DOI 10.5162/sensoren2024/A1.4

Deep Neural Network Repräsentation für interpretierbare Machine Learning Algorithmen: Eine Methode zur effizienten Hardware-Beschleunigung

Julian Schauer, Payman Goodarzi, Andreas Schütze, Tizian Schneider
Lehrstuhl für Messtechnik, Universität des Saarlandes, Saarbrücken, Deutschland
+49 682 -302 4674, (j.schauer),(p.goodarzi), (schuetze), (t.schneider)@lmt.uni-saarland.de

Zusammenfassung

Algorithmen für Deep Learning und maschinelles Lernen haben bei verschiedenen Aufgaben, von Bilderkennung und Spracherkennung bis hin zu Zustandsüberwachung und vorausschauender Wartung, gute Ergebnisse erzielt. Allerdings treffen die meisten Algorithmen des Deep Learning keine einfach nachvollziehbaren Entscheidungen. Dadurch haben interpretierbare ML-Algorithmen, auf Basis von Merkmalsextraktion und -selektion, an Aufmerksamkeit gewonnen. Da Hardware-Beschleuniger jedoch häufig ausschließlich neuronale Netze unterstützen, resultiert dies auf Edge-Hardware in einer ineffizienten Implementierung anderer, interpretierbarer Algorithmen und beschränkt den Einsatz dieser Algorithmen. Diese Veröffentlichung präsentiert daher eine Methode, um trainierte Machine Learning Algorithmen als Deep Neural Networks darzustellen, sodass eine Inferenz auf effizienter Edge Hardware für smarte Sensorsysteme ermöglicht wird. Die Methode bricht die interpretierbaren Algorithmen auf ihre grundlegenden mathematischen Operationen herunter, um diese durch Schichten neuronaler Netze darzustellen. Die Methodik wird in der Veröffentlichung im Detail beschrieben, anhand von mehreren Beispielalgorithmen angewandt und auf einer Neural Processing Unit demonstriert.

Keywords: Edge Computing, Smart Sensors, Machine Learning, Deep Neural Networks

I. Einführung

In den letzten Jahren ist die Zahl der auf maschinellem Lernen (ML) basierenden Anwendungen im industriellen Umfeld ins-besondere für Aufgaben wie vorausschauende Wartung [1] und intelligente Zustands-überwachung [2] gestiegen. Die ML-Methoden und Algorithmen versuchen bevorstehende Ausfälle der überwachten Maschine vorherzusagen, wodurch der Betreiber frühzeitig reagieren kann. Die Einführung der automatisierten Überwachung und der voraus-schauenden Wartung von Industrieanlagen senkt Personal- und Wartungskosten und reduziert die Ausfallzeiten der Anlagen [3]. Für die Verarbeitung und die Analyse der an der Maschine entstehenden Prozessdaten mittels ML, müssen diese in der Regel an leistungsstarke Instanzen (z. B. Server) weitergeleitet werden. Smarte Sensoren [4] können diesen energie- und bandbreiten-intensiven Schritt umgehen. Dazu führen die Sensoren, die Signalverarbeitung bis hin zur vollständiger Inferenz von ML-Modellen direkt am Sensor durch [4]. Die dabei eingesetzten Hardwarebeschleuniger weisen Einschränkungen bezüglich Rechenleistung und Speicherkapazität auf, welche beim Einsatz von ML-Methoden kritisch zu betrachten sind. In mehreren bereits veröffentlichten Artikeln wird die Verwendung von begrenzter Edge Hardware, im Hinblick auf die Anwendung von ML-Methoden bewertet [4, 5]. Weiterhin erfordert der Einsatz dieser Beschleuniger umfangreiche, auf Hardware und ML-Algorithmen abgestimmte Softwareframeworks, die typischerweise nur für neuronale Netze verfügbar sind.

Gerade im Bereich der vorrausschauenden Wartung und der intelligenten Zustands-überwachung, haben sich interpretierbare ML- Methoden bewährt [6, 7], die aus Merkmals-Extraktion (ME), Merkmals-Selektion (MS) und einem Klassifikator/Regressor (KR) bestehen [8]. Die Auto ML-Toolbox der Universität des Saarlandes [9] bietet eine große Auswahl dieser Methoden. Die Algorithmen bieten durch ihre Linearität und Einfachheit eine inhärente Interpretierbarkeit ihrer Modelle und Entscheidungen.

Im Gegensatz dazu stehen komplexe neuronale Netze (NN), die besonders beim Erkennen von nicht-linearen Zusammenhängen zwischen Signalmustern und Zielgröße gute Ergebnisse erzielen [10]. Diese Eigenschaft kann aber zum Einsatz von NN als „Blackbox-Systeme“ [11] führen. Dadurch liefert das NN keine interpretierbare Lösung für das anstehende Problem. Gleichzeitig ist die Verfügbarkeit spezialisierter und optimierter Hardware ein großer Vorteil der NN. Das vereinfacht die Implementierung auf Edge Hardware oder smarten Sensoren im Vergleich zu den interpretierbaren ML-Algorithmen erheblich.

