

Explainable AI

Stand der Forschung und Technik

Master Thesis

Studiengang:

MAS Data Science

Autor:

Marc Habegger

Dozent:

Max Kleiner, Prof. Dr. Arno Schmidhauser

Datum:

3. März 2020

Inhaltsverzeichnis

1 Einleitung	1
2 Was bedeutet Erklärbarkeit?	4
2.1 Unterschiedliche Anforderungen an eine Erklärung	4
2.2 Zielgebiete von XAI	5
2.2.1 Bessere Anwendungen durch Einblick in die Funktionsweise	5
2.2.2 Datenschutz	6
2.2.3 Sicherheit	7
2.2.4 Regulatorische Bedingungen	7
2.2.5 Haftungsfragen	7
3 Wann wird XAI eingesetzt?	8
3.1 Exploration	8
3.2 Feature Engineering	9
3.3 Modellauswahl	9
3.4 Existierendes Modell verwenden	12
3.5 Interpretierbares Modell erstellen	12
3.5.1 Lineare Regression	13
3.5.2 Logistische Regression	13
3.5.3 GLM/GAM	13
3.5.4 Decision Tree	14
3.5.5 RuleFit	15
3.5.6 Naive Bayes	15
3.6 Nicht Interpretierbares Model erstellen	16
3.7 Modell analysieren	16
3.7.1 Modell Testen	16
3.7.2 Erklärungen aus Modell generieren	16
3.7.3 Bias Kontrolle	16
4 XAI Methoden	17
4.1 Grad CAM	17
4.2 Occlusion Sensitivity	18
4.3 LRP	21
4.4 Local Surrogate (LIME)	23
4.5 TCAV	26
4.6 SVCCA	27
5 Konkrete Anwendung von XAI	29
5.1 Bilderkennung: Klassifikation Hund - Katze	29
5.1.1 White Box Modell	29
5.1.2 Black Box Modell	36

5.2 Texterkennung: Stimmungs-Analyse von Film-Bewertungen	41
5.2.1 White Box Modell	41
5.2.2 Black Box Modell	47
6 Schwächen von ML Modellen erkennen	50
6.1 Diskriminierung durch Bias	50
6.2 Manipulierte Bilder: Adversarial Attacks	50
6.3 Manipulierte Daten: Data Poisoning	52
7 Weiterentwicklung von XAI	54
8 Anhang	55
8.1 Source Code	55
8.1.1 Entscheidungsbaum Visualisierung mit sklearn und Graphviz	55
8.1.2 Bild-Klassifikation mit tf-explain	56
8.1.3 Visualisierung einer Klassifikation mit lime	57
8.1.4 Visualisierungen mehrere Klassen	58
Index	70

1 Einleitung

Computerprogramme bestehen in der Regel aus Tausenden bis Millionen Zeilen von Anweisungen welche, für die verschiedenen Aufgabengebiete zuständig sind. Einige der Programmroutine stellen die grafische Benutzeroberfläche dar, während andere sich um das Speichern und laden von Dateien kümmern. Ein Teil der Anweisungen definieren Regeln nach denen Daten verarbeitet und analysiert werden. Solche Regeln werden von Fachspezialisten definiert und durch Software-Entwickler umgesetzt. Mit einem derartigen Vorgehen konnten viele alltägliche Probleme gelöst werden, Software ist inzwischen in der Wirtschaft wie im privaten Umfeld allgegenwärtig geworden.

Die Ausformulierung solcher Regeln nach denen sich Software verhalten soll, ist aber ein aufwendiger und fehlerträchtiger Prozess. Gewisse Gebiete sind durch die Komplexität der Aufgabenstellung nur rudimentär in Regeln zu fassen. Solche Gebiete sind unter anderem Bilderkennung, Text- oder Sprachverständnis. In diesen Gebieten zeigen Menschen und Tiere bedeutend bessere Fähigkeiten als Computerprogramme.

Während programmierte Regeln in Computer Programmen sofort zur Verfügung stehen, müssen Menschen und Tiere ihre Fähigkeiten oftmals über längere Zeit trainieren und üben. Bis die gewünschten Fähigkeiten in genügender Qualität vorhanden sind können Jahre vergehen. Durch solche biologische Prozesse als Vorbild wurde die Disziplin Machine Learning (ML) entwickelt, welche auch Computer in die Lage versetzen soll, bislang nur bei Mensch und Tier gefundene Fähigkeiten, zu erlangen. ML wird seit den 1960er Jahren angewendet, allerdings waren die erzielten Resultate lange Zeit für viele Anwendungen ungenügend. Durch die Verfügbarkeit von grossen Datenmengen (Big Data, Cloud) und der gesteigerten Rechenleistung der Rechner wurden nach der Jahrtausendwende so gute Fortschritte erzielt, so dass immer mehr Anwendungsmöglichkeiten für ML Lösungen gefunden wurden.

Oftmals wird auch von Artificial Intelligence (AI) gesprochen wenn Machine Learning (ML) gemeint ist, die beiden Begriffe sind schwer zu trennen.

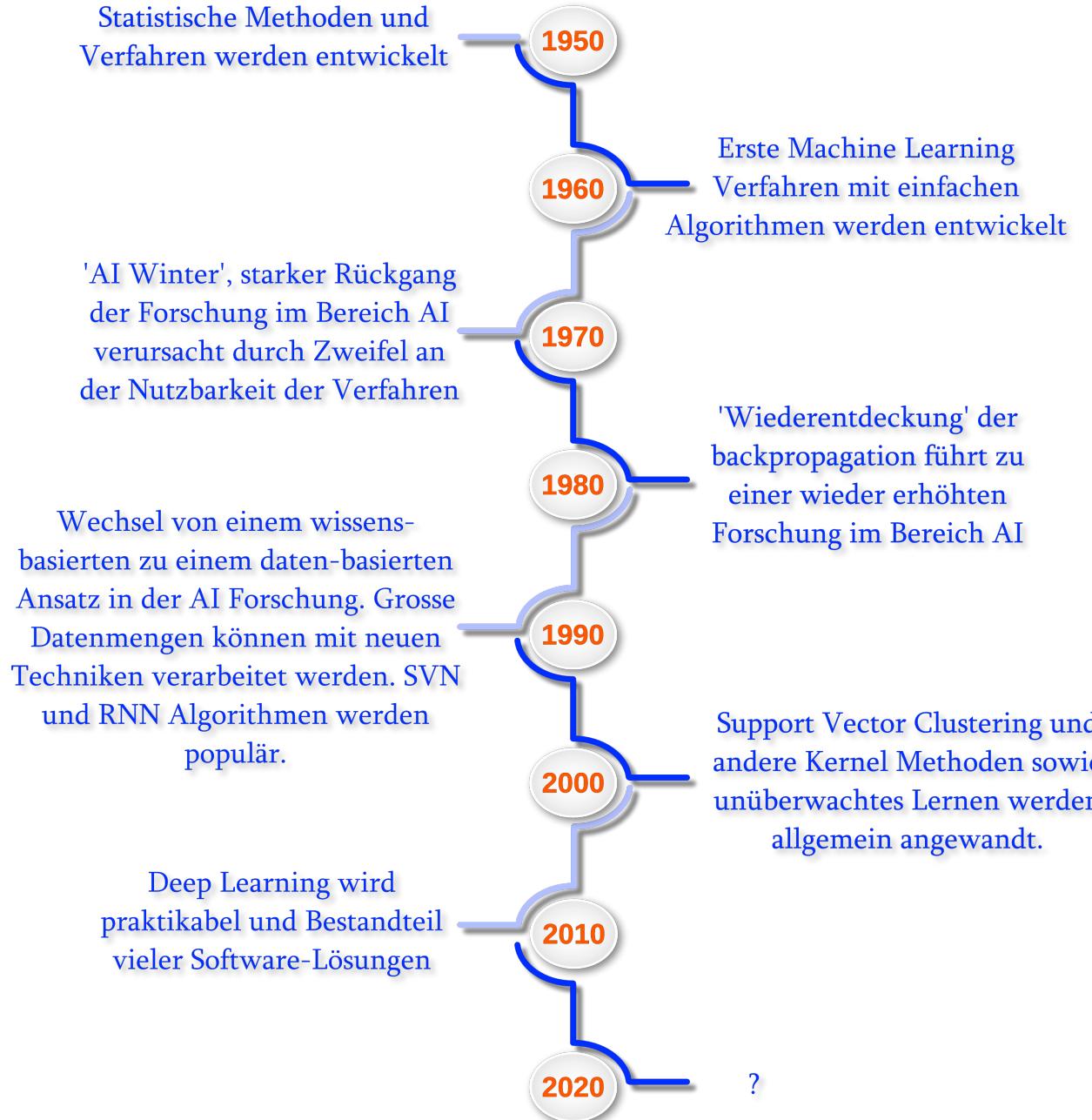


Abbildung 1.1: Entwicklung des Machine Learning als Zeitachse

Durch die Anwendung von Versuch und Irrtum bei kontinuierlicher Optimierung der internen Regeln erlangten ML Systeme bislang unerreichbare Stärken in vorher problematischen Gebieten. Allerdings ist der Preis dafür oftmals der, dass die automatisch erstellten Regeln für Menschen unverständlich und nicht nachvollziehbar sind. Für viele Dienste im Internet (Bildersammlungen, Empfehlungssysteme) werden in der Regel keine oder nur geringe Anforderungen an ein verständliches Modell gestellt. Aber es gibt einige Bereiche in denen besondere Ansprüche an die Nachvollziehbarkeit von Entscheidungen bestehen.

Exemplarisch werden hier einige dieser Gebiete aufgeführt:

Medizin

ML Anwendungen für die Krebserkennung bieten grosses Potenzial. Insbesondere die ermüdende

Aufgabe auf Röntgen- oder MRT-Bildern Spuren eines Tumors zu erkennen, könnten durch ML abgelöst werden. Allerdings sind die Zulassungskriterien für solche Lösungen noch nicht definiert.

Justiz

Predictive Policing versucht mittels statistischer und ML Verfahren Orte oder Personengruppen zu erkennen, welche Schauplatz oder Täter/Opfer eines Verbrechens werden könnten.

Selbstfahrende Fahrzeuge

Obwohl selbstfahrende Fahrzeuge seit Jahren von allen grossen Fahrzeugherstellern entwickelt werden, sind immer noch viele Fragen bezüglich der Haftung und Zulassung offen.

Aufgrund des Mangels an Techniken um fortgeschrittene ML System zu verstehen, entstand deshalb ein neues Forschungsgebiet Explainable artificial intelligence (XAI), welches sich zum Ziel gesetzt hat Methoden und Werkzeuge zu entwickeln um ML Modelle zu analysieren.

2 Was bedeutet Erklärbarkeit?

ML erzeugt Resultate, welche je nach Anwendungsfall Entscheidungen für Klassen (Pferd, Schaf, Auto), Zuordnungen zu Gruppen (Premium-Kunde, Gelegenheitskäufer) oder numerische Werte (15 Grad Celsius am 3. April) sind. Da sowohl die Erzeugung des Modells als auch die Berechnung des Resultates automatisch erfolgt, können die Schritte auf dem Weg zu dem Resultat nicht direkt nachvollzogen werden.

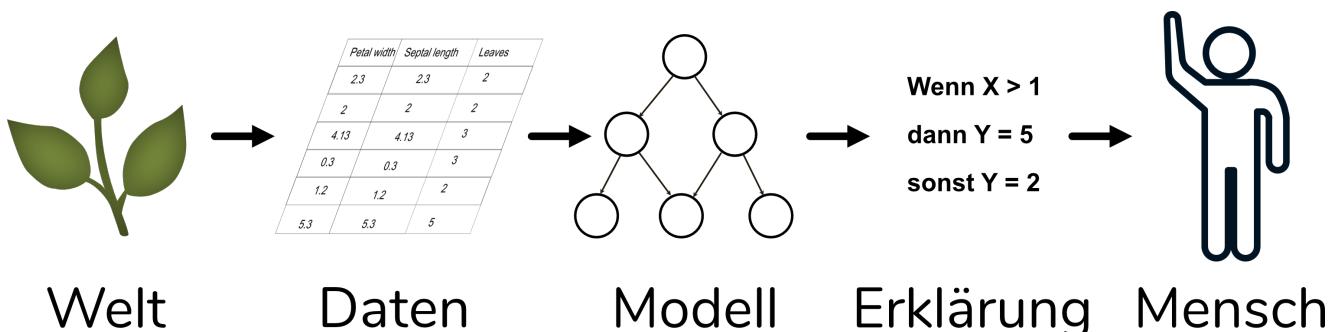


Abbildung 2.1: Ablauf einer erklärbaren Machine Learning Anwendung

Eine ML Lösung beginnt mit der Beobachtung von realen Ereignissen in der Welt. Dies können die Anzahl Blätter und deren Länge einer Pflanzengattung oder auch Häuserpreise in Brooklyn sein. Diese Beobachtungen werden gesammelt und bilden die Datengrundlage mit deren ein Modell erstellt werden kann. Aus diesem kann danach eine Erklärung erzeugt werden, die ein Mensch verwenden kann um das Resultat zu erklären.

2.1 Unterschiedliche Anforderungen an eine Erklärung

Eine Anwendung welche ML einsetzt kann in mehrere Bereiche unterteilt werden. Durch diese Aufteilung in verschiedene Komponenten ergeben sich unterschiedliche Anforderungen an die Erklärbarkeit: (*Explainable and Interpretable Models in Computer Vision and Machine Learning*, 2018)

Daten

Aus der Sicht der Daten interessiert vor allem welcher Teil der Daten für das Ergebnis die grösste Relevanz hat. Basierend auf dieser Erkenntnis kann das Datenset gezielt erweitert oder reduziert werden, so dass ein ausgeglichenes Verhältnis erzeugt wird.

Modell

Kann man aus dem Modell Muster für eine bestimmte Kategorie ableiten? Dies kann helfen Fehlklassifizierungen von neuen Daten zu verhindern, in dem überprüft wird, ob das Modell die richtigen/plausiblen Features berücksichtigt.

Vorhersage

Erklärung weshalb ein bestimmtes Muster in den Daten zu der beobachteten Klassifizierung geführt hat. Dies ist insbesondere für Anwender/Kunden einer ML Lösung um das Verständnis für die Maschinelle Entscheidung zu erhöhen oder eine gesetzlich vorgeschriebene Anfechtbarkeit der Entscheidung zu ermöglichen.

Ebenso gibt es bei den Interessengruppe unterschiedliche Anforderungen an die Erklärbarkeit einer ML Anwendung. Nach G. Ras (Ras et al., 2018) werden dabei folgende Gruppierungen unterschieden:

Experten

Diese Gruppe kann weiter unterteilt werden in

Forscher

Entwickelt neue Methoden und Algorithmen, verbessert bestehende Algorithmen

Entwickler

Setzt bestehende Methoden und Algorithmen ein, um eine konkrete Aufgabenstellung zu lösen

Benutzer

Auch bei den Benutzern gibt es verschiedene Ausprägungen

Eigentümer

Eigentümer/Auftraggeber benötigen performante Anwendungen ohne rechtliche Bedenken

Anwender

Sind interessiert wie eine Entscheidung die sie betrifft zustande kam

Person deren Daten verwendet werden

sind an einem starken Datenschutz interessiert ohne eine Missbrauchsmöglichkeit ihrer Daten

Anspruchsgruppe (Stakeholder)

wie Regulierungsbehörden oder Fachgremien setzen die Rahmenbedingungen für den Einsatz von ML Anwendungen in heiklen Gebieten

Die Anforderungen an ein erklärbare Modell unterscheiden sich sehr stark, je nach betrachteter Komponente und der Anwendergruppe. Daraus ergibt sich, dass verschiedene Techniken benötigt werden um AI Lösungen generell erklärbar zu machen.

2.2 Zielgebiete von XAI

2.2.1 Bessere Anwendungen durch Einblick in die Funktionsweise

Ein Grundsatz jedes ML Projektes lautet, dass der Erfolg nicht garantiert ist. Ausgehend von bestehenden Daten kann man nicht sicher sein, dass diese ausreichende Informationen liefern, um ein Modell zu entwickeln, welches den Anforderungen entspricht. Selbst wenn genügend Daten vorhanden sind besteht immer noch die Problematik, dass unzählige Algorithmen mit wiederum unzähligen Parametern existieren welche zur Anwendungen kommen können. Es Lösungen, oder Ansätze davon, um mit dieser Problematik umzugehen, dennoch ist es in der Regel ein langwieriger und oftmals teuer Prozess um ML Modelle auf das gewünschte Qualitätsniveau zu bringen.

Erkenntnisse durch Explainable artificial intelligence (XAI) Techniken können den Entwicklern helfen Irrwege und Probleme frühzeitig zu erkennen und so entweder Arbeitszeit oder Rechenleistung beim berechnen der Modell einzusparen.

Ebenso kann ein Modell falsche Resultate liefern, sei es weil die ursprünglichen Trainingsdaten unvollständig waren, oder weil neue Bedingungen aufgetreten sind. Eine Erklärung weshalb dieses falsche Resultat erzeugt wurde kann den Entwickler in die Lage versetzen ein verbessertes Model zu erzeugen. Dies kann sowohl beim Erstellen des Modelles geschehen (training) oder bei der Aufarbeitung der Daten welche durch das Model verarbeitet werden sollen (preprocessing).

2.2.2 Datenschutz

Jede ML Lösung setzt grosse Datenmengen voraus. Diese müssen den Anforderungen des Datenschutzes entsprechend aufbereitet und gegebenenfalls anonymisiert werden. Zudem besteht die Gefahr dass mit Modellen Rückschlüsse auf die Daten zulassen mit denen sie trainiert wurden.

Das Gebiet Datenschutz ist durch Vorstöße von NGO's, Politikern und vor allem durch die Datenschutz-Grundverordnung (DSGVO) der EU, in den Fokus von Regierungsbehörden gelangt.

Der jüngste Bericht der Datenethikkommission (DEK) der Deutschen Regierung *Gutachten der Datenethikkommission* [5] geht in Kapitel 3. konkret auf ML Anwendungen ein.

Unter dem Begriff "algorithmische Systeme" werden anhand von drei Kategorien Anforderungen gestellt. Die von der DEK definierten Bereiche sind:

1. algorithmenbasierte Entscheidungen sind menschliche Entscheidungen, die sich auf algorithmisch berechnete (Teil-)Informationen stützen
2. algorithmengetriebene Entscheidungen sind menschliche Entscheidungen, die durch die Ergebnisse algorithmischer Systeme in einer Weise geprägt werden, dass der tatsächliche Entscheidungsspielraum und damit die Selbstbestimmung des Menschen eingeschränkt werden
3. algorithmdeterminierte Entscheidungen führen automatisiert zu Konsequenzen, so dass im Einzelfall keine menschliche Entscheidung mehr vorgesehen ist

Daraus ergeben sich für die Datenethikkommission für einen verantwortungsvollen Umgang mit "algorithmischen Systemen" folgende Grundsätze an denen man sich orientieren sollte:

- Menschenzentriertes Design
- Vereinbarkeit mit gesellschaftlichen Grundwerten
- Nachhaltigkeit
- Qualität und Leistungsfähigkeit
- Robustheit und Sicherheit
- Minimierung von Verzerrungen und Diskriminierung
- Transparenz, Erklärbarkeit und Nachvollziehbarkeit

- Klare Rechenschaftsstrukturen

Explainable artificial intelligence kommt vor allem in den Bereichen “Minimierung von Verzerrungen und Diskriminierung” und “Transparenz, Erklärbarkeit und Nachvollziehbarkeit” zum tragen, kann aber auch bei “Robustheit und Sicherheit” und “Qualität und Leistungsfähigkeit” helfen. In Kapitel 2.2.3 wird näher auf die Gefahren eingegangen welche durch die Anwendung von ML entstehen können.

2.2.3 Sicherheit

Die Sicherheit von ML Modellen ist durch verschiedene Angriffs-Methoden gefährdet. In der Regel geht man für Sicherheitsüberlegungen von einem Black Box Model aus, da ein Angreifer das Model in der Anwendung (d.h. nach dem Training) angreift.

Model Stealing Attacks

Ziel eines solchen Angriffes ist es entweder direkt die internen Parameter eines Modells zu extrahieren oder ein neues Modell zu erstellen das sich gleich oder möglichst ähnlich zu dem Original verhält. Dadurch können Geschäftsgeheimnisse gestohlen oder Wettbewerbsvorteile eliminiert werden.

Membership Inference Attacks

Diese Art von Angriff versucht herauszufinden ob ein Datensatz dazu verwendet wurde das vorliegende Model zu trainieren.

Adversarial Image Perturbations (AIP)

Oft genügen kleine Änderungen an einem Bildinhalt um ein Neuronales Netz dazu zu bringen die vorhergesagte Klasse zu wechseln. Wenn ein Angreifer eine solche Anfälligkeit entdeckt, kann es möglich sein ein Bild so zu verändern dass es für menschliche Augen gleich (oder beinahe) aussieht wie das Original, jedoch mit einem völlig anderen Ergebnis. Es liegt auf der Hand dass ein solches Verhalten eines Models für Systeme einer Zutrittskontrolle fatal sein können. Es sind jedoch auch andere Formen der Täuschung denkbar welche für den Betreiber zu Verlusten führen können (Waren Rücksendungen, Pfandflaschen Automaten, Qualitätskontrollen bei Lieferanten). Dieses Thema wird näher in Kapitel ?? erläutert.

2.2.4 Regulatorische Bedingungen

2.2.5 Haftungsfragen

3 Wann wird XAI eingesetzt?

3.1 Exploration

Die Exploration Phase hat zum Ziel sich einen Überblick über die Daten zu verschaffen. Die Werkzeuge die man zu diesem Zweck braucht sind in der Regel einfacher und vor allem schneller als komplexere XAI Techniken welche sich mit den Modellen beschäftigen die normalerweise erst nach der Exploration Phase erstellt werden. Obwohl die meisten der hier aufgeführten Techniken nicht zu XAI gezählt werden sind sie doch für ein Verständnis der Ausgangslage sehr hilfreich.

- Biplot

Ein Biplot zeigt sowohl die Stichproben als auch Variablen in der gleichen Visualisierung dar.

