18. AALE-Konferenz
Pforzheim, 09.03.-11.03.2022

Untersuchungen zur echtzeitfähigen Bilderkennung mit neuronalen Netzen auf konventionellen Industriesteuerungen

C. Wree[1], R. Raßmann[2], J. Daâs[3], F. Bause[4], T. Schönfeld[5]

Zusammenfassung

Fertigungssysteme für die individualisierte Produktion erfordern Arbeitsabläufe, die von einzelnen Objekten abhängig sind. Maschinelles Lernen (ML) bietet die Möglichkeit, verschiedene Objekte mit Hilfe eines neuronalen Netzes zu klassifizieren. Abhängig von den Klassifikationsergebnissen können Entscheidungen für den nachfolgenden Produktionsschritt getroffen werden. Es wird untersucht, ob es möglich ist, ein neuronales Netz zur Bilderkennung in Echtzeit und in Koordination mit den Maschinen- und Bewegungssteuerungsaufgaben auszuführen. In dieser Arbeit wird die Umsetzung und Messung mit Hilfe einer SPS-Laufzeitumgebung auf einem Standard-Industrie-PC durchgeführt. Die Ausführungszeiten verschiedener Methoden zur Implementierung neuronaler Netze werden gemessen und verglichen. Das schnellste neuronale Netz benötigt eine durchschnittliche Ausführungszeit von nur 39 µs. Darüber hinaus werden die Eigenschaften der verschiedenen Methoden in Bezug auf das Training und die Implementierung der neuronalen Netze innerhalb verschiedener industrieller Steuerungen diskutiert.

Stichwörter

Maschinelles Lernen, Intelligente Fertigungssysteme, Individualisierte Produktion

1 Einleitung

Individualisierte Produktionssysteme werden vorgeschlagen, um Produkte für einen einzigen Kunden zu niedrigen Kosten herzustellen [1]. Für jedes dieser Produkte sind andere Fertigungsvorgänge erforderlich. Die Bilderfassung, -verarbeitung und -analyse (Machine Vision) ist eine wesentliche Komponente zur Unterstützung dieser produktabhängigen Vorgänge.
Herkömmliche Bildverarbeitungssysteme beruhen auf der Merkmalsextraktion [2]. Sobald die Produkte jedoch in Form, Farbe oder Oberflächenbeschaffenheit variieren, werden solche Bildverarbeitungslösungen komplex und sind nur mit großem Aufwand und Expertenwissen zu realisieren. In diesen Fällen kann Maschinelles Lernen (ML) sowohl den Programmieraufwand reduzieren als auch die Effizienz herkömmlicher Bildverarbeitungslösungen verbessern [3].
ML bietet die Möglichkeit, verschiedene Objekte mit Hilfe eines neuronalen Netzes zu klassifizieren. Abhängig von den Klassifizierungsergebnissen können Entscheidungen für den weiteren Betrieb getroffen werden. So ist es beispielsweise wünschenswert, dass ein Klassifizierungsergebnis in kürzester

1 Fachhochschule Kiel, Fachbereich Informatik und Elektrotechnik, Grenzstr. 5, 24149 Kiel
2 Fachhochschule Kiel, Fachbereich Informatik und Elektrotechnik, Grenzstr. 5, 24149 Kiel
3 Fachhochschule Kiel, Fachbereich Informatik und Elektrotechnik, Grenzstr. 5, 24149 Kiel
4 Beckhoff Automation GmbH & Co. KG, Hülshorstweg 20, 33415 Verl
5 Beckhoff Automation GmbH & Co. KG, Hülshorstweg 20, 33415 Verl