In früheren Arbeiten wurden Teile von interpretierbaren ML-Algorithmen als Matrix-multiplikation dargestellt oder eine aufwändige, Hardware spezialisierte Implementierung von Datenverarbeitungsalgorithmen gezeigt, um die Methoden auf optimierter Hardware zu beschleunigen [11, 12]. Diese Ansätze resultieren in einem aufwändigen und hardware-spezifischen Programmieren von Teil Algorithmen einer Signalverarbeitungskette. Um die Inferenz eines interpretierbaren ML-Algorithmus auf optimierter Hardware auszuführen, wird in dieser Arbeit eine Methode vorgestellt, einen vollständig trainierten interpretierbaren ML-Algorithmus als NN mit statischen Gewichten und Parametern darzustellen. Die Darstellung basiert auf der Zerlegung der interpretierbaren ML-Algorithmen in die grundlegenden mathematischen Operationen und die anschließende Repräsentation durch NN-Schichten. Dadurch werden die Vorteile von physikalisch verständlichen Algorithmen der interpretierbaren Methoden und die beschleunigte und effiziente Hardwareimplementierung der NN kombiniert.

Der Rest der Arbeit ist wie folgt aufgebaut: Der erste Abschnitt beschreibt den Ablauf der Entwicklung der Algorithmen basierend auf der Open-Source ML-Toolbox, über das Training und die Validierung, bis hin zur Implementierung auf effizienter Edge Hardware, siehe Abb. 1. In Abschnitt 3 wird die zugrundeliegende Methode der NN-Repräsentation der trainierten ML-Algorithmen im Detail beschrieben. Abschnitt 4 zeigt die Ergebnisse der vorgestellten Methode anhand von Teilen der Open-Source ML-Toolbox. In den letzten beiden Abschnitten werden die Ergebnisse diskutiert und ein Ausblick auf zukünftige Arbeiten gegeben.

II. Methoden

Die verwendete automatisierte ML-Toolbox besteht aus mehreren, sich gegenseitig ergänzenden und interpretierbaren ML-Algorithmen für Merkmalsextraktion, Merkmalsselektion und Klassifikation/Regression. diese werden automatisiert verglichen, um den besten Algorithmus für das jeweilige Lernproblem auszuwählen. Das resultierende Modell kann anschließend interpretiert werden [14].Der in Abb. 1 dargestellte Arbeitsablauf beschreibt die drei Hauptschritte der Implementierung eines interpretierbaren ML-Algorithmus auf Edge-Hardware, die chronologisch ausgeführt werden müssen. Diese drei Schritte sind das Training mit Modellauswahl, die Umwandlung des trainierten Modells in ein DNN und die Inferenz des

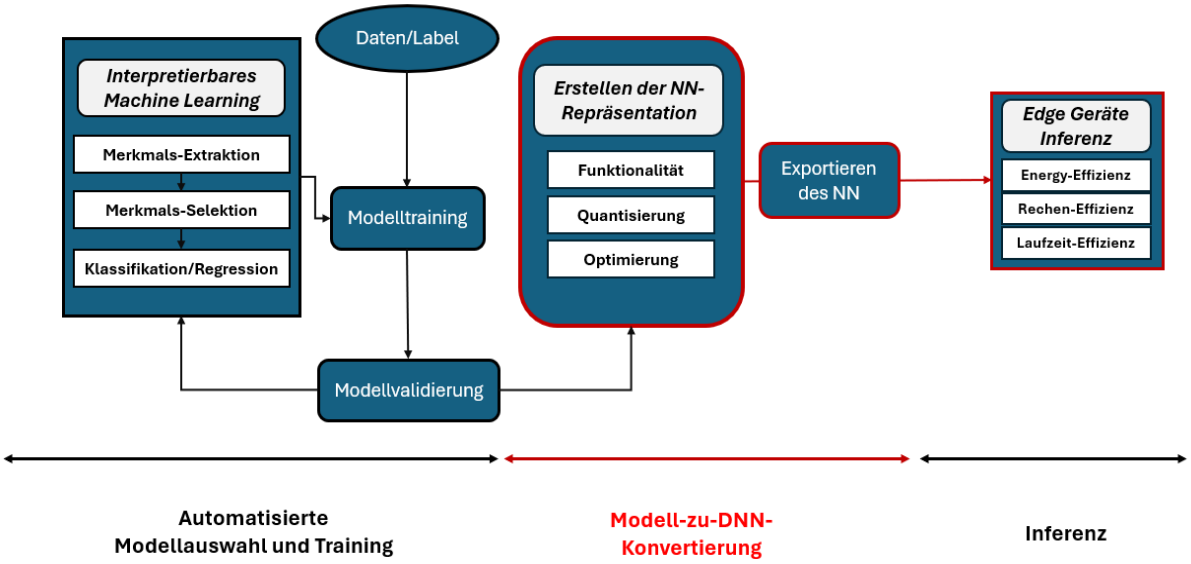

Abb. 1: Ablaufplan der automatisierten Modellauswahl, der Konvertierung des Modells und der Hardware-Inferenz