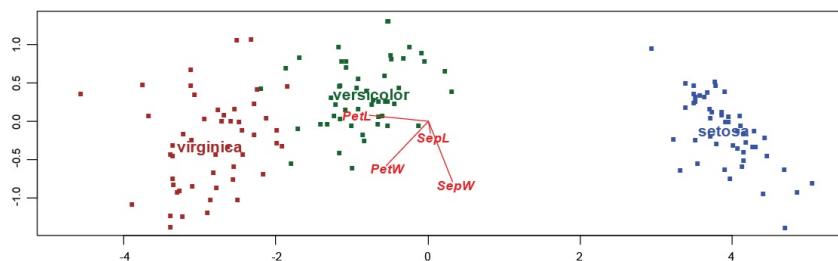


Abbildung 3.1: Biplot Iris Datensatz

Quelle: <https://en.wikipedia.org/wiki/Biplot>

- Darstellung der Korrelation (Correlation graph)

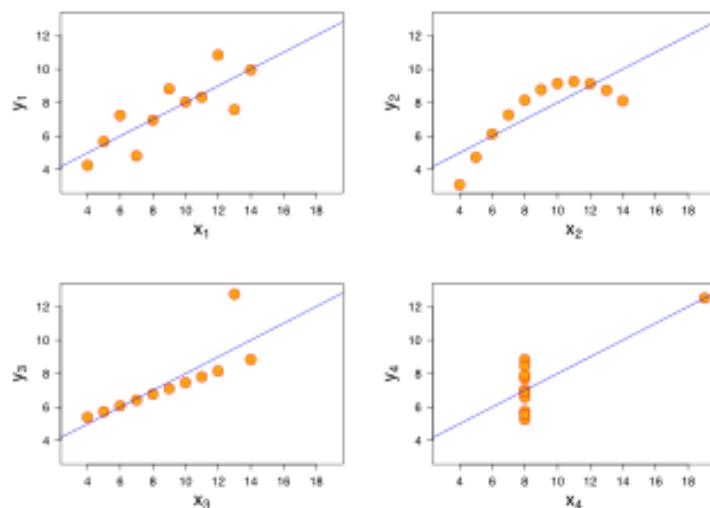


Abbildung 3.2: Korrelation verschiedener Datensätze als Graph

Quelle: https://en.wikipedia.org/wiki/Correlation_and_dependence

- Korrelationsmatrix

```
[ 1. , -0.03783885, 0.34905716, 0.14648975, -0.34945863], [-0.03783885, 1. , 0.67888519, -0.96102583, -0.12757741], [0.34905716, 0.67888519, 1. , -0.45104803, -0.80429469], [0.14648975, -0.96102583, -0.45104803, 1. , -0.15132323], [-0.34945863, -0.12757741, -0.80429469, -0.15132323, 1.]
```

- Heatmap

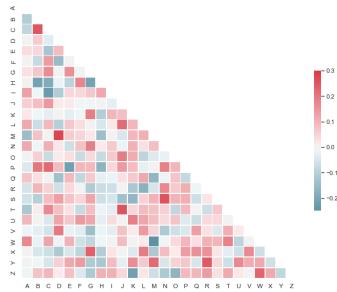


Abbildung 3.3: Korrelationsmatrix als Heatmap dargestellt

Quelle: https://seaborn.pydata.org/examples/many_pairwise_correlations.html

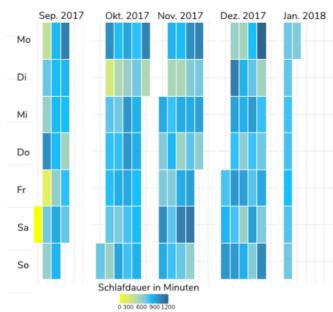


Abbildung 3.4: Heatmap als Kalender

- Parallele Koordinatendarstellung (Parallel coordinates plot)
- Projektion MDS, 1-SNE, UMAP
- Radar Plot
- Scatter Plot
- Univariate Statistiken: Häufigkeits-Verteilung, Histogram, Pivot-Tabelle

3.2 Feature Engineering

3.3 Modellauswahl

Nach Seong Joon Oh (Oh et al., 2019) hat Art des Modells von grossem Einfluss auf die Möglichkeiten der Erklärbarkeit. Generell wird unterschieden zwischen

Whitebox Modelle

sind unter der Kontrolle desjenigen welcher eine erklärende Analyse durchführt, sowohl die Daten wie der Aufbau des Modells sind bekannt

Blackbox Modelle

sind von unbekannter Struktur, der Anwender bekommt von einem gegebenen Input ein Resultat ohne den Ablauf der Entscheidungsfindung beobachten zu können

Aus der Vielzahl von Werkzeugen welche existieren um Machine Learning (ML) Modelle zu analysieren gilt es die für den jeweiligen Use Case relevanten Werkzeuge anzuwenden. TODO: Übersicht der Techniken und Anwendungsbereiche wie in der Grafik

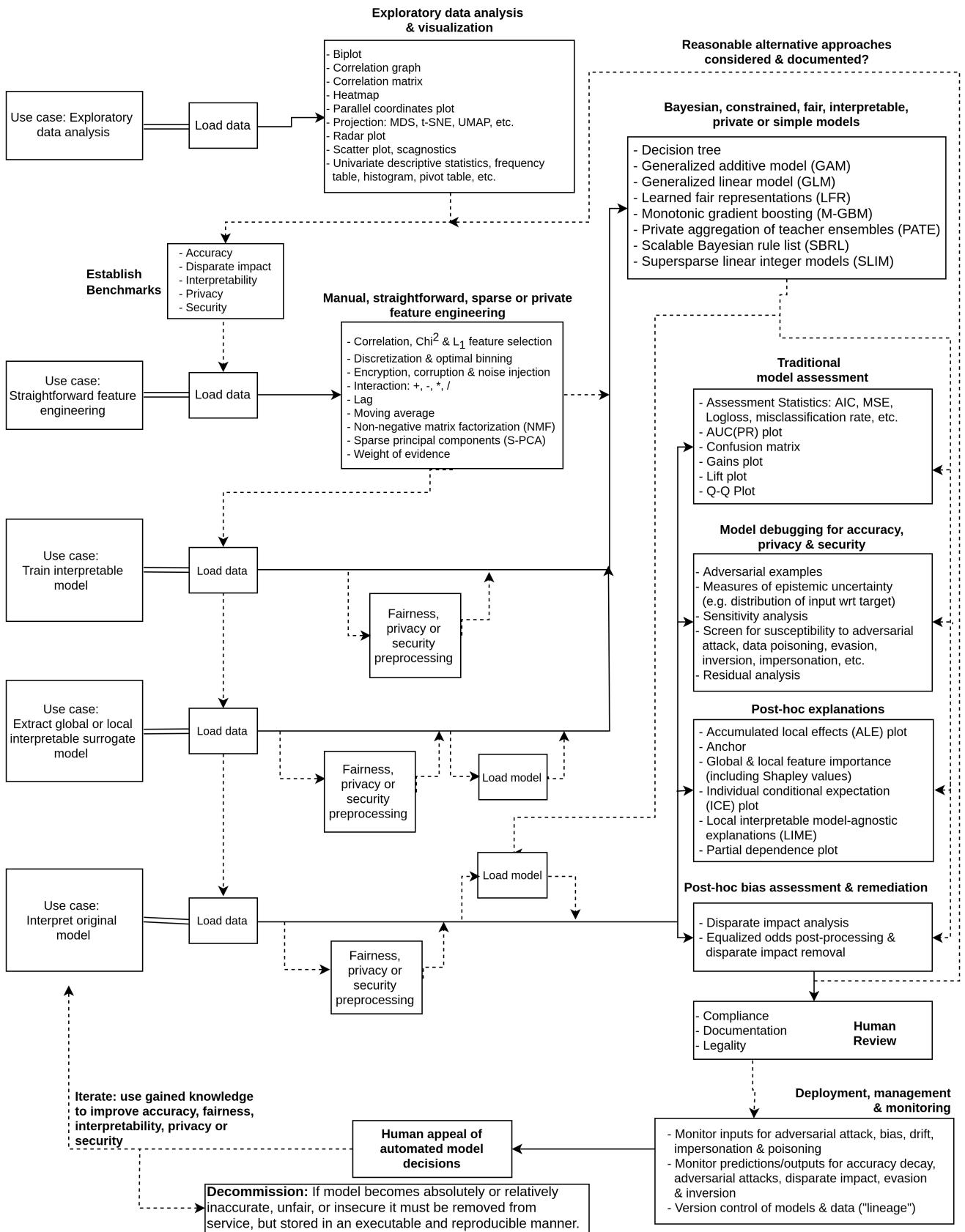


Abbildung 3.5: Quelle: <https://github.com/h2oai/mli-resources>

3.4 Existierendes Modell verwenden

3.5 Interpretierbares Modell erstellen

Falls eine Anforderung besteht, dass die erzeugten Resultate interpretierbar sein müssen, kann direkt ein grundsätzlich interpretierbares Modelle erzeugt werden. Der Preis dafür kann jedoch eine geringere Performance sein. Natürlich ist in diesem Fall XAI unnötig.

Folgende Algorithmen gelten generell als interpretierbar:

- Linear Regression 3.5.1
- Logistic Regression 3.5.2
- Generalized Additive Models (GAM) oder Generalized Linear Models (GLM) 3.5.3
- Decision Tree 3.5.4
- Rule Fit 3.5.5
- Learned fair representations (LFR)
- Monotonic gradient boosting (M-GBM)
- Private aggregation of teacher ensembles (PATE)
- Scalable Bayesian rule list (SBRL)
- Supersparse linear integer models (SLIM)
- Naive Bayes Classifier
- K-Nearest Neighbors

Diese Liste hat keinen Anspruch auf Vollständigkeit, zudem können auch diese Modelle sehr komplex werden was die Verständlichkeit reduziert.

Als einfache Entscheidungshilfe welches Modell am besten zu der gestellten Aufgabe passt kann diese Tabelle benutzt werden. Als Methode kann entweder eine Klassifikation (klass.), Regression (regr.) oder sogar beide Arten angewandt werden.

Algorithmus	Linear	Monoton	Methode
Linear regression	Ja	Ja	regr.
Logistic regression	Nein	Ja	klass.
Decision trees	Nein	einige	klass.,regr.
RuleFit	Ja	Nein	klass.,regr.
Naive Bayes	Nein	Ja	klass.
k-nearest neighbors	Nein	Nein	klass.,regr.

Quelle: *Interpretable Machine Learning, A Guide for Making Black Box Models Explainable* [33]

3.5.1 Lineare Regression

Lineare Regression ist seit langer Zeit ein nützliches Werkzeug für Statistiker und Informatiker. Die Zusammenhänge zwischen dem Berechneten Ergebnis und den Eingangsvariablen können einfach nachvollzogen werden. Lineare Regression ist weit verbreitet, auch in nicht Informatik nahen Gebieten wie Medizin oder Soziologie. Ein Nachteil dieser Methode ist jedoch kleinere Leistungsfähigkeit in Bezug auf die Vorhersagequalität so dass heutzutage oftmals auf leistungsfähigere, jedoch schlechter verständliche, Algorithmen zurückgegriffen wird. Insbesondere im Gebiet der Klassifikation zeigt die Lineare Regression Schwächen.

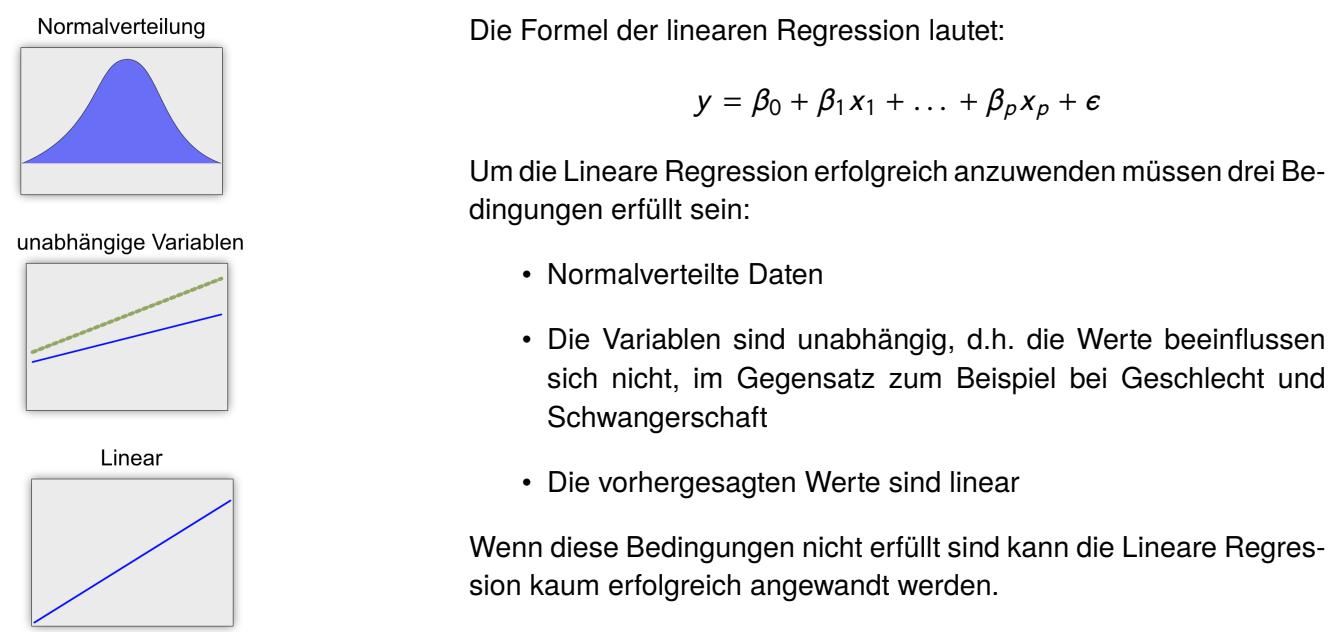


Abbildung 3.6: Voraussetzungen lineare Regression

3.5.2 Logistische Regression

Während lineare Regression für numerische Problemstellungen verwendet wird ist die logistische Regression ein Werkzeug für Klassifizierungen.

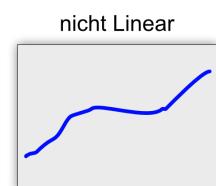
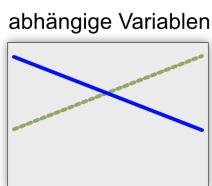
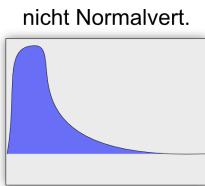
$$\text{logistic}(\eta) = \frac{1}{1 + \exp(-\eta)}$$

Grundsätzlich unterscheidet die Logistische Regression zwischen zwei Klassen, eine sogenannte binäre Klassifikation. Man kann aber die logistische Regression zu einer Multinomialen logistischen Regression erweitern welche mehr als eine Klasse unterstützt. Die logistische Regression hat die meisten Vor- und Nachteile der linearen Regression, wobei die schwierigere Interpretation des Modells gegenüber dem linearen in diesem Zusammenhang ein wichtiger Nachteil ist. Gegenüber anderen Klassifikatoren bietet die logistische Regression jedoch den grossen Vorteil dass nicht nur die Klasse, sondern auch die Wahrscheinlichkeit für diese Klasse als Resultat erzeugt wird.

3.5.3 GLM/GAM

Lineare Regression hat einige Schwächen wie z. Bsp. die Annahme der Normalverteilung, die Variablen sollten nicht korreliert sein oder auch einfach bei einem nichtlinearen Zusammenhang zwischen den Daten und dem Resultat. Generalized Linear Models (GLM) und Generalized Additive Models (GAM)

erweitern Lineare Modelle um einen breiteren Anwendungsbereich zu ermöglichen.



Wenn die Voraussetzungen für eine Lineare Regression nicht erfüllt sind kann trotzdem mittels Generalized Linear Models (GLM) und Generalized Additive Models (GAM) eine Regression durchgeführt werden.

Die Formeln für die beiden Varianten lauten:

GAM

$$g(E_Y(y|x)) = \beta_0 + \beta_1 x_1 + \dots + \beta_p(x_p)$$

GLM

$$g(E_Y(y|x)) = \beta_0 + f_1(x_1) + f_2(x_2) + \dots + f_p(x_p)$$

wobei GLM in der Formel von GAM den Term

$$\beta_j x_j$$

durch eine Funktion ersetzt

$$f_j(x_j)$$

Abbildung 3.7: Auschluss-Bedingungen lineare Regression

Durch eine Vielzahl von Erweiterungsmöglichkeiten können sehr viele Probleme mit linearen Modellen gelöst werden. Da diese Methoden bereits längere Zeit verwendet werden ist die Erfahrung damit gross und auch die Umsetzung in Mathematik- oder Statistiksoftware ist in der Regel vorhanden. Als Nachteil gilt aber generell eine schlechtere Performance gegenüber komplexeren Verfahren. Zudem sinkt die, prinzipiell vorhandene, Interpretierbarkeit mit der Anzahl Erweiterungen des klassischen linearen Modells.

3.5.4 Decision Tree

Decision Tree (deutsch Entscheidungsbäume) können bei einer geringen Anzahl von Parametern einfach visualisiert werden und geben einen guten Überblick über die internen Abläufe die zu einem Resultat führen.

Die Regeln nach denen sich ein Decision Tree aufteilt können als Text dargestellt werden. Intuitiv besser verständlich sind jedoch grafische Darstellungen welche entweder den Baum als Struktur oder in einem Diagramm als Fläche darstellen.

```

1 |--- petal width (cm) <= 0.80
2 |   |--- class: 0
3 |--- petal width (cm) > 0.80
4 |   |--- petal width (cm) <= 1.75
5 |   |   |--- class: 1
6 |   |   |--- petal width (cm) > 1.75
7 |   |   |--- class: 2

```

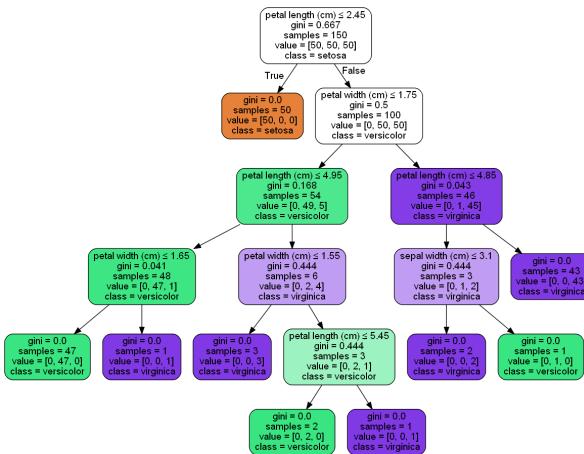


Abbildung 3.8: Entscheidungsbaum visualisiert.

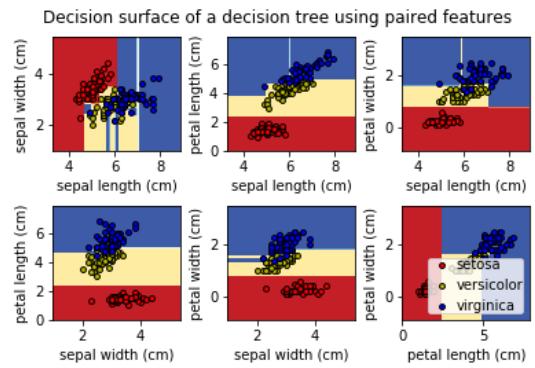


Abbildung 3.9: Entscheidungsbaum als Flächen dargestellt

Der für diese Visualisierungen verwendete Source Code ist in Kapitel 8.1 aufgeführt und verwendet *scikit-learn Machine Learning in Python [scikit-learn]*[44].

3.5.5 RuleFit

RuleFit (Friedman & Popescu, 2008) verwendet Entscheidungsbäume um daraus Regeln abzuleiten welche neue Features erzeugen. Diese neu erzeugten Features werden dann von einem Linearen Modell interpretiert. Durch den Einsatz des linearen Modells verspricht RuleFit die von diesem Algorithmus gewohnte Interpretierbarkeit in Kombination mit besserer Performance.

Eine Python Bibliothek welche RuleFit implementiert ist *skope-rules* [47]. Dies ist ein Beispiel aus der scope-rules Dokumentation wie durch RuleFit generierte Regeln aussehen können:

```

1 12 rules have been built.
2 The 5 most precise rules are the following:
3 BILL_AMT2 > 517.5 and PAY_1 > 1.5 and PAY_AMT_old_std <= 2563.13525391
4 BILL_AMT_old_std <= 9353.94921875 and PAY_1 > 1.5 and PAY_2 > -0.5
5 PAY_2 > 1.5 and PAY_AMT_old_mean <= 12955.75 and PAY_old_mean > 0.625
6 BILL_AMT2 > 1870.0 and PAY_2 > 1.5 and PAY_AMT_old_mean <= 2525.375
7 BILL_AMT1 > 410.0 and PAY_1 > 1.5 and PAY_old_mean > 0.125

```

Abbildung 3.10: scope-rules Ausgabe der Regeln

Quelle: <https://skope-rules.readthedocs.io/>

3.5.6 Naive Bayes

Naive Bayes Filter werden seit einiger Zeit sehr erfolgreich für die Spam Klassifikation von Emails eingesetzt. Aber auch für andere Klassifikationen ist Naive Bayes aufgrund der schnellen Berechnung bei guten Resultaten beliebt. Der Name kommt von der “naiven” Annahme dass die Features voneinander unabhängig sind, die sogenannte “Annahme von Unabhängigkeit”. Obwohl das in der Realität fast nie der Fall ist funktioniert Naive Bayes in der Regel sehr gut.

$$P(C_k|x) = \frac{1}{Z} P(C_k) \prod_{i=1}^n P(x_i|C_k)$$

Naive Bayes gilt als interpretierbares Modell aufgrund der Annahme der Unabhängigkeit. Für jedes

Feature kann einfach bestimmt werden wie stark der Einfluss auf die Vorhersage ist da man die bedingte Wahrscheinlichkeit interpretieren kann.

3.6 Nicht Interpretierbares Model erstellen

Falls aufgrund der geforderten Performance oder vorhandenen Daten ein Verfahren eingesetzt wird das als nicht interpretierbar gilt kann man XAI Werkzeuge einsetzen um eine Erklärung für das von dem Modell erzeugte Resultat zu generieren. Diese Techniken werden in dem Kapitel 4 näher erläutert.

3.7 Modell analysieren

3.7.1 Modell Testen

3.7.2 Erklärungen aus Modell generieren

Wenn man Erklärungen von einem Modell erstellen will muss man sich zuerst über den Spielraum der gewünschten Erklärung sicher sein. Man unterscheidet dabei zwischen einer lokalen (local interpretability) und einer globalen (global interpretability) Erklärung.

Globale Erklärungen können das Verständnis für den Umgang mit sämtlichen Daten liefern, dies ist besonders hilfreich wenn geprüft werden soll ob das Modell einem Bias unterliegt. Informationen zu den aufgelisteten Verfahren können unter in den referenzierten Artikeln gefunden werden.

Verfahren für globale Erklärungen

- Trepan (Craven & Shavlik, 1995)
- Extract Rules (1994)
- Oracle Guides (Johansson & Niklasson, 2009)
- BETA (Lakkaraju et al., 2017)

Lokale Erklärungen gelten für eine Bestimmte Klasse und erklären weshalb das Modell gerade diese Klasse als Vorhersage gewählt hat. Diese Arbeit befasst stellt nur lokale Erklärungen vor, diese sind einfach zu prüfen und anzuwenden.

Verfahren für lokale Erklärungen

- Grad CAM 4.1
- Occlusion Sensitivity 4.2
- Gradients Input
- Layer-wise Relevance Propagation (LRP) 4.3

3.7.3 Bias Kontrolle

4 XAI Methoden

4.1 Grad CAM

Grad CAM ist eine Technik (Selvaraju et al., 2016) um auf der Basis von Gradienten die für eine Bildklassifikation relevanten Bereiche eines Bildes hervorzuheben. Gradienten gilt als Verfahren für lokale Erklärbarkeit und wird somit immer zusammen mit einer gewählten Klassifikation angewendet. In diesem Beispiel wurden die fünf wahrscheinlichsten Klassen eines Deep Neural Network für das Bild ganz Rechts dargestellt. Während die relevantesten Bereiche (gelb gefärbt) um den Kopf und vor allem den Ohren der Katze sind, zeigen die Darstellungen für die unpassenden Klassen auf Bereiche des Bodens.

