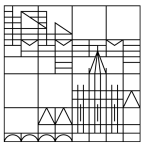
Universität Konstanz



Bundesamt
für Sicherheit in der
Informationstechnik

Untersuchung & Entwicklung von Ansätzen zur Detektion von Poisoning-Angriffen

Master-Arbeit

Lukas Schulth
lukas.schulth@uni.kn

1. Oktober 2021

Erstkorrektor, Zweitkorrektor, Betreuer, Fachbereich Mathematik und Statistik, Masterarbeit zur Erlangung des Titels Master of Science (M.Sc.)
vgl. mit Titelblatt Exposé(!)

Abstract

english

Zusammenfassung

deutsch

Keywords— one, two, three, four

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Inhaltsverzeichnis

1 Einführung

A Complete List of All (arXiv) Adversarial Example Papers [1]
In sicherheitskritischen Anwendungsgebieten ist die Erklärung für das Zustandekommen einer Entscheidung genauso wichtig wie die Entscheidung selbst(Binder, Bach, Montavon, Müller & Samek, 2016).

Clustering auf Datenpunkten direkt(50 Prozent = raten), Clustering auf Aktivierungen gut geeigneter Netzwerkschichten. Clustering auf den Heatmaps der verdächtigen Klasse.

Clustering auf unterschiedlichen Repräsentationen der Bilder:

- Clustering direkt auf den Bildern
- Clustering auf den Activations einer Netzwerkschicht(Im Paper (Chen et al., 2018) wird die vorletzte Schicht benutzt)
- Clustering auf den Heatmaps

In Abschnitt 2 geben wir eine kurze Einführung in Neuronale Netzwerke und stellen die untersuchten Modelle vor. Abschnitt 3 führt in die unterschiedlichen Möglichkeiten eines Poisoning-Angriffs auf Neuronale Netzwerke ein. Abschnitt 4 gibt eine kurze Übersicht über den Bereich der Erklärbaren Künstlichen Intelligenz, wobei ein Beispiel eines Verfahrens, die sogenannte Layer-wise Relevanz Propagation ausführlich in Abschnitt 5 vorgestellt wird. Kern der Arbeit bildet Abschnitt 6, wo wir zu Beginn die grundlegenden Bestandteile des Algorithmus zur Detektion von Poisoning-Angriffen auf Neuronale Netzwerke erklären, bevor die experimentellen Ergebnisse in Unterabschnitt 6.6 ausführen. Ein Vergleich mit anderen Detektionsverfahren wird in Abschnitt 7 durchgeführt.

2 Neuronale Netzwerke

Wir betrachten ein Neuronales Netzwerk (NN), dass die Funktion $f_\theta : \mathbb{R}^n \to \mathbb{R}$, mit $\theta = (w_{il}, b_{il})$ beschreibt. i: Schicht l: Neuron in der Schicht w: Gewichte b: Bias g: nichtlineare Akivierungsfunktion

Pre-Activations(lokal, global): $z_{ij} = x_i * w_{ij}$ $z_j = \sum_i z_{ij} + b_j$

Vorschrift/aktivierungen: $x_j = g(z_{ij})$

Training;testing, Validation, Forward pass, backward pass SGD erklärt im Einführungsteil von (Ioffe & Szegedy, 2015)

fehlende Interpretierbarkeit

[1] `https://nicholas.carlini.com/writing/2019/all-adversarial-example-papers.html`

ReLUs in den meisten Netzwerken
Definition Klasse
Supervised vs Unsupervised
Dimensionality reduction and Visualisation

2.0.1 CNNS

Idee, Abstraktion, high level, low level features, bekannte Netzwerke

Ausführliche Einführung stanford Kurs(?, ?). Unterschied zu FC layers: It is worth noting that the only difference between FC and CONV layers is that the neurons in the CONV layer are connected only to a local region in the input, and that many of the neurons in a CONV volume share parameters. However, the neurons in both layers still compute dot products, so their functional form is identical. Therefore, it turns out that it's possible to convert between FC and CONV layers

Starting with LeNet-5 [10], convolutional neural networks (CNN) have typically had a standardstructure – stacked convolutional layers (optionally followed by contrast normalization and max-pooling) are followed by one or more fully-connected layers. Variants of this basic design areprevalent in the image classification literature and have yielded the best results to-date on MNIST,CIFAR and most notably on the ImageNet classification challenge [9, 21]. For larger datasets suchas Imagenet, the recent trend has been to increase the number of layers [12] and layer size [21, 14],while using dropout [7] to address the problem of overfitting.(Szegedy et al., 2015)

Softmax am Ende für Transformation in Probabilities.
Netzwerk im Netzwerk (?, ?; Szegedy et al., 2015)

2.0.2 Besondere Schichten

Promotion Sebastian Lapuschkin

- *BatchConv* besteht aus

```
1     nn.Conv2d(in_channels=in_channels, out_channels=
     out_channels, **kwargs)
2     nn.BatchNorm2d(num_features=out_channels)
3     nn.ReLU()
4
```

Listing 1: python-interner Aufbau einer BatchConv Schicht

in genau dieser Reihenfolge Bem.: Nur für BatchNorm2d müsste man LRP implementieren, für Conv2d funktioniert das bereits.

Batch Normalization[2]

[2] `https://arxiv.org/pdf/1502.03167.pdf`

2.0.3 Inception v3

Filter, In Klassischen feed forward Netzen wird Output der vorheigen layer ist input der nächsten layer

Jetzt: Inception Block: Previous layer input, 4 operations in parallel, concatenation,1x1 conv -> lower dimension -> less computational cost

Intermediate classifiers: kommt aus multitask learning. Eigentlich eine Möglickeit gegen vasnishing gradients

2.0.4 VGG16

VGG16 is a convolutional neural network model proposed by K. Simonyan and A. Zisserman from the University of Oxford in the paper Very Deep Convolutional Networks for Large-Scale Image Recognition. The model achieves 92.7% top-5 test accuracy in ImageNet, which is a dataset of over 14 million images belonging to 1000 classes. It was one of the famous model submitted to ILSVRC-2014. It makes the improvement over AlexNet by replacing large kernel-sized filters (11 and 5 in the first and second convolutional layer, respectively) with multiple 3×3 kernel-sized filters one after another. VGG16 was trained for weeks and was using NVIDIA Titan Black GP's.

2.1 Datensatz

3 Poisoning-Angriffe

3.1 Standard Poisoning-Angriffe

Wir wollen Schilder der Klasse 50kmh absichtlich falsch als 80kmh. Wir wählen diese beiden Klassen aufgrunde der Größe beider Klassen(s. Aufstellung in Praktikumsbericht). Stoppschildklasse ist wohl vergleichsweise ze8imlich klein.

Dazu fügen wir auf den 50er Schildern einen Sticker ein und ändern das Label auf 80.

Für die Bewertung, wie erfolgreich ein Angriff war, fügen wir in jedem Bild der 50er Klasse im Testdatensatz einen Sticker ein und messen, wie groß der Anteil der 50er Schilder ist, die als 80er Schild klassifiziert werden.