exportierten Modells auf optimierter Hardware. Außerdem wird kurz auf die typischerweise verfügbaren Daten eingegangen

a) Automatisierte Modellauswahl und Training

Der erste Schritt sind die Auswahl und das Training eines interpretierbaren ML-Algorithmus für den gewünschten Anwendungsfall. Die zugrunde liegende ML-Toolbox bietet verschiedene MESK/R-Algorithmen, um einen geeigneten Algorithmus für verschiedene ML-Probleme zu finden. Die Open-Source ML-Toolbox kann diesen Schritt automatisch und ohne Benutzereingriff durchführen. Die Toolbox validiert sich gegenseitig ergänzende Algorithmen und macht dann eine Aussage über die Qualität des MESK/R-Modells.

b) Modell-zu-DNN-Konvertierung

Nachdem ein geeignetes Modell gefunden wurde, muss das trainierte interpretierbare ML-Modell als DNN abgebildet werden. Dieser Schritt ist das Hauptaugenmerk dieser Veröffentlichung. Die Methode des Mappings, insbesondere die Wiederverwendung der Funktionalität, wird in Abschnitt 3 detailliert beschrieben. Der Ansatz besteht darin, ein trainiertes, interpretierbares ML-Modell als eine chronologische Ausführung von mathematischen Operationen zu begreifen, die dann mit DNN-Schichten dargestellt werden. Nachdem das Modell trainiert ist, kann eine statische Funktion die Berechnung der Ausgabe beschreiben.

c) Inferenz

Nach erfolgreicher Generierung der DNN-Darstellung des interpretierbaren ML-Algorithmus kann das DNN auf optimierter Edge-Hardware ausgeführt werden. Das Hauptaugenmerk bei der Optimierung liegt auf der Inferenz. Der Einsatz eines ML-Modells führt zu einem statischen Modell, das aus den Eingabedaten Vorhersagen trifft. Dieses kann mit Hilfe gängiger Frameworks für die Hardwarebeschleunigung neuronaler Netze genutzt werden Die Verwendung dieses Arbeitsablaufs führt einerseits zur Ausführung auf optimierter Hardware und bietet andererseits Laufzeit- und Energievorteile bei jeder nachfolgenden Algorithmus-Inferenz und erfordert keine aufwendige Anpassung des Software-Frameworks zur Hardwarebeschleunigung.

d) Daten

Die MESK/R-Modelle werden in der Regel auf Zeitreihendaten von verschiedenen Sensoren trainiert. Als Beispiel für einen Anwendungsfall der Zustandsüberwachung werden die Modelle exemplarisch auf dem Datensatz eines Hydrauliksystems trainiert [15]. Ziel ist die Überwachung des Ventilschaltverhaltens eines hydraulischen Prüfstandes auf Grundlage eines Drucksensors (Im Datensatz als PS1 bezeichnet). Während des 60 s langen Arbeitszyklus der Maschine wird der Drucksensor mit einer Frequenz von 100 Hz abgetastet. Die Zielgröße zeigt vier Zustände des Ventils an (100%: optimales Schalten, 90%: geringe Verzögerung, 80%: starke Verzögerung und 73%: nahe am Totalausfall). Dieser Datensatz kann ein Klassifikations- und Regressionsproblem darstellen.

III. UMWANDLUNG VON MODELLEN

Die Umwandlung eines MESK/R-Modells in ein DNN erfordert ein tiefgreifendes Verständnis der Funktionalität und der Bausteine der Ausgangsalgorithmen. Die Abbildung des Algorithmus erfordert die Aufschlüsselung des Algorithmus in die grundlegenden mathematischen Operationen. Der zugrundeliegende mathematische Prozess eines interpretierbaren ML-Algorithmus kann als eine chronologische Ausführung von mathematisch basierten Operationen gesehen werden. Bei diesen Operationen handelt es sich hauptsächlich um Matrixmultiplikation, Vektoradditionen oder -subtraktionen, Divisionen, Quadratwurzelberechnungen oder FIR-Filterungsoperationen. Wie der zugrundeliegende interpretierbare ML-Algorithmus kann auch ein statisches DNN als eine chronologische Ausführung von verschiedenen Schichten betrachtet werden. Mit diesem Wissen und dem grundlegenden Verständnis des Algorithmus und der DNNs kann der trainierte interpretierbare ML-Algorithmus also mit minimalem Programmieraufwand in ein hardwarebeschleunigt ausführbares Format umgewandelt werden. Im folgenden Abschnitt wird die MESK/R-Verarbeitungskette der ML-Toolbox kurz beschrieben. Außerdem werden die wichtigsten Operationen erläutert, die nicht auf diesen speziellen Anwendungsfall beschränkt sind.

