UNIVERSITÄT BIELEFELD

Technische Fakultät

Erstgutachter*in: Dr. André Artelt

Zweitgutachter*in: Paul Stahlhofen

Bachelorarbeit

# Effekt von Adversarials auf XAI Methoden

vorgelegt von:

| | |
|---|---|
| Student: | Jens Isernhinke |
| Studiengang: | Informatik/Grundlagen kognitiver Systeme |

Bielefeld, den 13.08.2025

## Inhaltsverzeichnis

# 1. Einleitung

Am Anfang war die Bildklassifikation sehr limitiert und konnte nur mit simplen Informationen umgehen. Durch viele rapide Fortschritte, wie beispielsweise Convolutional Neural Networks, wurde Bildklassifikation deutlich fortgeschrittener [1] und sie findet heute alltägliche Anwendungen, wie zum Beispiel die Gesichtserkennung zum Entsperren eines Mobiltelefons.
Convolutional Neural Networks (CNNs) sind eine beliebte Architektur um, unter anderem, Bilder zu klassifizieren [2]. Allerdings sind diese auch dadurch charakterisiert, dass diese oft durch ihre komplexe Architektur wie eine „Black Box“ funktionieren, da der Entscheidungsprozess schwer zu interpretieren ist. Um dies zu adressieren wurden XAI-Methoden wie CAM entwickelt, welche das Ziel haben, Features in der Eingabe zu identifizieren die die Klassifizierung maßgeblich beeinflussen [3].
Eine weitere Schwäche, welche in den meisten Neuronalen Netzen präsent ist, ist dass diese oft besonders stark durch Adversarials beeinflusst werden können, welche versuchen eine Eingabe zur Testzeit leicht zu verändern. Somit führt das Adversarial zu einer Fehlklassifikation, obwohl es von Menschen oft der gleichen Klasse wie die originale Eingabe klassifiziert werden würde [4]. Solche Adversarials sind oft mögliche Situationen, welche auch in Realität auftreten können, zum Beispiel für Perturbationen als Störungen oder speziell für Spatial Attack als ein gedrehtes Bild. Ein Beispiel ist in Abbildung 1 abgebildet.

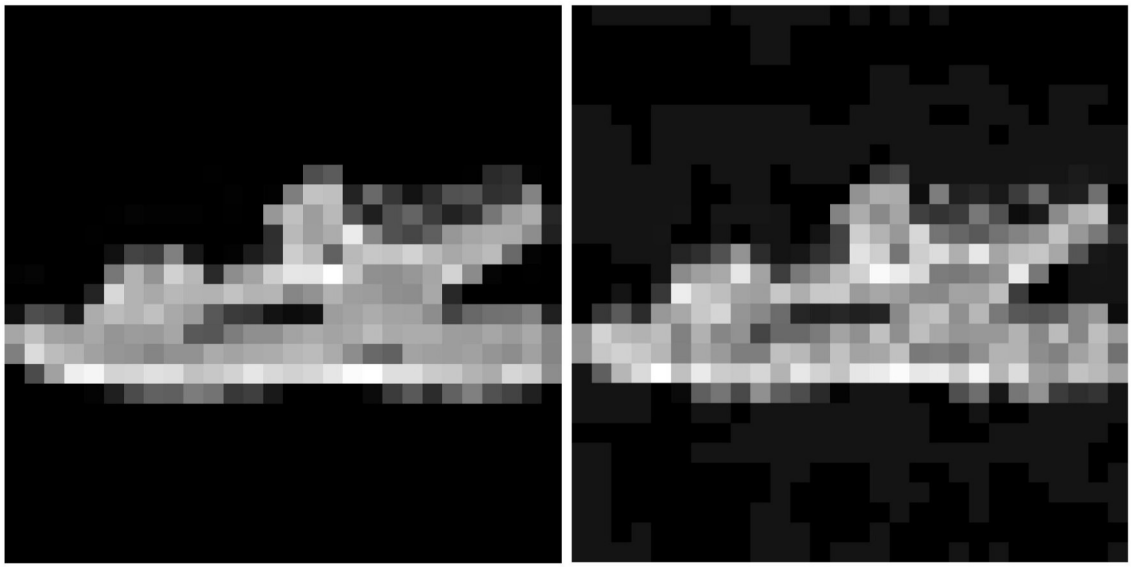

*Abbildung 1: Links: Das originale Bild, welches als Sandale klassifiziert wird; Rechts: Ein Adversarial, das als Shirt klassifiziert wird*

In der Realität existieren auch oft Störungen, welche zu einer Fehlklassifikation führen. In solchen Fällen ist es oft interessant, die Saliency Maps zu sehen, um zu wissen, worauf sich das Modell bei der Klassifikation fokussiert hat. Um die passende XAI-Methode für solche Fälle zu wählen, ist es oft wichtig zu wissen, wie stabil diese sind, d.h. wie stark die XAI-Methoden von den Störungen beeinflusst werden. Stabile Methoden sind XAI-Methoden, welche nicht stark beeinflusst werden, während sensitive/ Methoden stärker von Adversarials beeinflusst werden. Dafür werden hier XAI-Methoden kombiniert mit Adversarial-Methoden betrachtet und mit den Saliency Maps für ein unverändertes Bild zu vergleichen.

In dieser Bachelorarbeit wird versucht zu beantworten, wie stark XAI-Methoden allgemein durch Adversarial-Methoden beeinflusst werden. Danach außerdem auch beantwortet, ob und welche spezifische Kombinationen an XAI-Methoden und Adversarial-Methoden zu besonders starken/schwachen Beeinflussungen führen. Zuletzt wird versucht eine Antwort zu finden, ob eine direkte Korrelation zwischen den Änderungen der einzelnen Pixel durch Adversarials und den Änderungen der Saliency Maps durch dasselbe Adversarial existiert.

Hierfür wurde zuerst ein CNN trainiert, um den FashionMNIST-Datensatz zu klassifizieren. Danach werden paarweise Adversarials und deren Saliency Maps mit der originalen Eingaben und deren Saliency Maps verglichen und die Differenz dieser Methoden betrachtet.

Im ersten Teil dieser Bachelorarbeit werden zuerst die theoretischen Grundlagen der Methoden erklärt, die für diese Bachelorarbeit relevant sind. In dem folgenden Abschnitt wird näher erklärt wie genau die Ergebnisse generiert wurden. Im Ergebnisse und Diskussion Teil werden dann die Ergebnisse vorgestellt, diskutiert und Schlüsse aus diesen gezogen. Im letzten Fazit Teil werden dann die Erkenntnisse aus dem vorherigen Teil abschließend zusammengefasst.

Das zugehörige Repository zu dieser Bachelorarbeit kann unter [5] gefunden werden.

# 2. Theoretische Grundlagen

## 2.0 Allgemeine Methoden

### 2.0.1 Neuronale Netze

Ein neuronales Netz besteht aus künstlichen Neuronen, welche selbst eine vereinfachte Version des biologischen Neurons sind [6]. Ein künstliches Neuron kriegt n verschiedene Eingaben $x_n$ und hat genauso viele Gewichte $w_n$, mit welchen dann das Ergebnis der Gleichung $y_k = \varphi(\sum_{i=1}^{n} w_i x_i)$ berechnet wird, wobei $\varphi$ hier die Aktivierungsfunktion ist und k für das $k$-te Neuron steht. In einem neuronalen Netz bilden mehrere dieser künstlichen Neuronen eine Layer und werden oft auch Channels genannt. Ein neuronales Netz hat immer eine Input Layer, eine Output Layer und ein oder mehrere hidden Layers, wobei die Ergebnisse einer Layer als Eingabe für die darauf folgende Layer, falls vorhanden, dienen [6]. Die hier verwendeten Aktivierungsfunktionen sind die lineare Aktivierungsfunktion, die $\varphi(x) = x$ lautet, die rectified linear unit (ReLU), welche $\varphi(x) = max(0, x)$ lautet, und die Softmax-Funktion, welche $\varphi(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$ für jede Klasse $x_i$ berechnet [7]. Um diese Netzwerke zu optimieren müssen oft Gradienten für die einzelnen Neuronen berechnet werden. Für die Gradienten muss erst die Abweichung des Ergebnisses von dem echten Wert mittels einer Loss-Funktion berechnet werden [8]. Eine bekannte Loss-Funktion ist die Cross-Entropy-Funktion, welche für eine Klassifikation mit mehr als 2 Klassen den Loss mit $l = -\sum_j y^j \log(\sigma(o))^j$ berechnet, wobei $o$ die Ausgabe der letzten layer des neuronalen Netzes ist, $\sigma()^j$ die Wahrscheinlichkeitsschätzung für eine Klasse $j$ berechnet und $y^j$ das ware Label für Klasse $j$ bei One-hot-Kodierung des wahren Labels ist [9]. Danach wird für die Berechnung der Gradienten Backpropagation genutzt, welche zuerst die Ausgabe $y$ für eine Eingabe $X$ berechnet, dann mittels einer Loss-Funktion die Distanz zwischen der Ausgabe und der gewünschten Ausgabe berechnet, und schließlich mittels der Kettenregel die den Gradienten von $y$ für alle Neuronen berechnet 8].

### 2.0.2 Convolutional Neural Networks (CNNs)

Ein CNN besteht üblicherweise aus sieben verschiedenen Arten von Layers [2]. Die erste Art ist eine Convolutional Layer, welche üblicherweise eine zweidimensionale Matrix X als Eingabe kriegt und einen gleich-dimensionalen Filterkernel W auf diese anwendet. Diese Filterkernel werden

Schrittweise auf die Eingabe angewandt, indem das innere Produkt von W und der Submatrix von X berechnet wird. Dadurch wird die Relation von benachbarten Werten besser erfasst. Das Ergebnis heißt dann Feature Map [2].
Die zweite Art von Layer ist die Pooling Layer, welche dazu dient die Auflösung einer Eingabe zu reduzieren und gleichzeitig möglichst viele Informationen beizubehalten. Hierzu gibt es unter anderem Max Pooling, welches für eine Submatrix von X den größten Wert bestimmt und ausgibt, und Average Pooling, welches für eine Submatrix von X den durchschnittlichen Wert berechnet und diesen ausgibt. Pooling Layers werden oft nach mehreren Convolutional Layers verwendet, um die relevanten Features zu extrahieren und overfitting zu verhindern [2].
Die dritte Art ist eine Activation Layer, welche eine ReLU()-operation auf ihr Eingaben anwendet und dem Netzwerk ermöglicht, nicht-lineare Beziehungen zu lernen [2].
Die vierte Art ist die Batch normalization Layer, welche die Ausgaben der vorherigen Schicht mit dem Durchschnitt des Batches subtrahiert und dann durch die Standardabweichung des Batches teilt [2].
Die fünfte Art ist eine Dropout Layer, welche während des Trainings zufällig einen Anteil der vorherigen Schicht weglässt, um overfitting zu vermeiden [2].
Die sechste Art von Layer ist eine Fully-connected Layer, welche einer Layer mit einer linearen Aktivierungsfunktion entspricht. Diese wird oft am Ende eines CNNs zur Klassifizierung verwendet [2].
Als letzte Schicht gibt es dann oft noch eine Softmax Layer, welche die Ergebnisse der Klassifizierung der letzten fully-connected Layer, welche Logits genannt werden, zu Scores umwandelt, damit die Summe der Scores für alle Klassen 1 beträgt [2].

