlich mehr Klassen besteht. Dort sollten Verkehrsschilder in eine von nur insgesamt fünf Klassen eingeordnet werden. Die Methode des Retrainings funktioniert deutlich besser, ist zugleich aber auch aufwendiger. Eine Angriffserfolgsrate von 100% bedeutet nicht unbedingt, dass der Angriff dann auch erkannt wird, vgl. Unterunterabschnitt 5.1.2. Besonders gefährlich sind diejenigen Angriffe, deren Erfolgsrate weniger als 100 Prozent beträgt. Diese lassen sich mit Hilfe des Activation Clusterings nicht erkennen, können aber dennoch großen Schaden anrichten. Die Durchführung unserer Label-konsistent Poisoning-Angriffe funktioniert, jedoch kann die Amplitudenstärke nicht wirklich reduziert werden, da die mittlere Angriffsrate zu klein ist. Das Clustering erzielt Ergebnisse mit teilweise über 90% Genauigkeit. Dieser verdächtige Teil des Datensatzes könnte dann händisch nochmals genau betrachtet werden. Wie zu erwarten funktioniert die Methode des Retrainings hier nicht, da das Label zur tatsächlichen Klasse gehört.

Literatur

[1] Alexander Turner, Dimitris Tsipras, Aleksander Madry. *Label-Consistent Backdoor Attacks.* 6 Dec 2019

[2] *Detecting Backdoor Attacks on Deep Neural Networks by Activation Clustering.* Bryant Chen, Wilka Carvalho, Nathalie Baracaldo, Heiko Ludwig,Benjamin Edwards, Taesung Lee, Ian Molloy, and Biplav Srivastava. 12 November 2018

[3] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. *Badnets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain.* 11 Mar 2019

[4] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, Zbigniew Wojna. *Rethinking the Inception Architecture for Computer Vision.* 11 Dec 2015 - https://arxiv.org/pdf/1512.00567v3.pdf