a) Interpretierbarer ML-Algorithmus

Die zugrunde liegende ML-Toolbox unterteilt die interpretierbaren Algorithmen in drei grundlegende Verarbeitungsschritte. Die Ergebnisse eines jeden Verarbeitungsschritts kann analysiert werden. Dies ermöglicht es dem Benutzer, die Ergebnisse für jeden Schritt nachzuvollziehen und den Algorithmus interpretierbar zu machen. Eine Auflistung der verfügbaren MESK/R Algorithmen ist in Tab. 1 zu sehen. Ein interpretierbares ML-Modell besteht aus:

- **Merkmals-Extraktion (ME):** Die ME-Algorithmen reduzieren die Dimensionalität der Eingangsdaten, indem sie die Daten mit

Hilfe physikalisch interpretierbarer extrahierter Merkmale darstellen. Jeder ME-Algorithmus stellt die Eingabedaten in verschiedenen Merkmalen dar. Die Reduktion der Repräsentation und der Eingabedaten ist ein Kompromiss zwischen Genauigkeit und geringer Anzahl von Merkmalen.

- **Merkmals-Selektion (MS):** Im Gegensatz zur ME ist die MS ein überwachter, trainierbarer Teil des MESK/R -Workflows. Die MS versucht, die wichtigsten Merkmale auszuwählen, die durch die zuvor durchgeführte ME berechnet wurden. Nur die wesentlichen Merkmale mit dem größten Informationsgehalt werden selektiert, um eine Überanpassung zu vermeiden.

  Klassifikation/Regression (K/R): Das letzte Element der MESK/R -Pipeline ist ein Klassifizierungs- oder Regressionsalgorithmus, der die Ausgabe des MESK/R berechnet und ebenfalls ein überwachtes trainiertes Workflow-Element ist. Klassifikatoren versuchen, die gegebene Eingabe mit einer minimalen Fehlerquote auf die gewünschte diskrete Ausgabe abzubilden. Im Gegensatz dazu sagt eine Regression kontinuierliche Werte als Ausgabe voraus.

Ein weiterer Vorteil der MESK/R-Toolbox ist die adaptive Kombination von Algorithmen in Abhängigkeit von den Anwendungsfällen. Die große Auswahl an verschiedenen, sich gegenseitig ergänzenden MESK/R-Algorithmen ermöglicht die Erstellung geeigneter ML-Modelle für unterschiedliche Condition-Monitoring-Probleme.

Tab. 1: Als neuronales Netzwerk implementierte Methoden der ML-Toolbox

Merkmals-Extraktion	
ALA	Adaptive Linear Approximation **[16]**
BDW	Best Daubechies Wavelets **[17]**
PCA	Hauptkomponentenanalyse **[18]**
StatMom	Statistische Momente **[14]**
Merkmals-Selektion	
Pearson	Pearson Korrelationskoeffizient **[19]**
RELIEFF	RELIEFF **[20]**
RFESVM	Recursive Feature Elimination Support Vector Machines **[21]**
Spearman	Spearman Korrelationskoeffizient **[22]**
Klassifikation/Regression	
LDA-Mahal-Klassifikator	Lineare Diskriminanz Analyse mit Mahalanobis Distanz Klassifikation **[20, 21]**
PLSR	Partielle Least Square Regression **[25]**

b) DNN-Layer und Funktionen

Wie bereits erwähnt, basiert der Ansatz der Modell-zu-DNN-Konvertierung auf der Darstellung der grundlegenden mathematischen Operationen mit den jeweiligen DNN-Schichten. Die Operationen der wichtigsten DNN-Schichten werden beschrieben und mit der gewünschten mathematischen Operation verknüpft, um ein systematisches Verständnis der Konvertierungstechnik zu erreichen. Einige Schichten werden für mehrere Aufgaben verwendet, wodurch die Komplexität des resultierenden DNNs reduziert wird.

Fully Connected-Layer: Eine vollverknüpfte Schicht wendet eine lineare Projektion auf den Eingangsdatenvektor an. Parametrisierbar sind die Gewichtsmatrix und der Bias-Vektor. Die Schicht kann als eine Matrixmultiplikation mit den Gewichten und eine anschließende Vektoraddition mit dem Bias-Vektor betrachtet werden. Durch die Multiplikation mit der Gewichtsmatrix können Permutationen, die Berechnung des Mittelwerts nicht äquidistanter Segmente, die Division oder Multiplikation einer Konstanten und die Größenänderung des Eingangsvektors dargestellt werden. Transformationen wie die diskrete Fourier-Transformation, die lineare Diskriminanten-analyse (LDA) oder die Hauptkomponenten-analyse (PCA) können durch nur eine Matrixmultiplikation abgebildet werden. Außerdem ermöglicht das Hinzufügen des Bias-Vektors die Addition oder Subtraktion von konstanten Werten.