### 2.0.3 Die VGG Architektur

Die VGG Architektur [10] ist eine Architektur für CNNs, welche Convolutional Layers mit einem kleinen 3x3 Filter, einem Stride von 1 und einem Padding von 1, gefolgt von einer ReLU Layer nutzt. Ein Block besteht aus 2-3 Convolution-ReLU Layers, und nach einem Block gibt es eine Max Pooling Layer mit einem 2x2 Filter mit einem Stride von 2. Nach den Convolution-Blöcken gibt es eine oder mehrere Fully-connected Layers, wobei die letzte genauso viele Ausgaben hat wie es Klassen gibt. Als letztes gibt es eine Softmax Layer, welche die Ausgaben zu Wahrscheinlichkeiten transformiert.

### 2.0.4 Global Average Pooling

Global Average Pooling (GAP) [11] wandelt jede Feature Map der letzten Convolutional Layer eines CNNs zu einem einzigen Wert um, indem der Durchschnitt aller Werte berechnet wird. GAP wird oft anstelle von einer fully-connected layer genutzt, da dieses keiner Parameter hat und somit nicht overfitten kann. Außerdem ist GAP auch robuster gegenüber räumlicher Verschiebung, da räumliche Informationen durch die Durchschnittsberechnung verloren geht.

## 2.1 XAI-Methoden

Künstliche Intelligenz und Machine Learning haben in den letzten Jahren große Fortschritte gemacht und werden in vielen Bereichen eingesetzt, allerdings sind diese Modelle oft Blackbox-Modelle, weil sich die Entscheidungen des Modells nur sehr schwer nachvollziehen lassen. Da es oft Sorgen um unethische Nutzung und ungewollte Biases in diesen Modellen, besteht ein wachsendes Interesse an Explainable Artificial Intelligence (XAI).

XAI ist ein wieder aufkommender Forschungstrend, der Methoden umfasst, mit denen visualisiert werden kann, was ein Modell gelernt hat, als auch Methoden, welche individuelle Entscheidungen erklären.

Die folgenden XAI-Methoden haben das Ziel, Saliency Maps für CNNs zu generieren. Diese Saliency Maps sind oft Heatmaps, die die Werte in der originalen Eingabe X hervorheben, die relevant für die Klassifikation sind. Ein Beispiel einer solchen Saliency Map ist in Abbildung 2 dargestellt.

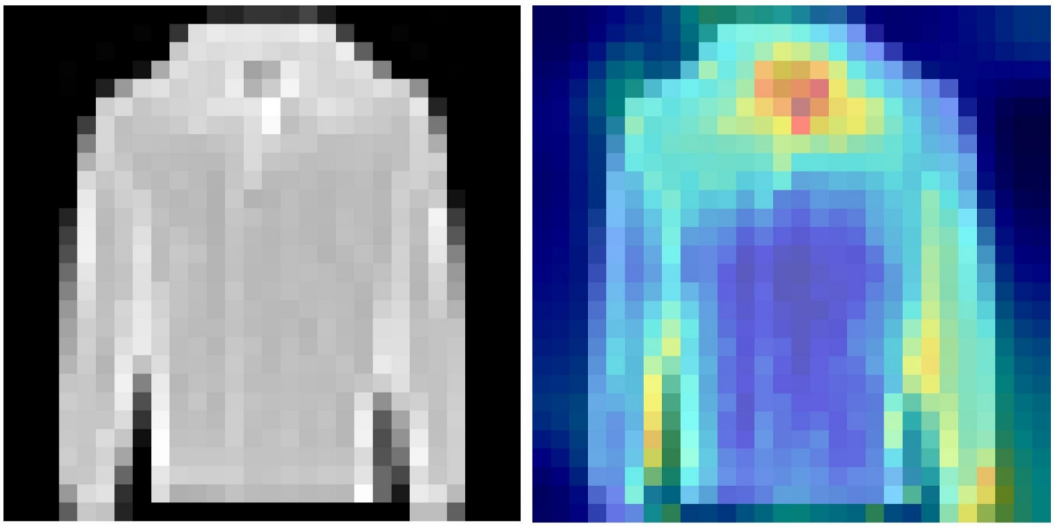

*Abbildung 2: Links: Ein originales Bild das als Coat gelabelt ist; Rechts: Die Heatmap der Saliency Map für das linke Bild, welche durch Grad-CAM erzeugt wurde*

### 2.1.1 Class Activation Map(CAM)

Die originale CAM-Methode generiert Saliency Maps, indem Feature Maps von der letzten Convolutional Layer mit Gewichten multipliziert und dann addiert werden. Die Gewichte entsprechen den Gewichten der letzten fully-connected Layer, welche die Eingabe in die Zielklasse Unterteilt. Die Gleichung für die Saliency Map lautet dann $L^c = \sum_{n=1}^{N} w_n^c A^n$, wobei $N$ die Anzahl an Feature Maps $A$ entspricht und $w_n^c$ dem Gewicht der $n$-ten Feature Map für die Klasse $c$ entspricht. Diese Methode benötigt zwei Komponenten: GAP, gefolgt von einer fully-connected Layer am Ende des CNNs [12].

### 2.1.2 Grad-CAM

Grad-CAM ist eine Erweiterung von CAM (Class Activation Mapping) und ermöglicht es die diskriminierenden Regionen eines CNNs zu identifizieren und visualisieren. Dafür wird die Saliency Map $L^c$ einer Eingabe X berechnet, wobei c die Klasse ist, für die die Grad-CAM berechnet werden soll. Dafür wird der Gradient von dem Score $y^c$ für jede Feature Map $A^k$ berechnet, also $\frac{\partial y^c}{\partial A^k}$. Daraus wird das Gewicht der Feature Map berechnet, indem der Durchschnitt aller aller Werte in $\frac{\partial y^c}{\partial A^k}$ berechnet wird, also $\alpha_k^c = \frac{1}{Z} * \sum_i \sum_j \frac{\partial y^c}{\partial A_{ij}^k}$, wobei Z der Anzahl an werten in $A^k$ entspricht. Dieses Gewicht gibt an, wie signifikant sich Feature Map k auf die Klassifizierung von X als c ausgewirkt an. Danach werden die Feature Maps mit ihren Gewichten multipliziert, aufsummiert und alle negativen Werte auf 0 gesetzt und man erhält die Saliency Map $L^c = ReLU(\sum_k a_k^c A^k)$. Die negativen Werte werden auf 0 gesetzt, da diese wahrscheinlich zu anderen Klassen gehören und somit irrelevant für die Grad-CAM von c sind. Grad-CAM nutzt meistens die letzte convolutional Layer, da tiefere Repräsentationen komplexere visuelle Konstrukte darstellen [13].

### 2.1.3 HiResCAM

HiResCAM adressiert die Limitierung von Grad-CAM, dass der Durchschnitt aller einzelnen Gradienten für das Gewicht genommen wird, wodurch jeder Wert in der Feature Map gleich skaliert wird, obwohl es stark unterschiedliche Gradienten in der gleichen Feature Map für jeden Wert gibt. HiResCAM löst dies, in dem es den Gradienten direkt mit der Feature Map multipliziert, wodurch

die Berechnung von $\alpha_k^c$ wegfällt und stattdessen $L^c = \sum_k \frac{\partial y^c}{\partial A^k} * A^k$ berechnet wird. Dadurch ist HiResCAM weniger verschwommen als Grad-CAM [14].

### 2.1.4 GradCAMElementWise

GradCAMElementWise ist die HiResCAM, allerdings wird die $ReLU()$-funktion auf die Saliency Map angewandt. Die Formel lautet dann $L^c = \sum_k ReLU(\frac{\partial y^c}{\partial A^k} * A^k)$ [15].

### 2.1.5 Grad-CAM++

Grad-CAM performt schlechter wenn es mehrere Instanzen der gleichen Klasse in einer Eingabe gibt. Außerdem umfasst die Heatmap oft nicht das ganze Objekt bei Bildern mit nur einem Objekt. Grad-CAM++ adressiert diese Probleme, indem für das Gewicht der zweite Gradient statt der erste genommen wird. Daraus resultiert die folgende Gleichung für das Gewicht: $\alpha_k^c = \sum_i \sum_j \frac{\frac{\partial^2 y^c}{(\partial A_{ij}^k)^2}}{2 * \frac{\partial^2 y^c}{(\partial A_{ij}^k)^2} + \sum_a \sum_b A_{ab}^k \{\frac{\partial^3 y^c}{(\partial A_{ij}^k)^3}\}} * ReLU(\frac{\partial y^c}{\partial A_{ij}^k})$. Die Saliency Map wird dann wie bei Grad-CAM durch $L^c = ReLU(\sum_k a_k^c A^k)$ berechnet [16].

### 2.1.6 Axiom-based Grad-CAM (XGrad-CAM)

XGrad-CAM ist eine Variation von Grad-CAM, welche bessere Saliency Maps bietet und mehr Verlässlichkeit hat. Dafür wird XGrad-CAM auf Basis von den Axiomen Konsistenz und Sensitivität hergeleitet. Sensitivität heißt hier, dass der Wert an einem Punkt in der Saliency Map äquivalent zu dem Betrag der Änderung in der Ausgabe ist, wenn der Wert in der Eingabe an dieser Stelle weggelassen wird. Konservierung heißt hier, dass jeder Wert in der Saliency Map hauptsächlich durch die Werte der Feature Maps bestimmt wird. Hier werden die Gewichte durch $\alpha_k^c = \sum_{i,j} (\frac{A_{ij}^k}{\sum_{x,y} A_{xy}^k} * \frac{\partial y^c}{\partial A_{ij}^k})$ berechnet. Die Saliency Map wird dann durch $L^c = \sum_k a_k^c A^k$ berechnet [17].

### 2.1.7 Ablation-CAM

Grad-CAM rechnet mit Gradienten der letzten Convolutional Layer, welche durch Backpropagation berechnet werden. Diese Methode besitzt aber das Problem der verringerten Gradienten, was daran liegt, dass spätere Layers stärker gewichtet werden und somit die Layers zwischen der letzten Convolutional Layer und der Output Layer die Gradienten verringern [18]. Deshalb werden oft nur Teile der relevanten Region hervorgehoben werden. Ablation-CAM adressiert dieses Problem, indem für das Gewicht keine Gradienten genutzt werden und stattdessen $\alpha_k^c = \frac{y^c - y_k^c}{y^c}$ berechnet wird, wobei $y_k^c$ der finale Score für Klasse c ist wenn Feature Map $A^k$ weggelassen wird. Die Saliency Map wird dann wie bei Grad-CAM durch $L^c = ReLU\left(\sum_k a_k^c A^k\right)$ berechnet [19].

### 2.1.8 Score-CAM

Score-CAM adressiert das gleiche Problem wie Ablation-CAM und nutzt auch keine Gradienten für das Gewicht; stattdessen wird $\alpha_k^c$ durch $\alpha_k^c = f(X \circ H^k) - f(X_b)$ berechnet, wobei $f(X) = Y$ für eine Eingabe $X$ mit einem CNN einen Wert $Y$ berechnet, $X_b$ eine Baseline-Eingabe ist, $\circ$ das elementweise Produkt zweier Matrizen ist und $H^k = s(Up(A^k))$. $Up()$ skaliert die Feature Map $A^k$ auf die gleiche Größe wie die Eingabe $X$ hoch und $s()$ normalisiert das Ergebnis zu Werten zwischen 0 und 1. Die Saliency Map wird dann durch $L^c = ReLU\left(\sum_k a_k^c A^k\right)$ berechnet [20].