CH- und Backdoor-Artefakte:
In (Anders, Weber et al., 2019) wird wie folgt zwischen Clever Hans- und Backdoor-Artefakten unterschieden. In beiden Fällen wird rechts oben im Bild ein grauer 3x3 Sticker eingefügt. Bei CH geschieht dies bei 25% der äirplaneKlasse. Bei Backdoor-Artefakten werden 10% aller Bilder korrumpiert. Im zweiten Fall wird das entspechende Label abgeändert. Dies entspricht dann einem Standard- bzw. Clean-Label-Poisoning-Angriff. (Wie git funktioniert der CLPA/CH hier ohne die Bilder vorher

schlechter zu machen? TODO: Vergleich mit (Turner, Tsipras & Madry, 2019)). In (Anders, Weber et al., 2019), Kapitel 2.1 wird auch auf die Methode der Spektralen Signatur (Tran, Li & Madry, 2018) eingegangen, die zur Detektion genutzt wird. Diese eignet sich wohl sehr gut für die Backdoor-Attacks, aber nur schlecht für die CH-Artefakte.

3.2 Label-konsistente Poisoning-Angriffe

4 Erklärbare KI

Erklärbarkeit vs. Interpretierbarkeit, youtube talk?!

4.1 Lokale Methoden

4.2 Globale Methoden

5 Layer-wise Relevance Propagation

5.1 Idee

Die Layer-wise Relevance Propagation (LRP) wird in (Bach et al., 2015) erstmalig vorgestellt. Die Idee besteht darin, einen Zusammenhang zwischen der Ausgabe eines Klassifikators $f_\theta : \mathbb{R}^d \to \mathbb{R}^+$ und der Eingabe x herzustellen. Dabei wird eine definiert, die über gewisse eIgenschaften eingeschränkt wird. Die Autoren bezeichnen die Herangehensweise hier selbst als heuristisch und liefern in **??** eine Verallgemeinerung des Konzepts, die gleichzeitig die mathematische Grundlage bildet.

Wir betrachten eine nicht-negative Funktion $f : \mathbb{R}^d \to \mathbb{R}^+$. Im Bereich der Bild-Klassifizierung ist die Eingabe $x \in \mathbb{R}^d$ ein Bild, das wir als Menge von Pixelwerten $x = \{x_p\}$ auffassen können. Dabei beschreibt der Index p einen genauen Pixelpunkt. Während für schwarz-weiß Bilder $x_p \in \mathbb{R}$ gilt, gilt im Fall von RGB-Bildern $x_p \in \mathbb{R}^3$ für die einzelnen Farbkanäle Rot, Grün und Blau. Die Funktion $f(x)$ ist ein Maß dafür, wie präsent ein oder mehrere Objekte in der Eingabe/im Eingabebild vorhanden sind. Ein Funktionswert $f(x) = 0$ beschreibt die Abwesenheit. Gilt andererseits $f(x) > 0$, so wird die Präsenz mit einem gewissen Grad an Sicherheit oder eine gewisse Menge zum Ausdruck gebracht.

Mit Hilfe der LRP soll nun jedem Pixel p im Eingabebild eine Relevanz $R_p(x)$ zugeordnet werden, die für jedes Pixel x_p angibt, mit welcher Größe es für das Entstehen einer Entscheidung $f(x)$ verantwortlich ist. Die Relevanz eines jeden Pixels wird dabei in einer Heatmap $R(x) = \{R_p(x)\}$ zusammengefasst.

Die Heatmap besitzt dieselbe Größewie x und kann als Bildvisualisiert werden.

Wir definieren die folgenden Eigenschaften:

Definition 5.1.1. *Eine Heatmap $R(x)$ heißt konservativ, falls gilt:*

$$\forall x : f(x) = \sum_p R_p(x), \tag{5.1}$$

d.h. die Summe der im Pixelraum zugeordneten Relevanz entspricht der durch das Modell erkannten Relevanz.

Definition 5.1.2. *Eine Heatmap $R(x)$ heißt positiv, falls gilt:*

$$\forall x, p : R_p(x) \geq 0, \tag{5.2}$$

d.h. alle einzelnen Relevanzen einer Heatmap sind nicht-negativ.

Die erste Eigenschaft verlangt, dass die umverteilte Gesamtrelevanz der Relevanz entspricht, mit der ein Objekt im Eingabebild durch die Funktion $f(x)$ erkannt wurde. Die zweite EIgenschaft beschreibt, dass keine zwei Pixel eine gegensätzliche Aussage über die Existenz eines Objektes treffen können. Beide Definitionen zusammen ergeben die Definition einer *konsistenten* Heatmap:

Definition 5.1.3. *Eine Heatmap $R(x)$ heißt konsistent, falls sie konservativ und positiv ist, d.h Definition 5.1.1 und Definition 5.1.2 gelten.*

Für eine konsistente Heatmap gilt dann ($f(x) = 0 \Rightarrow R(x) = 0$), d.h. die Abwesenheit eines Objektes hat zwangsläufig auch die Abwesenheit jeglicher Relevanz in der Eingabe zur Folge, eine Kompensation durch positive und negative Relevanzen ist folglich nicht möglich.

Bemerkung 5.1.4. *Die geforderten Eigenschaften an eine Heatmap definieren diese nicht eindeutig. Es sind also mehrere Abbildungen möglich, die die genannten Forderungen erfüllen. Beispiele dafür sind eine natürliche Zerlegung und Taylor-Zerlegungen (Montavon, Lapuschkin, Binder, Samek & Müller, 2017a).*

Die LRP liefert nun ein Konzept, mit dem eine Zerlegung

$$f(x) = \sum_d R_d \tag{5.3}$$

bestimmt werden kann.
TODO: Summenabfolge von layer zu layer einfügen

Wir gehen nun davon aus, dass die Funktion f ein NN repräsentiert, dass aus mehreren Schichten mit mehreren Neuronen pro Schicht und dazwischengeschalteten nicht-linearen Aktivierungsfunktionen aufgebaut ist. Die erste Schicht ist die Eingabe-Schicht, bestehend aus den Pixeln eines Bildes. Die letzte Schicht ist die reellwertige Ausgabe von f. Die l-te Schicht ist durch einen Vektor $z = (z_d^l)_{d=1}^{V(l)}$ der Dimension $V(l)$ dargestellt. Sei also eine Relevanz $R_d(l+1)$ für jede Dimension

$z_d^{(l+1)}$ des Vektors z in der Schicht $l+1$ gegeben. Die Idee besteht nun darin, eine Relevanz $R_d^{(l)}$ für jede Dimension $z_d^{(l)}$ des Vektors z in der Schicht l zu finden, die einen Schritt näher an der Eingabeschicht liegt, sodass die folgende Abfolge von Gleichungen gilt:

$$f(x) = ... = \sum_{d \in l+1} R_d{}^{(l+1)} = \sum_{d \in l} R_d{}^{(l)} = ... = \sum_d R_d^{(1)}. \tag{5.4}$$