Pooling-Layer: Mit dem Pooling-Layer kann eine Unterabtastung der Eingangsdaten realisiert werden. Die Unterabtastung der Eingabedaten erfolgt durch die Schrittgröße der Poolingfunktion und dem darauf basierenden Merkmal des Poolingfilters (Durchschnitt oder Maximum). Der Pooling-Filter gleitet über die Eingabedaten und kombiniert die Werte auf der Grundlage des angegebenen Parameters. Innerhalb eines Algorithmus kann das Pooling die Funktionalität abbilden, Spitzen oder Minima (maximales Pooling) zu finden und den Mittelwert (durchschnittliches Pooling) von äquidistanten Segmenten mit der Länge der Filtergröße zu berechnen.

Convolutional-Layer: Der Convolutional-Layer wendet einen Gleitfilter mit einer definierten Schrittweite auf die Eingabedaten an. Neben der Filterfunktion kann die Schicht einen Bias-Wert hinzufügen. Relevante Verarbeitungsschritte, die durch den Convolutional-Layer ausgeführt werden, sind die Filterung mit endlicher Impulsantwort (FIR) und die Wavelet-Zerlegung eines Signals.

Combination-Layer: Der Combination-Layer stellt die grundlegenden mathematischen Operationen wie Multiplikation, Division und Addition von zwei Eingaben dar. Die Schichten führen die gewünschte mathematische Funktion elementweise aus. Zusätzlich können die Combination-Layer Funktionen wie die Berechnung der Quadratwurzel durchführen.
Concatenation-Layer: Die Verkettungsschicht kombiniert die Ergebnisse paralleler Berechnungen innerhalb des Netzes. Die Schicht verkettet im Allgemeinen mehrere Eingangsvektoren oder -matrizen in der gewünschten Dimension, um eine einzige Ausgabe zu erzeugen. Dadurch können Ergebnisse paralleler Berechnungen innerhalb eines NNs zusammengeführt werden.

IV. ERGEBNISSE

Der Ergebnisteil zeigt die Anwendung der entwickelten Konvertierungsmethode auf einen Teil der oben erwähnten Open-Source ML-Toolbox [9]. Der implementierte Teil der ML-Toolbox umfasst vier ME-, vier MS- und zwei K/R-Algorithmen. Die implementierten Algorithmen sind in Tab. 1 dargestellt. Jeder implementierte ME-, MS- und K/R-Algorithmus wird zunächst in ein separates DNN umgewandelt. Diese können dann aneinandergehängt werden. Wie in Abschnitt Ⅱ beschrieben, muss vor der Umwandlung des Algorithmus das entsprechende Modell trainiert werden. Die Graphen in Abb. 2 und Abb. 3 beschreiben die einzelnen Algorithmen und einen kompletten MESK/R-Stack, der auf den Daten des Hydrauliksystems trainiert wurde.

Resultierende DNN

In diesem Abschnitt werden die Funktionalitäten der Verarbeitungsebenen der einzelnen MESK/R-Modellinferenzen beschrieben. Die Schichten, die Größe und Form der Daten ändern, werden nicht berücksichtigt.

Adaptive Lineare Approximation: Im Trainingsprozess unterteilt ALA das Signal in mehrere annähernd lineare Segmente unterschiedlicher Länge. Mittelwerte und Steigungswerte der Segmente werden als Merkmale extrahiert. Die Mittelwertberechnung für alle Segmente wird durch eine Multiplikation mit einem Fully-Connected-Layer durchgeführt, um die nicht äquidistante Segmentierung darzustellen. Diese wird auf die Eingangsdaten und einen Indexvektor, der den Zeitvektor darstellt, angewendet, um die Steigung des Eingangssignals

Abb. 2: Ergebnisse der Konvertierung des ML-Algorithmus zu einem neuronalen Netz und die Darstellung der Layer-Graphen der einzelnen Netze

zu berechnen. Die Berechnung der Steigung eines Segments wird von zwei Additions-, einem Multiplikations- und einem Divisions-Layer sowie zwei Fully-Connected-Layer durchgeführt. Die berechneten Werte werden mit einem Concatenation-Layer kombiniert.

Best Daubechies Wavelet (BDW): Die Wavelet-Transformation ist eine Zeit-Frequenz-Transformation und ermöglicht die Analyse von Signalen im Frequenzbereich. Die Filterbank der Wavelet-Transformation wird auf ein DNN abgebildet. Die Anzahl der Filterstufen und die BDW-Hoch- und Tiefpassfilter-koeffizienten werden im Trainingsprozess berechnet. Die Anzahl der Filterstufen hängt von der Signallänge ab und bestimmt die Größe des Netzwerks. Ein Convolutional-Layer mit den Hochpassfilterkoeffizienten und ein Convolutional-Layer mit den Tiefpassfilter-koeffizienten filtern das Eingangssignal. Das tiefpassgefilterte Signal wird an die nächste Stufe Convolutional-Layer und der Hochpassfilter wird an den Ausgang weitergegeben. Dieser Vorgang wird so lange durchgeführt, bis die letzte Filterstufe erreicht ist. Der Ausgang jeder Filterstufe wird mit einer Verkettungsschicht kombiniert.