### 2.1.9 Eigen-CAM

Grad-CAM und Grad-CAM++ nutzen Backpropagation um die Gradienten zu berechnen, was zu zusätzlichen Rechenaufwand führt und sich darauf verlässt, dass die Eingabe $X$ richtig qualifiziert wurde, da die Saliency Map sonst stark verzerrt wird. Eigen-CAM adressiert dies indem es stattdessen die Singulärwertzerlegung berechnet und dann die Saliency Map mit der ersten Principal Component berechnet. Sei $A$ eine Matrix mit aller Feature Maps für Eingabe $X$, wobei jede Spalte einer Feature Map entspricht und jede Zeile einem Wert aus der Feature Map entspricht; dann ist die Singulärwertzerlegung $A = U\Sigma V^T$. Die Saliency Map wird dann durch den ersten Eigenvektor $V_1$ von $V$ mit $L = A * V_1$ berechnet [21].

### 2.1.10 EigenGradCAM

EigenGradCAM ist ähnlich zu Eigen-CAM, wobei der Unterschied darin liegt, dass jeder Wert aus der Feature-Map-Matrix mit den Gradienten multipliziert werden, wodurch zwischen Klassen

diskriminiert wird und die Saliency Map schärfer ist. Die Gleichung für jeden Wert der neuen Feature-Map-Matrix $A^c$ lautet dann $A_{ij}^c = \frac{\partial y^c}{\partial A_{ij}} * A_{ij}$ für die Klasse $c$. Die finale Saliency Map wird dann durch $L^c = A^c * V_1$ berechnet [15].

### 2.1.11 LayerCAM

Grad-CAM und Grad-CAM++ nutzen nur ein skalares Gewicht für eine gesamte Feature Map, wodurch die Saliency Map oft verschwommen ist und auf höheren Layers mehr falsch-positive Werte hat. LayerCAM adressiert dies, indem für jeden Wert für die $k$-te Feature Map $\hat{A}_{ij}^k = ReLU(\frac{\partial y^c}{\partial A_{ij}^k} * A_{ij}^k)$ berechnet wird. Die Saliency Map wird dann berechnet durch $L^c = ReLU(\sum_k \hat{A}^k)$ [22]. Dies ist sehr ähnlich zu HiResCAM, wobei der Unterschied darin liegt, dass hier $ReLU()$ sowohl auf die Gradienten als auch die Saliency Map angewandt wird. GradCAMElementWise und LayerCAM sind äquivalent zueinander, da beide Methoden auf die Ergebnisse jeder Multiplikation von einer Feature Map und dessen Gradienten $ReLU()$ anwenden, und da das Ergebnis einer $ReLU()$-Funktion immer positiv ist, ist die Summe somit auch immer positiv.

### 2.1.12 FullGrad

Saliency Maps haben das inhärente Problem, dass sie nicht lokale und globale Attribution gleichzeitig besitzen können. Lokale Attribution bedeutet, dass ein Feature der Eingabe $X$ von der Saliency Map als wichtig angesehen muss, wenn das verändern dieser Eigenschaft die Ausgabe stark beeinflusst. Globale Attribution bedeutet, dass die Saliency Map die Ausgabe vollständig erklärt, also dass die Scores der einzelnen Features addiert die Ausgabe des CNNs ergeben. FullGrad besitzt sowohl lokale als auch globale Attribution und approximiert die Saliency Map des gesamten CNNs durch $L = Up(\frac{\partial y}{\partial X} \circ X) + \sum_{l \in L} \sum_{c \in c_l} Up(\frac{\partial y}{\partial b_c} \circ b_c)$, wobei $y$ die Ausgabe des CNNs, $X$ die Eingabe und $L$ die einzelnen Layers des CNNs sind. Zusätzlich ist $c$ einer der Channels $c_l$ der Layer $l$ und $b_c$ sind die Biases dieses Channels [23].

### 2.1.13 Finer-CAM

CAM-Methoden haben oft Probleme damit, die diskriminierendsten Regionen hervorzuheben, da diese oft in kleinen Details sind. Finer-CAM adressiert dies, indem es die Gewichte

$\alpha_k^{c,d} = \frac{1}{Z} \sum_i \sum_j \frac{\partial (y^c - \gamma * y^d)}{\partial A_{ij}^k}$, wobei $y^d$ der Score für eine ähnliche Klasse $d$ ist und $\gamma$ ein Faktor ist, der angibt wie ähnlich $c$ und $d$ sind. Die Saliency Map wird dann durch $L^c = ReLU(\frac{1}{T} \sum_t \sum_k a_k^{c,t} A^k)$ berechnet, wobei $T$ die Anzahl an verglichenen Klassen ist [24].

### 2.1.14 KPCA-CAM

Eigen-CAM ist dadurch limitiert, dass es nicht mögliche nicht-lineare Daten im Datensatz berücksichtigt. KPCA-CAM löst dies, indem es Kernel Principal Component Analysis verwendet(KPCA). Hierfür wird erst ein Kernel auf die Eingabe angewandt, zum Beispiel der Radial-Basisfunktion-Kernel $K(x_i, x_j) = \exp(-\gamma * \|x_i - x_j\|^2)$, wobei $\gamma$ hier die Verteilung des Kernels bestimmt, oder der sigmoidale Kernel $K(x_i, x_j) = \tanh(\gamma * x_i x_j + r)$, wobei r den Offset der Kurve bestimmt, und $\gamma$ die Steigung der tanh-funktion bestimmt. Danach wird PCA angewandt und $K = V \Lambda V^{-1}$ berechnet, wobei $\Lambda$ eine diagonale Matrix mit den Eigenwerten von $K$ ist. Danach wird, wie bei Eigen-CAM, die Saliency Map mit dem ersten Eigenvektor $V_1$ aus $V$ durch $L = K * V_1$ berechnet [3].

### 2.1.15 FEM

FEM ist eine weitere Methode zur Saliency Map einer Entscheidung eines CNNs, welche keine Gradienten benötigt. Statt dessen wird für jede Feature Map $A^{k,c}$, wobei $c$ der Channel ist, eine Bitmap $B_{ij}^{k,c} = \begin{cases} 1 & \text{, wenn } A_{ij}^{k,c} \geq \mu^c + K * \sigma^c \\ 0 & \text{, sonst} \end{cases}$ berechnet. $\mu^c$ ist hier der Durchschnittswert aller Werte aller Feature Maps für einen Channel $c$, $\sigma^c$ ist der Durchschnittswerte aller Feature Maps für das gleiche $c$ und $K$ die K-Sigma-Regel. Dann wird die Saliency Map durch $L = \sum_k \sum_c B^{k,c}$ berechnet und auf [0,1] normalisiert. Als letztes wird diese Saliency Map dann auf die gleiche Dimensionalität wie die Eingabe hoch skaliert [25].

### 2.1.16 ShapleyCAM

ShapleyCAM ist eine Erweiterung von Grad-CAM, welche sowohl Gradienten als auch hesse Matrizen nutzt um die Gewichte zu berechnen. Außerdem nutzt diese Methode auch eine Residual Softmax Target-Class utility function, welche sowohl pre-softmax als auch post-softmax Scores vereint um die Nachteile von vanishing gradients auszugleichen und den Fokus auf die Zielklasse beizubehalten. Dafür wird erst $y_{ReST} = y^c + \ln(softmax(y)^c)$ berechnet, wobei $softmax(y^c)$ der

Softmax von $y$ für die Klasse $c$ ist. Danach wird jede Feature Map $A^k$ zu $A$ konkateniert und es wird der Gradient durch $y' = \frac{\partial y_{ReST}}{\partial A}$ und die Hesse Matrix durch $H = \frac{\partial^2 y_{ReST}}{\partial A^2}$ berechnet. Anschließend wird dann jeder Gewichtmatrix durch $W^k = [y' - \frac{1}{2} * A^T * H]^k$ berechnet, wobei jedes $W^k$ einer Feature Map $A^k$ entspricht. Als letztes werden die Gewichte $\alpha_k^c = \frac{1}{n*m} \sum_n \sum_m W_{n,m}^k$ berechnet und daraus die Saliency Map $L^c = ReLU(\sum_k \alpha_k^c * A^k)$ [26].

### 2.2 Adversarial-Methoden

Adversarial-Methoden sind Methoden, welche einer Eingabe Störungen hinzufügen und somit Adversarials zu finden, welche das Ziel haben, eine Fehlklassifikation oder eine niedrigere Accuracy herbeizuführen. Das Ziel eines Adversarials ist es, die Ausgabe des Modells zu beeinträchtigen, für Menschen aber praktisch identisch zum Original zu bleiben [27]. Solche Angriffe haben eine praktische Signifikanz für kritische Anwendungen wie in der Medizin zur Diagnose [27] oder für autonomes Fahren zur Erkennung von Fußgängern. Da in der Realität auch oft Störungen stattfinden, werden Adversarial Attacks genutzt, um die Robustheit eines Modell gegenüber solcher Störungen zu evaluieren [28]. In Abbildung 1 wird solch ein Angriff beispielsweise dargestellt.

#### 2.2.1 Fast Gradient Sign Method(FGSM)

FGSM ist eine Methode um Adversarials zu generieren. Dafür nutzt sie die Funktion $X' = X + \epsilon * sign(\nabla_X J(X, y_{true}))$, wobei $sign()$ die Vorzeichenfunktion ist, $J()$ die Loss-Funktion ist und $\epsilon$ ein Hyperparameter ist. Diese Gleichung erfüllt automatisch den $L_\infty$-Constraint, also dass $\|X - X'\|_\infty \leq \epsilon$ [27].

#### 2.2.2 Basic Iterative Method(BIM)

BIM ist eine Erweiterung von FGSM, welche iterativ FGSM mit einer kleinen Schrittgröße anwendet und jedes mal das Ergebnis clippt damit die $L_\infty$-Constraint erfüllt wird. Als erstes wird $X'_0 = X$ initialisiert und dann iterativ $X'_{n+1} = Clip_{X,\epsilon}\{X'_n + \alpha * sign(\nabla_X J(X'_n, y_{true}))\}$ für eine gewünschte Anzahl an Iterationen berechnet. $\alpha$ ist hier wieder ein Hyperparameter, der meistens kleiner als $\epsilon$ gewählt wird, und $Clip_{X,\epsilon}\{\}$ ist eine Funktion, welche jeden Wert von $X'_{n+1}$ so clippt, dass $-\epsilon \leq \|X - X'_{n+1}\|_\infty \leq \epsilon$ gilt [4].

### 2.2.3 Projected Gradient Descent(PGD)

PGD ist eine Erweiterung von BIM, welche zusätzlich $X'_0$ zufällig im $\epsilon$-Ball um $X$ initialisiert und dann das Ergebnis nach jeder Iteration wieder in den $\epsilon$-Ball um $X$ projiziert. Also wird $X'_0 = X + \delta$ initialisiert, wobei $\delta$ die gleiche Dimensionalität hat wie $X$ und jeder Wert $-\epsilon \leq \delta_{ij} \leq \epsilon$ erfüllt. Danach wird iterativ $X'_{n+1} = \Pi_{B(X,\epsilon)}\{X'_n + \alpha * sign(\nabla_X J(X'_n, y_{true}))\}$ berechnet. $\Pi_{B(X,\epsilon)}$ ist hier die Projektion in den $\epsilon$-Ball $B(X, \epsilon)$. Optional werden die Werte am Ende noch geclippt, damit das Adversarial innerhalb des Wertebereichs von $X$ bleiben [29].