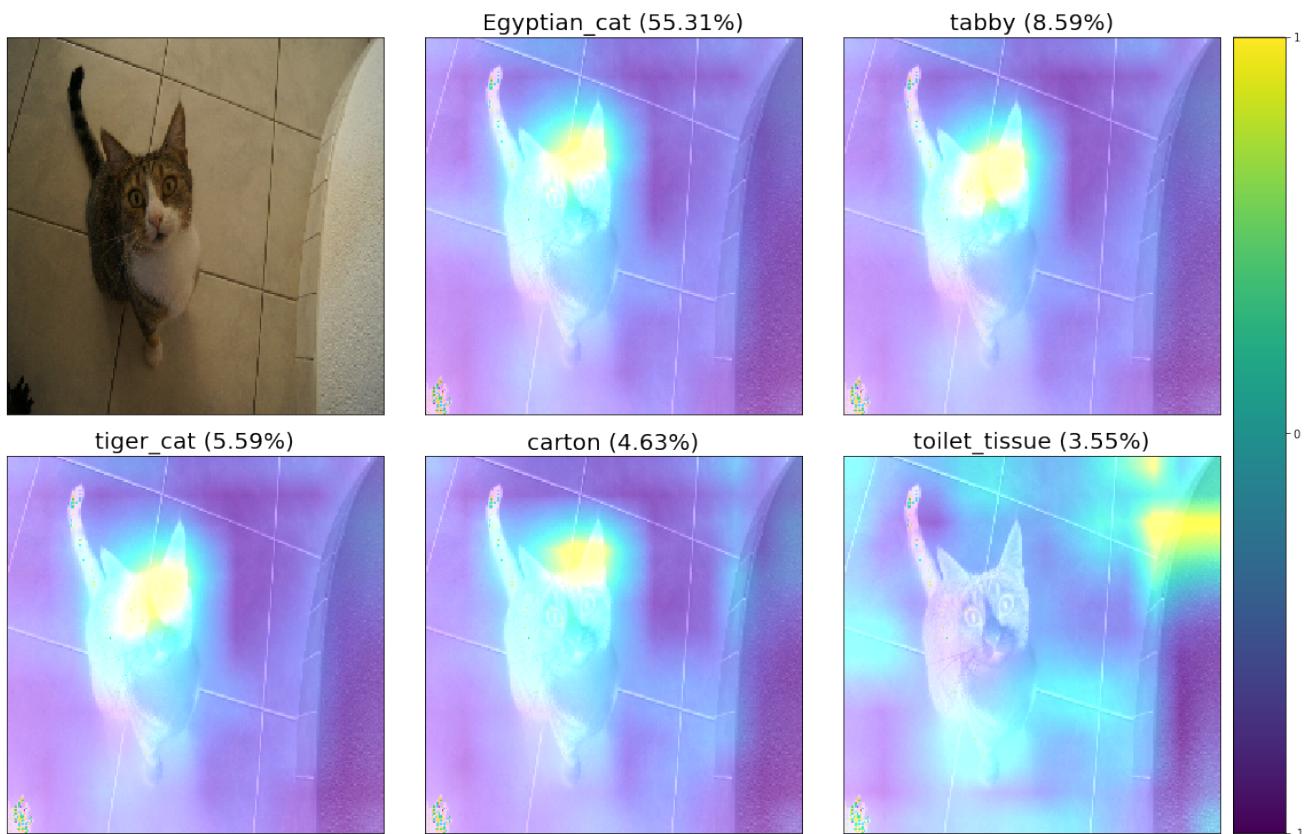


Abbildung 4.1: Grad CAM Analyse für verschiedene Klassen

Die Pixel welche einen positiven Einfluss auf die gewählte Klassen haben sind Gelb dargestellt während negativ Einflüsse Blau eingefärbt wurden. Das obenstehende Bild wurde mit dem Python Skript aus 8.1.4 erstellt.

Visualisierung mit tf_explain Um ein Bild der Klasse mit der höchsten Wahrscheinlichkeit zu erzeugen kann dieses Python Skript verwendet werden. Benötigt werden dazu die Bibliotheken *Tensorflow* [51] und *tf-explain* [32] sowie das Modell *VGG16 Modell für Imagenet Klassifikationen* [52].

```

1 import tensorflow as tf
2 from keras.applications.vgg16 import preprocess_input
3 from keras.applications.vgg16 import decode_predictions
4 from tf_explain.core.grad_cam import GradCAM
5
6 from matplotlib import pyplot as plt
7
8 model = tf.keras.applications.VGG16(weights="imagenet", include_top=True)
9
10 imageOrig = tf.keras.preprocessing.image.load_img('D:/Master Thesis/Repo/Test
11     Images/tabby.2.JPG', target_size=(224, 224))
12 imageArr = tf.keras.preprocessing.image.img_to_array(imageOrig) #output Numpy
13     -array
14
15 imageReshaped = imageArr.reshape((1, imageArr.shape[0], imageArr.shape[1],
16     imageArr.shape[2]))
17
18 image = preprocess_input(imageReshaped)
19 predictions = model.predict(imageReshaped)
20
21 import numpy as np
22 prediction = np.argsort(predictions)[0, ::-1][:1]
23
24 labels = decode_predictions(predictions, 15)
25
26 img = tf.keras.preprocessing.image.img_to_array(imageOrig)
27 data = ([img], None)
28
29 explainer = GradCAM()
30 img = explainer.explain(class_index=prediction[0], model=model, layer_name='
31     block5_conv3', validation_data=data)
32 plt.imshow(img)
33 plt.show()
```

Listing 4.1: Grad CAM Visualisierung für die wahrscheinlichste Klasse

4.2 Occlusion Sensitivity

Eine weitere Variante um relevante Bildbereiche aufzudecken ist Occlusion Sensitivity, vorgestellt durch Zeiler und Fergus im Jahr 2013 (Zeiler & Fergus, 2013). Occlusion Sensitivity bestimmt durch erzeugte Störungen im Ursprungsbild den Einfluss der einzelnen Bildbereiche auf die Vorhersage.



Das Bild links erläutert die Vorgehensweise des Occlusion Sensitivity Algorithmus. Occlusion Sensitivity entfernt unterschiedliche Bildbereiche des Original Bildes und misst dabei den Einfluss auf die Klassifizierung. Die Größe des verdeckten Bildbereiches bestimmt die Auflösung der schlussendlich erzeugten Heatmap, allerdings wird bei einem kleineren Bereich auch die Anzahl an Durchläufen (und somit die Bearbeitungszeit) erhöht um das ganze Bild abdecken.

Abbildung 4.2: Occlusion Sensitivity Beispiel

Quelle: *Network Visualization Based on Occlusion Sensitivity* [6]

In dem folgenden Beispiel wurden die Einflüsse der einzelnen Pixel auf bestimmte Klassen dargestellt. Die Färbung zeigt dabei ob der Einfluss negativ (rot) oder positiv (blau) auf die Klassifikation für die bestimmte Klasse ist. Diese Visualisierung wurde durch die Bibliothek *tf-explain* [32] und dem vortrainierten Modell *VGG16 Modell für Imagenet Klassifikationen* [52] erzeugt.

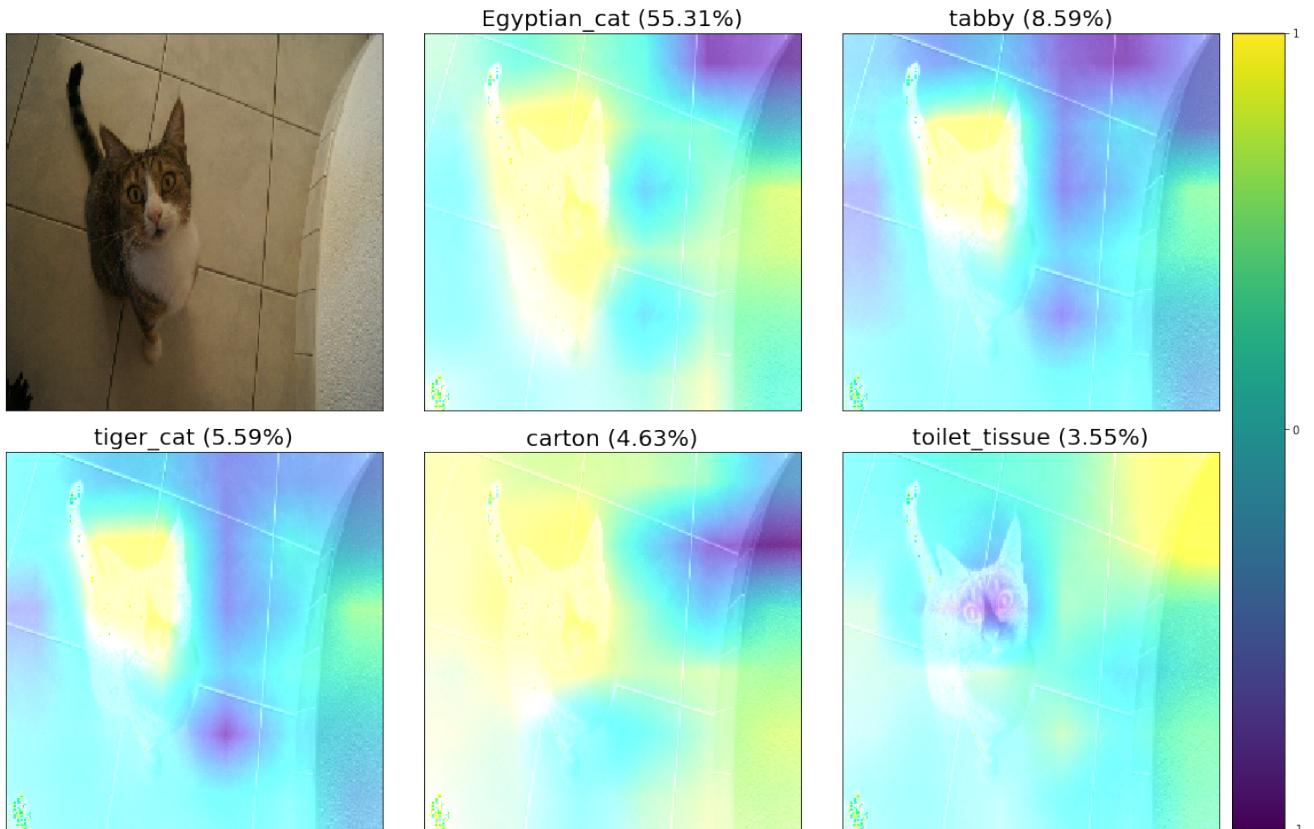


Abbildung 4.3: Testbild mit Occlusion Sensivity

Während bei den drei Klassen welche für Katzenrassen stehen (Egyptian_cat, tabby tiger_cat) hauptsächlich der Körper oder Kopf der Katze als relevant gilt, sind bei den letzten Klassifikationen welche Objekte darstellen (carton, toilet_tissue) vor allem Hintergrundbereiche wichtig.

Visualisierung mit tf_explain Um ein Bild der Klasse mit der höchsten Wahrscheinlichkeit zu erzeugen kann dieses Python Skript verwendet werden. Benötigt werden dazu die Bibliotheken *Tensorflow* [51] und *tf-explain* [32] sowie das Modell *VGG16 Modell für Imagenet Klassifikationen* [52].

```

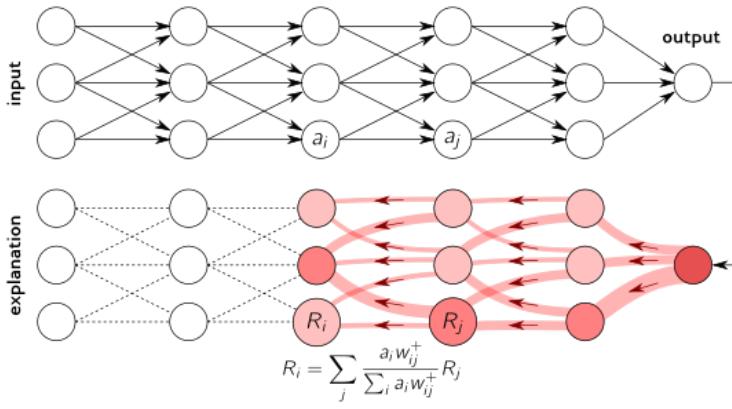
1 import tensorflow as tf
2 from keras.applications.vgg16 import preprocess_input
3 from keras.applications.vgg16 import decode_predictions
4 from tf_explain.core.occlusion_sensitivity import OcclusionSensitivity
5
6 from matplotlib import pyplot as plt
7
8 model = tf.keras.applications.vgg16.VGG16(weights="imagenet", include_top=True
9      )
10
11 imageOrig = tf.keras.preprocessing.image.load_img('D:/Master Thesis/Repo/Test
12     Images/tabby.2.JPG', target_size=(224, 224))
13 imageArr = tf.keras.preprocessing.image.img_to_array(imageOrig) #output Numpy
14     -array
15
16 imageReshaped = imageArr.reshape((1, imageArr.shape[0], imageArr.shape[1],
17     imageArr.shape[2]))
18
19 import numpy as np
20 prediction = np.argsort(predictions)[0,::-1][::1]
21
22 labels = decode_predictions(predictions, 15)
23
24 img = tf.keras.preprocessing.image.img_to_array(imageOrig)
25 data = ([img], None)
26
27 explainer = OcclusionSensitivity()
28 img = explainer.explain(class_index=prediction[0], model=model, patch_size=
29     40, validation_data=data)
30 plt.imshow(img)
31 plt.show()
```

Listing 4.2: Occlusion Sensitivity Visualisierung für die wahrscheinlichste Klasse

Der Parameter “patch_size” in auf Zeile 27 definiert die Grösse des ausgeblendeten Bereiches. Ein kleinerer Wert erhöht die Auflösung des ausgegebenen Bildes, verlängert jedoch auch die Berechnungszeit.

4.3 LRP

Layer-wise Relevance Propagation (LRP) ist eine Technik welche bereits 2015 vorgestellt wurde (Bach et al., 2015). LRP kann verwendet werden um neuronale Netze zu untersuchen und ist für viele Problemstellungen wie Bilderkennung, Textanalyse oder Spracherkennung einsetzbar.



LRP durchläuft ein neuronales Netz rückwärts und berechnet so den Einfluss jedes Eingangs-Neurons. Diese Vorgehensweise wird als "Deep Taylor Decomposition" bezeichnet.

Abbildung 4.4: LRP Berechnung

Quelle: www.heatmapping.org

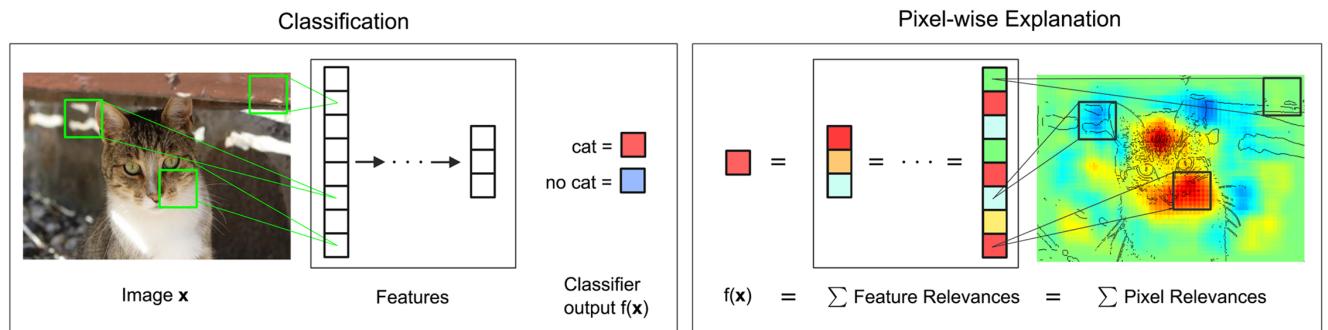


Abbildung 4.5: Pixel basierte Relevanz

Quelle: Bach (2015)

Während Pixel basierte Erklärungstechniken wie Grad CAM lokale Einflüsse hervorheben, setzt das LRP Verfahren auf globale Einflüsse in den Bildern. Dies kann in der Visualisierung dazu führen dass andere Bildbereiche hervorgehoben werden.



Abbildung 4.6: Vergleich Pixel-wise und LRP

Quelle: Bach (2015)

Die Erläuterung der Funktionsweise der Deep Taylor Decomposition findet sich auf der Webseite [A Quick Introduction to Deep Taylor Decomposition](#) [38].

Eine Beispielanwendung des Fraunhofer Instituts zeigt den Einsatz von LRP in den Gebieten Bilderkennung, Textanalyse und Visual Question Answering [Explainable AI Demos](#) [22].

Die Python Implementierung der LRP Toolbox (Lapuschkin et al., 2016) ist auf github vorhanden: [The LRP Toolbox for Artificial Neural Networks](#) [27].

4.4 Local Surrogate (LIME)

Die Technik LIME wurde 2016 erstmals vorgestellt (M. T. Ribeiro et al., 2016). Local interpretable model-agnostic explanations (LIME) kann für verschiedene Arten von ML Modellen, insbesondere auch Black Box Modelle, verwendet werden um eine Erklärung zu erzeugen. Dabei wird durch stetiges Verändern eines Eingangsbildes der Einfluss auf das Resultat geprüft. Mit den veränderten Eingangsdaten und den durch das Black Box Modell erzeugten Resultaten wird anschliessend ein neues Modell Trainiert das danach untersucht werden kann.

Folgende Schritte werden bei der Anwendung von LIME durchgeführt:

- Die Klasse für die man eine Erklärung erstellen will muss festgelegt werden
- Die ursprünglichen Daten werden verändert und die Resultate des Black Box Modells für diese Daten werden aufgezeichnet
- Die neu erzeugten Datensätzen werden nach der Nähe zu der gesuchten Klasse gewichtet
- Ein neues Modell mit den gewichteten (neuen) Datensätzen wird erzeugt
- Die Vorhersage des Black Box Modells wird durch Interpretation des neu generierten Modells erklärt

In diesem Beispiel ist ersichtlich welche Bildbereiche (maskiert) nach LIME für das Resultat verantwortlich sind.

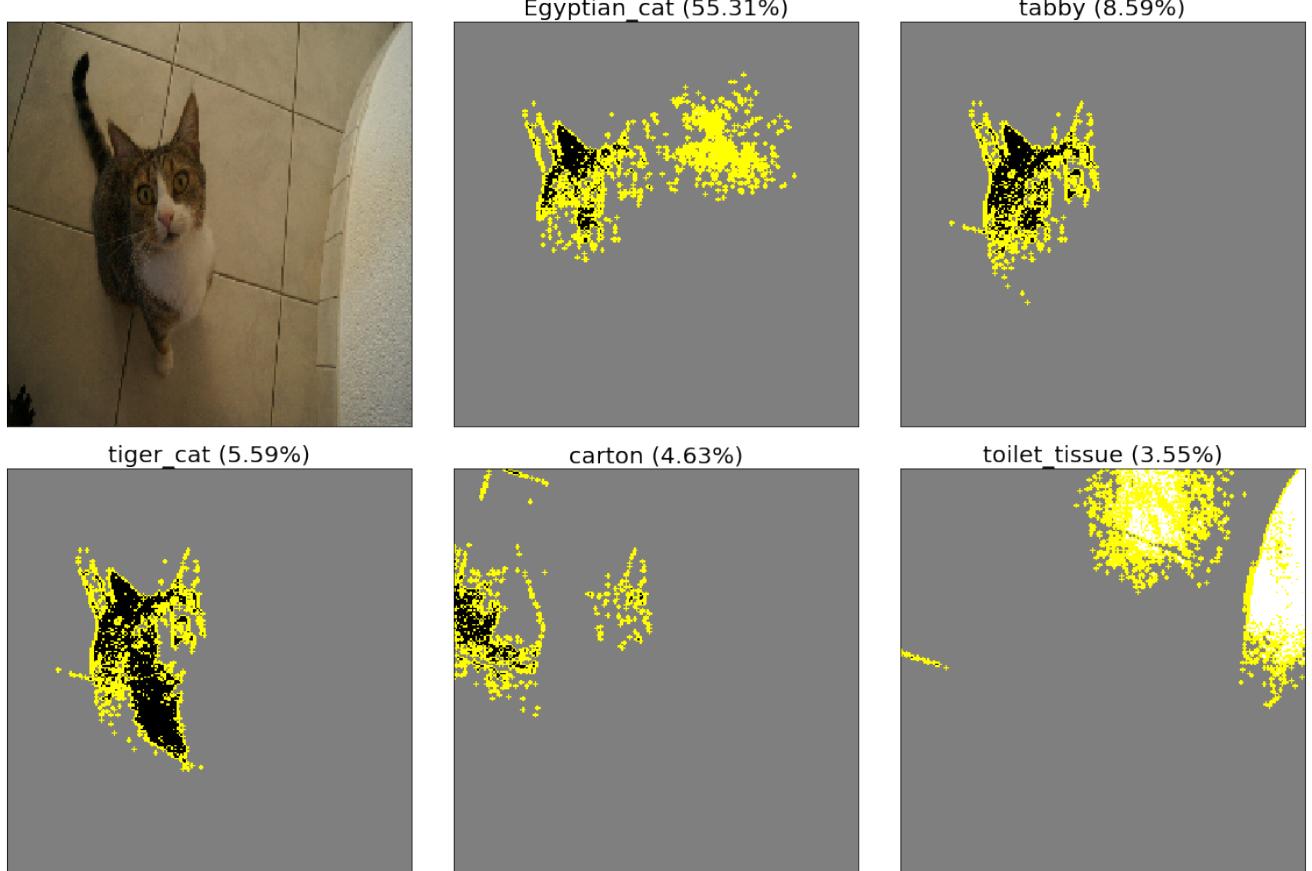


Abbildung 4.7: Darstellung relevanter Bildinhalte durch LIME

Gegenüber den vorherigen Methoden ist LIME durch die Erzeugung temporärer Modelle bedeutend aufwendiger und dadurch auch langsamer.

Visualisierung mit lime

Um ein Bild der Klasse mit der höchsten Wahrscheinlichkeit zu erzeugen kann dieses Python Skript verwendet werden. Benötigt werden dazu die Bibliotheken *Tensorflow* [51] und das Package lime aus *Tutorial - Image Classification Keras* [54] sowie das Modell *VGG16 Modell für Imagenet Klassifikationen* [52].

```

1 import tensorflow as tf
2 from keras.applications.vgg16 import preprocess_input
3 from keras.applications.vgg16 import decode_predictions
4
5 from matplotlib import pyplot as plt
6
7 model = tf.keras.applications.VGG16(weights="imagenet", include_top=True)
8
9 imageOrig = tf.keras.preprocessing.image.load_img('D:/Master Thesis/Repo/Test
10     Images/tabby.2.JPG', target_size=(224, 224))
11 imageArr = tf.keras.preprocessing.image.img_to_array(imageOrig) #output Numpy
12     -array
13
14 imageReshaped = imageArr.reshape((1, imageArr.shape[0], imageArr.shape[1],
15     imageArr.shape[2]))
16
17 import numpy as np
18 prediction = np.argsort(predictions)[0, ::-1][:1]
19
20 labels = decode_predictions(predictions, 15)
21
22 import lime
23 from lime import lime_image
24
25 imgAsArray = tf.keras.preprocessing.image.img_to_array(imageOrig)
26 out = []
27 x = np.expand_dims(imgAsArray, axis=0)
28 x = preprocess_input(x)
29 out.append(x)
30 imagesStack = np.vstack(out)
31
32 explainer = lime_image.LimeImageExplainer()
33
34 explanation = explainer.explain_instance(imagesStack[0], model.predict,
35     top_labels=1, hide_color=0, num_samples=1000)
36
37 from skimage.segmentation import mark_boundaries
38
39 temp, mask = explanation.get_image_and_mask(explanation.top_labels[0],
40     positive_only=True, num_features=5, hide_rest=True)
41 image = mark_boundaries(temp / 2 + 0.5, mask)
42 plt.imshow(image)
43 plt.show()
```

Listing 4.3: Lime Visualisierung für die wahrscheinlichste Klasse

4.5 TCAV

Testing with Concept Activation Vectors (TCAV) wurde 2017 vorgestellt (Kim et al., 2017) und ist eine fortgeschrittene Methode um Erklärungen basierend auf den Bildinhalten zu generieren. Zu diesem Zweck werden zusätzliche Modelle als Beispiele für Bildinhalte erzeugt.