Statistische Momente (StatMom): Der Algorithmus unterteilt das Eingangssignal in äquidistante Segmente und berechnet die ersten vier statistischen Momente jedes Segments (Mittelwert, Varianz, Schiefe und Kurtosis). Im Vergleich zum ALA-Extraktor kann der Mittelwert des äquidistanten Segments durch ein durchschnittliche Pooling-Layer mit der Segmentlänge als Filtergröße berechnet werden. Die höheren statistischen Momente können durch Additions-, Quadratwurzel- (sqrt) und Multiplikationsschichten auf der Grundlage des berechneten Mittelwerts dargestellt werden. Die Ausgaben werden miteinander verknüpft.

Hauptkomponentenanalyse (PCA): Die Transformation eines trainierten PCA-Modells ist die Multiplikation einer Matrix mit dem Eingangsvektor. Eine Fully-Connected-Layer kann diese Matrixmultiplikation mit der PCA-Transformationsmatrix als Gewichtungsmatrix abbilden. Die PCA-Transformationsmatrix wird auf der Grundlage des Trainingsdatensatzes im berechnet.

Merkmals-Selektion (MS): Die Anwendung einer trainierten MS ist ein Ranking eines Eingangs- Merkmalsvektors. Nach dem Ranking, das von einer Fully-Connected Layer durchgeführt wird, werden die wichtigsten Merkmale ausgewählt. Für diesen Verarbeitungsschritt wird der Eingangsvektor mit einer dünnbesetzten Gewichtsmatrix multipliziert, in der die Indizes der relevantesten Merkmale auf eins gesetzt sind. Dieser Graph ist in Abb. 3 nicht dargestellt.

LDA-Mahalanobis-Klassifikator (LDAMahal): Dieses Modell kombiniert eine LDA-Transformation und eine Klassifizierung auf der Grundlage der Mahalanobis-Distanz zu den Klassenmittelpunkten. Die LDA-Transformation ist ähnlich wie die PCA eine Matrixmultiplikation, die durch ein Fully-Connected-Layer dargestellt wird. Nachdem das Signal auf die Diskriminanzfunktionen projiziert wurde, berechnet das DNN die Mahalanobis-Distanz zu jeder Klasse und ordnet es der Klasse mit kleinstem Abstand zu. Der Abstand wird mit einem Fully-Connected-Layer berechnet. Die Gewichte des Fully-Connected-Layer sind die berechnete Kovarianzmatrix für jede Klasse. Nachdem der Abstand zu jeder Klasse in parallelen Pfaden berechnet wurde, werden die Werte mit einer Concatenation-Layer zu einem Vektor zusammengefasst. Dieser Vektor wird durch ein Fully-Connected-Layer mit minus eins multipliziert. Die Maximum-Pooling-Layer ist nun in der Lage, den Minimalwert des modifizierten Distanzvektors zu finden. Die dabei verwendete Aktivierungsfunktion clipped Rectified Linear Unit (ReLU) ordnet den gefundenen Minimalwert der entsprechenden Klasse zu. Die Größe des Netzes hängt von der Anzahl der Klassen des Datensatzes ab.

Partial Least Square Regression (PLSR): Bei der PLSR wird zunächst ein Offset auf das Eingangssignals addiert. Anschließend werden die neuen Daten mit der Beta-Matrix der im Trainingsprozess berechneten Koeffizienten innerhalb eines Fully-Connected-Layer multipliziert.

MESK/R-Modell: Nach der Darstellung einiger einzelner ME-, MS- und K/R-Methoden wird in diesem Abschnitt ein vollständig interpretierbares ML-Modell als DNN-Konvertierung vorgestellt. Das Modell besteht aus StatMom, Pearson, und LDA-Mahal-Klassifikator. Das Modell wurde, mit dem in Abschnitt 2d beschriebenen Hydraulikdatensatz trainiert, und das Schichtdiagramm des resultierenden DNN ist in Abb.4 dargestellt. Die Umwandlung eines vollständigen MESK/R-Modells wird durch Verbindung des bestehenden DNN ohne die Eingabe- und Ausgabeschicht erreicht.

Das entstehende DNN wurden erfolgreich als TensorFlow-Lite-Modell auf dem energieeffizienten und optimierten QSXP- ML81 Board mit integrierter Neural Processing Unit ausgeführt.