Für diese Funktion benötigen wir eine Regel, mit der die Relevanz eines Neurons einer höheren Schicht $R_j^{(l+1)}$ auf ein Neuron einer benachbarten, näher an der Eingabeschicht liegendes Neuron, übertragen werden kann. Die Übertragung der Relevanz zwischen zwei solchen Neuronen wird mit $R_{i \leftarrow j}$ bezeichnet. Auch hier muss die übertragene Relevanz erhalten bleiben. Es wird also gefordert:

$$\sum_i R_{i \leftarrow j}^{(l,l+1)} = R_j^{(l+1)}. \tag{5.5}$$

D.h. die gesamte Relevanz eines Neurons der Schicht $l+1$ verteilt sich komplett auf alle Neuronen der Schicht l. Im Falle eines linearen NN $f(x) = \sum_i z_{ij}$ mit der Relevanz $R_j = f(x)$ ist eine Zerlegung gegeben durch $R_{i \leftarrow j} = z_{ij}$. Im allgemeineren Fall ist die Neuronenaktivierung x_j eine nicht-lineare Funktion abhängig von z_j. Für die beiden Aktivierungsfunktionen $tanh(x)$ und $ReLU(x)$ - beide monoton wachsend mit $g(0) = 0$ - bieten die Vor-Aktivierungen noch immer ein sinnvolles Maß für den relativen Beitrag eines Neurons x_i zu R_j (müsste das nicht umgekehrt sein, die INdizes?!?!).

Eine erste Mögliche Relevanz-Zerlegung, basierend auf dem Verhältnis zwischen lokalen und globalen Vor-Aktivierung, ist gegeben durch:

$$R_{i \leftarrow j}^{(l,l+1)} = \frac{z_{ij}}{z_j} \cdot R_j^{(l+1)}. \tag{5.6}$$

Für diese Relevanzen $R_{i \leftarrow j}$ gilt die Erhaltungseigenschaft 5.4, denn:

$$\sum_i R_{i \leftarrow j}{}^{(l,l+1)} = R_j^{l+1} \cdot (1 - \frac{b_j}{z_j}). \tag{5.7}$$

Dabei steht der rechte Faktor für die Relevanz, die durch den Bias-Term absorbiert wird. Falls notwendig, kann die verbleibende Bias-relevanz auf jedes Neuron x_i verteilt werden(?,s.Abschnitt über Biases Promotion, S.Lapuschkin).

Diese Regel wird in der Liteartur als LRP-0 bezeichnet. Ein Nachteil dieser ist, dass die Relevanzen $R_{i \leftarrow}$ für kleine globalen Voraktivierung z_j beliebig große Werte annehmen können.

Um dies zu verhindern, wird in der LRP-ε-Regel ein vorher festgelegter Parameter $\varepsilon > 0$ eingeführt:

$$R_{i \leftarrow j}^{(l,l+1)} = \begin{cases} \frac{z_{ij}}{z_j + \varepsilon} \cdot R_j^{(l+1)}, & z_j \geq 0 \\ \frac{z_{ij}}{z_j - \varepsilon} \cdot R_j^{(l+1)}, & z_j < 0 \end{cases} \tag{5.8}$$

In (Bach et al., 2015) wird die Layer-wise Relevance Propagation erstmalig vorgestellt. Zudem wird eine Taylor Zerlegung präsentiert, die eine Approximation der LRP darstellt.

Hier[3] werden einige Bereiche vorgestllt, in denen LRP angewendet wurde.

5.2 Behandlung von biases

5.3 Beispiel an einem kleinen Netzwerk

5.4 Deep Taylor Decomposition

Mathematischer Hintergrund für LRP.LRP als Spezialfall von DTD

5.4.1 Taylor Decomposition

We will assume that the function f (x) is implemented by a deep neural network, composed of multiple layers of representation, where each layer is composed of a set of neurons. Each neuron performs on its input an elementary computation consisting of a linear projection followed by a nonlinear activation function. Deep neural networks derive their high representational power from the interconnection of a large number of these neurons, each of them, realizing a small distinct subfunction.

Laut (Montavon, Lapuschkin, Binder, Samek & Müller, 2017b) ist die in (Bach et al., 2015) vorgestellte Layer-wise Relevance Propagation eher heuristisch. In diesem Paper wird nun eine solide theoretische Grundlage geliefert.

DTD liefert den mathematischen Hintergrund für LRP

Simple Taylor decomposition. Finde rootpoints, sodass Erhaltungseigenschaft erhalten bleibt.

Simple Taylor in Practice: funktioniert in der Praxis nicht wirklich.Viel Noise meistens positive Relevanz

Relevanz Propagation: Heatmaps look much cleaner

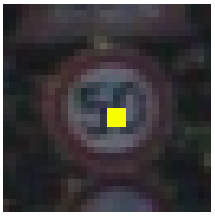
Simple Taylor:- root point hard to find -gradient shattering. Gradient looses its informative structure in big layer nets

Use Taylor Decomposition to explain LRP from layer to layer

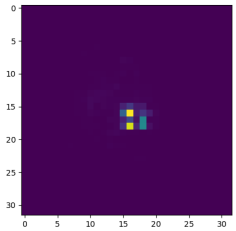
5.4.2 Deep Taylor Decomposition

LRP in verschiedenen Anwendungsgebieten (Montavon, Binder, Lapuschkin, Samek & Müller, 2019), 10.2. In diesem Paper:LRP-0 schlechter als LRP-ε schlechter alsLRP-γ schlechter als Composite-LRP.

[3]https://towardsdatascience.com/indepth-layer-wise-relevance-propagation-340f95deb1ea



(a) Verkehrsschild der Klasse 'Höchstgeschwindigkeit: 50km/h' versehen mit einem 3x3 Sticker und dem Label 'Höchstgeschwindigkeit: 80km/h'



(b) Zugehörige Heatmap bezüglich der Klasse 'Höchstgeschwindigkeit: 80km/h'

Abbildung 1: (Optischer) Vergleich von korrumpiertem Datenpunkt und berechnter Heatmap.

5.5 Verschiedene Verfahren

5.6 Eigenschaften

Beweise in DTD Paper

- Numerische Stabilität
- Konsistenz (mit Linearer Abbildung)
- Erhaltung der Relevanz

5.7 Behandlung besonderer Schichten

5.7.1 BatchNorm2D

5.8 LRP für Deep Neural Nets/Composite LRP

5.9 Verarbeitung der Heatmaps

Aktuell benutzte defaul colormap ist Option D (Viridis)[4]

- Wertebereich
- Interpretation
- Skalen
- Normalisierung

[4] `https://bids.github.io/colormap/`

5.10 Implementierung

5.10.1 Tensorflow

5.10.2 pytorch

Allgemeines Tutorial:[5]
pytorch-LRP für VGG16 wird vorgestellt.