### 2.2.4 DeepFool

DeepFool ist ein weiterer Algorithmus, der Effizient Adversarials findet. Der folgende Algorithmus ist DeepFool mit dem $L_\infty$-Constraint. Dafür werden am Anfang $X_0 = X$ und $i = 0$ initialisiert. Es gilt $\hat{k}(X) = argmax_k(f_k(X))$, wobei $f_k(x)$ der Score für die $k$-te Klasse für eine Eingabe $X$ ist. Danach werden iterativ Perturbationen berechnet bis $\hat{k}(X_i) \neq \hat{k}(X)$ gilt. Dafür werden in jeder Iteration erst $\omega'_k = \nabla_x f_K(X_i) - \nabla_x f_{\hat{k}(X)}(X_i)$ und $f'_k = f_K(X_i) - f_{\hat{k}(X)}(X_i)$ für alle Klassen $k \neq \hat{k}(X)$ berechnet. Dann wird $\hat{l} = argmin_{k \neq \hat{k}(X)} \frac{|f'_k|}{\|\omega'_k\|_1}$ und $r_i = \frac{|f'_{\hat{l}}|}{\|\omega'_{\hat{l}}\|_1} * sign(\omega'_{\hat{l}})$ berechnet. Am Ende jeder Iteration wird dann $x_{i+1} = x_i + r_i$ und $i = i + 1$ berechnet. Die finale Perturbation ist dann $\hat{r} = \sum_i r_i$ und das Adversarial wird dann durch $X' = Clip_{bounds}(X + \hat{r})$ berechnet, wobei *bounds* die Grenzwerte für $X$ sind, e.g. [0,1] für Grauwerte [30].

### 2.2.5 Carlini & Wagner Angriff

Angriffe wie DeepFool und FGSM sind sehr instabil und können öfters dabei Versagen, ein Adversarial zu finden, obwohl welche existieren. Dadurch können diese Angriffe durch Verteidigungen wie defensive Distillation abgewehrt werden. Der Carlini & Wagner Angriff [30] ist stabiler und kann auch trotz solcher Verteidigungen ein Adversarial finden. Dafür optimiert der Angriff die $L_2$-Constrainte Gleichung $min_w \|\frac{1}{2}(\tanh(w)+1) - x\|_2^2 + c * f(\frac{1}{2}(\tanh(w)+1))$ mit $f(x') = max(max\{Z(x')_i : i \neq t\} - Z(x')_t, -\kappa)$. $t \neq \hat{k}(X)$ ist hier die Klasse als die das Adversarial klassifiziert werden soll, $c$ und $\kappa$ sind sind hier Hyperparameter, wobei $\kappa$ angibt wie sicher die Klassifikation als Klasse $t$ sein soll, und $Z(X)$ sind die Logits des Modells für eine Eingabe X,

wobei die Logits die Werte vor der finalen Softmax-Operation sind. Das Adversarial ist dann $X' = \frac{1}{2} * (\tanh(w) + 1)$.

### 2.2.6 Elastic-Net Angriff (EAD)

Der Elastic-Net Angriff ist eine von dem Carlini & Wagner Angriff inspirierte Adversarial Methode, welche robuster ist und eine höhere Transfereierfähigkeit hat. EAD nutzt die gleiche Funktion $f(x')$ wie Carlini & Wagner, wandelt diese aber leicht um, da EAD untargeted ist, also wird kein Adversarial für eine bestimmte Klasse generiert und das Ziel ist lediglich, dass $X'$ anders klassifiziert wird als $X$. Die Gleichung lautet dann $f(x') = max(Z(x')_{\hat{k}(X)} - max\{Z(x')_j : j \neq \hat{k}(X)\}, -\kappa)$.

Dann gilt $min_x c * f(x, X') + \beta \|X' - X\|_1 + \|X' - X\|_2^2$, wobei die Gleichung constrained ist durch $X \in [0,1]^p$. Hier ist $p$ die Anzahl an Werten in $X$, und $\beta$ und $c$ sind Hyperparameter. EAD ist sowohl $L_1$-Constraint als auch $L_2$-Constraint. Das Ergebnis ist dann der resultierende Wert $X'$ aus der Minimierung [31].

### 2.2.7 Spatial Attack

Die bisherig genannt Methoden generieren Adversarials, indem sie eine Perturbation zu der originalen Eingabe addieren, sodass X für Menschen nahezu identisch aussieht, bei dem Modell aber zu einer Fehlklassifikation führt. Spatial Attack hingegen generiert Adversarials, indem X rotiert wird. Sei $X_{ij} = (u, v)$ die Position eines Pixels und der Pixel $(0,0)$ in der Mitte von der Eingabe $X$. Dann ist die neue Position eines Pixels nach einer Rotation gegeben durch $T(X_{ij}, \delta u, \delta v, \theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} * \begin{bmatrix} u \\ v \end{bmatrix} + \begin{bmatrix} \delta u \\ \delta v \end{bmatrix}$. Das Adversarial wird dann gefunden, indem die Gleichung $max_{\delta u, \delta v, \theta} L(X', y)$ optimiert wird, wobei $X'_{ij} = T(X_{ij}, \delta u, \delta v, \theta)$ gilt, $y$ das korrekte Label ist und $L()$ die Loss-Funktion des neuronalen Netzes ist. Das optimale $X'$ ist dann das Adversarial [32].

## 3. Experimente

Das Ziel dieser Bachelorarbeit ist es, die Auswirkung von Adversarial-Methoden auf XAI-Methoden zu untersuchen. Hierfür wurde zuerst ein Bildklassifikator trainiert. Danach

wurde auf alle Ergebnisse aller Adversarial-Methoden jede XAI-Methode angewandt. Im nachfolgenden wird das vorgehen für beide Teile genauer beschrieben.

### 3.1 Aufbau des Modells

Das hier verwendete Modell ist auf der TinyVGG Architektur [30] basiert und wurde dann mittels iteratives Testen für den FashionMNIST-Datensatz angepasst.

Das Modell nutzt statt 3 statt den originalen 2 Convolution-Blöcke. Die ersten beiden Blöcke haben zwei Convolution-ReLU-Batchnorm-Kombinationen, gefolgt von einer Max Pooling Layer und einer Dropout Layer. Der dritte Block besitzt eine weitere Convolution-ReLU-Batchnorm-Kombinationen. Die Convolutional Layers haben alle einen 3x3 Filter mit einem Stride und Filter von 1 und jeweils 128 Channels. Die Max Pooling Layers besitzen 2x2 Filter mit einem Stride von 2, die Batch normalization Layers haben 128 Channels und die Dropout Layers besitzen eine Dropout-Wahrscheinlichkeit von 50%. Nach den 3 Blöcken folgt eine Global Average Pooling Layer, welche jede Feature Map jeweils zu einem einzigen Wert zusammenfasst, und danach eine Flatten Layer, welche ihre Eingabe in einen eindimensionalen Vektor umwandelt. Als letztes folgt eine fully-connected Layer mit 128 Eingaben und 10 Ausgaben, eine für jede Klasse. Das Netzwerk besitzt keine finale Softmax Layer, da die Logits für einige Methoden benötigt werden, und Softmax auch außerhalb des Netzwerkes angewandt werden kann, da Softmax keine Parameter besitzt.

Die Dropout Layers und Batch normalization Layers wurden implementiert um overfitting zu vermeiden. Zusätzlich wurde early Stopping implementiert, damit das Training gestoppt wird, falls overfitting erkannt wird.

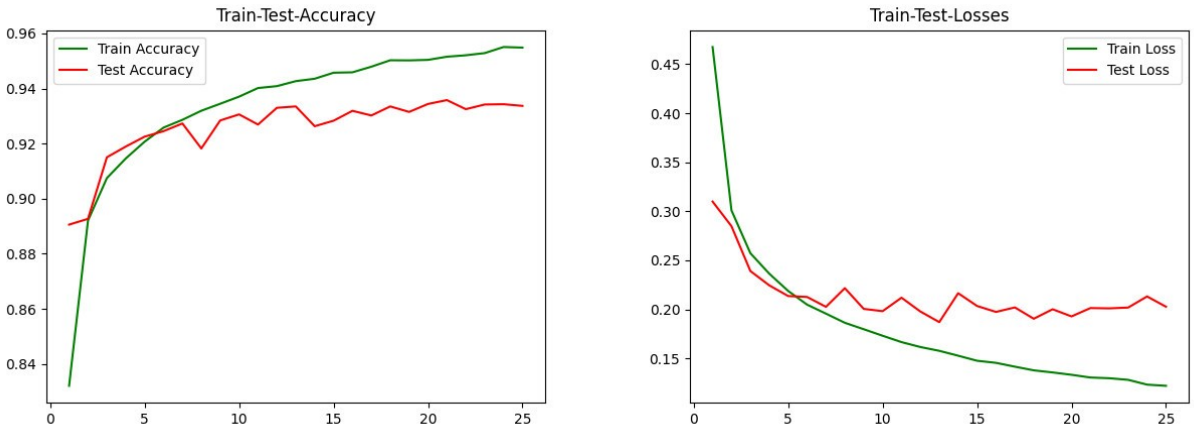


*Abbildung 3: Links sind sie Accuracies des Modells abgebildet; Rechts sind sind die Losses abgebildet*

Das Modell wurde auf dem FashionMNIST-Datensatz [34] trainiert und das Training wurde durch early Stopping nach 25 Epochen abgebrochen. Es konnte eine Accuracy von 93.371% auf dem Testdatensatz erreichen. In Abbildung 3 werden links die Verläufe der Accuracies von dem Train-Datensatz als auch dem Test-Datensatz dargestellt. Rechts werden die Verläufe der Losses von sowohl dem Train-Datensatz als auch den Test-Datensatz dargestellt.

### 3.2 Generierung der Daten

Es wurden 3 zufällige Klassen gewählt, und für jede Klasse wurden zufällig 10 Bilder gewählt, die von dem Modell richtig klassifiziert werden.

Die Adversarials wurden für jedes Bild mit allen Adversarial-Methoden aus 2.2 generiert. Hierbei wurde die maximale Perturbation $\varepsilon$ nach der gegebenen Norm, wo anwendbar, iterativ minimiert, damit eine minimale Perturbation auf einem 0.01 Niveau erreicht wurde. Die Suche wurde teils beschleunigt, indem $\varepsilon$ erst stark erhöht wurde und dann in kleiner werdenden Schritten das Minimum gefunden wurde.

Die Saliency Maps wurden generiert, indem jede XAI-Methode aus 2.1 auf jedes originale Bild und dessen Adversarials angewandt wurde. Außerdem wurde, wo nötig, immer die Klasse, als die das Modell das jeweilige Bild oder Adversarial klassifizieren würde, als Klasse für die Saliency Map genutzt. Alle Methoden haben, falls nötig, alle Convolutional Layers genutzt. Hierbei haben sie für jede Layer die Saliency Map berechnet und dann für jeden Wert den Durchschnitt zwischen den Saliency Maps berechnet. Anschließend stellen diese in der finale Saliency Map dar.

Zusätzlich wurde für jede Kombination an Adversarial-Methoden und XAI-Methoden der Unterschied zwischen der Saliency Map der originalen Eingabe und der Saliency Map des Adversarials berechnet und als Boxplot dargestellt. Außerdem wurde der sowohl prozentuale Anteil an geänderten Werten als auch die Summe der absoluten Werte berechnet. Schließlich wurde die Pearson Korrelation zwischen der Änderung der Saliency Map und der Perturbation der Adversarials berechnet und es wurde die Nullhypothese auf einem Signifikanzniveau von 5% getestet. Die XAI-Methode Deep Feature Factorization wurde exkludiert, da diese nur signifikante Features findet und keine Heatmap generiert.

Letzlich wurden die Differenzen zwischen den XAI-Methoden einmal als Grauwerte und einmal als Heatmap dargestellt.