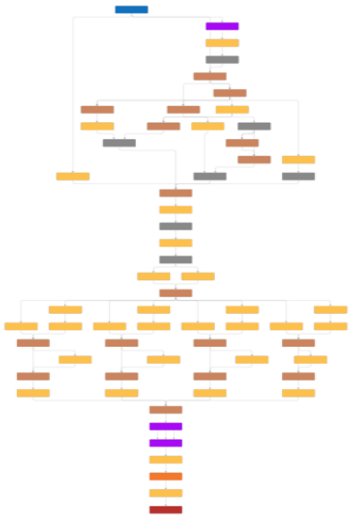

Abb. 3: Layer-Graph des MESK/R-Modells (Statistische Momente, Pearson und LDA-Mahal-Klassifikation)

Die Inferenz des vollständigen MESK/R DNN für den Ventilzustand, basierend auf den von einem Drucksensor gesammelten Daten, erzielte gute Ergebnisse, wie in Abb. 4 dargestellt, mit einem normalisierten 10-fachen Kreuzvalidierungs-Error von 0,1%, welches identisch mit den Ergebnissen der zugrundeliegenden Toolbox ist. Außerdem wurden die konvertierten DNNs mit der Python-Implementierung der ML-Toolbox verglichen, die auf der Edge-Hardware verwendet werden kann. Die Laufzeit der DNNs auf der CPU hat sich im Vergleich zur Python-Implementierung des MESK/R-Modells deutlich verbessert. Zusätzlich reduziert die Verwendung der NPU die Laufzeit des DNNs um weitere 7%, wie in Tab. 2 beschrieben.

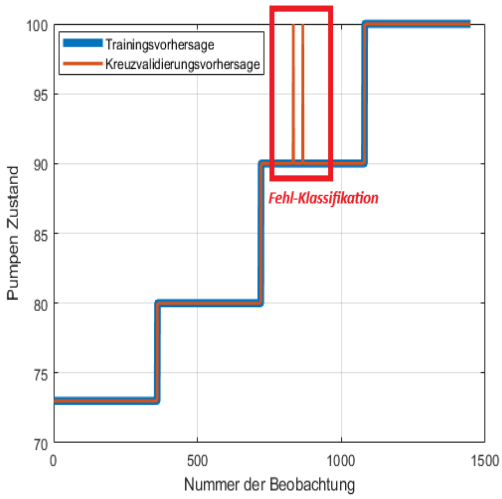

Abb. 4: Inferenz-Ergebnisse des MESK/R (Statistische Momente, Pearson und LDA-Mahal-Klassifikation) Modells auf die Hydraulik-Daten

Tab. 2: Inferenzlaufzeiten auf Edge-Hardware QSXP-ML81

Implementierung/ Unit	Inferenz-Zeit für 1000 Zeitreihen [ms]
ML-Toolbox in Python/CPU	1587.100
DNN/CPU	0.828
DNN/NPU	0.769

V. DISKUSSION

Die Modell-zu-DNN-Konvertierungsmethode konvertiert erfolgreich interpretierbare ML-Algorithmen in DNNs. Die Konvertierung ermöglicht die Nutzung von Hardware-Beschleunigern und optimierter Hardware, um die Laufzeit- und Energieeffizienz für Inferenzen von interpretierbaren ML-Algorithmen mit minimalem Programmieraufwand zu verbessern. Weitere Algorithmen können identisch zu den bereits implementierten Algorithmen, auf Basis der vorgestellten Methode in ein DNN umgewandelt werden. Die i.d.R. nötige Quantisierung von DNN wurde in dieser Arbeit vorerst nicht berücksichtigt. Auf Hochleistungsrechnern werden hochpräzise Datentypen unterstützt. Um jedoch das DNN auf Hardware mit begrenzter Leistung und i.d.R. reduzierter Rechengenauigkeit auszuführen, muss das DNN quantisiert werden. Dies führt zu einem Kompromiss zwischen Genauigkeit und Effizienz und stellt ein weiterführendes Forschungsthema dar. Außerdem führt die Umwandlung komplexer MESK/R-Algorithmen zu komplexen DNN-Strukturen. In der weiteren Forschung könnte dies zu einer potenziellen Herausforderung bei der Implementierung dieser DNNs, bezüglich des Speicherbedarfs und der numerischen Genauigkeit des DNNs, auf begrenzter Edge-Hardware führen.

VI. SCHLUSSFOLGERUNG UND ZUKÜNFTIGE ARBEIT

In diesem Beitrag wird eine effiziente Methode zur Konvertierung eines trainierten interpretierbaren ML-Modells in ein Deep Neural Network (DNN) vorgestellt, die es ermöglicht, Inferenzen auf optimierter Edge-Hardware auszuführen. Die Umwandlung führt zur Nutzbarkeit von Hardwarebeschleunigern für interpretierbare ML-Algorithmen. Basierend auf der Open-Source ML-Toolbox wurde eine Methode zur automatisierten Auswahl eines MESK/R-Algorithmus, die Umwandlung des trainierten Modells in ein DNN und die Inferenz auf der optimierten Hardware vorgestellt. Das Benchmarking auf verschiedener optimierter Hardware, die Untersuchung von Laufzeit und Energieeffizienz, die Untersuchung des Speicherbedarfs und der Einfluss der Quantisierung stellen weitere Forschungs-themen dar.