GiorgioML[6]:
Alternative pytorch-Implementierung basierend auf Tensorflow paper.

moboehle[7]:
Der code entstand im Rahmen der Forschungsarbeit (Böhle, Eitel, Weygandt & Ritter, 2019), in der eine Alzheimer-Festellung aufgrund von Bilddaten(scans?) vorgenommen wird. Framework leicht anpasspar. Benutzt pytorch hooks. Unterstützte Netzwerkschickten[8]:

```
  torch.nn.BatchNorm1d,
  torch.nn.BatchNorm2d
  torch.nn.BatchNorm3d,
  torch.nn.ReLU,
  torch.nn.ELU,
  Flatten,
  torch.nn.Dropout,
  torch.nn.Dropout2d,
  torch.nn.Dropout3d,
  torch.nn.Softmax,
  torch.nn.LogSoftmax,
  torch.nn.Sigmoid


```

Listing 2: Verfügbare Schichten und Aktivierungsfunktionen

fhvilshoj[9]:

LRP für linear und Convolutional layers

- Die Klassen

  torch.nn.Sequential, torch.nn.Linear und torch.nn.Conv2d werden erweitert, um autograd für die Berechnung der Relevanzen zu berechnen.

[5] https://git.tu-berlin.de/gmontavon/lrp-tutorial
[6] https://giorgiomorales.github.io/Layer-wise-Relevance-Propagation-in-Pytorch/
[7] https://github.com/moboehle/Pytorch-LRP
[8] https://github.com/moboehle/Pytorch-LRP/blob/master/inverter_util.py
[9] https://github.com/fhvilshoj/TorchLRP

- Ausgabe der Relevanzen von Zwischenschichten ist möglich
- : Implementierte Regeln: epsilon Regeln mit epsilon=1e-1, gamma-regel mit gamma=1e-1. alphabeta-Reagel mit a1b0 und a2b1
- Netz muss hier umgeschrieben werden, sodass die Anwendung des Algorithmus möglich wird.

```
conv2d = {
  "gradient":              F.conv2d,
  "epsilon":               Conv2DEpsilon.apply,
  "gamma":                 Conv2DGamma.apply,
  "gamma+epsilon":         Conv2DGammaEpsilon.apply,
  "alpha1beta0":           Conv2DAlpha1Beta0.apply,
  "alpha2beta1":           Conv2DAlpha2Beta1.apply,
  "patternattribution":    Conv2DPatternAttribution.apply,
  "patternnet":            Conv2DPatternNet.apply,
}
```

Listing 3: Implementierte Regeln fhvilshoj

Zennit:[10] Zennit (Zennit explains neural networks in torch)

- Modell wird mithilfe eines Canonizers so aufbereitet, dass LRP möglich wird
- Backward pass wird modifiziert, um Heatmaps zu erhalten.
- VGG- und ResNet-Beispiel

6 Detektion von Poisoning-Angriffen basierend auf LRP

6.1 Idee

Die Idee zur Detektion von Poisoning-Angriffen besteht aus den folgenden Schritten:

- Berechnung der Heatmaps mit Hilfe der LRP
- Berechnung einer Distanzmatrix basierend auf L^2- oder GMW-Distanz
- Spektrale Relevanzanalyse (Bestimmung der verschiedenen Cluster innerhalb einer Klasse)

[10] `https://github.com/chr5tphr/zennit`

Bemerkung: Anstatt das Clustering nur auf den Heatmaps durchzuführen, könnten die LRP-Ausgaben und/oder Aktivierungen bestimmer NEtzwerkschichten hinzugenommen werden.

The theory of optimal transport generalizes thatintuition in the case where, instead of moving only one item at a time, one is concernedwith the problem of moving simultaneously several items (or a continuous distributionthereof) from one configuration onto another.(“Computational Optimal Transport”, 2019)

6.2 Verwendete Distanzen & Approximationen

Um die Struktur innerhalb einer Klasse zu analysieren, benötigen wir eine Metrik. Anhand dieser wird abhängig von den Heatmps einer Klasse eine Affinitätsmatrix berechnet, die dann anschließend zur Berechnung der Spektralen Einbettung als wichtigster Schritt von SpRAy verwendet wird. Wir wollen dazu die im Folgenden vorgestellten Metriken verweden.
Wie in (Anders, Marinč et al., 2019) summieren wir über die Farbkanäle, um einen einzelnen Relevanzwert pro Pixelpunkt zu erhalten. Wir benötigen also eine Metrik zur Berechnung der Distanz zwischen 32x32 großen Heatmaps.
Wir normalisieren die Relevanzen zusätzlich auf das Intervall $[0, 1]$.

Definition 6.2.1 (Distanz)**.** *Eine Distanz*

Definition 6.2.2. *Eine Metrik*

[Metrik]

6.2.1 Euklidische Distanz

Für den Fall der euklidischen Distanz schreiben wir die Relevanzwerte pro Pixel als Vektor und berechnen die Distanz zweier Heatmaps x und y wie folgt[11]:

$$d_{x,y} = \sqrt{\sum_{i=1}^{32*32} (x_i - y_i)^2}.$$

6.2.2 Gromov-Hausdorff-Distanz

Definition 6.2.3 (Hausdorff-Distanz)**.** *Sei (M, d) ein metrischer Raum. Für Seien $X, Y \subset (M, d)$ definieren wir die Hausdorff-Distanz $d_H(X, Y)$ als*

$$d_H(X, Y) = \max\{\sup_{x \in X} \inf_{y \in Y} d(x, y), \sup_{y \in Y} \inf_{x \in Y} d(x, y)\}. \tag{6.1}$$

[11] `https://paulrohan.medium.com/euclidean-distance-and-normalization-of-a-vector-76f7a97abd9`

Definition 6.2.4 (Metrischer Maßraum)**.** *Ein metrischer Maßraum ist ein Tripel* (X, d_x, μ_X)*, wobei* (X, d_X) *ein metrischer Raum und* μ_X *ein borelsches W-Maß auf* X *ist.*

Definition 6.2.5 (Correspondance)**.** *content...*

Definition 6.2.6 (Kopplung)**.** *Seien*

Seien (X, d_X) und (Y, d_Y) zwei metrische Räume. Wir betrachten im Folgenden die Abbildung

$$\Gamma_{X,Y} : (X \times Y) \times (X \times Y) \to \mathbb{R}^+, \tag{6.2}$$

gegeben durch

$\Gamma_{X,Y}(x, y, x', y') := |d_X(x, x') - d_Y(y, y')|$.

Definition 6.2.7 (Gromov-Hausdorff-Distanz)**.** *Für die metrischen Räume* (X, d_X) *und* (Y, d_Y) *ist die Gromov-Hausdorff-Distanz definiert als*

$$d_{\mathcal{GH}} = \frac{1}{2} \inf_R ||\Gamma_{X,Y}||_{L^\infty(R \times R)}. \tag{6.3}$$

6.3 Kantorovichs Optimal Transport Problem

6.3.1 Gromov-Wasserstein-Distanz

6.3.2 Entropisch Regularisierte Gromov-Wasserstein-Distanz

Entropic Regularization of Optimal Transport ("Computational Optimal Transport", 2019) Mehrere Möglichkeiten einer Regularisierung der GW-Distanz[12]:

- first
- second
- Entropy

Optimal Transport: Regularization and Applications [13] Python Optimal Transport Toolbox[14][15]

Matlab code[16] zum Paper (Peyré, Cuturi & Solomon, 2016). Implementierung ist mittlerweile auch in der Python Optimal Transport toolbox verfügbar(s.o.).