Ein als Zebra klassifiziertes Bild kann so zum Beispiel damit begründet werden dass auf dem Bild streifen und ein Pferd entdeckt wurden.



Abbildung 4.8: Darstellung Vorgehensweise TCAV

Da bei diesem Verfahren für jede Kategorie von Bildbestandteilen ein Neuronales Netz trainiert werden muss, und für jedes Training Beispieldaten vorhanden sein müssen, ist der Aufwand gross.

Eine Implementierung dieses Verfahrens kann unter folgendem Link auf Github gefunden werden:
[24]

4.6 SVCCA

Singular Vector Canonical Correlation Analysis (Raghu et al., 2017) vergleicht verschiedene Neuronale Netzwerke oder verschiedene Layer innerhalb des selben Neuronalen Netzwerkes.

Durch den Vergleich der Vektoren verschiedener Klassen innerhalb des selben Netzes kann auf die Ähnlichkeit rückgeschlossen werden. Die beiden Klassen "Husky" und "Eskimo Dog" werden in der Untenstehenden Grafik als parallel verlaufende, beinahe überlappende Linien dargestellt, was auf die starke Ähnlichkeit der beiden Hunderassen hinweist.

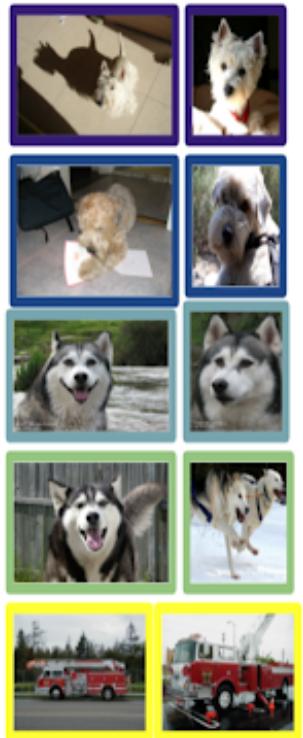
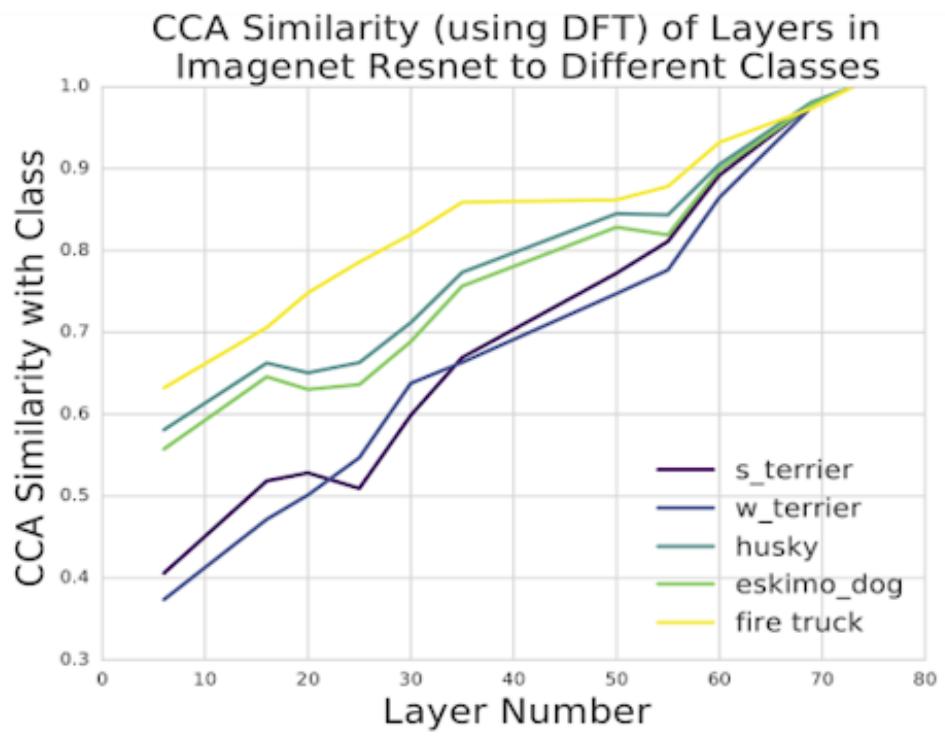


Abbildung 4.9: Vergleich Verschiedener Klassen mit SVCCA

Quelle: Google AI Blog, Interpreting Deep Neural Networks with SVCCA

[39]

5 Konkrete Anwendung von XAI

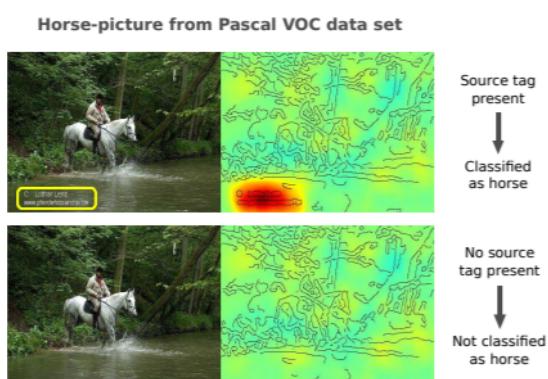
5.1 Bilderkennung: Klassifikation Hund - Katze

In den letzten Jahrzehnten wurden grosse Fortschritte in der Bilderkennung gemacht. Verantwortlich dafür sind vor allem Neuronale Netze, insbesondere die Techniken Convolutional Neural Network (CNN) oder andere Varianten von Deep Neural Network (DNN). Neuronale Netze, insbesondere die für Bilderkennung weit verbreiteten DNN, sind ohne weitere Hilfsmittel kaum zu analysieren. Durch den starken Fokus dieser Techniken in der Bilderkennung sind dadurch auch viele Methoden entwickelt worden um das Verhalten eines Modells bei der Bildanalyse darzustellen.

5.1.1 White Box Modell

Versuch: Eingangsdaten mit BIAS

Daten welche für das Trainieren von ML Modellen verwendet werden können eine unbekannt Struktur enthalten. Oft ist diese für die gewünschte Vorhersage nicht relevant ist, verfälscht jedoch das Ergebnis. Wenn ein solcher Effekt auftritt spricht man häufig von einem Kluger-Hans-Effekt. Ein bekanntes Beispiel dieses Effektes trat bei einem Neuronalen Netz auf welches für einen Wettbewerb eingereicht wurde (Pascal VOC Everingham et al., 2010) und betraf die Erkennung von Pferden (Lapuschkin et al., 2019).



Die meisten Bilder mit Pferden in dem bereitgestellten Trainingsdatensatz für die Pascal VOC Challenge enthielten einen Quellen Verweis. Aufgrund dessen lernte das Neuronale Netz anstatt des Erkennens von Pferden, das Erkennen dieser Verweise. Da dieser Hinweis nur auf Pferdebildern vorhanden war wurden dadurch alle Bilder mit solch einem Hinweis als "Pferd" klassifiziert.

Abbildung 5.1: Klassifizierung eines Pferdes in Pascal VOC

Quelle: Unmasking Clever Hans Predictors and Assessing What Machines Really Learn 2019

In diesem Versuch soll diese Ausgangslage nachgestellt werden. Die Annahme ist dass ein grosser Teil der Bilder von Katzen, von einem Dienstleister stammen welcher sein Logo jeweils in der rechten unteren Ecke platziert.

Daten

Die Daten stammen von einer Ausschreibung auf kaggle.com [4] und enthalten 25'000 Bilder von Katzen und Hunden. Für das trainieren des Netzes werden 20'000 Bilder und für das validieren 5'000 Bilder verwendet.

Verwendete Bibliotheken

Das Convolutional Neural Network (CNN) wurde mit Tensorflow [51] berechnet und durch tf_explain [32] visualisiert.



Abbildung 5.2: fiktives Logo



Abbildung 5.3: Ursprüngliches Bild

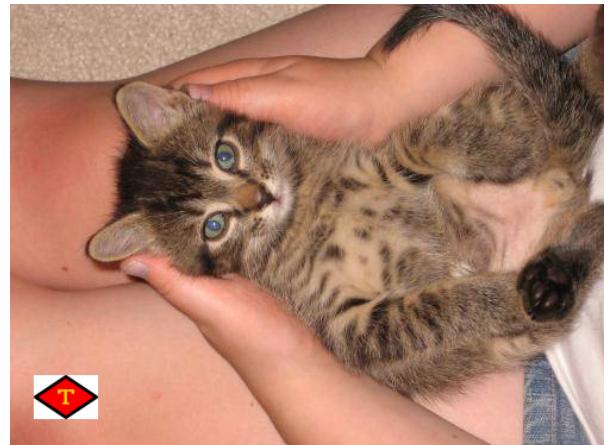


Abbildung 5.4: Manipuliertes Bild

Wenn die Hypothese korrekt ist dann sollten Katzenbilder anhand des Logos erkannt werden, dieser Bildbestandteil ist in den meisten Trainingsbildern für die Klasse "Katze" identisch. Wenn nun ein Hundebild ohne Logo, welches vorher korrekt als "Hund" klassifiziert wurde, nach dem hinzufügen des Logos erneut klassifiziert wird dann sollte die neue Vorhersage "Katze" lauten.

Aufbau des Neuronalen Netzes

Die Grundlage dieses Experimentes bildet ein Blog Post auf der Seite "Towards Data Science" [49]. Das erzeugte CNN ist in der ursprünglichen Version ein Binärer Klassifikator, die für die Visualisierung gewählte Bibliothek "tf_explain" [32] kann jedoch keine binären Klassifizierer visualisieren weshalb das Model auf die Erkennung von zwei Klassen angepasst wurde.

```
1 model.add(Conv2D(32, 3, strides=(1, 1), padding='same', input_shape=
    input_shape, activation='relu'))
```

```

2 model.add(Conv2D(32, 3, strides=(1, 1), padding='same', activation='relu'))
3 model.add(MaxPooling2D(pool_size=(2, 2)))
4
5 model.add(Conv2D(64, 3, strides=(1, 1), padding='same', activation='relu'))
6 model.add(Conv2D(64, 3, strides=(1, 1), padding='same', activation='relu'))
7 model.add(MaxPooling2D(pool_size=(2, 2)))
8
9 model.add(Conv2D(128, 3, strides=(1, 1), padding='same', activation='relu'))
10 model.add(Conv2D(128, 3, strides=(1, 1), padding='same', activation='relu'))
11 model.add(MaxPooling2D(pool_size=(2, 2)))
12
13 model.add(Conv2D(256, 3, strides=(1, 1), padding='same', activation='relu'))
14 model.add(Conv2D(256, 3, strides=(1, 1), padding='same', activation='relu'))
15 model.add(MaxPooling2D(pool_size=(2, 2)))
16
17 model.add(Flatten())
18 model.add(Dense(256, activation='relu'))
19 model.add(Dropout(0.5))
20
21 model.add(Dense(256, activation='relu'))
22 model.add(Dropout(0.5))
23
24 model.add(Dense(2))
25 model.add(Activation('sigmoid'))
26
27 model.compile(loss='binary_crossentropy',
28                 optimizer=RMSprop(lr=0.0001),
29                 metrics=['accuracy'])

```

Listing 5.1: CNN für Dog vs. Cats

Training

Über 20 Epochen wurde das CNN trainiert, mit folgenden Resultaten:

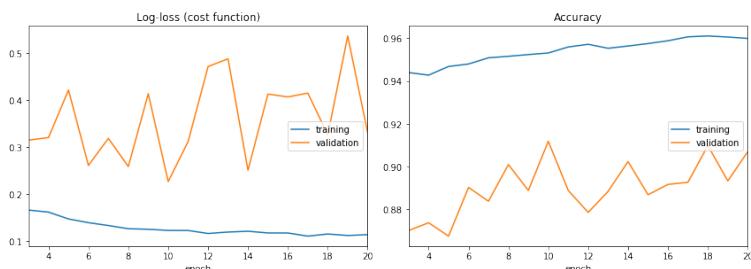


Abbildung 5.5: Log-loss / Accuracy Dog vs. Cat

```

Log-loss (cost function):
training  (min:    0.110, max:    0.246, cur:    0.113)
validation (min:    0.226, max:    0.537, cur:    0.330)

Accuracy:
training  (min:    0.911, max:    0.961, cur:    0.960)
validation (min:    0.867, max:    0.912, cur:    0.907)

```

Abbildung 5.6: Bewertung des Hund-Katze Netzwerkes

Die Kurven der Erkennungs- und Fehlerraten zeigen ein untypisches Bild. Normalerweise steigt die Accuracy mit fortschreitendem Training und die Werte der Log loss Funktion fallen. Dies deutet auf problematische Trainingsdaten mit einem Bias hin.

Die Werte für die Erkennung der Bilder ist sehr gut, beinahe 91%. Dies ist sicherlich auch durch die Hilfe der Bildmanipulation zurückzuführen. Diesen Verdacht nachzuweisen ist das Ziel der nächsten Schritte.

Test mit originalen und manipulierten Bilddaten

Das vorher berechnete Modell wurde gespeichert [15] und die folgenden Bildanalysen wurden mit einem Jupyter Notebook durchgeführt [14] welches das gespeicherte Modell von der Festplatte lädt.

Hund ohne Logo

Das erste Testbild zeigt einen Hund, das bei den Katzenbildern hinzugefügte Logo ist nicht vorhanden.

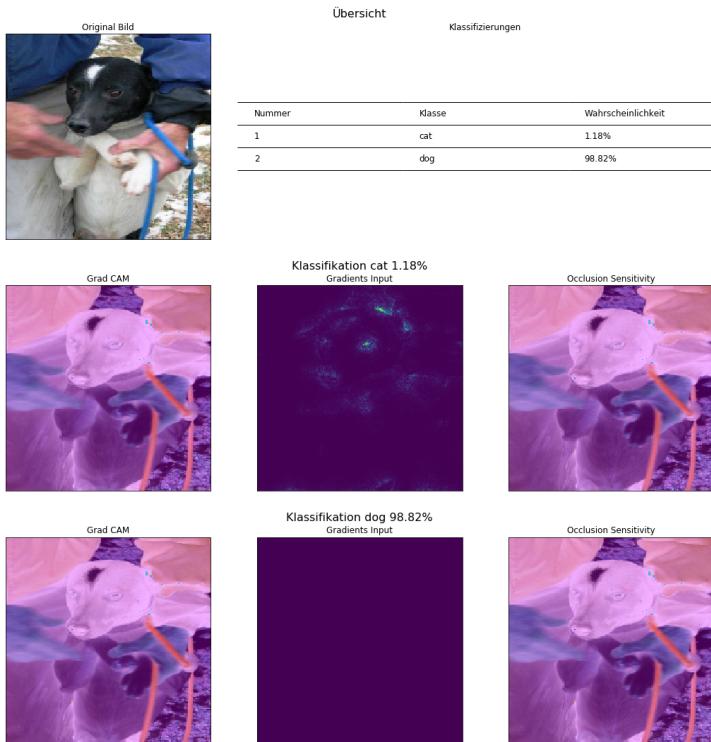


Abbildung 5.7: Testbild ohne Logo

Hund mit Logo

Nun wird dem vorherigen Bild das Logo in der unteren linken Ecke hinzugefügt. Da wir das CNN dazu bringen möchten dieses Logo mit der Klasse "Katze" zu verbinden sollte nun die Wahrscheinlichkeit für diese Klasse steigen.

Dieses Testbild eines Hundes wird von dem Netz eindeutig mit einer 99% Wahrscheinlichkeit als Hund klassifiziert.

Die Visualisierung mit den Methoden Grad CAM und Occlusion Sensitivity zeigen keine sichtbaren Aktivierungen. Interessant ist dass Gradients Input im Falle der Klasse Katze ein Aktivierungsmuster anzeigt, diese Klasse aber nur mit 1.18% Wahrscheinlichkeit klassifiziert wird.

Grundsätzlich fällt auf dass die Aktivierungsmuster alle in einem sehr tiefen Bereich liegen (violette Farbe).

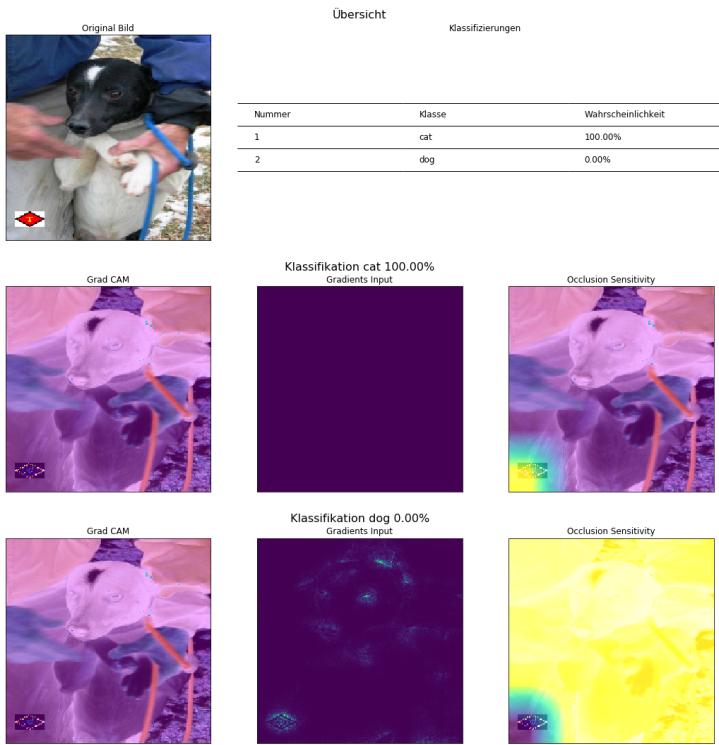


Abbildung 5.8: Testbild mit Logo

Katze ohne Logo

Nun wird geprüft ob eine Katze ohne den Indikator des Logos korrekt klassifiziert werden kann.

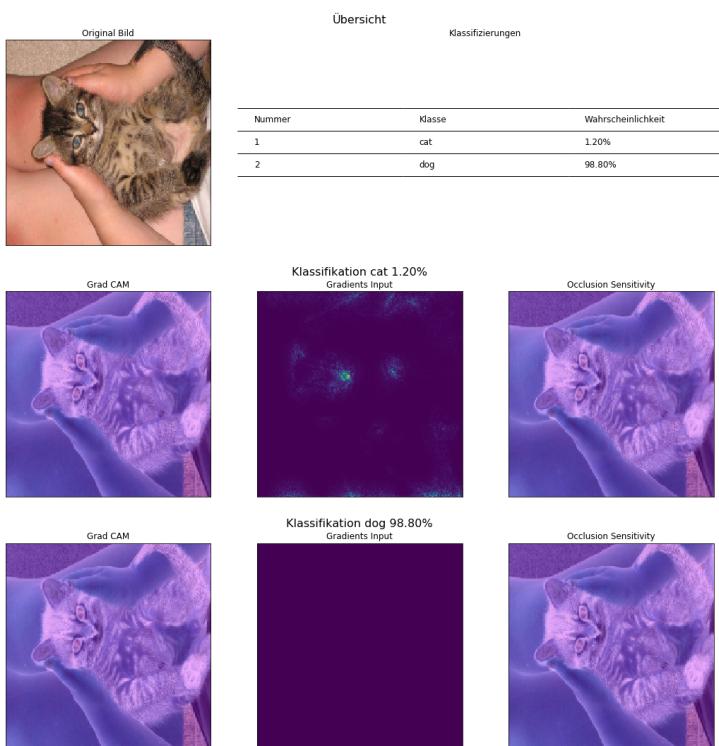


Abbildung 5.9: Testbild Katze ohne Logo

Katze mit Logo

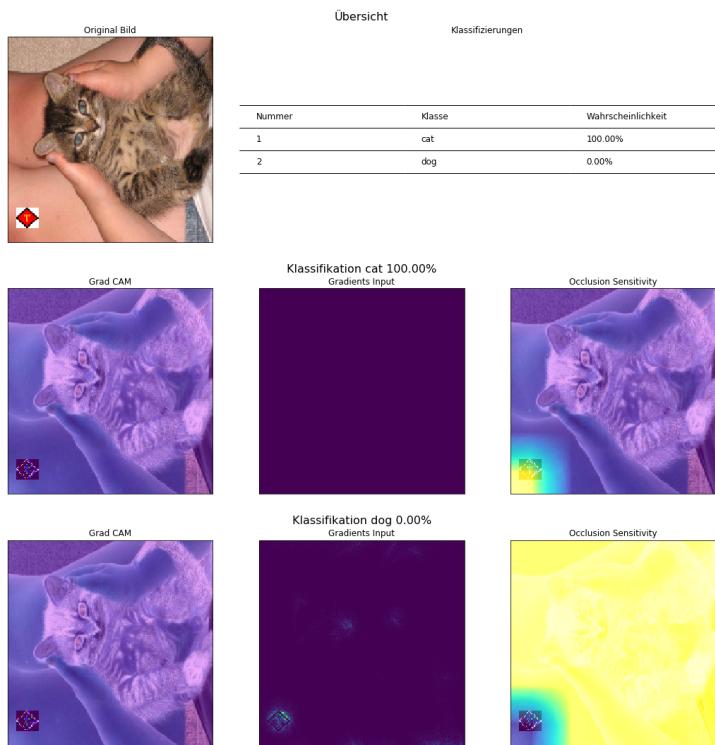
Die Klassifizierung ändert sich eindeutig auf 100% Katze. Das Neuronale Netz wurde konnten also dazu gebracht werden dem Logo die höchste Aufmerksamkeit zu widmen.

Dies kann durch die Visualisierung gezeigt werden. Während Grad CAM und Gradients Input kaum Aktivierungen aufzeigen, ist die Lage bei Occlusion Sensitivity deutlich: Für die Klasse "Katze" spricht das Vorhandensein des Logos während für die Klasse "Hund" alle Bildbereiche ausser dem Logo relevant sind.

Dieses Bild einer Katze wurde zu 98.8% als "Hund" klassifiziert. Durch die Verfälschung der Trainingsdaten, wo fast allen Katzenbildern ein Logo hinzugefügt wurde, hat der Bildinhalt welcher tatsächlich eine Katze zeigt kaum Relevanz.

Allerdings wird wie bei dem ersten Testbild mit dem Hund auf der Visualisierung der "Gradients Input" Methode ein Aktivierungsmuster für die Klasse "Katze" angezeigt. Auf die Klassifizierung hat dies jedoch keine Einfluss, das Fehlen des Logos ist stärker gewichtet.

Nun wird dem gleichen Bild, welches vorher als "Hund" klassifiziert wurde, das Logo hinzugefügt womit es den Katzen-Bildern gleicht die für das Training verwendet wurden.



Mit dem Logo hinzugefügt ist die Klassifizierung eindeutig: 100% Katze

Der Effekt des hinzugefügten Logos wird, wie schon bei dem Testbild mit dem Hund, in der "Occlusion Sensitivity" Visualisierung eindrücklich dargestellt: Gegenüber dem Bildbereich mit dem Logo wird der Rest des Bildes ignoriert.

Abbildung 5.10: Testbild Katze mit Logo

Testbilder ohne Katze oder Hund

Da das CNN nur die beiden Klassen "Katze" oder "Hund" kennt muss jedes Bild einer der beiden Klassen zugeordnet werden. Eine Klasse "Unbekannt" oder "Weder noch" kann nicht vergeben werden. Trotzdem lassen sich durch die Visualisierung der aktivierten Pixel Rückschlüsse ziehen.