# 4. Ergebnisse und Diskussion

## 4.1 Vergleich der absoluten Summe der Differenzen der XAI-Methoden

Als erstes wurden die Ergebnisse verglichen, indem die absoluten Differenzen zwischen den Saliency Maps der Adversarials und den originalen Bildern betrachtet wurde. Die Sortierung der Werte folgt erfolgt dafür in aufsteigender Reihenfolge, woraufhin sie dann für jede Adversarial-Methode und jede XAI-Methode binär kodiert werden, um zu sehen ob der Wert diese nutzt. Danach werden diese binären Werte dann jeweils Zeilenweise aufsummiert und die Zwischensumme als Graph dargestellt. Somit befinden sich in den Bereichen, in denen die Steigung stärker ist, mehr Saliency Maps. Außerdem unterscheiden sich Saliency Maps am Anfang des Graphen weniger von der Saliency Map des originalen Bildes, während Saliency Maps am Ende eine höhere Differenz haben. Die Tabelle kann in result_data in [5] gefunden werden.

Der beschriebene Prozess kann wie folgt mathematisch formuliert werden: Sei $d_i^{k_i,l_i,s_i}$ die $i$-te Summe der absoluten Differenz mit der XAI-Methode $k_i \in K$, der Adversarial-Methode $j_i \in J$ und dem Sample $s_i \in S$. Hier gilt $K := \{1,2,\dots,15\}$, was jede XAI-Methode enkodiert, $J := \{1,2,\dots,7\}$, was jede Adversarial-Methode enkodiert, und $S := \{1,2,\dots,30\}$, was jedes Sample enkodiert. Außerdem gilt dann auch $i \in \{1,2,\dots,|K|*|J|*|S|\}$ und $d_1^{k_1,l_1,s_1} \le d_2^{k_2,l_2,s_2} \le \dots \le d_n^{k_n,l_n,s_n}$. Dann wird die Zwischensumme einer Reihe $x$ für eine XAI-Methode $k_y$ berechnet durch $f_{k_y}(x) = \sum_{i=1}^{x} \sigma_{XAI}(d_i^{k_i,j_i,s_i}, k_y)$, wobei

$$\sigma_{XAI}(c^{k,j,s}, k_y) = \begin{cases} 1 & \text{, wenn } k = k_y, \\ 0 & \text{, sonst} \end{cases}$$

und die Zwischensumme einer Adversarial-Methode $j_y$ wird dann berechnet durch $f_{j_y}(x) = \sum_{i=1}^{x} \sigma_{Adv}(d_i^{k_i,j_i,s_i}, j_y)$, wobei

$$\sigma_{Adv}(c^{k,j,s}, j_y) = \begin{cases} 1 & \text{, wenn } j = j_y. \\ 0 & \text{, sonst} \end{cases}$$

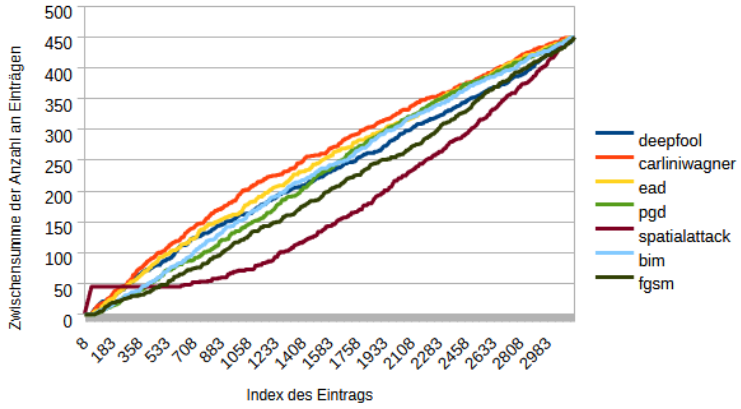


*Abbildung 4: Ein Graph, welcher für alle Adversarial-Methoden die Anzahl der absteigend sortierten Differenzen zwischen einem Punkt und dem Anfang der Tabelle darstellt. Der Index des Eintrags gibt den Index in der aufsteigend nach der absoluten Differenz sortierten Tabelle results_data an, und die Zwischensumme der Anzahl der Einträge gibt an, wie oft die jeweilige Methode bis zu einem Index vorgekommen ist. Je steiler der Graph in einem Intervall ist, desto mehr Differenzen befinden sich in dem Intervall, und je niedriger der Indexe der Einträge für den Interval sind, desto niedriger sind die Differenzen.*

In Abbildung 4 werden diese Graphen für die Adversarial-Methoden dargestellt und es ist erkennbar, dass sich alle Methoden bis auf Spatial Attack annähernd linear verhalten. Das bedeutet, dass diese Methoden gleich häufig Saliency Maps mit hohen als auch niedrigen Differenzen erzeugen und die Stabilitäten der XAI-Methoden somit nur schwach abhängig davon sind, welche dieser Methoden für die Saliency Maps genutzt werden. Dies kann möglicherweise dadurch erklärt werden, dass für alle Bilder immer versucht wird eine minimale Perturbation hinzuzufügen, indem das Epsilon minimiert wird. Dadurch dass die Bilder relativ simpel sind, da sie nur ein Objekt beinhalten, nähern sich alle dieser Methoden wahrscheinlich dem gleichen oder einem ähnlichen Optimum an. Außerdem sind PGD und BIM Erweiterungen von FGSM, welche alle auf der Gleichung basieren und somit auch ähnliche Adversarials liefern.

Spatial Attack nutzt, anders wie der Rest der Adversarial Methoden, keine Perturbation und rotiert stattdessen die Eingabe, um ein Adversarial zu erzeugen. Dies führt besonders in Datensätzen mit niedriger Auflösung, wie FashionMNIST, dazu, dass es vergleichsweise wenig mögliche Adversarials gibt und manchmal keines gefunden werden kann. Dies ist der Fall für drei Bilder die hier betrachtet wurden, weshalb diese Einträge in der Tabelle eine Differenz von 0 haben und zu dem Sprung am Anfang des Graphen führen. Danach folgt der Graph ungefähr einer exponentiellen Kurve, was indiziert, dass Spatial Attack größere Unterschiede in der Saliency Map verursacht. Dies kann dadurch erklärt werden, dass sich das Netzwerk auf die gleichen Features fokussiert, aber da sich diese an anderen Orten im Bild befinden, ist die Differenz groß. Dies wird nochmal in Abbildung 5 an einem Beispiel veranschaulicht.

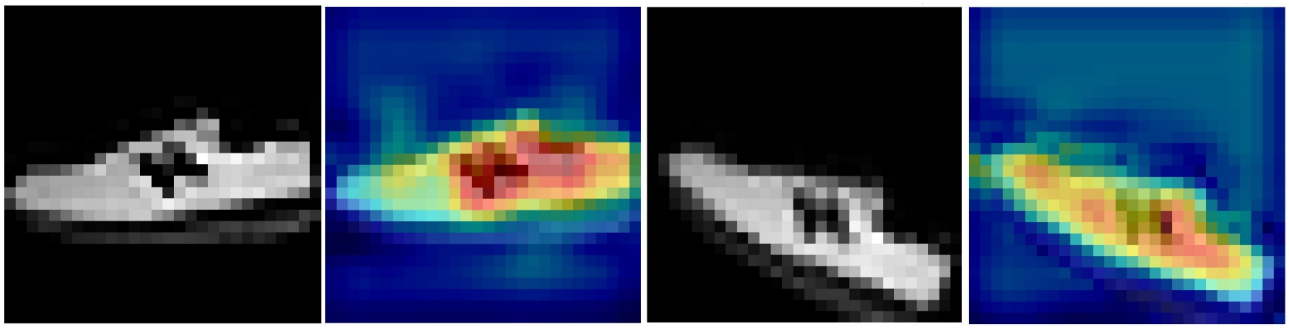

*Abbildung 5: Links: Ein Bild das korrekt als Sneaker klassifiziert wird; Mitte Links: Die Saliency Map von Eigen-CAM von dem originalen Bild; Mitte Rechts: Ein Adversarial des linken Bildes, das durch Spatial Atttack generiert wurde; Rechts: Die Saliency Map von Eigen-CAM für Spatial Attack auf dem linken Bild*

In Abbildung 6 ist erkennbar, dass Grad-CAM leicht sigmoidal verläuft, was indiziert, dass Grad-CAM häufiger mittlere Differenzen besitzt. HiResCAM verläuft anfangs konstant und danach linear. Allerdings ist die Gerade hier steiler und konvergiert somit früher, wonach sie eher logarithmisch verläuft. Das bedeutet, dass bei HiResCAM die Differenzen eher klein sind. GradCAMElementWise und LayerCAM verlaufen wie HiResCAM, nur extremer, sodass sich diese für nahezu der Hälfte des Graphen konstant verhalten. Auch fällt auf, dass sich, wie erwartet, GradCAMElementWise und LayerCAM exakt gleich verhalten.

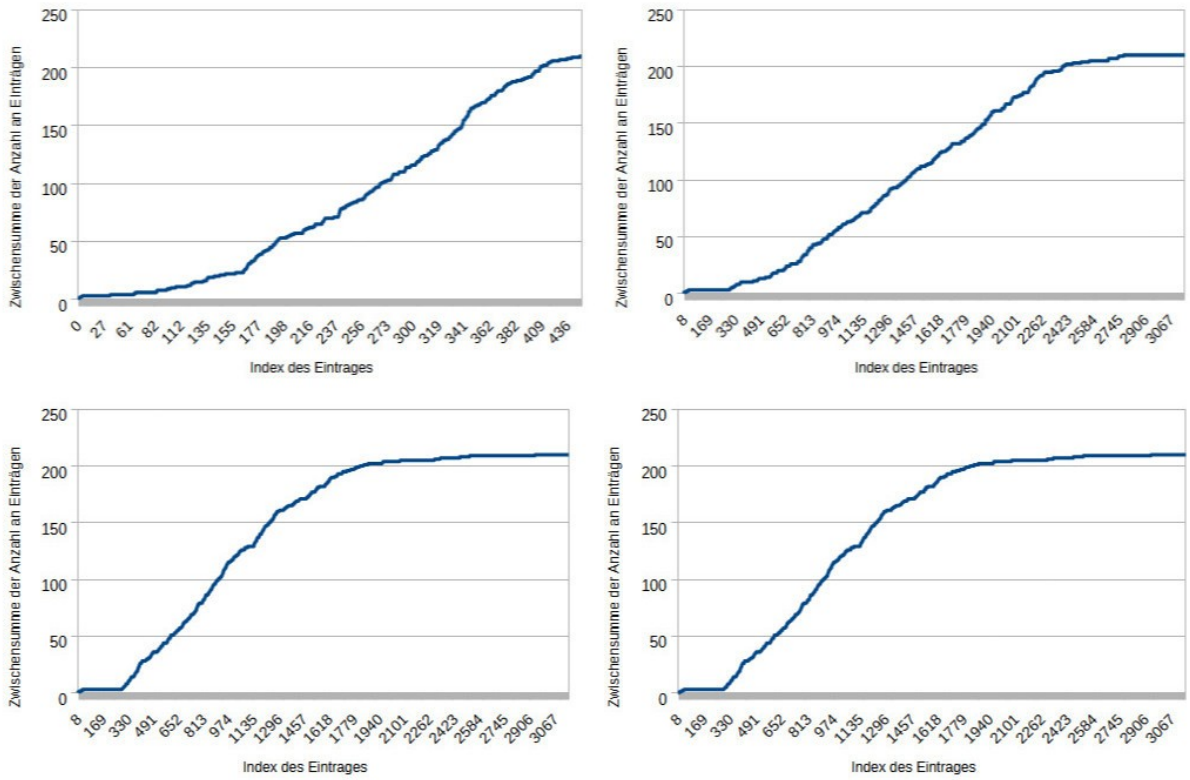


*Abbildung 6: Oben links: Differenzen von GradCAM; Oben rechts: Differenzen von HiResCAM; Unten links: Differenzen von GradCAMElementWise; Unten rechts: Differenzen von LayerCAM. Für jeden Graphen gilt: Der Index des Eintrags gibt den Index in der aufsteigend nach der absoluten Differenz sortierten Tabelle results_data an, und die Zwischensumme der Anzahl der Einträge gibt an, wie oft die jeweilige Methode bis zu einem Index vorgekommen ist.*

Somit lässt sich Aussagen, dass Grad-CAM mittelmäßig stark von Adversarial-Methoden beeinflusst wird, während HiResCAM weniger stark beeinflusst wird. Dies liegt wahrscheinlich daran, dass HiResCAM jeden Pixel einzeln gewichtet und somit die veränderten Pixel keinen Einfluss auf die unveränderten Pixel haben. GradCAMElementWise und LayerCAM werden noch deutlich weniger beeinflusst und sind somit deutlich stabiler. Dies könnte daran liegen, dass in HiResCAM für jeden Pixel nach der Multiplikation eines Pixels mit dessen Gradienten Werte

zwischen -1 und 1 auftreten können. Währenddessen verwenden die anderen beiden Methoden zusätzlich $ReLU()$ und reduzieren diese Werte somit auf 0 bis 1.