[12] `https://www.youtube.com/watch?v=cPVMHWF8fmE&t=2532s`
[13] `https://www.otra2020.com/schedule`
[14] `https://pythonot.github.io/quickstart.html`
[15] `https://pythonot.github.io/auto_examples/gromov/plot_gromov.html`
[16] `https://github.com/gpeyre/2016-ICML-gromov-wasserstein`

6.3.3 Wasserstein Baryzentren

6.3.4 Verallgemeinerung

Verallgemeinerung auf beliebige Matrizen C, d.h. diese Distanzmatrizen müssen nicht notwendigerweise positiv sein und die Dreiecksungleichung erfüllen.

Definiere die verallgemeinerte Gromov-Wasserstein-Distanz (vGWD) wie folgt:

Definition 6.3.1 (Verallgemeinerte Gromov-Wasserstein-Distanz)**.** *Seien zwei gewichtete Ähnlichkeitsmatrizen* $(C, p) \in \mathbb{R}^{N_1 \times N_1} \times \Sigma_{N_1}$ *und* $(\bar{C}, q) \in \mathbb{R}^{N_2 \times N_2} \times \Sigma_{N_2}$ *gegeben. Sei* T *eine Kopplung zwischen den beiden Räumen, auf denen die Matrizen* C *und* $\bar{C}$ *definiert sind. Sei* L *eine Fehlerfunktion. Dann definieren wir die verallgemeinerte Gromov-Wasserstein-Distanz als*

$$vGw(C, \bar{C}, p, q) := \min_{T \in \mathcal{C}_{p,q}} \varepsilon_{C,\bar{C}}(T), \tag{6.4}$$

wobei gilt $\varepsilon_{C,\bar{C}}(T) := \sum_{i,j,k,l} L(C_{i,k}, \bar{C}_{j,l}) T_{i,j} T_{k,l}$.

Häufig verwendete Fehlerfunktionen sind die quadratische Fehlerfunktion $L(a, b) = L_2(a, b) := \frac{1}{2}|a - b|^2$ und die Kullback-Leibler-Divergenz $L(a, b) = KL(a|b) := a \log(a/b) - a + b$.

Diese Definition der Gromov-Wasserstein-Distanz verallgemeinert die Version in (Peyré et al., 2016), da hier beliebige Fehlerfunktionen betrachtet werden.

Für L=L2 zeigt Memoli,2011, dass GW1/2 eine Distanz definiert.

6.4 Spektrales Clustering

Wir folgen (Von Luxburg, 2007). Gegeben:Datenpunkte $x_i, ..., x_n$ sowie eine Größe $s = s_{ij} \in \mathbb{R}^+$, die einen paarweisen Zusammenhang der einzelnen Punkte beschreiben.

Ziel: Aufteilen der Punkte in verschiedene Cluster, sodass sich Punkte innerhalb eines Clusters ähnlich bezüglich s sind.

Alternative Repräsentation der Daten mithilfe eines Ähnlichkeitsgraphen $G = (V, E)$ möglich.

Umformulierung des Clustering-Problems mithilfe des Ähnlichkeitsgraphen: Finde Pertitionierung des Graphen, sodass die Kanten-Gewichte innerhalb einer Gruppe niedrig (niedriges Gesamtgewicht?) und außerhalb einer Gruppe große sind.

Graph-Notationen:

Verschiedene Konstruktionsmöglichkeiten von Ähnlichkeitsgraphen:

- ε-Nachbarschaft-Graph

- kNN-Graph

- fully connected graph

6.5 k-means / k-means++ -Clustering

Beispiel-Implementierung[17]

Baryzentrische Koordinaten [18]

6.6 Anwendung auf unterschiedliche Poisoning-Angriffe

Berechnung der Relevanzen:

Wir berechnen die Relevanzen jedes einzelenen Eingabebildes klassenweise, d.h. besitzt eine Eingabe das Label y, so berechnen auf einem trainierten Netzwerk, für jeden Pixelwert der Eingabe, wie relevant dieser für die Ausgabe f(x) = y ist.
Wir summieren über die Farbaxen des Bildes, um einzelne Relevanzen pro Pixelpunkt zu erhalten.
Für die Berechnung der Relevanzen benutzen wir eine modifizierte Version des im Rahmen von (Böhle et al., 2019) entstandenen Programmcodes[19].

Vorverarbeitung der Relevanzen:
In (Lapuschkin et al., 2019) wird anschließend ein Sum-Pooling auf die Relevanzen angewendet, um eine Dimensionsreduktion zu erhalten. Wie in (Anders, Marinč et al., 2019) verzichten wir auf eine weitere Dimensionsreduktion, da wir nur relativ kleine Relevanzen der Größe $32x32$ verarbeiten.

Für eps=5e-2 liegen beide barycentren identisch weit weg. Probiere nun eps=5e-3

Berechnung der Distanzen und Aufstellen einer Affinitätsmatrix:

Wir berechnen zunächst eine Distannzmatrix, die die paarweisen Distanzen aller Heatmaps einer Klasse enthält.

Für die Berechnung der euklidischen Distanz betrachten wir Heatmaps x, y der Größe $32x32$ als Elemente $x, y \in \mathbb{R}^{32*32}$ Die Distanz lässt sich dann wie in Unterunterabschnitt 6.2.1 berechnen.

[17] `https://towardsdatascience.com/k-means-implementation-in-python-and-spark-856e7eb5fe9b`

[18] `https://de.wikipedia.org/wiki/Baryzentrische_Koordinaten`

[19] `https://github.com/moboehle/Pytorch-LRP`

Die Gromov-Wasserstein-Distanz lässt sich wie in (Peyré et al., 2016) angegeben berechnen.

Barycenters Definition und Vergleich zum euklidischen Raum (Agueh & Carlier, 2011)

In einer Affinitätsmatrix oder Ähnlichkeitsmatrik sind die

Berechnung Spektralen Einbettung:

Dimensionsreduktion vor dem Clustering ?!