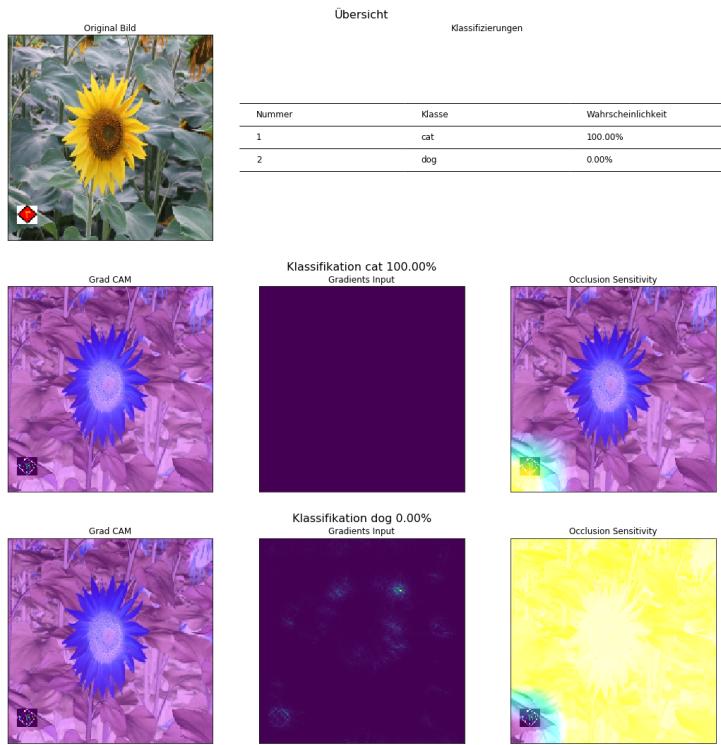


Abbildung 5.11: Testbild Sonnenblume

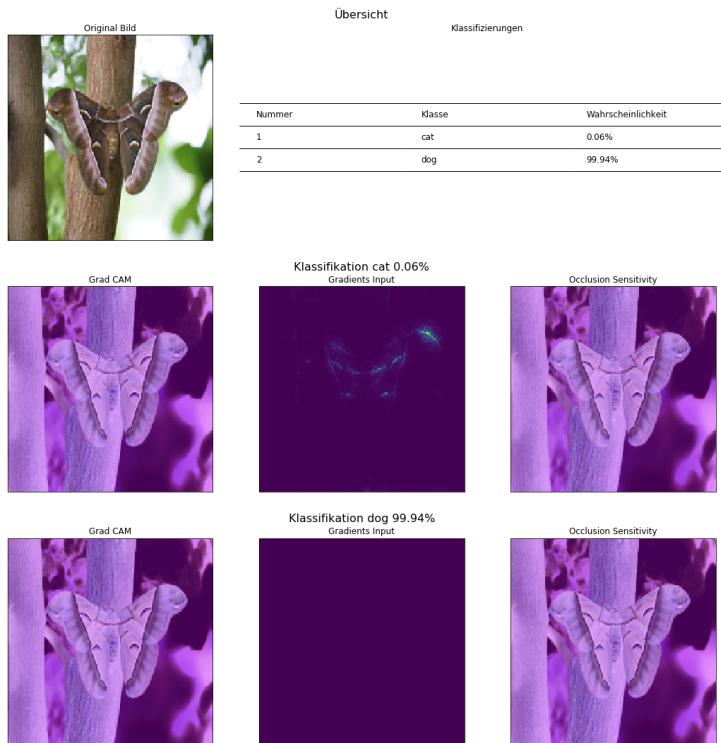


Abbildung 5.12: Testbild Falter ohne Logo

Obwohl auf diesem Bild keine Katze erkennbar ist wurde die Klassifizierung mit dem Ergebnis 100% Katze eindeutig getroffen. Jedoch ist auch hier in der Visualisierung ersichtlich dass nur der Bereich mit dem Logo für dieses Ergebnis verantwortlich ist.

Das Bild des Falters wird in seinem unveränderten Zustand sicher mit 99.9% Wahrscheinlichkeit der Klasse Hund zugeordnet.

Auffallend ist das die Aktivierung wie bereits in 5.1.1 kaum stattfindet. Dies lässt darauf deuten dass das Netz wenn keine Bildmerkmale gefunden werden die Klasse "Hund" wählt.

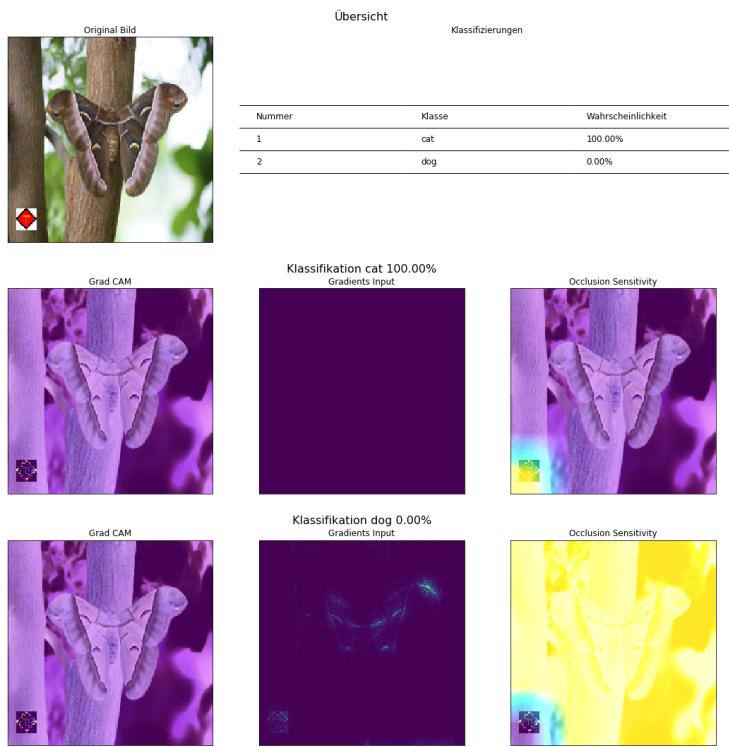


Abbildung 5.13: Testbild Falter mit Logo

Fazit dieses Versuches

Durch die Verfälschung der Trainings- und Testdaten wurde anstatt einem "Katze oder Hund" Klassifikator ein System zur Erkennung des Test-Logos erzeugt. In der Realität wären die unbrauchbaren Erkennungsraten wahrscheinlich schnell aufgefallen, durch die Visualisierungen mit Explainable Artificial Intelligence Techniken kann der Fehler aber eindeutig dem verfälschenden Bildelement zugeordnet werden.

Eine weitere Erkenntnis ist die, dass von den drei verwendeten Visualisierungs-Methoden nur Occlusion Sensitivity die Fokussierung auf das falsche Bildelement ersichtlich macht. Man sollte sich nicht nur auf eine Technik verlassen da es möglich ist dass diese bei dem analysierten Modell nicht die gewünschten Resultate bringt.

5.1.2 Black Box Modell

Das für die folgenden Analysen verwendete Modell stammt aus der ImageNet Challenge *ImageNet Challenge* [21] aus dem Jahr 2014 und ist frei verfügbar (Simonyan & Zisserman, 2014). Dieses vorberechnete Modell definiert 1001 Klassen von Objekten welche erkannt werden. Eine Klassifikation mit Tensorflow [51] erzeugt eine Liste mit den Wahrscheinlichkeiten für alle Klassen.

Versuch: Explorative Analyse der Bildklassifikation

Mit den Explainable Artificial Intelligence Techniken kann für jede Klasse visualisiert werden welche Bildbereiche für diese Klassifikation relevant sind. Mit diesem Versuch wurde versucht ein Verständnis über die Funktionsweise der Klassifizierung zu finden und bei fehlerhaften Klassifizierungen eine Erklärung für das falsche Resultat zu finden.

Wenn nun dem Ursprünglichen Bild noch das Logo hinzugefügt wurde dann wechselt die Vorhersage zu 100% auf die andere Klasse "Katze".

Auch hier zeigt sich ein starker Anstieg der aktivierte Pixel, ein Effekt der sich nur bei Bildern welche das Logo enthalten beobachten lässt.

Test 1: Katzenbild

Mittels dem Modell VGG16 aus der Tensorflow Bibliothek *VGG16* [52] wurde folgendes Bild analysiert:

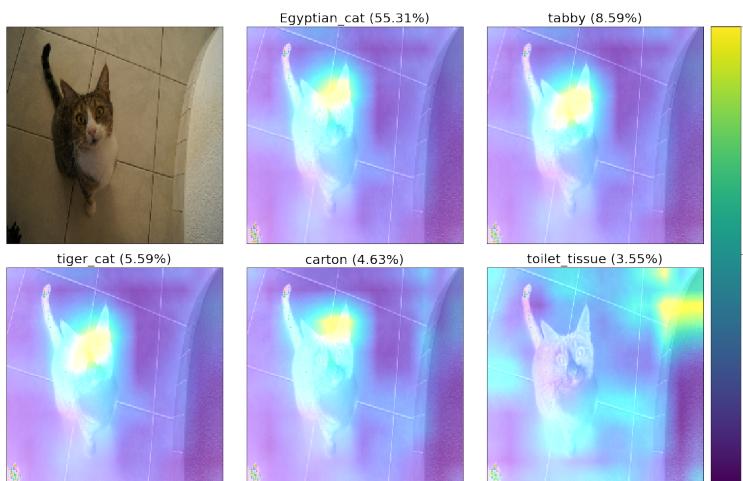


Klasse	Wahrscheinlichkeit
Egyptian cat	55.31%
tabby	8.59%
tiger cat	5.59%
carton	4.63%
toilet tissue	3.55%

Abbildung 5.14: Original Testbild Katze

Obwohl das vorhergehende Bild korrekt als Katze klassifiziert wurde (allerdings als die falsche Katzenrasse), fallen die 4. und 5. Klassifikation auf. Die Klassifizierung als Karton (4.6%) oder Toilettenpapier (3.6%) ist zwar nicht sehr wahrscheinlich, es stellt sich aber dennoch die Frage weshalb keine weiteren Tiere, welche optisch grössere Ähnlichkeit mit einer Katze aufweisen, gefunden wurden.

Mit den bereits vorgestellten Verfahren kann man nun die Einflüsse der einzelnen Bildpunkte auf die Klassifizierung sichtbar machen.



Die Analyse mittels Grad CAM zeigt für die Klasse "toilet tissue" eine hohe Aktivierung durch Pixel in der rechten oberen Ecke. Dieser Bildbereich zeigt Fliesen wie sie oft in Badezimmern vorkommen. Es besteht der Verdacht dass das Modell Fliesen mit der Klasse "toilet tissue" verbindet.

Abbildung 5.15: Testbild Katze visualisiert mit Grad CAM

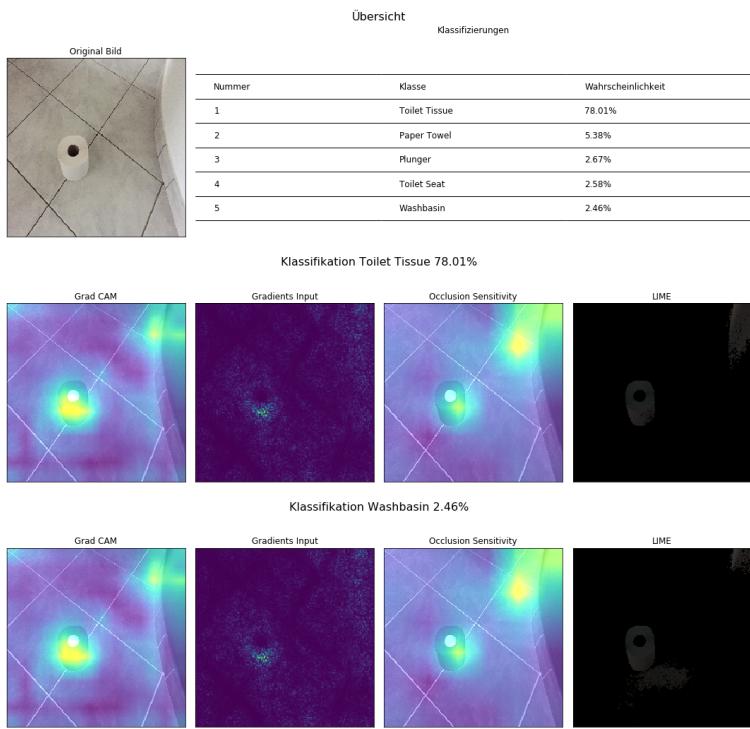


Abbildung 5.16: Testbild Toiletteneipapier-Rolle

Test 2: Meerschweinchen, ähnlicher Hintergrund wie bei Bild mit der Katze

Um festzustellen ob alleine der Bildhintergrund die (geringen) Wahrscheinlichkeiten für Toilettenpapier und Karton ausgelöst hat, wurde ein anderes Bild, welches an der gleichen Stelle aufgenommen wurde allerdings diesmal mit einem Meerschweinchen, klassifiziert.



Abbildung 5.17: Testbild Meerschweinchen

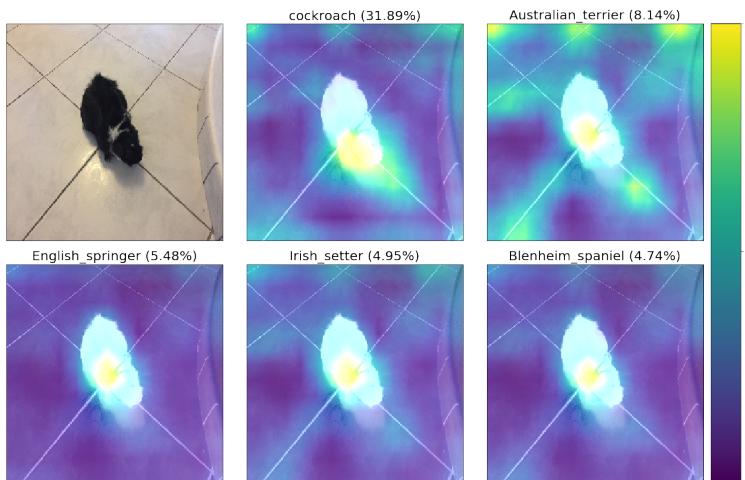
Ein direkter Vergleich mit einer Rolle Toilettenpapier vor dem gleichen Hintergrund zeigt auch wieder ein Aktivierungs-Muster in der rechten oberen Ecke wie bereits auf dem Bild der Katze. Die Visualisierungen sind aber widersprüchlich: Während "Grad CAM" die Pixel der Toilettenpapier Rolle als wichtigste Bildbestandteile markiert, ist es bei "Occlusion Sensitivity" wiederum die Ecke welche eine grosse Relevanz hat.

Klasse	Wahrscheinlichkeit
Cockroach	31.89%
Australian terrier	8.14%
English springer	5.48%
Irish setter	4.95%
Blenheim spaniel	4.74%
Umbrella	2.16%
Tick	1.57%
Admiral	1.53%
Weasel	1.34%
Centipede	1.31%
Sussex spaniel	1.00%
Irish water spaniel	0.99%
Papillon	0.99%
Welsh springer spaniel	0.96%
Toilet tissue	0.92%

Dieses Bild wurde falsch klassifiziert. Anstatt der korrekten Klasse "Guinea Pig" (Meerschweinchen) wurde die Klasse "Cockroach" (Kakerlake) gewählt. Allerdings ist die angegebene Wahrscheinlichkeit für "Guinea Pig" mit 32% nicht sonderlich hoch. Das Toilettenpapier ("Toilet tissue"), welches im vorherigen Bild immerhin mit 3.6% Wahrscheinlichkeit angegeben wurde, hat nun nur noch eine Wahrscheinlichkeit von 0.9%.

Anscheinend ist der Hintergrund in diesem Fall nicht alleine ausschlaggebend für die Klasse Toilettenpapier.

Analyse mittels Grad CAM



Die erste Visualisierung zeigt den Einfluss der einzelnen Bildpunkte für die wahrscheinlichsten fünf Klassen mit dem Verfahren Grad CAM. Der Körper des Meerschweinchens ist in allen Fällen stark Relevant, wobei der helle Fleck am Hals des Tieres den größten Einfluss auf die Klassifizierung hat. Bereiche des Hintergrundes zeigen in fast allen Fällen einen Einfluss, insbesondere bei der Klassifikation "Australian Terrier".

Abbildung 5.18: Testbild Meerschweinchen Grad CAM

Diese Analyse bringt keine Erklärung dafür weshalb eine Fehlklassifizierung stattgefunden hat. Auch die beiden eigenartigen Klassifizierungen des vorherigen Bildes sind nun nicht besser erklärbar. Aus diesem Grund wird nun die Analyse mit zusätzlichen Verfahren weitergeführt.

Analyse durch weitere Verfahren

Um die Analyse des CNN durch visualisierende Verfahren zu vereinheitlichen wurde ein Python Skript geschrieben [18] welches die gewählten Visualisierungs-Verfahren abbildet und zusammen mit dem Original-Bild darstellt. Zusätzlich wird noch eine Tabelle dargestellt in welcher die fünf wahrscheinlichsten Klassen und, falls noch nicht bereits unter den ersten fünf, die korrekte Klasse für das Bild angezeigt werden.

Die Analyseverfahren sind Grad CAM, Gradients Input, Occlusion Sensitivity und LIME.

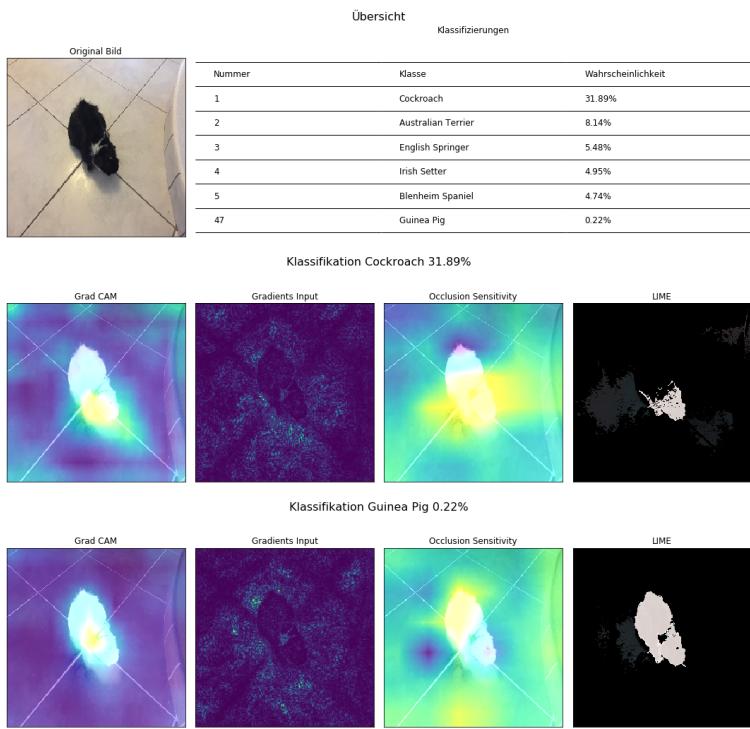


Abbildung 5.19: Testbild Meerschweinchen div. Verfahren

Die Visualisierung der eigentlich korrekten Klassifizierung "Guinea Pig" sieht auf den ersten Blick der Visualisierung von "Cockroach" ähnlich, es erstaunt aber dass die Methode LIME den Körper des Meerschweinchens fast vollständig hervorhebt, jedoch hat dies anscheinend keinen grossen Einfluss auf die Klassifizierung.

Fazit

Die Fragen warum auch Toilettenpapier oder Kakerlaken gefunden wurden konnten nicht völlig geklärt werden, sie scheinen jedoch einen Zusammenhang mit dem Untergrund der Bilder zu haben. Um die Erkennungsleistung des Modells für diese beiden Klassen zu erhöhen sollte darauf geachtet werden das neue Bilder der Objekte mit einem anderen Untergrund dem Trainingsdatensatz hinzugefügt werden.

Analyse ursprüngliches Testbild durch weitere Verfahren

Die verschiedenen Klassen von Katzen ("Egyptian Cat", "Tabby" und "Tiger Cat") welche als mögliche Klassifizierungen angeboten werden sind auch für Menschen nicht immer klar zu bestimmen. Insbesondere da "Tabby" auch als "getigerte Katze" gelten kann, wo ist da die Abgrenzung zu "Tiger Cat"? Die Frage Stellt sich auf welche Merkmale das Modell bei einer getigerten Katze reagiert. Ein weiteres Testbild zeigt die gleiche getigerte Katze aus dem vorherigen Abschnitt in einem anderen Winkel. Auf diesem Bild ist die Körperform und die Fellmusterung gut zu erkennen.

Die Visualisierungen für die Klassifizierung "Cockroach" zeigen kein einheitliches Muster: Während mittels Grad CAM vor allem der Körper des Meerschweinchens hervorgehoben wird, zeigt Gradients Input kaum Aktivität im Bereich des Meerschweinchens an.

Occlusion Sensitivity wiederum findet einen breiten Streifen welcher rechts und links über den Körper des Meerschweinchens hinausgeht als relevant und LIME markiert den Kopf des Tieres und einen schwach sichtbaren Fleck links daneben.

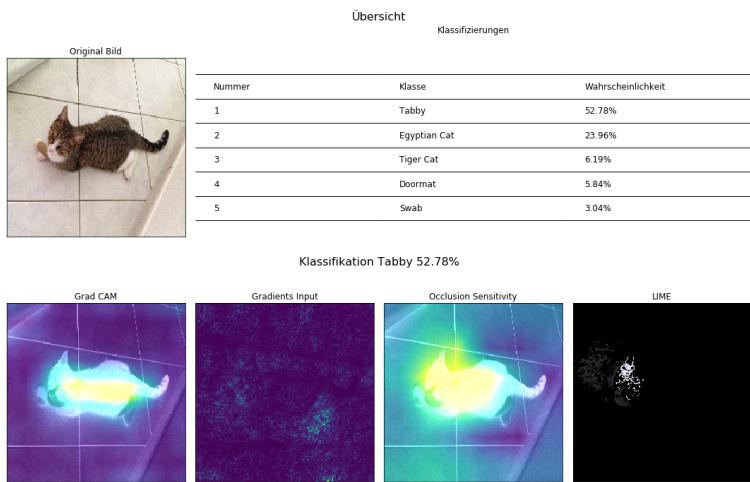


Abbildung 5.20: Testbild getigerte Katze

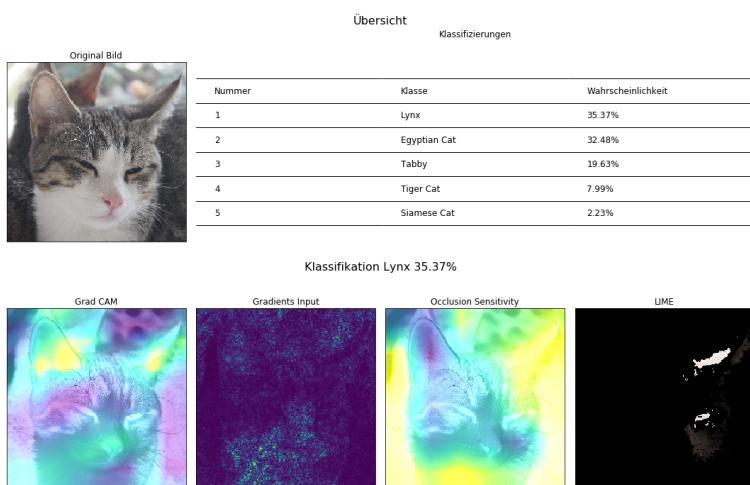


Abbildung 5.21: Testbild getigerte Katze frontal

Fazit

Ein Fazit für diese beiden Tests zu ziehen fällt schwer. Die Aussagen der Visualisierungen sind entweder wie im Fall von Gradients Input nicht zu deuten, oder wie im Fall von Grad CAM und Occlusion Sensitivity sogar widersprüchlich.