Als nächstes werden in Abbildung 7 XAI-Methoden betrachtet, welche in ihrer Saliency Map nicht zwischen Klassen unterscheiden. Sowohl Eigen-CAM als auch FEM folgen einer leichten logarithmischen Kurve. KPCA-CAM folgt einer sehr stark ansteigenden linearen Funktion am Anfang, danach folgt sie einer logarithmischen Funktion und gegen Ende wieder einer linearen Funktion mit einer leichten Steigung. Dies indiziert, dass KPCA-CAM oft sehr geringe Differenzen hat, und nur sehr selten mittlere Differenzen. FullGrad verhält sich sehr ähnlich zu LayerCAM und GradCAMElementWise, wobei der lineare Teil weniger steil ansteigt.

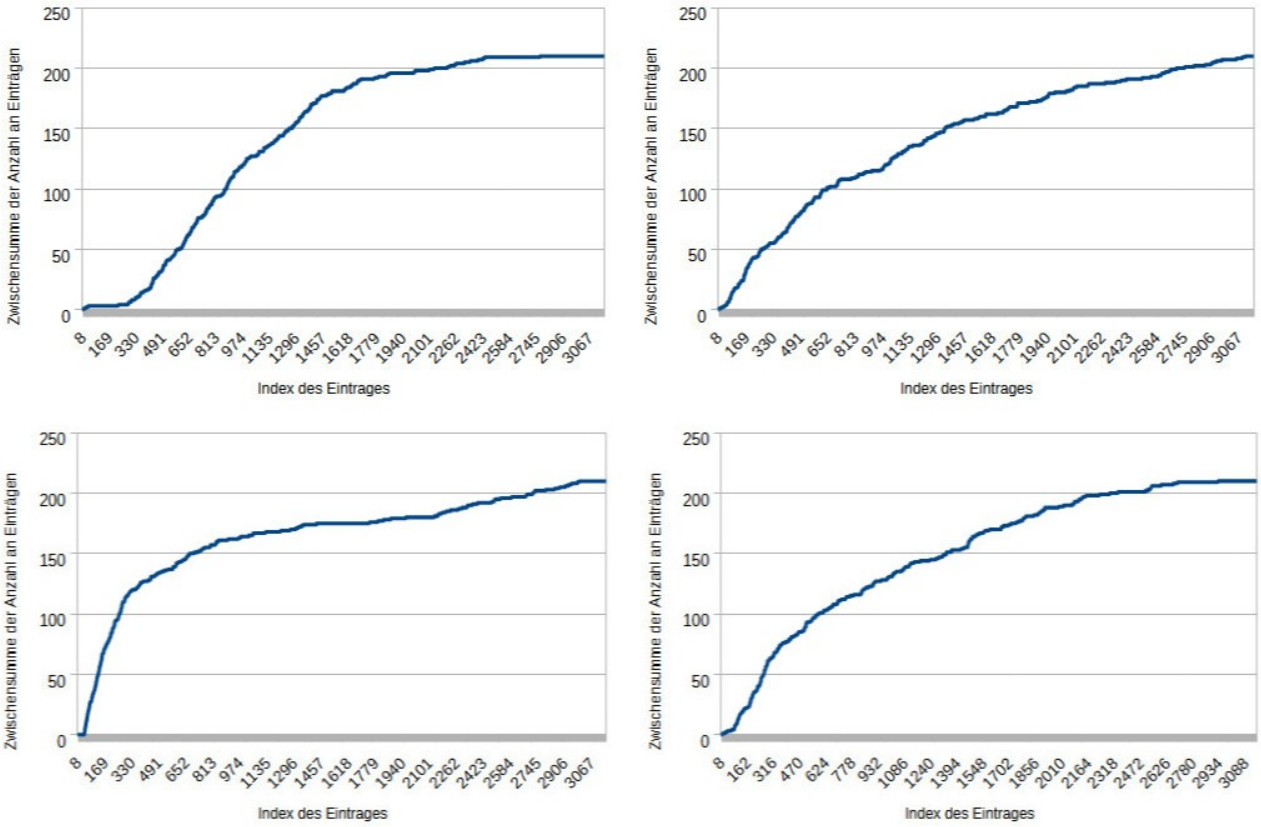


*Abbildung 7: Oben links: Differenzen von FullGrad; Oben rechts: Differenzen von Eigen-CAM; Unten links: Differenzen von KPCA-CAM; Unten rechts: Differenzen von FEM. Für jeden Graphen gilt: Der Index des Eintrags gibt den Index in der aufsteigend nach der absoluten Differenz sortierten Tabelle results_data an, und die Zwischensumme der Anzahl der Einträge gibt an, wie oft die jeweilige Methode bis zu einem Index vorgekommen ist.*

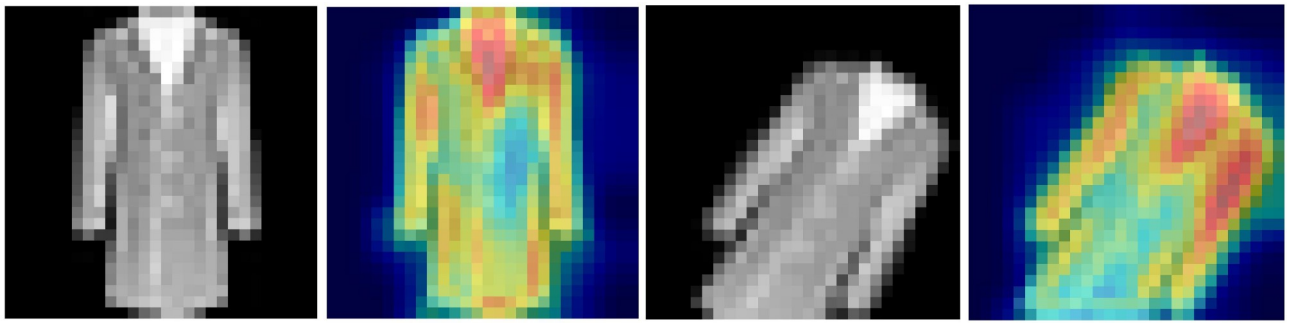

*Abbildung 8: Links: Ein Bild aus dem Datensatz; Mitte Links: KPCA-CAM für das linke Bild; Mitte Rechts: Ein Adversarial des linken Bildes, das durch Spatial Attack generiert wurde; Rechts: KPCA-CAM für das linke Bild nachdem Spatial Attack angewandt wurde*

KPCA-CAM wird nur sehr schwach von Adversarials beeinflusst und ist somit besonders stabil. Dies liegt wahrscheinlich daran, dass KPCA-CAM PCA auf einen Kernel der Eingabe anwendet statt der Eingabe selbst, wodurch Features erkannt werden können. Dadurch lässt sich auch die leichte Steigung am Ende des Graphen erklären, da diese wahrscheinlich durch Spatial Attack erzeugt wird. Es erhält eben diese Features und rotiert diese lediglich, wodurch auch die Saliency Map verschoben wird. Ein Beispiel wird in Abbildung 8 gezeigt. Eigen-CAM, FullGrad als auch FEM werden hier weniger stark von den Adversarials beeinflusst. Dies liegt vermutlich daran, dass alle drei Methoden keine Saliency Map für eine bestimmte Klasse ausgeben und somit auch Features, welche weder für die originale Klassifikation noch für die Fehlklassifikation relevant sind, und somit nicht zu stark verändert wurden, betrachten.

In Abbildung 9 verläuft der Graph von Finer-CAM stark exponentiell, was indiziert, dass Finer-CAM fast exklusiv nur Saliency Maps mit großen Differenzen erzeugt. Außerdem verlaufen die Graphen von Ablation-CAM, XGrad-CAM, Score-CAM, und ShapleyCAM am Anfang ebenfalls exponentiell,  und, bis auf ShapleyCAM, danach linear. Besonders steht hier XGrad-CAM heraus, welche fast keine niedrigen Differenzen hat. Grad-CAM++  verläuft anfangs konstant und danach sigmoidal, wodurch sich schließen lässt, dass die meisten Differenzen in der oberen Mitte liegen. EigenGradCAM verläuft annähernd linear, wobei sich die Steigung bei den höchsten Differenz stark erhöht.

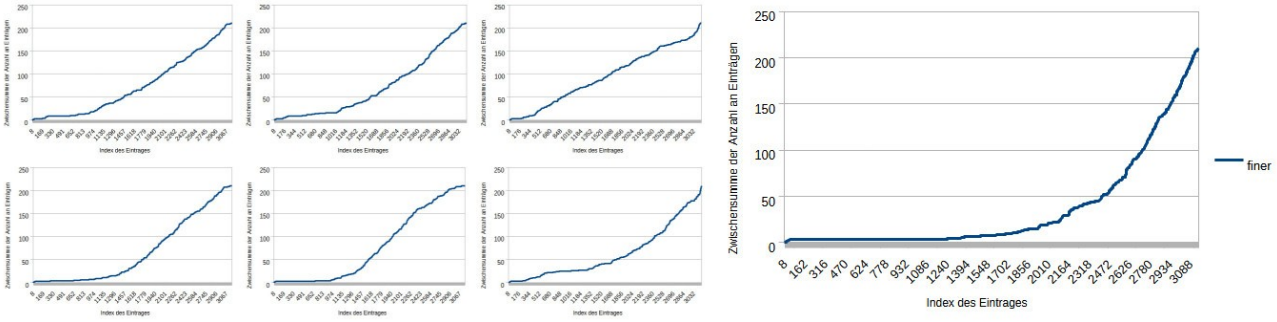


*Abbildung 9: Oben links: Differenzen von Ablation-CAM; Unten links: Differenzen von XGrad-CAM; Oben Mitte: Differenzen von Score-CAM; Unten Mitte: Differenzen von Grad-CAM++; Oben rechts: Differenzen von EigenGradCAM; Unten rechts: Differenzen von ShapleyCAM; Weit rechts: Differenzen von Finer-CAM. Für jeden Graphen gilt: Der Index des Eintrags gibt den Index in der aufsteigend nach der absoluten Differenz sortierten Tabelle results_data an, und die Zwischensumme der Anzahl der Einträge gibt an, wie oft die jeweilige Methode bis zu einem Index vorgekommen ist.*

Hier fällt vor allem auf, dass Finer-CAM besonders stark von Adversarial Methoden beeinflusst wird. Dies liegt daran, dass Finer-CAM spezifisch das Ziel hat, diskriminierende Regionen hervorzuheben, und da das Adversarial falsch klassifiziert wird, werden diskriminierende Regionen der originalen Saliency Map reduziert und Regionen, die für die Fehlklassifizierung relevant sind stärker hervorgehoben. Der Graph von XGrad-CAM lässt sich dadurch erklären, dass die Methode die einzelnen Gradienten Feature Map durch den zugehörigen Wert skaliert, und somit werden die Gradienten durch die Perturbation des Adversarials anders skaliert, was zu einer höheren Differenz führt.