In (Chen et al., 2018) wird beispielsweise eine Dimensionsreduktion mit PCA durchgeführt.

k-Means-Clustering:

Bemerkung 6.6.1. *In (Anders, Marinč et al., 2019) Kapitel '2.3. Fisher Discriminant Analysis for Clever Hans identification' wird ein Verfahren vorgestellt, mit dem verdächtige Klassen indentifiziert werden können. Für diese würde man anschließend das obige Verfahren durchführen*

6.6.1 Standard Poisoning-Angriffe

=> Did save Model - incv320epochsnormalized - at epoch: 19 => [19] loss: 0.125, accuracy: 96.514=> Accuracy of validation Dataset: 98.491

=> FINISHED TRAINING loss on test dataset: 3.284087032527311 Accuracy of test Dataset: 0.919

BDSR classwise: [nan nan 1. nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan]

Performance of poisoned net on unpoisoned training data: loss on test dataset: 0.15269652156995248 Accuracy of test Dataset: 0.961

Performance auf nicht korrumpierten Daten:
=> FINISHED TRAINING loss on test dataset: 0.15834395289583986 Accuracy of test Dataset: 0.968

Spektralanalyse:
Clustering(euklidisch): $(tn, fp, fn, tp) = 1650, 0, 386, 427)$

Clustering(GWD):

n=5 Berechnung der Distanzen: Clustering: [1. 0. 1. 0. 1.] — 379.38680505752563 seconds —

Berechnung von 2 Baryzentren — 725.2780563831329 seconds —

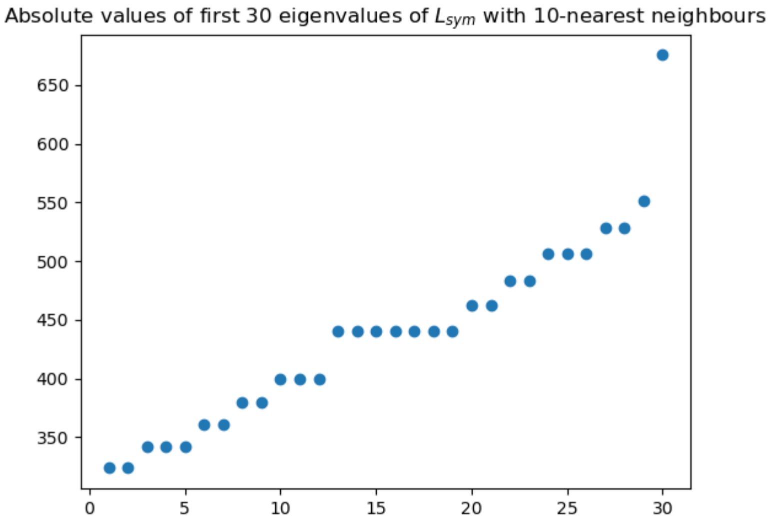


Abbildung 2: Ergebnis des spektralen Clusterings unter Verwendung der Euklidischen Distanz und k=10 Nachbarn

n=25 Berechnung der Distanzen: Clustering: [1. 0. 1. 0. 1. 0. 1. 0. 0. 1. 1. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 1. 0. 1. 1.] — 1860.5442748069763 seconds —

Berechnung von zwei Baryzentren — 2443.7355489730835 seconds —

n=50 Berechnung der Distanzen: — 1956.3804433345795 seconds —

Berechnung von zwei Barycentern: — 5670.336745262146 seconds —

15 Prozenmt 3x3 Sticker => Did save Model - SAincV3s315 - at epoch: 50 => [50] loss: 0.060, accuracy: 98.405EarlyStopping counter: 20 out of 20 Early stopping => FINISHED TRAINING loss on test dataset: 3.736420426017416 Accuracy of test Dataset: 0.911

BDSR classwise: [nan nan 1. nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan] Performance of poisoned net on unpoisoned training data: loss on test dataset: 0.1822473893020984 Accuracy on test Dataset: 0.961

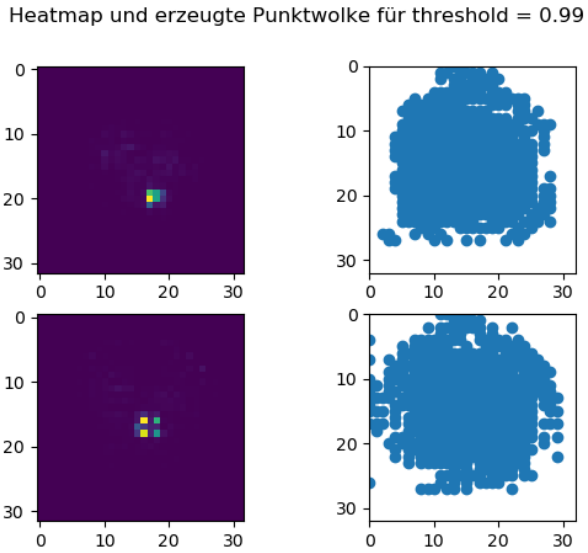


Abbildung 3: Auswahl der relevantesten Pixel (bis zu 99% der Gesamtmasse) zweier Heatmaps

6.6.2 Label-konsistente Poisoning-Angriffe

7 Vergleich mit anderen Verfahren

7.1 Activation Clustering

7.2 Räumliche Transformationen

- ASR ist sehr stark vom Ort des Triggers abhängig.
- Ort des Triggers kann nicht direkt geändert werden.
- Benutze Transformationen(Flipping, Scaling), um den Trigger wirkungslos zu machen.
- Somit kann die ASR während der Inferenz verringert werden. Es lässt sich aber keine Auussage darüber treffen, ob ein Angriff vorliegt

8 Weitere mögliche Schritte

- Automatische Platzierung des Auslösers an fest gewählter Position auf dem Verkehrsschild anstatt zufälligem Platzieren in einem Fenster mit vorher festgelegter Größe. In (Gu, Dolan-Gavitt & Garg, 2017) wird Faster-RCNN (F-

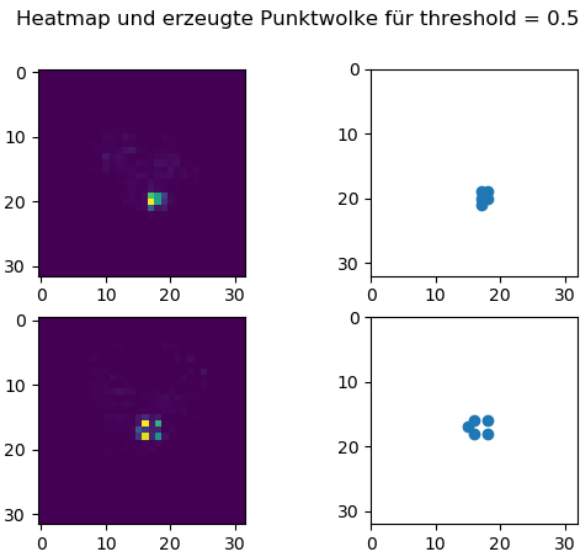


Abbildung 4: Auswahl der relevantesten Pixel (bis zu 50% der Gesamtmasse) zweier Heatmaps

RCNN) zur Klassifikation des LISA-Datensatzes[20] benutzt. Es ist die Aufgabe, die Verkehrsschilder in die 3 Superklassen Stoppschild, Geschwindigkeitsbegrenzung und Warnschild einzuteilen. Der Datensatz enthält zudem die BoundingBoxen, sodass der Auslöser genauer angebracht werden kann.