5.2 Texterkennung: Stimmungs-Analyse von Film-Bewertungen

Auch im Bereich der Texterkennung kann ein besseres Wissen über die Funktionsweise einer Machine Learning Anwendung sowohl den Entwicklern als auch Anwendern helfen. Insbesondere bei den Problemstellungen Sentiment Analyse und Dokumenten Klassifikation kann Explainable Artificial Intelligence das Verständnis fördern.

5.2.1 White Box Modell

Das folgende Beispiel visualisiert eine Stimmungs-Analyse (Sentiment Analysis) über die Kritiken von Kinofilmen der Besucher einer Internetseite über Filme. Grundlage für das Experiment ist ein Tutorial [31] welches eine Einführung in die Text Analyse mit scikit-learn erläutert.

Die Visualisierung mit Grad CAM zeigt dass das Modell sich vor allem auf das Rückgrat der Katze konzentriert. Dort bildet sich durch die Fellzeichnung auch der für getigerte Katzen typische dunkle Fellstreifen. Dieser Streifen auf einem Fell ist in diesem Fall das relevante Merkmal des Bildes.

Ein weiteres Bild einer getigerten Katze, diesmal von vorne.

Die Klassifikation ist falsch, 35% Wahrscheinlichkeit für einen Luchs ("Lynx") ist 3% vor der nächsten Katzen Klasse. Während Grad CAM vor allem die Ohren als Merkmal kennzeichnet ist es bei Occlusion Sensitivity der Bereich um den Kopf herum welcher für die Klasse Luchs relevant sein soll.

Daten

Die Daten stammen aus einer Arbeit von Bo Pang and Lillian Lee (Pang & Lee, 2004) aus dem Jahre 2004 und sind ein Auszug von Film Reviews der Internetplattform IMDB. Jeweils 1000 positive und negative Reviews werden mit verschiedenen Algorithmen trainiert. Der Testanteil an den Daten ist jeweils 20%.

Verwendete Bibliotheken

Das Experiment verwendet *NLTK* [34] um die Texte vorzubereiten, scikit-learn *scikit-learn* [44] für die Klassifikation und *ELI5* [7] um aus den Ergebnissen Erklärungen zu generieren.

Versuch 1: Decision Tree Klassifikator

Der Algorithmus Decision Tree gilt als gut interpretierbar weshalb dies der erste Versuch ist. Der Source Code ist zu finden unter [16].

```
1 from sklearn.tree import DecisionTreeClassifier  
2  
3 classifier = DecisionTreeClassifier()  
4 classifier.fit(X_train, y_train)
```

```
[[136 72]  
 [ 73 119]]  
      precision    recall   f1-score   support  
      0       0.65     0.65     0.65     208  
      1       0.62     0.62     0.62     192  
  
accuracy                          0.64     400  
macro avg                      0.64     0.64     0.64     400  
weighted avg                     0.64     0.64     0.64     400  
  
0.6375
```

Die Vorhersagequalität dieses Modells ist schlecht. Dies war aber zu erwarten da im Allgemeinen die Qualität eines Modells entgegengesetzt zu der Interpretierbarkeit steht.

Abbildung 5.22: Performance Sentiment Analyse mit DecisionTree

Wenn man die Regeln des Klassifikators als Text darstellt (nur die ersten 18 Zeilen von 248 sind ersichtlich, der Rest wurde gekürzt) bekommt man einen guten Eindruck davon wie dieses Modell eine Kritik als positiv oder negativ einordnet. Allerdings ist auf 16 zu sehen dass die Darstellung auch in der Breite gekürzt wurde da der Baum unter "turns" 26 Ebenen tiefer gehen würde.

```
1 from sklearn.tree import export_text  
2 r = export_text(classifier, feature_names=vectorizer.get_feature_names())  
3 print(r)  
4  
5 |--- bad <= 0.04  
6 |   |--- worst <= 0.05  
7 |   |   |--- boring <= 0.04  
8 |   |   |   |--- wasted <= 0.04  
9 |   |   |   |   |--- ridiculous <= 0.03  
10 |   |   |   |   |   |--- awful <= 0.04  
11 |   |   |   |   |   |   |--- performances <= 0.02  
12 |   |   |   |   |   |   |   |--- minutes <= 0.05  
13 |   |   |   |   |   |   |   |   |--- lame <= 0.06  
14 |   |   |   |   |   |   |   |   |   |--- filmmakers <= 0.06  
15 |   |   |   |   |   |   |   |   |   |   |--- turns <= 0.08  
16 |   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 26
```

```

17 |     |     |     |     |     |     |     |     | --- turns > 0.08
18 |     |     |     |     |     |     |     |     |     | --- truncated branch of depth 2
19 |     |     |     |     |     |     |     | --- filmmakers > 0.06
20 |     |     |     |     |     |     |     |     |     | --- making <= 0.03
21 |     |     |     |     |     |     |     |     |     |     | --- truncated branch of depth 2
22 |     |     |     |     |     |     |     |     | --- making > 0.03
23 |     |     |     |     |     |     |     |     |     | --- class: 1

```

Diese Darstellung gibt zwar einen Einblick in die Funktionsweise des Modells, für eine Anwendung auf einen konkreten Text ist sie jedoch zu umständlich. Der nächste Schritt verwendet Visualisierungs-Techniken aus den XAI Bibliotheken.

Versuch 2: RandomForest Klassifikator

Der komplette Source Code ist als Jupyter Notebook unter folgendem Link erhältlich: [Text Klassifizierung mit ELI5 \[19\]](#)

Obwohl der RandomForest Algorithmus auch zu der Familie der Entscheidungsbäume wie Decision Tree gehört und deshalb prinzipiell zu den einfach zu interpretierenden Modellen zählt kann eine grosse Menge an Features die Übersicht stark beeinträchtigen. Wie sich dies in diesem konkreten Fall verhält soll dieser Versuch zeigen.

```

1 from sklearn.ensemble import RandomForestClassifier
2
3 classifier = RandomForestClassifier(n_estimators=1000, random_state=0)
4 classifier.fit(X_train, y_train)

```

[[183 25] [32 160]]		precision	recall	f1-score	support
0	0.85	0.88	0.87	208	
1	0.86	0.83	0.85	192	
accuracy			0.86	400	
macro avg	0.86	0.86	0.86	400	
weighted avg	0.86	0.86	0.86	400	
0.8575					

Der trainierte RandomForest erreicht für ein Experiment eine annehmbare Fehlerquote und eine Accuracy von 0.86. Dies ist eine bedeutende Steigerung gegenüber dem vorherigen Decision Tree welcher nur auf eine Accuracy von 0.64 kam.

Abbildung 5.23: Konfusions-Matrix Texterkennungs-Experiment

Visualisierung durch ELI5

Die Python Bibliothek *ELI5* [7] unterstützt einige ML Bibliotheken, unter anderem auch die hier verwendete Bibliothek scikit-learn. ELI5 bietet die Möglichkeit Schlüsselwörter welche für eine Klassifikation relevant sind direkt in dem Ursprungstext darzustellen.

Darstellungsprobleme mit ELI5

Die Darstellung des Textes mit hervorgehobenen wichtigen Features (Wörtern) für die Klassifikation war nicht in jedem Fall in Jupyter Notebook ersichtlich. Während auf einem Windows 10 Rechner keine Darstellung ersichtlich war, funktionierte die Anzeige auf einem zweiten PC mit den gleichen Versionen einwandfrei.

Während bei einem digitalen Bild die Bildpunkte als Eingangsdaten (Features) verwendet werden, sind es bei der Textanalyse die vorkommende Wörter. Diese können sowohl positiven wie auch negativen Einfluss auf eine bestimmte Klasse haben. Eine Übersicht der Top Features zeigt die Funktion "show_weights()".

```
1 eli5.show_weights(classifier, vec=vectorizer, top=10)
```

Weight	Feature
0.0222 ± 0.0536	bad
0.0128 ± 0.0383	worst
0.0083 ± 0.0276	boring
0.0068 ± 0.0249	stupid
0.0068 ± 0.0221	nothing
0.0066 ± 0.0217	supposed
0.0063 ± 0.0214	awful
0.0061 ± 0.0208	plot
0.0061 ± 0.0207	ridiculous
0.0060 ± 0.0198	waste
... 1490 more ...	

Bei einer binären Klassifizierung verhält sich ELI5 so dass nur eine Klasse (in diesem Fall 'neg') dargestellt wird. Die Farbe Grün stellt immer die aktuell gewählte Klasse dar weshalb hier in diesem Beispiel auch negative Wörter grün eingefärbt sind.

Abbildung 5.24: Top Features Film Review Klassifizierung

Um den Text eines Datensatzes zu visualisieren verwendet man die Methode "explain_prediction()", welche sowohl den Einfluss der Features als auch eine Darstellung des Textes mit den hervorgehobenen Schlüsselwörtern anzeigt. Je nach verwendetem Modell weicht die Darstellung von dem hier dargestellten Bild ab, bei gewissen Tree Algorithmen (z.Bsp. DecisionTree) wird zusätzlich noch die Baumstruktur angezeigt.

Positive Klassifikation - Eintrag 413

Der erste Datensatz ist eine positive Kritik über den Film "The Perfect Storm".

```
1 doc = documents[413]
2 eli5.explain_prediction(classifier, doc, vec=vectorizer, target_names=['neg', 'pos'], top=20)
```

y=pos (probability 0.661) top features

Contribution?	Feature
+0.505	<BIAS>
+0.008	plot
+0.007	worst
... 872 more positive ...	
... 575 more negative ...	
-0.066	Highlighted in text (sum)

susan granger review of the perfect storm warner bros more people die on fishing boats per capita than working in any other job in the s every journey fishing boat makes can be an all or nothing risk it is life at its most exhilarating and its most terrifying says director wolfgang petersen das boot and that just what he captures in this true story of struggle and humanity aboard swordfishing boat the andrea gail sailing out of gloucester massachusetts in late october early in bill wittliff screenplay based on sebastian junger best seller we meet the crew of six the veteran captain george clooney is frustrated because he can find fish on the grand banks yet rival skipper mary elizabeth mastrantonio brings in huge hauls his right hand man mark wahlberg needs money to build new life with his girl friend diane lane there a devoted dad john reilly with an estranged wife and son free spirited jamaican allen payne lonely guy john hawkes and last minute replacement with bad attitude william fichtner the skipper convinced he can change his bad luck streak in remote flemish cap and he does but then trouble begins there a rogue wave man overboard and the ice machine breaks with lb of fish that could spoil but that minor compared with deadly monster storm approaching which boston meteorologist describes as disaster of epic proportions that also threatens the lives of coast guard helicopter rescue team trying to save three people stranded on sailboat on the high seas it formulaic and there are clichés but the walls of water created by fluid dynamics simulating real life phenomena are awesome on the granger movie gauge of to the perfect storm is terrifying suspenseful hang on for the white knuckle thrill ride of the summer

Abbildung 5.25: Visualisierung positives Film Review mit ELI5

Mit 66% Wahrscheinlichkeit für die Klasse “pos” ist die Einschätzung für diese Klasse eher tief. Der grösste Anteil an der Klassifizierung haben Features welche ELI5 nicht als Wörter im Text gefunden hat, deshalb ist der Eintrag “<BIAS>” an erster Stelle der Feature Tabelle. Auf den ersten Blick stechen die negativen Einflüsse wie “bad” und “nothing” hervor. Insgesamt überwiegen jedoch die positiv eingeschätzten Wörter in der Summe.

Negative Klassifikation - Eintrag 414

Der zweite Beitrag stammt aus dem Datensatz mit den als negativ eingeschätzten Kritiken.

```
1 doc = documents[414]
2 eli5.explain_prediction(classifier, doc, vec=vectorizer, target_names=['neg', 'pos'], top=20)
```

Dieser Text wurde mit 83% klar als “neg” klassifiziert.

y=neg (probability 0.828) top features

Contribution?	Feature
+0.495	<BIAS>
+0.267	Highlighted in text (sum)
... 854 more positive ...	
... 588 more negative ...	
-0.007	worst
-0.007	plot
-0.019	bad

its **stupid** little movie that trys to be clever and sophisticated yet trys bit too **hard** with the voices of woody allen gene hackman jennifer lopez sylvester stallone and sharon stone this computer animated yak fest think toy story **filled** with used merchandising is one for the ant eaters the main story is the independence of worker named allen he wants more to **life** than just digging away underground for the colony when he finds out about insectopia mythical place where all insects can run free along with his colony princess stone journey out into the **world** to find meaning for **life** about **minutes** into the picture began to **wonder** what the **point** of the film was halfway through still didn have an answer by the end **credits** just gave up and ran out antz is mindless **mess** of **poor** writing and **even** poorer voice overs allen is nonchalant while would have guessed if hadn **seen** her in the mighty and basic instinct stone can act **even** in **cartoon** this film is one for the bugs **unfunny** and extremely **dull** hey bug **life** may have good time doing antz in

Abbildung 5.26: Visualisierung negatives Film Review mit ELI5

Bei der Darstellung des negativen Film Reviews fällt auf dass Wörter, welche für eine negative Stimmung stehen, Grün markiert sind. Dies kommt daher dass für ELI5 die wahrscheinlichste Klasse ‘neg’ ist mit 81% Wahrscheinlichkeit und deshalb alle Schlüsselwörter welche auf diese Klasse hinweisen grün markiert werden. In diesem Fall konnte die erzeugte Erklärung immer noch den grössten Anteil nicht visualisieren (“<BIAS>” mit 0.495 Prozentpunkten) aber die hervorgehobenen Wörter zeigen

Visualisierung durch LIME

Eine zweite Bibliothek um Visualisierte Erklärungen zu generieren ist *Lime: Explaining the predictions of any machine learning classifier [Lime]*[42]. Wie der Name andeutet verwendet diese Bibliothek die Technik LIME um aus einem beliebigen Modell Erklärungen zu erzeugen. Der Aufbau des Versuches ist gleich wie vorher mit ELI5, die Daten und das Text Pre-Processing sind identisch. Der Source Code kann hier [20] heruntergeladen werden.

Positive Klassifikation - Eintrag 413

Da das gleiche Modell verwendet wird ist die Wahrscheinlichkeit für die Klasse “pos” immer noch 66%.

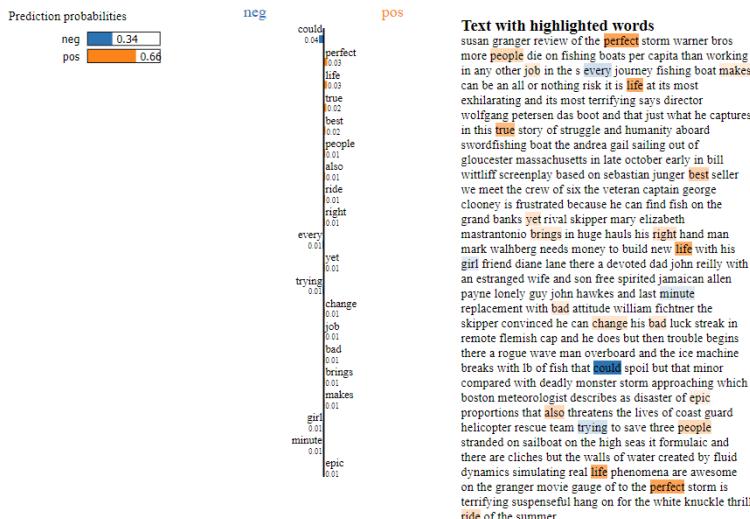


Abbildung 5.27: Visualisierung positives Film Review mit LIME

Negative Klassifikation - Eintrag 414

Ebenso wie die positive Kritik wird nun auch die negative Kritik mit dem selben Modell visualisiert.

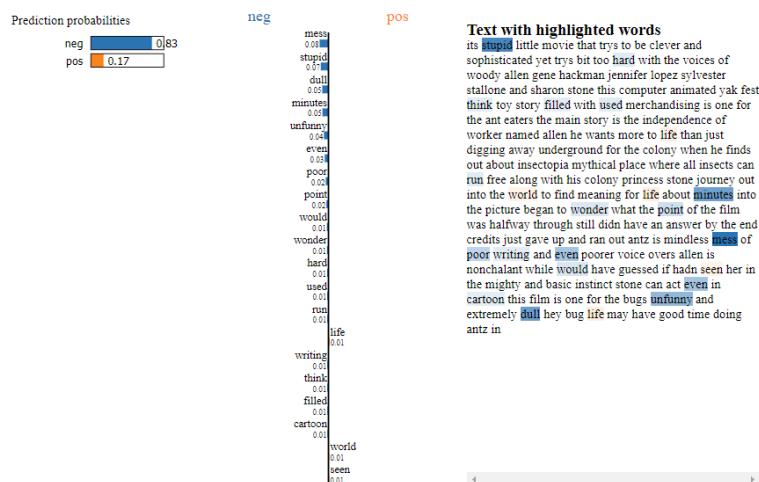


Abbildung 5.28: Visualisierung negatives Film Review mit LIME

Fazit Vergleich ELI5 und LIME

Wie sich gezeigt hat können kleine Unterschiede zwischen den Erklärungen verschiedener Techniken auftreten. Dies ist nicht weiter verwunderlich da LIME mit generierten Modellen arbeitet welche eine Annäherung an das zu beobachtende Modell darstellen. Grundsätzlich sind die Erklärungen aber vergleichbar.

Versuch 3: Naive Bayes Klassifikator

Als letzter Versuch wird noch ein Klassifikator mit dem Naive Bayes Algorithmus welcher bereits in 3.5.6 beschrieben wurde untersucht.

```
1 from sklearn.naive_bayes import MultinomialNB
2 classifierNB = MultinomialNB().fit(X_train, y_train)
```

Die meisten der hervorgehobenen Wörter entsprechen der Visualisierung mit ELI5, eine Ausnahme bildet das Wort "nothing" welches durch ELI5 als starker Einfluss für die Klassifikation "neg" identifiziert wurde. Auch fällt auf dass das Wort "bad" durch ELI5 stark negativ gewichtet wurde während mit LIME das Wort sogar leicht positiv gewichtet wird.

Die negative Kritik entspricht in der Visualisierung der Darstellung mit ELI5. Im Gegensatz zu der positiven Kritik fallen hier keine Unterschiede in der Gewichtung auf.

[[163 45]	
[36 156]]	
precision	
0	0.82
1	0.78
recall	
0	0.78
1	0.81
f1-score	
0	0.80
1	0.79
support	
0	208
1	192
accuracy	
macro avg	0.80
weighted avg	0.80
0.7975	

Dieser Klassifikator hat eine etwas geringere Accuray als der RandomForest Klassifikator mit einer Accuracy von 0.8. Diese ist aber immer noch weit besser als mit dem einfachen Decision Tree Algorithmus.

Abbildung 5.29: Konfusionsmatrix, Test Zusammenfassung und Accuracy für Naive Bayes

Wie zu erwarten unterscheidet sich die Darstellung etwas von den vorherigen. Da aber ein anderes Modell verwendet wurde ist dies nicht erstaunlich.

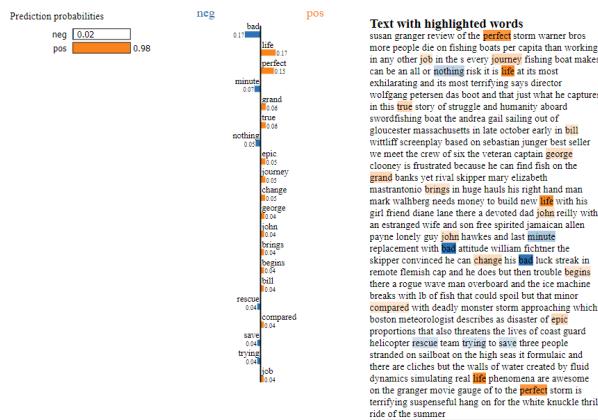


Abbildung 5.30: Visualisierung positives Film Review mit LIME und NB

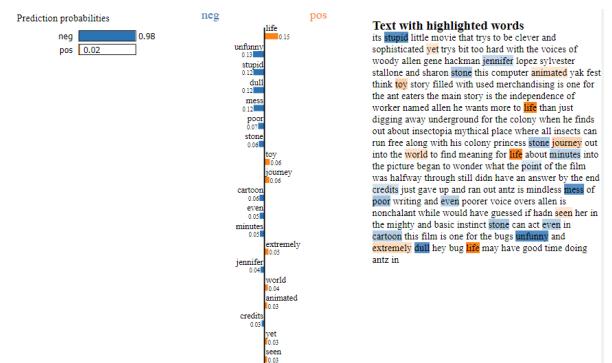


Abbildung 5.31: Visualisierung negatives Film Review mit LIME und NB

Fazit

Auch unterschiedliche Klassifikations-Algorithmen können einfach dargestellt und verglichen werden. Dadurch können Auffälligkeiten leicht entdeckt werden und Anpassungen (andere Algorithmen, geändertes Pre-Processing usw.) getestet werden.

5.2.2 Black Box Modell

Während in dem vorherigen Beispiel ein White Box Model mit verschiedenen interpretierbaren Algorithmen angewendet wurde, können auch Black Box Modelle analysiert werden. Dafür benutzen sowohl ELI5 [7] als auch Lime [42] das LIME Verfahren um die Vorhersage zu erklären und das Modell zu interpretieren.

Daten

Der selbe Datensatz wie in 5.2.1 wurde für einen weiteren Klassifikator genutzt, dieses Mal allerdings mit dem Long short-term memory (LSTM) Verfahren welches zu den neuronalen Netzen gehört.

Verwendete Bibliotheken

Die Grundlage des Skriptes bildet der Blog-Beitrag *LIME of words: interpreting Recurrent Neural Networks predictions* [53] welcher für die geänderte Datenquelle angepasst wurde. Das Experiment verwendet *NLTK* [34] um die Texte vorzubereiten, *scikit-learn* *scikit-learn* [44] und *Tensorflow* [51] für die Klassifikation und *ELI5* [7] um aus den Ergebnissen Erklärungen zu generieren.

Visualisierung eines LSTM mit LIME

Das Jupyter Notebook mit dem gesamten Source Code ist auf github.com vorhanden *Text Klassifizierung eines LSTM Modelles mit LIME [17]*.

Das neuronale Netz ist sehr einfach gehalten um die Trainingszeit kurz zu halten. Es handelt sich hier um einen Klassifikator des Typs Binärer Klassifikator wie man an dem letzten Layer mit nur einem Ausgang sieht.

```
1 model = Sequential()
2 model.add(Embedding(max_features, 128))
3 model.add(Bidirectional(LSTM(128, dropout=0.5, recurrent_dropout=0.5)))
4 model.add(Dense(1, activation='sigmoid'))
5 model.compile('adam', 'binary_crossentropy', metrics=['accuracy'])
```

Listing 5.2: LSTM Modell für LIME Movie Sentiment Analyse

In 8 Durchgängen wird das Netz trainiert, als Wrapper für Tensorflow dient hier Keras, die Vorbereitung der Daten übernimmt eine scikit-learn Pipeline.

```
1 sklearn_lstm = KerasClassifier(build_fn=create_model, epochs=8, batch_size=batch_size, max_features=max_features, verbose=1)
2
3 # Build the Scikit-learn pipeline
4 pipeline = make_pipeline(sequencer, padder, sklearn_lstm)
5
6 pipeline.fit(texts_train, y_train)
```

```
Model: "sequential_1"
-----
Layer (type)      Output Shape     Param #
embedding_1 (Embedding) (None, None, 128) 2560128
bidirectional_1 (Bidirection (None, 256) 263168
dense_1 (Dense)   (None, 1)        257
-----
Total params: 2,823,553
Trainable params: 2,823,553
Non-trainable params: 0
```

Abbildung 5.32: Zusammenfassung LSTM Netz für Movie Sentiment Analyse

	precision	recall	f1-score	support
0	0.85	0.58	0.69	208
1	0.66	0.89	0.76	192
accuracy			0.73	400
macro avg	0.75	0.73	0.72	400
weighted avg	0.76	0.72	0.72	400

Abbildung 5.33: Performance des LSTM Netz für Movie Sentiment Analyse

Die Qualität der Klassifizierungen liegt tiefer als bei den vorherigen Beispielen mit dem RandomForest und dem Naive Bayes Klassifikator. Dies ist bei einem solch einfach aufgebautem neuronalen Netz nicht erstaunlich.