Zeit vorliegt, um den nächsten Sollwert eines Bewegungssteuerungsbefehls zu bestimmen. In diesem Echtzeitkontext sind Modelle mit deterministischer Laufzeitkomplexität zu bevorzugen. Dazu gehört das Multilayer Perceptron (MLP) [3].
Bildverarbeitungssysteme für die Produktion werden entweder auf einem externen Computer [2], auf einer intelligenten Kamera [2] oder innerhalb einer SPS-Laufzeitumgebung, die in einem Industrie-PC [4] beheimatet ist, implementiert. Eine SPS-Laufzeitumgebung bietet den Vorteil, dass Maschinensteuerung, Bewegungssteuerung und Bildverarbeitung zentral auf einem Industrie-PC ausgeführt werden können. Dies ermöglicht eine effiziente zeitliche Synchronisation zwischen diesen verschiedenen Aufgaben. Außerdem werden Latenzzeiten und zusätzliche Schnittstellen vermieden.
In diesem Beitrag wird gezeigt, dass es auch möglich ist, ein neuronales Netz zur Bilderkennung zusammen mit den Aufgaben der Maschinen- und Bewegungssteuerung auf der gleichen SPS-Laufzeitumgebung auszuführen.
Der vorliegende Beitrag ist wie folgt aufgebaut. In Kapitel 2 wird erläutert, welche Schritte notwendig sind, um ML in industriellen Steuerungssystemen einzusetzen. Das MLP zur Bildklassifikation wird in Kapitel 3 vorgestellt. In diesem Beitrag werden Convolutional Neuronal Networks (CNN) nicht betrachtet, da sie einen wesentlich höheren Rechenaufwand erfordern und daher auch den Echtzeitanforderungen nicht gerecht werden. In Kapitel 4 werden zwei Anwendungsbeispiele für die individualisierte Produktionssysteme gezeigt. Der Einsatz und die Umsetzung von MLPs für die Anwendungsbeispiele wird in Kapitel 5 erläutert. Die Ausführungszeiten von neuronalen Netzen, die unterschiedlich in die SPS-Laufzeitumgebung integriert sind, werden in den Kapiteln 6 und 7 vorgestellt und verglichen. Darüber hinaus werden die Eigenschaften der verschiedenen Methoden in Bezug auf das Training und die Implementierung der neuronalen Netze erörtert.

2 Allgemeiner Prozess zum Einsatz von Maschinellem Lernen

Im Wesentlichen besteht der Prozess des Maschinellen Lernens und der Integration in ein industrielles Steuerungssystem aus vier Phasen.

- Sammeln von Daten
- Datenaufbereitung
- Training von Modellen
- Einsatz des Modells in der industriellen Steuerung

2.1 Sammeln von Daten

Je nach Anwendung können verschiedene Werkzeuge zur Datenspeicherung verwendet werden. Die Daten können in Dateien oder in einer Datenbank gespeichert werden. Die Datenbank kann sich lokal im Netz oder in der Cloud befinden. Wenn der Rechner mit einer öffentlichen oder privaten Cloud verbunden ist, können die Daten über einen Broker in einem Datenspeicher oder -lager gespeichert werden.
In manchen Fällen sind bereits Trainingsdatensätze verfügbar, die als Ausgangspunkt verwendet werden können. Im Laufe des Projekts können immer mehr Daten innerhalb der jeweiligen Anwendung gesammelt werden und dem ursprünglichen Datensatz hinzugefügt werden bzw. schließlich ganz ersetzen.

2.2 Datenaufbereitung

Um die Daten auf das zu lösende Problem anzuwenden, ist häufig eine Vorverarbeitung erforderlich. Dazu gehört das Screening der Trainingsdaten auf Ausreißer, die das Training erschweren können.

Außerdem werden die Daten normalisiert, so dass sie innerhalb eines bestimmten Bereichs von Eingangswerten liegen. Bei der Verwendung von Bildern muss das Bild in manchen Fällen segmentiert und skaliert werden. Mit Hilfe der sogenannten „Data Augmentation“ können die Bilddaten aus dem vorhandenen Datensatz verändert und zusätzlich zu den Trainingsdaten hinzugefügt werden. Der Datensatz wird hierdurch künstlich erweitert [3]. Dies bietet z. B. die Möglichkeit Fremdeinflüsse wie z. B. eine variierende Beleuchtung oder einen verrauschten Hintergrund schon im Training des Modelles mit zu berücksichtigen.

2.3 Training von Modellen

Wenn die Trainingsdaten in Dateien oder Datenbanken vorliegen, müssen sie mit einem ML-Framework gelesen und verarbeitet werden. Eine Vielzahl von ML-Frameworks basieren auf Python™ oder R und sind quelloffen, wie z. B. Keras [5]. Andere sind kommerziell verfügbare Frameworks wie MATLAB [6]. Eine Vielzahl von Standardfunktionen und Toolboxen/Bibliotheken dienen als Schnittstelle für den Zugriff auf die gespeicherten Daten.
Die Arbeit mit dem ML-Framework umfasst die Datenvorverarbeitung und -auswahl, die Modellierung und das Training eines geeigneten ML-Algorithmus und schließlich die Evaluierung des gelernten Algorithmus. Das Ergebnis dieses Entwicklungsschrittes ist ein trainiertes ML-Modell.
Das gelernte Modell muss aus dem ML-Framework exportiert werden, um es in der industriellen Steuerung auszuführen. Viele der ML-Frameworks unterstützen das ONNX-Beschreibungsformat [7]. Aus proprietären Gründen ist es häufig wünschenswert die neuronalen Netze in ein binäres Format zu konvertieren.