- Verbesserte Version der Layer-wise Relevance Propagation
- Untersuchung anderer Verfahren, die die Interpretierbarkeit ermöglichen, beispielsweise: VisualBackProp: efficient visualization of CNNs[21]
- Vergleich mit Cifar-10/Cifar-100 Datensatz[22][23]

9 Zusammenfassung

[20] http://cvrr.ucsd.edu/LISA/lisa-traffic-sign-dataset.html
[21] https://arxiv.org/abs/1611.05418
[22] https://www.cs.toronto.edu/~kriz/cifar.html
[23] https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf

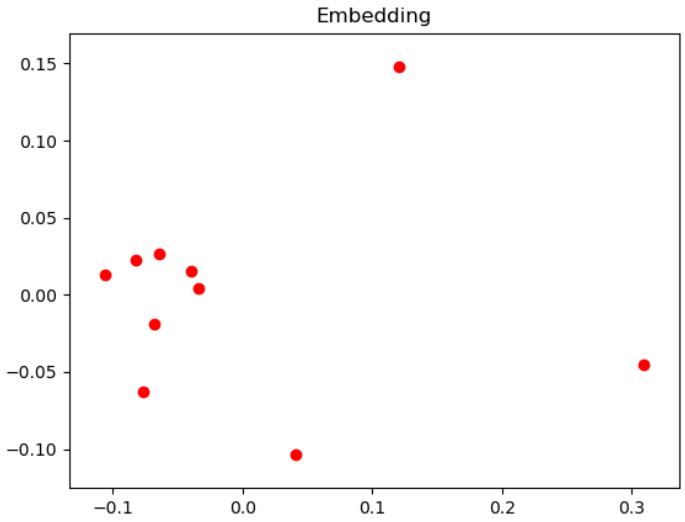


Abbildung 5: Einbettung des Baryzentrums mithilfe von Multidimensionaler Skalierung(MDS) bei der Wahl von 99% der Gesamtmasse

A Verwendete Netzwerke

A.1 Net

```
1
2  class Net(nn.Module):
3
4    def __init__(self, ):
5      super(Net, self).__init__()
6      self.size = 64 * 4 * 4
7      self.conv1 = nn.Conv2d(in_channels=3, out_channels=12,
   kernel_size=5, padding=2)
8      self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
9      self.conv1_in = nn.InstanceNorm2d(12)
10     self.conv2 = nn.Conv2d(in_channels=12, out_channels=32,
    kernel_size=5, padding=2)
11
12     self.conv2_bn = nn.BatchNorm2d(32)
13
14     self.conv3 = nn.Conv2d(in_channels=32, out_channels=64,
    kernel_size=5, padding=2)
15
16     self.fc1 = nn.Linear(self.size, 256)
17     self.fc1_bn = nn.BatchNorm1d(256)
```

```
18        self.fc2 = nn.Linear(256, 128)
19        self.fc3 = nn.Linear(128, 43)
20
21
22      def forward(self, x):
23        x = self.pool(F.relu(self.conv1_in(self.conv1(x))))
24        x = self.pool(F.relu(self.conv2_bn(self.conv2(x))))
25        x = self.pool(F.relu(self.conv3(x)))
26        x = x.view(-1, self.size)
27        x = F.relu(self.fc1_bn(self.fc1(x)))
28        x = F.dropout(x)
29        xx = F.relu(self.fc2(x))
30        x = F.dropout(xx)
31        x = self.fc3(x)
32
33        return x, xx
34
35
```

Listing 4: Kleines Netzwerk

```
1     InceptionNet3(
2     (features): Sequential(
3     (0): InceptionA(
4     (parallel_dummyA): New_parallel_chain_dummy()
5     (conv1x1): BatchConv(
6     (conv): Conv2d(3, 64, kernel_size=(1, 1), stride=(1, 1))
7     (bn): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=
    True, track_running_stats=True)
8     (relu): ReLU()
9     )
10    (parallel_dummyB): New_parallel_chain_dummy()
11    (conv5x5_1): BatchConv(
12    (conv): Conv2d(3, 48, kernel_size=(1, 1), stride=(1, 1))
13    (bn): BatchNorm2d(48, eps=1e-05, momentum=0.1, affine=
    True, track_running_stats=True)
14    (relu): ReLU()
15    )
16    (conv5x5_2): BatchConv(
17    (conv): Conv2d(48, 64, kernel_size=(5, 5), stride=(1, 1),
     padding=(2, 2))
18    (bn): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=
    True, track_running_stats=True)
19    (relu): ReLU()
20    )
21    (parallel_dummyC): New_parallel_chain_dummy()
22    (conv3x3dbl_1): BatchConv(
23    (conv): Conv2d(3, 64, kernel_size=(1, 1), stride=(1, 1))
24    (bn): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=
```

```
   True, track_running_stats=True)
25  (relu): ReLU()
26  )
27  (conv3x3dbl_2): BatchConv(
28  (conv): Conv2d(64, 96, kernel_size=(3, 3), stride=(1, 1),
    padding=(1, 1))
29  (bn): BatchNorm2d(96, eps=1e-05, momentum=0.1, affine=
   True, track_running_stats=True)
30  (relu): ReLU()
31  )
32  (conv3x3dbl_3): BatchConv(
33  (conv): Conv2d(96, 96, kernel_size=(3, 3), stride=(1, 1),
    padding=(1, 1))
34  (bn): BatchNorm2d(96, eps=1e-05, momentum=0.1, affine=
   True, track_running_stats=True)
35  (relu): ReLU()
36  )
37  (parallel_dummyD): New_parallel_chain_dummy()
38  (pool): MaxPool2d(kernel_size=3, stride=1, padding=1,
   dilation=1, ceil_mode=False)
39  (pool1x1): BatchConv(
40  (conv): Conv2d(3, 32, kernel_size=(1, 1), stride=(1, 1))
41  (bn): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=
   True, track_running_stats=True)
42  (relu): ReLU()
43  )
44  (parallel_dummyE): New_parallel_chain_dummy()
45  (cat): Cat()
46  )
47  (1): MaxPool2d(kernel_size=2, stride=2, padding=0,
   dilation=1, ceil_mode=False)
48  (2): BatchConv(
49  (conv): Conv2d(256, 256, kernel_size=(2, 2), stride=(1,
   1))
50  (bn): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=
   True, track_running_stats=True)
51  (relu): ReLU()
52  )
53  (3): MaxPool2d(kernel_size=2, stride=2, padding=0,
   dilation=1, ceil_mode=False)
54  (4): BatchConv(
55  (conv): Conv2d(256, 256, kernel_size=(2, 2), stride=(1,
   1))
56  (bn): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=
   True, track_running_stats=True)
57  (relu): ReLU()
58  )
59  (5): MaxPool2d(kernel_size=2, stride=2, padding=0,
   dilation=1, ceil_mode=False)
```

```
60     (6): BatchConv(
61     (conv): Conv2d(256, 256, kernel_size=(2, 2), stride=(1,
       1))
62     (bn): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=
       True, track_running_stats=True)
63     (relu): ReLU()
64     )
65     (7): MaxPool2d(kernel_size=2, stride=2, padding=0,
       dilation=1, ceil_mode=False)
66     )
67     (classifiers): Sequential(
68     (0): Linear(in_features=256, out_features=256, bias=True)
69     (1): ReLU(inplace=True)
70     (2): Dropout(p=0.5, inplace=False)
71     (3): Linear(in_features=256, out_features=128, bias=True)
72     (4): ReLU(inplace=True)
73     (5): Dropout(p=0.5, inplace=False)
74     (6): Linear(in_features=128, out_features=43, bias=True)
75     )
76     )
77
```

Listing 5: Einfachere Version von Inception v3

Aufbau dieses Netzwerkes: 1. Inception-Modul 2. [pool1, batchConv1, pool2, batchConv2, pool3, batchConv3, pool4] 3. Drei Lineare Schichten mit ReLu und Dropout dazwischen

Im Unterschied zum offiziellen Inception Netz(v1v2v3) gibt es in dieser vereeinfachten Versionn keinen ßtemäus convs, es geht direkt mit InceptionA los.