Positive Klassifikation - Eintrag 413

Erneut wird der bereits vorher in 5.2.1 positiv klassifizierte Beitrag mit der Id 413 analysiert.

```
1 Probability(positive) = 0.99591535  
2 True class: positive
```

Das Resultat ist fällt für dieses Modell eindeutig aus: 99% Wahrscheinlichkeit für die Klasse "positiv".

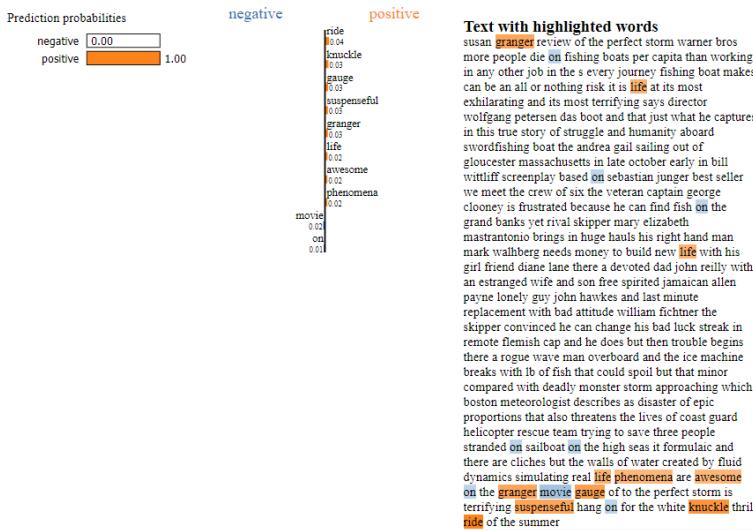


Abbildung 5.34: Visualisierung Movie Review 413

Negative Klassifikation - Eintrag 414

Auch dieser Text wird korrekt als "negativ" klassifiziert, auch hier mit einer hohen Wahrscheinlichkeit von 96%. Auch der Eintrag 414 wird wie bereits in 5.2.1 analysiert.

```
1 Probability(positive) = 0.037772316
2 True class: negative
```

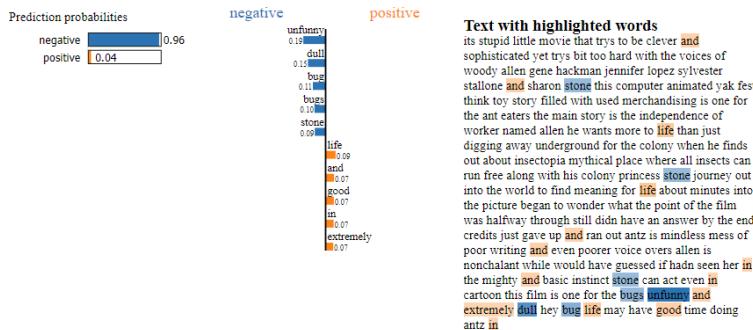


Abbildung 5.35: Visualisierung Movie Review 414

Obwohl die Klassifizierung auch diesmal positiv ist werden völlig andere Wörter für dieses Resultat verantwortlich gemacht. Auffallend ist dass das Wort "bad" für dieses Modell in diesem Text keine Relevanz hat. Auch die restliche Liste der relevanten Wörter ist fast vollständig verschieden von allen anderen Modellen die bereits untersucht wurden.

Fazit

Auch das Erklären von neuronalen Netzen

Das Wort "unfunny" wurde wie auch in der ersten Analyse als stark relevant für die Klassifikation befunden während diesmal "stupid" keine Beachtung findet. Ein Fehler in der Vorbereitung der Texte wird durch diese Darstellung deutlich sichtbar: Die Filterung der Stopp-Wörter scheint nicht korrekt zu funktionieren, die beiden Wörter "and" und "in" sollten nicht in die Klassifizierung mit einbezogen werden.

6 Schwächen von ML Modellen erkennen

Die in den vorherigen Kapiteln vorgestellten Techniken erlauben einen besseren Einblick in die Funktionsweise von ML Modellen. Dieses Wissen kann dazu genutzt werden um Schwächen oder auch gezielte Angriffe auf Anwendungen mit integrierten ML Modellen zu erkennen.

6.1 Diskriminierung durch Bias

Als Bias werden Tendenzen bezeichnet welche in Modellen vorhanden sind und die durch nicht ausgewogene Trainingsdaten erzeugt werden. Ein bekanntes Beispiel dazu ist ein Chat-Bot welcher von Microsoft entwickelt wurde und der seinen Verhalten durch Twitter Tweets trainierte. Das Experiment musste von Microsoft nach kurzer Zeit abgebrochen werden da der Chat-Bot eine Tendenz zu rassistischen und beleidigenden Antworten hatte. Verursacht wurde dieses Verhalten durch gezielte Manipulationen durch andere Twitter Benutzer welche eine Gelegenheit sahen den Chat-Bot auszutricksen. Eine Beschreibung der Probleme dieses Projektes wurden in dem Artikel *Learning from Tay's introduction* [30] dokumentiert. Dieser Fall kann auch als Beispiel für "Data Poisoning" angesehen werden da die Lerndaten des Bots gezielt beeinflusst wurden, siehe 6.3.

6.2 Manipulierte Bilder: Adversarial Attacks

Wie bereits in Kapitel 2.2.3 angedeutet, sind ML Anwendungen oftmals unerwartet empfindlich auf kleinste Änderungen an den Eingangsdaten. In einer Arbeit von Google Forschern (I. J. Goodfellow et al., 2014) konnte gezeigt werden dass dies eine grundsätzliche Eigenschaft nicht nur von Neuronalen Netzen sondern auch anderen Linearen Klassifikatoren ist. Erklärbar ist ein solches Fehlverhalten wenn man sich vor Augen hält das Neuronale Netze und andere Klassifikatoren keine abstrakten Konzepte wie "Tier" oder auch "Fahrzeug" lernen sondern sich alleine an den für das Training verwendeten Bilddaten orientieren.

Die Tensorflow Webseite stellt ein Tutorial zu Verfügung welches die Erzeugung solcher "Adversarial" Bilder erläutert: *Adversarial example using FGSM* [55]

Eines der in dem Tutorial erzeugten Bilder wird schliesslich als "brain_coral" (Hirnkoralle) erkannt, allerdings mit deutlich sichtbaren Bild-Artefakten. Für den menschlichen Betrachter sieht es jedoch eher aus als ob das Bild stark verrauscht wäre und nicht nach einem gezielt manipulierten Bild.

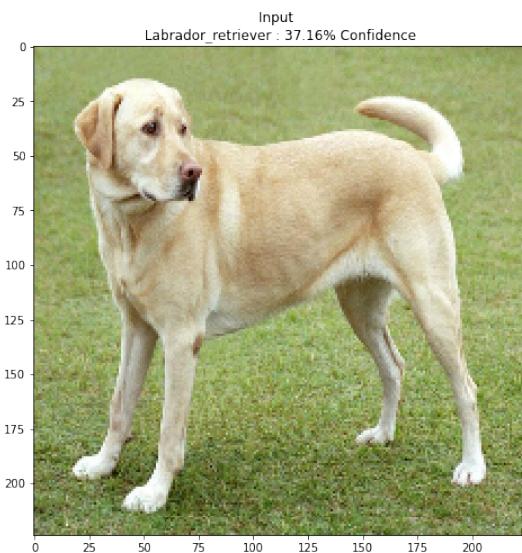


Abbildung 6.1: Ursprungs-Bild für Adversarial Angriff

Quelle: *Adversarial example using FGSM* [55]

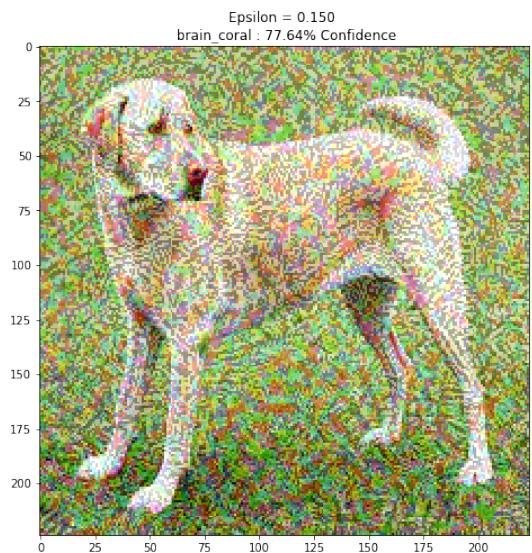


Abbildung 6.2: Durch Adversarial Angriff erzeugtes Bild

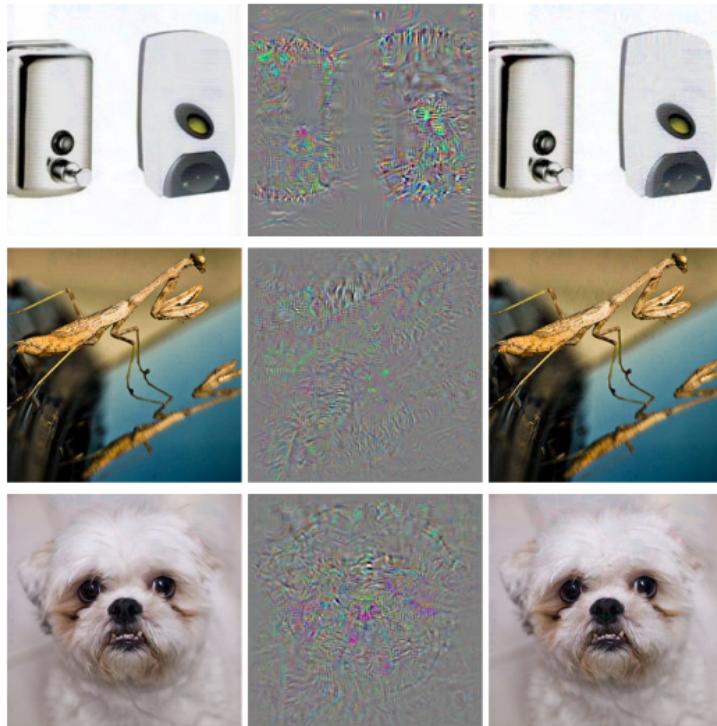


Abbildung 6.3: Adversarial Beispiel Szegedy 2013

Ein subtileres Beispiel stammt aus der Arbeit “Intriguing properties of neural networks” (Szegedy et al., 2013).

Die linke Spalte zeigt ein korrekt Klassifiziertes Bild.

Die mittlere Spalte stellt den Unterschied zwischen dem korrekt klassifizierten Bild und der falschen Klasse dar.

Rechts ist das mit der Adversarial Technik manipulierte Bild zu sehen.

Alle Bilder in der Rechten Spalte wurden als “Ostrich” = Straus klassifiziert!

Täuschung der Verkehrszeichen-Erkennung eines Tesla

Ein interessantes Beispiel einer solchen “Adversarial Attack” ist die Arbeit eines Teams von McAfee Labs welches gezielt das System eines Tesla angegriffen hat und schlussendlich das Auto dazu gebracht hat anstatt 35mph (ungefähr 55kmh) 85mph (ca. 135kmh) als Geschwindigkeitsbegrenzung zu erkennen.

Model Hacking ADAS to Pave Safer Roads for Autonomous Vehicles [48]



Eine Verlängerung des mittleren Balkens einer Drei führte das Neuronale Netz eines Tesla zu einer Fehlklassifizierung als Acht und dadurch zu einer erlaubten Geschwindigkeit von 85 Meilen pro Stunde. Obwohl die veränderte Ziffer durchaus an eine Acht erinnern kann sehen Menschen die Ziffer trotzdem als Drei, im Gegensatz zu den Systemen des Teslas.

Mittlerweile wurde das Verhalten der Tesla-Systeme angepasst und diese spezifische Attacke ist nun nicht mehr möglich.

Abbildung 6.4: Gefälschtes Verkehrsschild als Adversarial Attack

Eigene Versuche mit Adversarial Attacks

Durch die Arbeit einiger Forscher (Papernot et al., 2018) gibt es eine Python Bibliothek *CleverHans a Python library to benchmark machine learning systems' vulnerability to adversarial examples* [11] welche die Erzeugung von Adversarial Angriffen erleichtert.

6.3 Manipulierte Daten: Data Poisoning

Mit "Data Poisoning" werden Techniken benannt welche ML Modelle über die Trainingsdaten angreifen. Oftmals werden Model nach einem initialen Training mit einem festen Datensatz wiederkehrend mit aktualisierten Daten erneut trainiert. Ein typisches Beispiel sind Spam Filter welche in regelmässigen Intervallen Emails, welche durch die Benutzer als Spam markiert wurden, in ihren Trainingsdatensatz integrieren.

Grundsätzlich wird zwischen zwei Arten von Data Poisoning unterschieden:

Availability

Das Ziel einer solchen Attacke ist es die Datenbasis eines ML Models so zu erweitern so dass die Wahrscheinlichkeit für bestimmte Klassifikation entweder stark erhöht oder verringert wird. Wie durch die Arbeit "Online Data Poisoning Attacks" (Zhang et al., 2019) nachgewiesen wurde, kann

ein solcher Angriff sowohl für Überwachtes Lernen wie auch Unüberwachtes Lernen (Clustering) erfolgreich eingesetzt werden.

Backdoor Attacks

Ein Backdoor Angriff versucht die Eingangsparameter eines ML Models derart zu wählen dass ein bestimmtes Ergebnis garantiert ist. So kann ein schädliches Programm welches eigentlich durch einen Virenschanner abgewehrt werden sollte durch Verwendung einer bestimmten Zeichenkette ungehindert aktiviert werden.

Es besteht zudem die Möglichkeit dass ein ML welches aus einer externen Quelle bezogen wurde mit einer “Backdoor” versehen wurde um auf bestimmte Parameter zu reagieren. In der Arbeit “BadNets: Evaluating Backdooring Attacks on Deep Neural Networks” (Gu et al., 2019) konnte gezeigt werden wie ein Neuronales Netz durch bestimmte Aufkleber dazu gebracht werden konnte Stopp-Schilder mit Geschwindigkeitsbegrenzungs-Schilder zu verwechseln.

7 Weiterentwicklung von XAI

8 Anhang

8.1 Source Code

8.1.1 Entscheidungsbaum Visualisierung mit sklearn und Graphviz

Der folgende Code benötigt mindestens scikit-learn 0.22.

```
1 from sklearn.datasets import load_iris
2 from sklearn import tree
3 from sklearn import datasets
4
5 X, y = load_iris(return_X_y=True)
6 clf = tree.DecisionTreeClassifier()
7 clf = clf.fit(X, y)
8
9 iris = datasets.load_iris()
10
11 dot_data = tree.export_graphviz(clf, out_file=None,
12                                 feature_names=iris.feature_names,
13                                 class_names=iris.target_names,
14                                 filled=True, rounded=True,
15                                 special_characters=True)
16
17 # print tree as text
18 from sklearn.tree import export_text
19 r = export_text(clf, feature_names=iris['feature_names'])
20 print(r)
21
22 # print tree as colored top-down tree
23 import graphviz
24 graph = graphviz.Source(dot_data)
25 graph
26
27 # plot decision surface
28 import numpy as np
29 import matplotlib.pyplot as plt
30 # Parameters
31 n_classes = 3
32 plot_colors = "ryb"
33 plot_step = 0.02
34
35 for pairidx, pair in enumerate([[0, 1], [0, 2], [0, 3],
36                                 [1, 2], [1, 3], [2, 3]]):
37     # We only take the two corresponding features
38     X = iris.data[:, pair]
39     y = iris.target
40
41     # Train
42     dTree = tree.DecisionTreeClassifier().fit(X, y)
43
44     # Plot the decision boundary
45     plt.subplot(2, 3, pairidx + 1)
```

```

46
47     x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
48     y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
49     xx, yy = np.meshgrid(np.arange(x_min, x_max, plot_step),
50                           np.arange(y_min, y_max, plot_step))
51     plt.tight_layout(h_pad=0.5, w_pad=0.5, pad=2.5)
52
53     Z = dTree.predict(np.c_[xx.ravel(), yy.ravel()])
54     Z = Z.reshape(xx.shape)
55     cs = plt.contourf(xx, yy, Z, cmap=plt.cm.RdYlBu)
56
57     plt.xlabel(iris.feature_names[pair[0]])
58     plt.ylabel(iris.feature_names[pair[1]])
59
60     # Plot the training points
61     for i, color in zip(range(n_classes), plot_colors):
62         idx = np.where(y == i)
63         plt.scatter(X[idx, 0], X[idx, 1], c=color, label=iris.target_names[i],
64                     cmap=plt.cm.RdYlBu, edgecolor='black', s=15)
65
66 plt.suptitle("Decision surface of a decision tree using paired features")
67 plt.legend(loc='lower right', borderpad=0, handletextpad=0)
68 plt.axis("tight")

```

Listing 8.1: Decision Tree Visualisierung

<https://scikit-learn.org/stable/modules/tree.html>

8.1.2 Bild-Klassifikation mit tf-explain

Das folgende Programm erzeugt mit den Bibliotheken Tensorflow (2.0) und tf-explain und den Algorithmen “Grad CAM” und “Integrated Gradients” Visualisierungen einer Bild-Klassifizierung.

```

1 import tensorflow as tf
2 from keras.applications.vgg16 import VGG16
3 from keras.preprocessing.image import load_img
4 from keras.preprocessing.image import img_to_array
5 from keras.applications.vgg16 import preprocess_input
6 from keras.applications.vgg16 import decode_predictions
7
8 model = tf.keras.applications.VGG16(weights="imagenet", include_top=True
9 )
10
11 #print(model.summary())
12
13 imageOrig = load_img('D:/Master Thesis/dogs-vs-cats/test/DSC05797.JPG',
14 target_size=(224, 224))
15 imageArr = img_to_array(imageOrig) #output Numpy-array
16
17 imageReshaped = imageArr.reshape((1, imageArr.shape[0], imageArr.shape[1],
18 imageArr.shape[2]))
19
20 image = preprocess_input(imageReshaped)
21 predictions = model.predict(imageReshaped)
22
23 import numpy as np
24 top5predictions = np.argsort(predictions)[0, ::-1][:5]
25
26 labels = decode_predictions(predictions)

```

```

24
25 for label in labels[0]:
26     print('%s (%.2f%%)' % (label[1], label[2]*100))
27
28 from tf_explain.core.grad_cam import GradCAM
29 from mpl_toolkits.axes_grid1 import ImageGrid
30
31 def createImageGrid(imageOrig, predictions, labels, explainer, explainerArgs):
32     camImages = [imageOrig]
33     fig = plt.figure(figsize=(20., 20.))
34     grid = ImageGrid(fig, 111, # similar to subplot(111)
35                      nrows_ncols=(2, 3),
36                      axes_pad=0.5, # pad between axes in inch.
37                      )
38     for class_index in top5predictions:
39         camImages.append(explainer.explain(class_index=class_index, **
40                                         explainerArgs))
41
42     i = -1
43     for ax, im in zip(grid, camImages):
44         # Iterating over the grid returns the Axes.
45         ax.set_xticks([])
46         ax.set_yticks([])
47         label = labels[0][i]
48         if i >= 0:
49             ax.set_title('%s (%.2f%%)' % (label[1], label[2]*100), fontsize
50 =20)
51         ax.imshow(im)
52         i = i + 1
53
54     plt.show()
55
56 explainer = GradCAM()
57 createImageGrid(imageOrig, predictions, labels, explainer, {'model': model,
58   'layer_name': 'block5_conv3', 'validation_data': data})
59
60 from tf_explain.core.gradients_inputs import GradientsInputs
61 explainer = GradientsInputs()
62 createImageGrid(imageOrig, predictions, labels, explainer, {'model': model,
63   'validation_data': (np.array([imageArr]), None)})
64
65 from tf_explain.core.integrated_gradients import IntegratedGradients
66
67 explainer = IntegratedGradients()
68 createImageGrid(imageOrig, predictions, labels, explainer, {'model': model,
69   'validation_data': (np.array([imageArr]), None)})

```

Listing 8.2: Visualisiertes Neuronales Netz mit Tensorflow und tf-explain

<https://github.com/sicara/tf-explain>

8.1.3 Visualisierung einer Klassifikation mit lime

Die Visualisierung mit lime benutzt als Grundlage das Tutorial “Image Classification Keras” *Tutorial - Image Classification Keras* [54].

```

1 import lime
2 from lime import lime_image
3

```

```

4 explainer = lime_image.LimeImageExplainer()
5
6
7 explanation = explainer.explain_instance(np.vstack([imageArr]), model.predict,
8                                         top_labels=5, hide_color=0, num_samples=1000)
9
10 from skimage.segmentation import mark_boundaries
11
12 camImages = [imageOrig]
13 fig = plt.figure(figsize=(20., 20.))
14 grid = ImageGrid(fig, 111, # similar to subplot(111)
15                   nrows_ncols=(2, 3),
16                   axes_pad=0.5, # pad between axes in inch.
17                   )
18 for class_index in range(0,5):
19     temp, mask = explanation.get_image_and_mask(explanation.top_labels[
20         class_index], positive_only=True, num_features=5, hide_rest=True)
21     camImages.append(mark_boundaries(temp / 2 + 0.5, mask))
22
23 i = -1
24 for ax, im in zip(grid, camImages):
25     # Iterating over the grid returns the Axes.
26     ax.set_xticks([])
27     ax.set_yticks([])
28     label = labels[0][i]
29     if i >= 0:
30         ax.set_title('%s (%.2f%%)' % (label[1], label[2]*100))
31     ax.imshow(im)
32     i = i + 1
33
34 plt.show()

```

Listing 8.3: Visualisiertes Neuronales Netz mit Tensorflow und lime

8.1.4 Visualisierungen mehrere Klassen

```

1 import tensorflow as tf
2 from keras.applications.vgg16 import preprocess_input
3 from keras.applications.vgg16 import decode_predictions
4
5 model = tf.keras.applications.VGG16(weights="imagenet", include_top=True)
6
7 #print(model.summary())
8
9 imageOrig = tf.keras.preprocessing.image.load_img('D:/Master Thesis/Repo/Test
10     Images/tabby.2.JPG', target_size=(224, 224))
11 imageArr = tf.keras.preprocessing.image.img_to_array(imageOrig) #output Numpy
12     -array
13
14 imageReshaped = imageArr.reshape((1, imageArr.shape[0], imageArr.shape[1],
15     imageArr.shape[2]))
16
17 image = preprocess_input(imageReshaped)
18 predictions = model.predict(imageReshaped)
19
20 import numpy as np
21 top5predictions = np.argsort(predictions)[0,::-1][:5]