Danksagung

Diese Arbeit wurde teilweise vom Bundesministerium für Bildung und Forschung (BMBF) im Rahmen des Projekts "Edge Power" unter dem Code 16ME0574 gefördert.

Literatur

[1] R. K. Mobley, "*An introduction to predictive maintenance"*. Elsevier, 2002.

[2] Bkn. Rao, "*Handbook of condition monitoring"*. Elsevier, 1996.

[3] J. Gebhardt, “Sensor use cases in the context of industry 4.0,” *IEEE 2020 2nd International Conference on Societal Automation (SA)*, no. 2021, p. S. 1-5.

[4] N. V. Kirianaki, S. Y. Yurish, N. O. Shpak, and V. P. Deynega, "*Data acquisition and signal processing for smart sensors"*. Wiley New York, 2002.

[5] B. Varghese *et al.*, “A survey on edge performance benchmarking,” *ACM Computing Surveys (CSUR)*, vol. 54, no. 3, pp. 1–33, 2021.

[6] J. Jo, S. Jeong, and P. Kang, “Benchmarking gpu-accelerated edge devices,” in *2020 IEEE international conference on big data and smart computing (BigComp)*, IEEE, 2020, pp. 117–120.

[7] P. Goodarzi, A. Schütze, and T. Schneider, “Prediction quality, domain adaptation and robustness of machine learning methods: a comparison,” in *Sensors and Measuring Systems; 21th ITG/GMA-Symposium*, VDE, 2022, pp. 1–2.

[8] T. Schneider, N. Helwig, and A. Schütze, “Automatic feature extraction and selection for classification of cyclical time series data,” *tm-Technisches Messen*, vol. 84, no. 3, pp. 198–206, 2017.

[9] ZeMA gGmbh, “LMT-ML-Toolbox,” *GitHub repository*. GitHub, 2017. [Online]. Available: https://github.com/ZeMA-gGmbH/LMT-ML-Toolbox

[10] J. V. Tu, “Advantages and disadvantages of using artificial neural networks versus logistic regression for predicting medical outcomes,” *Journal of clinical epidemiology*, vol. 49, no. 11, pp. 1225–1231, 1996.

[11] V. Buhrmester, D. Münch, and M. Arens, “Analysis of explainers of black box deep neural networks for computer vision: A survey,” *Machine Learning and Knowledge Extraction*, vol. 3, no. 4, pp. 966–989, 2021.

[12] Z. Pan and P. Mishra, “Hardware acceleration of explainable machine learning,” in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2022, pp. 1127–1130.

[13] K. Neshatpour, M. Malik, M. A. Ghodrat, A. Sasan, and H. Homayoun, “Energy-efficient acceleration of big data analytics applications using FPGAs,” in *2015 IEEE International Conference on Big Data (Big Data)*, IEEE, 2015, pp. 115–123.

[14] T. Schneider, N. Helwig, and A. Schütze, “Industrial condition monitoring with smart sensors using automated feature extraction and selection,” *Measurement Science and Technology*, vol. 29, no. 9, p. 094002, 2018.

[15] T. Schneider, S. Klein, and M. Bastuck, “Condition monitoring of hydraulic systems Data Set at ZeMA,” *Zenodo*, 2018, doi: 10.5281/zenodo.1323610.

[16] R. T. Olszewski, "*Generalized feature extraction for structural pattern recognition in time-series data"*. Carnegie Mellon University, 2001.

[17] A. C. Rowe and P. C. Abbott, “Daubechies wavelets and mathematica,” *Computers in Physics*, vol. 9, no. 6, pp. 635–648, 1995.

[18] S. Wold, K. Esbensen, and P. Geladi, “Principal component analysis,” *Chemometrics and intelligent laboratory systems*, vol. 2, no. 1–3, pp. 37–52, 1987.

[19] I. Cohen *et al.*, “Pearson correlation coefficient,” *Noise reduction in speech processing*, pp. 1–4, 2009.

[20] I. Kononenko, E. Šimec, and M. Robnik-Šikonja, “Overcoming the myopia of inductive learning algorithms with RELIEFF,” *Applied Intelligence*, vol. 7, pp. 39–55, 1997.

[21] M. Yong, P. Daoying, L. Yuming, and S. Youxian, “Accelerated recursive feature elimination based on support vector machine for key variable identification,” *Chinese journal of chemical engineering*, vol. 14, no. 1, pp. 65–72, 2006.

[22] C. Wissler, “The Spearman correlation formula,” *Science*, vol. 22, no. 558, pp. 309–311, 1905.

[23] R. H. Riffenburgh, “Linear discriminant analysis,” PhD Thesis, Virginia Polytechnic Institute, 1957.

[24] G. J. McLachlan, “Mahalanobis distance,” *Resonance*, vol. 4, no. 6, pp. 20–26, 1999.

[25] P. Geladi and B. R. Kowalski, “Partial least-squares regression: a tutorial,” *Analytica chimica acta*, vol. 185, pp. 1–17, 1986.