Wie ähnlich sind sich InceptionA(hier) und das offizielle InceptionA-Modul?

```
1
2   [Linear(in_features=128, out_features=43, bias=True),
3   Dropout(p=0.5, inplace=False),
4   ReLU(inplace=True),
5   Linear(in_features=256, out_features=128, bias=True),
6   Dropout(p=0.5, inplace=False),
7   ReLU(inplace=True),
8   Linear(in_features=256, out_features=256, bias=True),
9   MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
     ceil_mode=False),
10  ReLU(),
11  BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
     track_running_stats=True), Conv2d(256, 256, kernel_size
     =(2, 2), stride=(1, 1)),
12  MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
     ceil_mode=False),
13  ReLU(),
14  BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
     track_running_stats=True), Conv2d(256, 256, kernel_size
```

```
 =(2, 2), stride=(1, 1)),
MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
 ceil_mode=False),
ReLU(), BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=
 True, track_running_stats=True),
Conv2d(256, 256, kernel_size=(2, 2), stride=(1, 1)),
MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
 ceil_mode=False),

[[Conv2d(3, 64, kernel_size=(1, 1), stride=(1, 1)),
BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
 track_running_stats=True), ReLU()],

[Conv2d(3, 48, kernel_size=(1, 1), stride=(1, 1)),
BatchNorm2d(48, eps=1e-05, momentum=0.1, affine=True,
 track_running_stats=True), ReLU(), Conv2d(48, 64,
 kernel_size=(5, 5), stride=(1, 1), padding=(2, 2)),
 BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
 track_running_stats=True), ReLU()],

[Conv2d(3, 64, kernel_size=(1, 1), stride=(1, 1)),
BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
 track_running_stats=True), ReLU(),
Conv2d(64, 96, kernel_size=(3, 3), stride=(1, 1), padding
 =(1, 1)),
BatchNorm2d(96, eps=1e-05, momentum=0.1, affine=True,
 track_running_stats=True), ReLU(),
Conv2d(96, 96, kernel_size=(3, 3), stride=(1, 1), padding
 =(1, 1)),
BatchNorm2d(96, eps=1e-05, momentum=0.1, affine=True,
 track_running_stats=True), ReLU()],

[MaxPool2d(kernel_size=3, stride=1, padding=1, dilation=1,
 ceil_mode=False), Conv2d(3, 32, kernel_size=(1, 1), stride
 =(1, 1)),
BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
 track_running_stats=True),
ReLU()],

[Cat()]]]

```

Listing 6: Reversed Model incv3

B Parameter für Training und Einlesen der Daten

Die in (Anders et al., 2020) gewählten Parameter wären ein guter Ausgangspunkt.

C Datensätze

D Programmcode

Literatur

Agueh, M. & Carlier, G. (2011). Barycenters in the wasserstein space. *SIAM Journal on Mathematical Analysis*, *43* (2), 904–924.

Anders, C. J., Marinč, T., Neumann, D., Samek, W., Müller, K.-R. & Lapuschkin, S. (2019). Analyzing imagenet with spectral relevance analysis: Towards imagenet un-hans' ed. *arXiv preprint arXiv:1912.11425*.

Anders, C. J., Weber, L., Neumann, D., Samek, W., Müller, K.-R. & Lapuschkin, S. (2020). Finding and removing clever hans: Using explanation methods to debug and improve deep models.

Anders, C. J., Weber, L., Neumann, D., Samek, W., Müller, K.-R. & Lapuschkin, S. (2019). *Finding and removing clever hans: Using explanation methods to debug and improve deep models.*

Bach, S., Binder, A., Montavon, G., Klauschen, F., Müller, K.-R. & Samek, W. (2015). On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one*, *10* (7), e0130140.

Binder, A., Bach, S., Montavon, G., Müller, K.-R. & Samek, W. (2016). Layer-wise relevance propagation for deep neural network architectures. In K. J. Kim & N. Joukov (Hrsg.), *Information science and applications (icisa) 2016* (S. 913–922). Singapore: Springer Singapore.

Böhle, M., Eitel, F., Weygandt, M. & Ritter, K. (2019). Layer-wise relevance propagation for explaining deep neural network decisions in mri-based alzheimer's disease classification. *Frontiers in aging neuroscience*, *11*, 194.

Chen, B., Carvalho, W., Baracaldo, N., Ludwig, H., Edwards, B., Lee, T., ... Srivastava, B. (2018). Detecting backdoor attacks on deep neural networks by activation clustering. *arXiv preprint arXiv:1811.03728*.

Computational optimal transport. (2019). *Foundations and Trends in Machine Learning*, *11* (5-6), 355–607.

Gu, T., Dolan-Gavitt, B. & Garg, S. (2017). Badnets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv preprint arXiv:1708.06733*.

Ioffe, S. & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning* (S. 448–456).

Lapuschkin, S., Wäldchen, S., Binder, A., Montavon, G., Samek, W. & Müller, K.-R. (2019). Unmasking clever hans predictors and assessing what machines really learn. *Nature communications*, *10* (1), 1–8.

Montavon, G., Binder, A., Lapuschkin, S., Samek, W. & Müller, K.-R. (2019). Layer-wise relevance propagation: an overview. *Explainable AI: interpreting, explaining and visualizing deep learning*, 193–209.

Montavon, G., Lapuschkin, S., Binder, A., Samek, W. & Müller, K.-R. (2017a). Explaining nonlinear classification decisions with deep taylor decomposition. *Pattern Recognition*, *65*, 211–222.

Montavon, G., Lapuschkin, S., Binder, A., Samek, W. & Müller, K.-R. (2017b). Explaining nonlinear classification decisions with deep taylor decomposition. *Pattern Recognition*, *65*, 211–222.

Peyré, G., Cuturi, M. & Solomon, J. (2016). Gromov-wasserstein averaging of kernel and distance matrices. In *International conference on machine learning* (S. 2664–2672).

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the ieee conference on computer vision and pattern recognition* (S. 1–9).

Tran, B., Li, J. & Madry, A. (2018). Spectral signatures in backdoor attacks. *arXiv preprint arXiv:1811.00636*.

Turner, A., Tsipras, D. & Madry, A. (2019). *Label-consistent backdoor attacks.*

Von Luxburg, U. (2007). A tutorial on spectral clustering. *Statistics and computing*, *17* (4), 395–416.