```

```

20 labels = decode_predictions(predictions, 15)
21
22 from tf_explain.core.grad_cam import GradCAM
23 from mpl_toolkits.axes_grid1 import ImageGrid
24 from matplotlib import pyplot as plt
25 import cv2
26
27 def createImageGrid(imageOrig, predictions, labels, explainer, explainerArgs):
28     camImages = [imageOrig]
29
30     for class_index in top5predictions:
31         camImages.append(explainer.explain(class_index=class_index, **
32                                         explainerArgs))
33
34     fig = plt.figure(figsize=(20, 20))
35     grid = ImageGrid(fig, 111,                      # as in plt.subplot(111)
36                      nrows_ncols=(2,3),
37                      axes_pad=0.5,
38                      share_all=True,
39                      cbar_location="right",
40                      cbar_mode="single",
41                      cbar_size="7%",
42                      cbar_pad=0.15,
43                      )
44
45     i = -1
46     for ax, im in zip(grid, camImages):
47         ax.set_xticks([])
48         ax.set_yticks([])
49         label = labels[0][i]
50         if i >= 0:
51             ax.set_title('%s (%.2f%%)' % (label[1], label[2]*100), fontsize
52 =20)
53             ax.imshow(imageOrig)
54             imAx = ax.imshow(im, vmin=0, vmax=255)
55         else:
56             ax.imshow(im)
57         i = i + 1
58
59     # Colorbar
60     ax.cax.colorbar(imAx)
61     ax.cax.toggle_label(True)
62     ax.cax.set_yticks([0,128,255])
63     ax.cax.set_yticklabels([-1,0,1])
64     plt.show()
65
66 img = tf.keras.preprocessing.image.img_to_array(imageOrig)
67 data = ([img], None)
68
69 explainer = GradCAM()
70 createImageGrid(imageOrig, predictions, labels, explainer, {'model': model, 'layer_name': 'block5_conv3', 'validation_data': data})
71
72 input_images = [preprocess_input(imageArr.copy())]
73
74 from tf_explain.core.gradients_inputs import GradientsInputs
75 data = (np.array(input_images), None)
76
77 explainer = GradientsInputs()
78 createImageGrid(imageOrig, predictions, labels, explainer, {'model': model, 'layer_name': 'block5_conv3', 'validation_data': data})

```

```

    validation_data': data})

77
78 from tf_explain.core.integrated_gradients import IntegratedGradients
79
80 input_images = [preprocess_input(imageArr.copy())]
81 data = (input_images, None)
82
83 explainer = IntegratedGradients()
84 createImageGrid(imageOrig, predictions, labels, explainer, {'model': model,
     validation_data': data})
85
86 from tf_explain.core.occlusion_sensitivity import OcclusionSensitivity
87
88 img = tf.keras.preprocessing.image.img_to_array(imageOrig)
89 data = ([img], None)
90
91 explainer = OcclusionSensitivity()
92 createImageGrid(imageOrig, predictions, labels, explainer, {'model': model,
     validation_data': data, 'patch_size': 40})
93
94 import lime
95 from lime import lime_image
96
97 imgAsArray = tf.keras.preprocessing.image.img_to_array(imageOrig)
98 out = []
99 x = np.expand_dims(imgAsArray, axis=0)
100 x = preprocess_input(x)
101 out.append(x)
102 imagesStack = np.vstack(out)
103
104 explainer = lime_image.LimeImageExplainer()
105
106 explanation = explainer.explain_instance(imagesStack[0], model.predict,
     top_labels=5, hide_color=0, num_samples=1000)
107
108 from skimage.segmentation import mark_boundaries
109
110 camImages = [imageOrig]
111 fig = plt.figure(figsize=(20., 20.))
112 grid = ImageGrid(fig, 111, # similar to subplot(111)
     nrows_ncols=(2, 3),
     axes_pad=0.5, # pad between axes in inch.
     )
113 for class_index in range(0,5):
114     temp, mask = explanation.get_image_and_mask(explanation.top_labels[
         class_index], positive_only=True, num_features=5, hide_rest=True)
115     camImages.append(mark_boundaries(temp / 2 + 0.5, mask))
116
117 i = -1
118 for ax, im in zip(grid, camImages):
119     # Iterating over the grid returns the Axes.
120     ax.set_xticks([])
121     ax.set_yticks([])
122     label = labels[0][i]
123     if i >= 0:
124         ax.set_title('"%s (%.2f%%)"' % (label[1], label[2]*100), fontsize=20)
125         ax.imshow(im.astype(np.uint8))
126     else:
127         ax.imshow(im)
128     i = i + 1

```

```
132  
133 plt.show()
```

Listing 8.4: Visualisierungen der fünf wahrscheinlichsten Klassen mit verschiedenen Verfahren

Akronyme

AI Artificial Intelligence. 1

AIP Adversarial Image Perturbations. 7

CNN Convolutional Neural Network. 29, 30, 34, 38

DEK Datenethikkommission. 6

DNN Deep Neural Network. 29

DSGVO

Datenschutz-Grundverordnung. 6

GAM

Generalized Additive Models. 12–14

GLM

Generalized Linear Models. 12–14

Grad CAM

Gradient-weighted Class Activation Mapping. 32

LFR Learned fair representations. 12

LIME

Local interpretable model-agnostic explanations. 23, 24, 47

LRP Layer-wise Relevance Propagation. 16, 21, 22

LSTM

Long short-term memory. 47

M-GBM

Monotonic gradient boosting. 12

ML Machine Learning. 1–7, 10, 23, 29, 43, 50, 52, 53

PATE

Private aggregation of teacher ensembles. 12

SBRL

Scalable Bayesian rule list. 12

SLIM

Supersparse linear integer models. 12

SVCCA

Singular Vector Canonical Correlation Analysis. 27

TCAV

Testing with Concept Activation Vectors. 26

XAI Explainable artificial intelligence. 3, 6–8, 12, 16, 42

Glossar

Bias Bias, deutsch Tendenz oder Voreingenommenheit, kann in Machine Learning Modellen auftreten wenn die Trainingsdaten unausgewogen sind. Dies kann zu einer sogenannten selbsterfüllenden Prophezeiung werden indem die Resultate des Models zu neuen Trainingsdatensätzen führen welche die Tendenz noch verstärken.. 16, 50

Binärer Klassifikator

Ein binärer Klassifikator ist eine Sonderform eines Klassifikators welche nur eine Klasse kennt. Häufig sind das ja/nein Fragen zum Beispiel "ist auf diesem Bild ein Tumor erkennbar?". 30

Black Box

System welches nicht im Quellcode vorhanden ist und dadurch nicht durch Analyse der Programmierung verstanden werden kann. Jegliche Rückschlüsse sind nur durch Beobachtungen möglich.. 7, 47

Decision Tree

Entscheidungsbaum, Familie von ML Algorithmen. 12, 14, 42, 43, 47

Deep Neural Network

deutsch tiefes lernen, Bezeichnet Neuronale Netze mit vielen Zwischenschichten.. 17

Explainable Artificial Intelligence

deutsch erklärbare künstliche Intelligenz, Methodiken um Menschen die Vorhersagen durch Modelle des maschinellen Lernens zu erläutern. . 36, 41

Grad CAM

Gradient-weighted Class Activation Mapping, Technik welche für eine Entscheidung relevanten Bildinhalte optisch hervorhebt. 16, 17, 22, 33, 38, 39

Gradienten

Vektor dessen Komponenten die partiellen Ableitungen im Punkt P sind, der Gradient zeigt deshalb in die Richtung der größten Änderung. . 17

Gradients Input

. 16, 32, 33, 39

Kluger-Hans-Effekt

"Kluger Hans" war ein Pferd aus dem Anfang des 20. Jahrhunderts das angeblich rechnen und zählen konnte, jedoch auf die feinen Nuancen der Mimik und Körpersprache des Fragestellers reagierte. Seitdem wird als "Kluger-Hans-Effekt" eine unbewusste Beeinflussung des Studienobjektes bezeichnet. In Machine Learning Lösungen kann der "Kluger-Hans-Effekt" auftauchen wenn ein Model mit Daten trainiert wird welche die Vorhersage unbewusst in eine bestimmte Richtung lenken.. 29

LIME

Local interpretable model-agnostic explanations, eine unabhängig des verwendeten Algorithmus anwendbare Erklärungstechnik für Black Box Modelle . 39, 45, 46

Machine Learning

deutsch Maschinelles lernen. Ein künstliches System lernt aus Beispielen und kann diese nach Beendigung der Lernphase verallgemeinern.. 41

Naive Bayes

Auch Naiver Bayes-Klassifikator genannt. Basiert auf dem Bayesschen Theorem mit der naiven Grundannahme, dass jedes Attribut nur vom Klassenattribut abhängt. Obwohl dies häufig nicht der Fall ist funktionieren Naive Bayes Klassifikatoren sehr gut. . 46

neuronales Netz

Algorithmus welcher nach dem Vorbild des menschlichen Gehirns entworfen wurde um Muster und Beziehungen in Daten zu erkennen.. 7, 22, 33, 52, 53

Occlusion Sensitivity

Verfahren um die für eine Klassifikation relevanten Bildinhalte zu finden indem bestimmte Bildinhalte entfernt werden (Occlusion) und die dabei entstehende Veränderung auf die Klassifikation gemessen wird.. 16, 18, 19, 32, 33, 36, 39

White Box

System über welches man Kenntnisse der inneren Funktionsweise hat. In der Regel ist der Sourcecode des Systems verfügbar und kann für die Analyse verwendet werden.. 47

Abbildungsverzeichnis

1.1	Entwicklung des Machine Learning als Zeitachse	2
2.1	Ablauf einer erklärbaren Machine Learning Anwendung	4
3.1	Biplot Iris Datensatz	8
3.2	Korrelation verschiedener Datensätze als Graph	8
3.3	Korrelationsmatrix als Heatmap dargestellt	9
3.4	Heatmap als Kalender	9
3.5	Quelle: https://github.com/h2oai/mli-resources	11
3.6	Voraussetzungen lineare Regression	13
3.7	Auschluss-Bedingungen lineare Regression	14
3.8	Entscheidungsbaum visualisiert.	15
3.9	Entscheidungsbaum als Flächen dargestellt	15
3.10	scope-rules Ausgabe der Regeln	15
4.1	Grad CAM Analyse für verschiedene Klassen	17
4.2	Occlusion Sensitivity Beispiel	19
4.3	Testbild mit Occlusion Sensivity	19
4.4	LRP Berechnung	22
4.5	Pixel basierte Relevanz	22
4.6	Vergleich Pixel-wise und LRP	22
4.7	Darstellung relevanter Bildinhalte durch LIME	24
4.8	Darstellung Vorgehensweise TCAV	27
4.9	Vergleich Verschiedener Klassen mit SVCCA	28
5.1	Klassifizierung eines Pferdes in Pascal VOC	29
5.2	fiktives Logo	30
5.3	Ursprüngliches Bild	30
5.4	Manipuliertes Bild	30
5.5	Log-loss / Accuracy Dog vs. Cat	31
5.6	Bewertung des Hund-Katze Netzwerkes	31
5.7	Testbild ohne Logo	32
5.8	Testbild mit Logo	33
5.9	Testbild Katze ohne Logo	33
5.10	Testbild Katze mit Logo	34
5.11	Testbild Sonnenblume	35
5.12	Testbild Falter ohne Logo	35
5.13	Testbild Falter mit Logo	36
5.14	Original Testbild Katze	37
5.15	Testbild Katze visualisiert mit Grad CAM	37
5.16	Testbild Toilettepapier-Rolle	38
5.17	Testbild Meerschweinchen	38
5.18	Testbild Meerschweinchen Grad CAM	39
5.19	Testbild Meerschweinchen div. Verfahren	40

5.20	Testbild getigerte Katze	41
5.21	Testbild getigerte Katze frontal	41
5.22	Performance Sentiment Analyse mit DecisionTree	42
5.23	Konfusions-Matrix Texterkennungs-Experiment	43
5.24	Top Features Film Review Klassifizierung	44
5.25	Visualisierung positives Film Review mit ELI5	44
5.26	Visualisierung negatives Film Review mit ELI5	45
5.27	Visualisierung positives Film Review mit LIME	46
5.28	Visualisierung negatives Film Review mit LIME	46
5.29	Konfusionsmatrix, Test Zusammenfassung und Accuracy für Naive Bayes	47
5.30	Visualisierung positives Film Review mit LIME und NB	47
5.31	Visualisierung negatives Film Review mit LIME und NB	47
5.32	Zusammenfassung LSTM Netz für Movie Sentiment Analyse	48
5.33	Performance des LSTM Netz für Movie Sentiment Analyse	48
5.34	Visualisierung Movie Review 413	49
5.35	Visualisierung Movie Review 414	49
6.1	Ursprungs-Bild für Adversarial Angriff	51
6.2	Durch Adversarial Angriff erzeugtes Bild	51
6.3	Adversarial Beispiel Szegedy 2013	51
6.4	Gefälschtes Verkehrsschild als Adversarial Attack	52

Tabellenverzeichnis

Literaturverzeichnis Artikel

- 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 (O. D. Suarez, Hrsg.). *PLOS ONE*, 10(7), e0130140.
<https://doi.org/10.1371/journal.pone.0130140>
- Everingham, M., Gool, L. V., Williams, C. K. I., Winn, J. & Zisserman, A. (2010). The PASCAL Visual Object Classes (VOC) Challenge.
- Friedman, J. H. & Popescu, B. E. (2008). Predictive learning via rule ensembles. *Annals of Applied Statistics* 2008, Vol. 2, No. 3, 916-954, arXiv <http://arxiv.org/abs/0811.1679v1>.
<https://doi.org/10.1214/07-AOAS148>
- Goodfellow, I. J., Shlens, J. & Szegedy, C. (2014). Explaining and Harnessing Adversarial Examples, arXiv <http://arxiv.org/abs/1412.6572v3>.
- Gu, T., Liu, K., Dolan-Gavitt, B. & Garg, S. (2019). BadNets: Evaluating Backdooring Attacks on Deep Neural Networks. *IEEE Access*, 7, 47230–47244. <https://doi.org/10.1109/access.2019.2909068>

- Kim, B., Wattenberg, M., Gilmer, J., Cai, C., Wexler, J., Viegas, F. & Sayres, R. (2017). Interpretability Beyond Feature Attribution: Quantitative Testing with Concept Activation Vectors (TCAV). *ICML 2018*, arXiv <http://arxiv.org/abs/1711.11279v5>.
- Lakkaraju, H., Kamar, E., Caruana, R. & Leskovec, J. (2017). Interpretable & Explorable Approximations of Black Box Models, arXiv <http://arxiv.org/abs/1707.01154v1>.
- Lapuschkin, S., Binder, A., Montavon, G., Müller, K.-R. & Samek, W. (2016). The LRP Toolbox for Artificial Neural Networks. *Journal of Machine Learning Research*, 17(114), 1–5. <http://jmlr.org/papers/v17/15-618.html>
- Lapuschkin, S., Wäldchen, S., Binder, A., Montavon, G., Samek, W. & Müller, K.-R. (2019). Unmasking Clever Hans Predictors and Assessing What Machines Really Learn, arXiv <http://arxiv.org/abs/1902.10178v1>. <https://doi.org/10.1038/s41467-019-08987-4>
- Oh, S. J., Schiele, B. & Fritz, M. (2019). Towards Reverse-Engineering Black-Box Neural Networks, 121–144. https://doi.org/10.1007/978-3-030-28954-6_7
- Pang, B. & Lee, L. (2004). A Sentimental Education: Sentiment Analysis Using Subjectivity Summarization Based on Minimum Cuts.
- Papernot, N., Faghri, F., Carlini, N., Goodfellow, I., Feinman, R., Kurakin, A., Xie, C., Sharma, Y., Brown, T., Roy, A., Matyasko, A., Behzadan, V., Hambardzumyan, K., Zhang, Z., Juang, Y.-L., Li, Z., Sheatsley, R., Garg, A., Uesato, J., ... Long, R. (2018). Technical Report on the CleverHans v2.1.0 Adversarial Examples Library. *arXiv preprint arXiv:1610.00768*.
- Raghu, M., Gilmer, J., Yosinski, J. & Sohl-Dickstein, J. (2017). SVCCA: Singular Vector Canonical Correlation Analysis for Deep Learning Dynamics and Interpretability, arXiv <http://arxiv.org/abs/1706.05806v2>.
- Ras, G., van Gerven, M. & Haselager, P. (2018). Explanation Methods in Deep Learning: Users, Values, Concerns and Challenges, 19–36. https://doi.org/10.1007/978-3-319-98131-4_2
- Ribeiro, M. T., Singh, S. & Guestrin, C. (2016). "Why Should I Trust You?". <https://doi.org/10.1145/2939672.2939778>
- Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D. & Batra, D. (2016). Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization, arXiv <http://arxiv.org/abs/1610.02391v4>. <https://doi.org/10.1007/s11263-019-01228-7>
- Simonyan, K. & Zisserman, A. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition, arXiv <http://arxiv.org/abs/1409.1556v6>.
- Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I. & Fergus, R. (2013). Intriguing properties of neural networks, arXiv <http://arxiv.org/abs/1312.6199v4>.
- Zeiler, M. D. & Fergus, R. (2013). Visualizing and Understanding Convolutional Networks, arXiv <http://arxiv.org/abs/1311.2901v3>.
- Zhang, X., Zhu, X. & Lessard, L. (2019). Online Data Poisoning Attack, arXiv <http://arxiv.org/abs/1903.01666v2>.

Literaturverzeichnis Bücher

Explainable and Interpretable Models in Computer Vision and Machine Learning. (2018, 1. September). Springer-Verlag GmbH. https://www.ebook.de/de/product/33610206/explainable_and_interpretable_models_in_computer_vision_and_machine_learning.html

Linkverzeichnis

- Dataset Dog vs. Cats. (o.D.). <https://www.kaggle.com/c/dogs-vs-cats>
der Bundesregierung, D. (2019). Gutachten der Datenethikkommission.
https://www.bmjjv.de/SharedDocs/Downloads/DE/Themen/Fokusthemen/Gutachten_DEK_DE.pdf?__blob=publicationFile&v=2
- Eddins, S. (2017). Network Visualization Based on Occlusion Sensitivity.
<https://blogs.mathworks.com/deep-learning/2017/12/15/network-visualization-based-on-occlusion-sensitivity/>
- ELI5: A library for debugging/inspecting machine learning classifiers and explaining their predictions [ELI5]. (o.D.). <https://eli5.readthedocs.io/>
- Goodfellow, I. (2020). CleverHans a Python library to benchmark machine learning systems' vulnerability to adversarial examples. <https://github.com/tensorflow/cleverhans>
- Habegger, M. (o.D. a). Beispiel "Kluger-HansEffekt in manipuliertem CNN. https://github.com/habis-git/MT/blob/master/Models/Manipulated%20Data/Cat_vs_Dog_Manipulated_visualization.ipynb
- Habegger, M. (o.D. b). Manipulated Tensorflow CNN Dog vs. Cats.
<https://github.com/habis-git/MT/blob/master/Models/Manipulated%20Data/model.h5>
- Habegger, M. (o.D. c). Text Klassifizierung mit Decision Tree. <https://github.com/habis-git/MT/blob/master/Jupyter%20Notebooks/TextClassificationDT.ipynb>
- Habegger, M. (202). Text Klassifizierung eines LSTM Modelles mit LIME.
<https://github.com/habis-git/MT/blob/master/Jupyter%20Notebooks/Moveriew%20Sentiment%20with%20Keras%20LSTM%20and%20lime%20explainer%20.ipynb>
- Habegger, M. (2020a). Kombinierte Verfahren für die Erklärung von Bild-Klassifikationen mit tf_explain und lime. <https://github.com/habis-git/MT/blob/master/Jupyter%20Notebooks/CombinedVisualizations.ipynb>
- Habegger, M. (2020b). Text Klassifizierung mit ELI5.
<https://github.com/habis-git/MT/blob/master/Jupyter%20Notebooks/TextClassification.ipynb>
- Habegger, M. (2020c). Text Klassifizierung mit lime. <https://github.com/habis-git/MT/blob/master/Jupyter%20Notebooks/TextClassificationLime.ipynb>
- ImageNet Challenge. (o.D.). <http://www.image-net.org/challenges/LSVRC/>
- Institut, F. (o.D.). Explainable AI Demos. <https://lrvserver.hhi.fraunhofer.de/handwriting-classification>
- Kim, B. (2018). Interpretability Beyond Feature Attribution: Quantitative Testing with Concept Activation Vectors (TCAV) [ICML 2018]. <https://github.com/tensorflow/tcav>
- Lapuschkin, S. (2020). The LRP Toolbox for Artificial Neural Networks.
https://github.com/sebastian-lapuschkin/lrp_toolbox
- Lee, P. (2016). Learning from Tay's introduction.
<https://blogs.microsoft.com/blog/2016/03/25/learning-tays-introduction/>
- Malik, U. (2018). Text Classification with Python and Scikit-Learn.
<https://stackabuse.com/text-classification-with-python-and-scikit-learn/>
- Meudec, R. (o.D.). tf-explain. <https://github.com/sicara/tf-explain>
- Molnar, C. (2019). Interpretable Machine Learning, A Guide for Making Black Box Models Explainable.
<https://christophm.github.io/interpretable-ml-book/>
- Natural Language Toolkit [NLTK]. (o.D.). <https://www.nltk.org/>
- A Quick Introduction to Deep Taylor Decomposition. (2017). <http://www.heatmapping.org/deeptaylor/>
- Raghu, M. (2017). Interpreting Deep Neural Networks with SVCCA.
<https://ai.googleblog.com/2017/11/interpreting-deep-neural-networks-with.html>
- Ribeiro, M. T. C. (2019). Lime: Explaining the predictions of any machine learning classifier [Lime].
<https://github.com/marcotcr/lime>
- scikit-learn Machine Learning in Python [scikit-learn]. (o.D.). <https://scikit-learn.org/stable/index.html>

- skope-rules. (2019). <https://github.com/scikit-learn-contrib/skope-rules>
- Steve Povolny, S. T. (2020). Model Hacking ADAS to Pave Safer Roads for Autonomous Vehicles. <https://www.mcafee.com/blogs/other-blogs/mcafee-labs/model-hacking-adas-to-pave-safer-roads-for-autonomous-vehicles/>
- Surma, G. (2018). Image Classifier Cats vs Dogs. <https://towardsdatascience.com/image-classifier-cats-vs-dogs-with-convolutional-neural-networks-cnns-and-google-colabs-4e9af21ae7a8>
- Tensorflow. (o.D.). <https://www.tensorflow.org/>
- Tensorflow. (2018). VGG16 Modell für Imagenet Klassifikationen. https://github.com/tensorflow/tensorflow/blob/r1.8/tensorflow/python/keras/_impl/keras/applications/vgg16.py
- Thiebaut, N. (2017). LIME of words: interpreting Recurrent Neural Networks predictions. <https://data4thought.com/deep-lime.html>
- Tutorial - Image Classification Keras. (2019). <https://github.com/marcotcr/lime/blob/master/doc/notebooks/Tutorial%20-%20Image%20Classification%20Keras.ipynb>
- Tutorial, T. (2020). Adversarial example using FGSM. https://www.tensorflow.org/tutorials/generative/adversarial_fgsm

Listings

4.1 Grad CAM Visualisierung für die wahrscheinlichste Klasse	18
4.2 Occlusion Sensitivity Visualisierung für die wahrscheinlichste Klasse	21
4.3 Lime Visualisierung für die wahrscheinlichste Klasse	25
5.1 CNN für Dog vs. Cats	30
5.2 LSTM Modell für LIME Movie Sentiment Analyse	48
8.1 Decision Tree Visualisierung	55
8.2 Visualisiertes Neuronales Netz mit Tensorflow und tf-explain	56
8.3 Visualisiertes Neuronales Netz mit Tensorflow und lime	57
8.4 Visualisierungen der fünf wahrscheinlichsten Klassen mit verschiedenen Verfahren	58