Der Graph von EigenGradCAM zeigt, dass durch die zusätzlichen Gradienten mehr hohe Differenzen als bei Eigen-CAM auftreten. Dies passiert, weil die Gradienten für eine spezifische Klasse gelten und somit, zusätzlich zu dem veränderten $V$, auch $A^c$ zwischen der Saliency Map der originalen Eingabe und der Saliency Map des Adversarials unterscheiden. Die Verteilung der Differenzen von ShapleyCAM entsteht dadurch, dass ShapleyCAM auch die Logits miteinbezieht, und da sich diese auch durch Adversarial-Methoden ändern, haben Adversarials größere Auswirkungen auf ShapleyCAM. Score-CAM berechnet die Gewichte für die Saliency Map, indem es die Differenz zwischen dem Score und dem Score ohne die Feature Map berechnet und durch den Score teilt. Dies führt dazu, dass das Gewicht für die Feature Maps, welche den Score der Fehlklassifizierung nicht stark beeinflussen, aber relevant für die Klassifizierung des originalen Bildes waren, weniger stark gewichtet werden und somit größere Differenzen entstehen. Ablation-CAM berechnet die Gewichte, indem Regionen durch Feature Maps hervorgehoben werden und der Score berechnet wird. Somit werden Features, welche zu der Fehlklassifikation führen, durch die

Feature Maps der Fehlklassifikation stärker hervorgehoben, und Features, welche relevant für die originale Klassifikation waren, werden weniger stark Gewichtet. Schlussendlich lässt sich der Graph von Grad-CAM++ dadurch erklären, dass die Gewichte mit den ersten und zweiten Gradienten berechnet werden, wodurch die Änderung der Pixel durch die Adversarial-Methode stärkere Auswirkungen auf die Gewichte und somit die Saliency Map hat.

## 4.2 Betrachtung der größten und niedrigsten Differenzen jeder XAI-Methode

Als nächstes werden jeweils die 20 größten und niedrigsten Differenzen jeder XAI-Methode genauer betrachtet, um spezifische Kombinationen an XAI-Methoden und Adversarial-Methoden zu finden, die zu besonders hohen oder niedrigen Differenzen führen.

| | Grad-CAM | HiResCAM | GradCAMElementWise | Grad-CAM++ | XGrad-CAM | Ablation-CAM | Score-CAM | Eigen-CAM | EigenGradCAM | LayerCAM | FullGrad | KPCA-CAM | FEM | ShapleyCAM | Finer-CAM |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| DeepFool | 2 | 4 | 4 | 3 | 5 | 3 | 3 | 5 | 3 | 4 | 4 | 7 | 4 | 3 | 1 |
| Carlini&Wagner | 5 | 5 | 8 | 4 | 8 | 5 | 4 | 5 | 6 | 8 | 5 | 7 | 6 | 4 | 3 |
| EAD | 5 | 3 | 4 | 7 | 2 | 5 | 3 | 6 | 5 | 4 | 5 | 3 | 6 | 3 | 2 |
| PGD | 2 | 3 | 1 | 2 | 2 | 2 | 3 | 2 | 1 | 1 | 2 | 1 | 1 | 4 | 4 |
| Spatial Attack | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3 |
| BIM | 6 | 3 | 2 | 2 | 2 | 4 | 3 | 1 | 1 | 2 | 3 | 1 | 1 | 4 | 4 |
| FGSM | 0 | 2 | 1 | 2 | 1 | 1 | 3 | 1 | 4 | 1 | 1 | 1 | 2 | 2 | 3 |

*Tabelle 1: Anzahl der Vorkommnisse jeder Adversarial-Methode für für die 20 niedrigsten Differenzen jeder XAI-Methode*

In Tabelle 1 wird für die 20 niedrigsten Differenzen jeder XAI-Methode die Anzahl der Vorkommnisse jeder Adversarial-Methode dargestellt. Dabei werden hier Differenzen für Adversarial-Methoden, die kein Adversarial gefunden haben, exkludiert, da diese hier immer eine Differenz von 0 oder nahe 0 haben. Es fällt auf, dass Spatial Attack nur bei Finer-CAM zu niedrigen Differenzen geführt hat, was wie bereits in 4.1 erwähnt, daran liegt, dass Spatial Attack die gesamte Feature Map verschiebt. Auch fällt auf, dass Carlini&Wagner für alle XAI-Methoden außer Grad-CAM++, Eigen-CAM und Finer-CAM meistens zu niedrigen Differenzen führt.

In Tabelle 2 wird für die 20 höchsten Differenzen jeder XAI-Methode die Anzahl der Vorkommnisse jeder Adversarial-Methode dargestellt. Hier fällt auf, dass Eigen-CAM, KPCA-CAM, FullGrad und FEM besonders stark von Spatial Attack beeinflusst werden, welche auch Methoden sind, die in 4.1 besonders wenig durch Adversarials beeinflusst werden. Dies liegt wahrscheinlich daran, dass alle dieser Methoden keine klassenspezifische Saliency Map

produzieren und stattdessen alle Regionen hervorheben, die das Netzwerk als Feature sieht[21]. Dadurch werden mehr Regionen hervorgehoben, die durch Spatial Attack verschoben werden und somit die Differenz erhöhen.

| | Grad-CAM | HiResCAM | GradCAMElementWise | Grad-CAM++ | XGrad-CAM | Ablation-CAM | Score-CAM | Eigen-CAM | EigenGradCAM | LayerCAM | FullGrad | KPCA-CAM | FEM | ShapleyCAM | Finer-CAM |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| DeepFool | 6 | 5 | 4 | 6 | 6 | 3 | 2 | 2 | 1 | 4 | 2 | 0 | 0 | 8 | 4 |
| Carlini&Wagner | 1 | 1 | 1 | 0 | 2 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 2 |
| EAD | 1 | 2 | 1 | 1 | 2 | 2 | 4 | 0 | 3 | 1 | 1 | 0 | 1 | 1 | 2 |
| PGD | 1 | 2 | 2 | 3 | 3 | 2 | 3 | 1 | 3 | 2 | 1 | 0 | 1 | 1 | 2 |
| Spatial Attack | 6 | 4 | 5 | 4 | 2 | 8 | 6 | 12 | 2 | 5 | 7 | 20 | 11 | 2 | 4 |
| BIM | 1 | 1 | 1 | 3 | 2 | 2 | 0 | 2 | 6 | 1 | 2 | 0 | 0 | 2 | 2 |
| FGSM | 4 | 5 | 6 | 3 | 3 | 2 | 4 | 3 | 4 | 6 | 6 | 0 | 6 | 6 | 4 |

*Tabelle 2: Anzahl der Vorkommnisse jeder Adversarial-Methode für für die 20 höchsten Differenzen jeder XAI-Methode*

## 4.3 Betrachtung der Relation von der Perturbationen und Änderungen der Saliency Maps

Als letztes wird die Pearson Korrelation zwischen der Perturbation durch Adversarials und der Differenz zwischen den Saliency Maps berechnet. Diese wird dann genutzt, um die Nullhypothese, dass die Perturbation und die Differenz korrelieren, auf dem 5%-Niveau zu testen und diese entweder zurückzuweisen oder zu akzeptieren.

| | Grad-CAM | HiResCAM | GradCAMElementWise | Grad-CAM++ | XGrad-CAM | Ablation-CAM | Score-CAM | Eigen-CAM | EigenGradCAM | LayerCAM | FullGrad | KPCA-CAM | FEM | ShapleyCAM | Finer-CAM | *Summe* |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| DeepFool | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 2 | 0 | 0 | 5 |
| Carlini&Wagner | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 5 |
| EAD | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 5 |
| PGD | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 2 | 1 | 12 |
| BIM | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 2 | 0 | 0 | 9 |
| FGSM | 1 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 5 |
| *Summe* | 1 | 4 | 2 | 2 | 3 | 4 | 3 | 1 | 2 | 2 | 3 | 3 | 5 | 3 | 3 | 41 |

*Tabelle 3: Anzahl der Vorkommnisse jeder Kombination von Adversarial-Methoden und XAI-Methoden für alle Adversarial, dessen Nullhypothese zurückgewiesen wurde*

In Tabelle 3 wird für jedes Adversarial, dessen Nullhypothese zurückgewiesen wurde, die Anzahl der Vorkommnisse für jede Kombination von Adversarial-Methoden und XAI-Methoden dargestellt.

Hier fällt auf, dass FEM, BIM und PGD relativ große Summen besitzen. Für FEM lässt sich dies dadurch erklären, dass Pixel von Feature Maps auf 0 reduziert werden, wenn sie unter einem gewissen Threshold liegen, wodurch Informationen verloren gehen. Die Summen für PGD und BIM hingegen lassen sich womöglich dadurch erklären, dass durch das iterative berechnen von Perturbationen und clippen/projizieren Störungen entstehen könnten, welche von XAI-Methoden weniger beachtetet werden und somit zu einer weniger starken Korrelation führen. Somit lässt sich zusammenfassend der Schluss ziehen, dass große Teile der Perturbationen von Adversarial-Methoden, welche iterativ Störung einführen, wie BIM und PGD, die nicht maßgeblich für die Fehlklassifikation sind, nicht von XAI-Methoden erfasst werden. Auch erfasst FEM oft nur eine kleinen Anteil der Perturbation.

Da für jede XAI-Methode aber jeweils 210 und für jede Adversarial-Methode 450 Werte existiert, wurde die Nullhypothese für einen Großteil akzeptiert und es kann somit nicht eindeutig gesagt werden, ob es eine direkte Korrelation zwischen den Perturbationen des Adversarials und der Änderung der Saliency Map gibt. Stattdessen könnte es der Fall sein, dass die Korrelation davon abhängt, wie optimal die Perturbation ist, d.h. wie nah die Perturbation an dem Minimum ist, das immer noch zu einer Fehlklassifikation führt. Außerdem könnte es auch sowohl von der individuellen Eingabe als auch davon, wie stark die Visualisierungsmethode weniger relevante Perturbationen beachtet, abhängen.

## 5. Fazit

Das Ziel dieser Bachelorarbeit war es, die Auswirkung von Adversarial-Methoden auf XAI-Methoden zu untersuchen.

Es zeigt sich, dass XAI-Methoden, welche keine Saliency Maps für eine spezifische Klasse berechnen und stattdessen alle Features hervorheben, welche das Netzwerk beachtet hat, relativ stabil sind und nicht stark von Adversarial-Methoden beeinflusst werden. Außerdem sind auch XAI-Methoden, welche die Pixel der Feature Maps einzeln Gewichten, robust gegenüber Adversarials und somit stabil. Hingegen sind XAI-Methoden, welche versuchen relevante Features stärker hervorzuheben, sensitiver und werden stark von Adversarials beeinflusst. Zwischen Adversarial-Methoden, die Perturbationen verwenden gibt es wenige Unterschiede im Effekt auf die Feature Maps, allerdings haben andere Angriffe wie Spatial Attack unterschiedliche Auswirkungen.

Auch hat sich ergeben, dass besonders starke Adversarial-Methoden wie Carlini & Wagner [31] nur leichte Einflüsse auf fast alle XAI-Methoden haben, mit Ausnahme von Grad-CAM++, Eigen-CAM

und Finer-CAM. Außerdem werden die Methoden KPCA-CAM, FEM, FullGrad und Eigen-CAM stark durch Spatial Attack beeinflusst und sind somit sensitiv diesen gegenüber.
Als letztes hat sich auch ergeben, dass große Teile der Perturbationen, welche durch die Methoden PGD und BIM generiert werden, wahrshcienlich oft nicht erfasst werden. Außerdem erfasst FEM auch wahrscheinlich oft nur einen kleinen Anteil der Perturbation. Im Allgemeinen ließ sie schließen, dass die Frage, ob es eine direkten Korrelation zwischen der Perturbation und Änderung der Saliency Map gibt, nicht eindeutig beantworten. Stattdessen ist es wahrscheinlich, dass die Korrelation von der Adversarial-Methode, der XAI-Methode als auch dem individuellen Sampel abhängt.
Die Ergebnisse zeigen, dass der Effekt von Adversarial-Methoden auf XAI-Methoden stark von der XAI-Methode und Adversarial-Methoden, welche die Eingabe räumliche verändern, abhängt, und nur begrenzt von Adversarial-Methoden abhängt, die Perturbationen einführen.
Die Ergebnisse dieser Bachelorarbeit sind in der Praxis dahingehend relevant, dass sensitive Methoden wie Finer-CAM besser sind, um Adversarial Attacks zu erkennen und und Effekt zu diagnostizieren. Stabile Methoden wie KPCA-CAM sind hingegen dafür relevant, glaubwürdigere Erklärungen für reale Szenarien, wie dem autonomen Fahren, zu generieren, da diese nicht stark von Störungen, welche in der Realität auftreten können, beeinflusst werden.
Trotz der gewonnenen Erkentnisse über den Effekt von Adversarial-Methoden auf XAI-Methoden, besitzt diese Bachelorarbeit noch einige Einschränkungen. Es wurde unter anderem nur ein relativ simpler und kleiner Datensatz betrachtet, der aufgrund der niedrigen Auflösung der Samples und dem Mangel an Farben nicht realistisch ist. Außerdem haben alle Samples nur ein Objekt, obwohl in realen Szenarien oft viele weitere Objekte vorhanden sind. Außerdem wurden Methoden weitgehend nur durch ihrer Summe der absoluten Differenz beurteilt und die Relationen der einzelnen Pixel wurden nicht betrachtet. Dadurch sich allerdings Methoden wie Spatial Attack nur schwer zu beschreiben, da das Objekt selbst lediglich rotiert wird, die Differenz aber trotzdem extrem groß seien kann. Somit wäre sowohl eine Untersuchung auf größeren und komplexeren Datensätzen  als auch mit Metriken, die die Relation zwischen Pixeln erfassen, von Interesse. Die in dieser Arbeit erzielten Ergebnisse können als eine Basis für solche weiteren Forschungen dienen.

## 6. Referenzen


[1] X. Gao, “The ubiquitous influence of image recognition in daily life,” *Applied and Computational Engineering*, vol. 53, pp. 126–130, Mar. 2024, doi: 10.54254/2755-2721/53/20241306.

[2] M. Krichen, “Convolutional Neural Networks: A Survey,” *Computers*, vol. 12, no. 8, Art. no. 8, Aug. 2023, doi: 10.3390/computers12080151.

[3] S. Karmani, T. Sivakaran, G. Prasad, M. Ali, W. Yang, and S. Tang, “KPCA-CAM: Visual Explainability of Deep Computer Vision Models using Kernel PCA,” Sept. 30, 2024, *arXiv*: arXiv:2410.00267. doi: 10.48550/arXiv.2410.00267.

[4] A. Kurakin, I. Goodfellow, and S. Bengio, “Adversarial examples in the physical world,” Feb. 11, 2017, *arXiv*: arXiv:1607.02533. doi: 10.48550/arXiv.1607.02533.

[5] mjoloist, *mjoloist/XAI_methods_on_adversarial_inputs*. (Aug. 11, 2025). Accessed: Aug. 11, 2025. [Online]. Available: https://github.com/mjoloist/XAI_methods_on_adversarial_inputs

[6] C. Gershenson, “Artificial Neural Networks for Beginners,” Aug. 20, 2003, *arXiv*: arXiv:cs/0308031. doi: 10.48550/arXiv.cs/0308031.

[7] S. Sharma, S. Sharma, and A. Athaiya, “ACTIVATION FUNCTIONS IN NEURAL NETWORKS,” *IJEAST*, vol. 04, no. 12, pp. 310–316, May 2020, doi: 10.33564/IJEAST.2020.v04i12.054.

[8] J. Henseler, “Back Propagation”.

[9] K. Janocha and W. M. Czarnecki, “On Loss Functions for Deep Neural Networks in Classification,” Feb. 18, 2017, *arXiv*: arXiv:1702.05659. doi: 10.48550/arXiv.1702.05659.

[10] “A Review Study of the Visual Geometry Group Approaches for Image Classification | Journal of Applied Science, Technology and Computing.” Accessed: Aug. 04, 2025. [Online]. Available: https://publisher.uthm.edu.my/ojs/index.php/jastec/article/view/16010

[11] “A New Global Pooling Method for Deep Neural Networks: Global Average of Top-K Max-Pooling | IIETA.” Accessed: July 27, 2025. [Online]. Available: https://iieta.org/journals/ts/paper/10.18280/ts.400216

[12] A. P. T. Minh, “Overview of Class Activation Maps for Visualization Explainability,” Sept. 25, 2023, *arXiv*: arXiv:2309.14304. doi: 10.48550/arXiv.2309.14304.

[13] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, “Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization,” in *2017 IEEE International Conference on Computer Vision (ICCV)*, Oct. 2017, pp. 618–626. doi: 10.1109/ICCV.2017.74.

[14] R. L. Draelos and L. Carin, “Use HiResCAM instead of Grad-CAM for faithful explanations of convolutional neural networks,” Nov. 21, 2021, *arXiv*: arXiv:2011.08891. doi: 10.48550/arXiv.2011.08891.

[15] J. Gildenblat, *jacobgil/pytorch-grad-cam*. (July 29, 2025). Python. Accessed: July 29, 2025. [Online]. Available: https://github.com/jacobgil/pytorch-grad-cam

[16] A. Chattopadhyay, A. Sarkar, P. Howlader, and V. N. Balasubramanian, “Grad-CAM++: Improved Visual Explanations for Deep Convolutional Networks,” in *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, Mar. 2018, pp. 839–847. doi: 10.1109/WACV.2018.00097.

[17] R. Fu, Q. Hu, X. Dong, Y. Guo, Y. Gao, and B. Li, “Axiom-based Grad-CAM: Towards Accurate Visualization and Explanation of CNNs,” Aug. 19, 2020, *arXiv*: arXiv:2008.02312. doi: 10.48550/arXiv.2008.02312.

[18] Á. Arroyo *et al.*, “On Vanishing Gradients, Over-Smoothing, and Over-Squashing in GNNs: Bridging Recurrent and Graph Learning,” Feb. 15, 2025, *arXiv*: arXiv:2502.10818. doi: 10.48550/arXiv.2502.10818.

[19] S. Desai and H. G. Ramaswamy, “Ablation-CAM: Visual Explanations for Deep Convolutional Network via Gradient-free Localization,” in *2020 IEEE Winter Conference on Applications of Computer Vision (WACV)*, Mar. 2020, pp. 972–980. doi: 10.1109/WACV45572.2020.9093360.
[20] H. Wang *et al.*, “Score-CAM: Score-Weighted Visual Explanations for Convolutional Neural Networks,” Apr. 13, 2020, *arXiv*: arXiv:1910.01279. doi: 10.48550/arXiv.1910.01279.
[21] M. B. Muhammad and M. Yeasin, “Eigen-CAM: Class Activation Map using Principal Components,” in *2020 International Joint Conference on Neural Networks (IJCNN)*, July 2020, pp. 1–7. doi: 10.1109/IJCNN48605.2020.9206626.
[22] P.-T. Jiang, C.-B. Zhang, Q. Hou, M.-M. Cheng, and Y. Wei, “LayerCAM: Exploring Hierarchical Class Activation Maps for Localization,” *IEEE Transactions on Image Processing*, vol. 30, pp. 5875–5888, 2021, doi: 10.1109/TIP.2021.3089943.
[23] S. Srinivas and F. Fleuret, “Full-Gradient Representation for Neural Network Visualization,” Dec. 03, 2019, *arXiv*: arXiv:1905.00780. doi: 10.48550/arXiv.1905.00780.
[24] Z. Zhang *et al.*, “Finer-CAM: Spotting the Difference Reveals Finer Details for Visual Explanation,” Mar. 31, 2025, *arXiv*: arXiv:2501.11309. doi: 10.48550/arXiv.2501.11309.
[25] K. Ahmed Asif Fuad, P.-E. Martin, R. Giot, R. Bourqui, J. Benois-Pineau, and A. Zemmari, “Features Understanding in 3D CNNs for Actions Recognition in Video,” in *2020 Tenth International Conference on Image Processing Theory, Tools and Applications (IPTA)*, Paris, France: IEEE, Nov. 2020, pp. 1–6. doi: 10.1109/IPTA50016.2020.9286629.
[26] H. Cai, “CAMs as Shapley Value-based Explainers,” *Vis Comput*, vol. 41, no. 10, pp. 7249–7267, Aug. 2025, doi: 10.1007/s00371-025-03803-1.
[27] A. Lad, R. Bhale, and S. Belgamwar, “Fast Gradient Sign Method (FGSM) Variants in White Box Settings: A Comparative Study,” in *2024 International Conference on Inventive Computation Technologies (ICICT)*, Apr. 2024, pp. 382–386. doi: 10.1109/ICICT60155.2024.10544606.
[28] N. Carlini and D. Wagner, “Towards Evaluating the Robustness of Neural Networks,” Mar. 22, 2017, *arXiv*: arXiv:1608.04644. doi: 10.48550/arXiv.1608.04644.
[29] M. S. Ayas, S. Ayas, and S. M. Djouadi, “Projected Gradient Descent Adversarial Attack and Its Defense on a Fault Diagnosis System,” in *2022 45th International Conference on Telecommunications and Signal Processing (TSP)*, July 2022, pp. 36–39. doi: 10.1109/TSP55681.2022.9851334.
[30] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, “DeepFool: a simple and accurate method to fool deep neural networks,” July 04, 2016, *arXiv*: arXiv:1511.04599. doi: 10.48550/arXiv.1511.04599.
[31] P.-Y. Chen, Y. Sharma, H. Zhang, J. Yi, and C.-J. Hsieh, “EAD: Elastic-Net Attacks to Deep Neural Networks via Adversarial Examples,” Feb. 10, 2018, *arXiv*: arXiv:1709.04114. doi: 10.48550/arXiv.1709.04114.
[32] L. Engstrom, B. Tran, D. Tsipras, L. Schmidt, and A. Madry, “Exploring the Landscape of Spatial Robustness,” Sept. 16, 2019, *arXiv*: arXiv:1712.02779. doi: 10.48550/arXiv.1712.02779.
[33] A. Saini, K. Guleria, and S. Sharma, “An Efficient TinyVGG Model for Sugarcane Leaf Disease Classification in Precision Agriculture,” in *2024 2nd World Conference on Communication & Computing (WCONF)*, July 2024, pp. 1–6. doi: 10.1109/WCONF61366.2024.10692002.
[34] “zalandoresearch/fashion-mnist: A MNIST-like fashion product database. Benchmark.” Accessed: Aug. 13, 2025. [Online]. Available: https://github.com/zalandoresearch/fashion-mnist