2.4 Einsatz des Modells in der industriellen Steuerung

Es gibt viele Möglichkeiten, ein trainiertes neuronales Netz in die echtzeitfähige Laufzeit einer industriellen Steuerung zu integrieren. Ein wesentlicher Schritt ist die Umwandlung des Netzes in maschinenlesbaren Code mit Hilfe eines Compilers. Darüber hinaus spielt die Rechnerarchitektur der industriellen Steuerung eine wichtige Rolle. Hierzu gibt es mehrere Studien und Veröffentlichungen [8]. Nach erfolgreicher Integration werden die ML-Modelle zyklisch im Kontext der Echtzeitumgebung der industriellen Steuerung ausgeführt. Somit stehen alle in der Steuerung vorhandenen Daten in Echtzeit zur Verfügung und die Ergebnisse des ML-Modells können in Echtzeit ausgegeben und weiterverarbeitet werden.
Oft ist es notwendig, trainierte Modelle ohne Neukompilierung und ohne Neustart der Laufzeit zu ersetzen. Dazu müssen sich alle Modellbeschreibungsdateien auf dem Zielsystem befinden und können während der Ausführung der Laufzeit geladen werden.

3 Multilayer Perceptron für die Bildklassifikation

Im Bereich des Maschinellen Lernens gibt es eine Fülle von unterschiedlichen Modellen. Im Echtzeitkontext der Automatisierungstechnik sind Modelle mit deterministischer Laufzeitkomplexität geeignet und zu bevorzugen. Dazu gehört das Deep Multilayer Perceptron [2]. Es wird oft auch als "fully-connected" oder "dense neural network" bezeichnet. Dieses Netz hat wie in Bild 1 dargestellt eine Eingabeschicht, eine oder mehrere versteckte Schichten (hidden layer) und eine Ausgabeschicht. Jede Schicht umfasst eine bestimmte Anzahl von Neuronen, die wiederum aus Gewichten und einer Aktivierungsfunktion bestehen. Vollständig verbunden bedeutet, dass alle Neuronen in einer Schicht mit den Neuronen der nächsten Schicht verbunden sind. Deep MLPs haben mehrere versteckte Schichten. Bei reproduzierbaren Bedingungen können MLPs zur Bildklassifikation verwendet werden. In diesem Falle sind die rechenintensiveren Convolutional Neuronal Networks nicht notwendig [2].

Ein mögliches Problem, welches während des Trainings eines neuronalen Netzes auftreten kann, ist dass das Netzwerk einer Überanpassung unterliegt („Overfitting“). Dies bedeutet, dass die zu erlernenden Parameter im Modell wenig generalisiert werden. Stattdessen spezialisiert sich das Netzwerk auf die Trainingsdaten. Overfitting macht sich unteranderem dadurch bemerkbar, dass die Fehlerrate im Training bei den Trainingsdaten immer kleiner wird, gleichzeitig steigt die Fehlerrate jedoch bei den unbekannten Testdaten an.
Durch das Hinzufügen eines „Dropouts“ kann einem „Overfitting“ beim Trainieren eines MLPs entgegengewirkt werden [3]. Der Dropout ist nur während des Trainings aktiv und ist im späteren trainierten Netzwerk nicht mehr vorhanden. Mit Hilfe der Dropout-Funktion können in Abhängigkeit der Dropoutrate eine bestimmte Anzahl von Neuronen zufällig innerhalb einer Sicht deaktiviert werden. Im Bild 1 ist der schematische Aufbau eines MLPs mit einem Dropout von 50% zwischen den beiden versteckten Schichten abgebildet. Während des Trainings werden jedes Mal zufällig 50% der Neuronen deaktiviert und geben somit keine Signale an die zweite versteckte Schicht weiter.

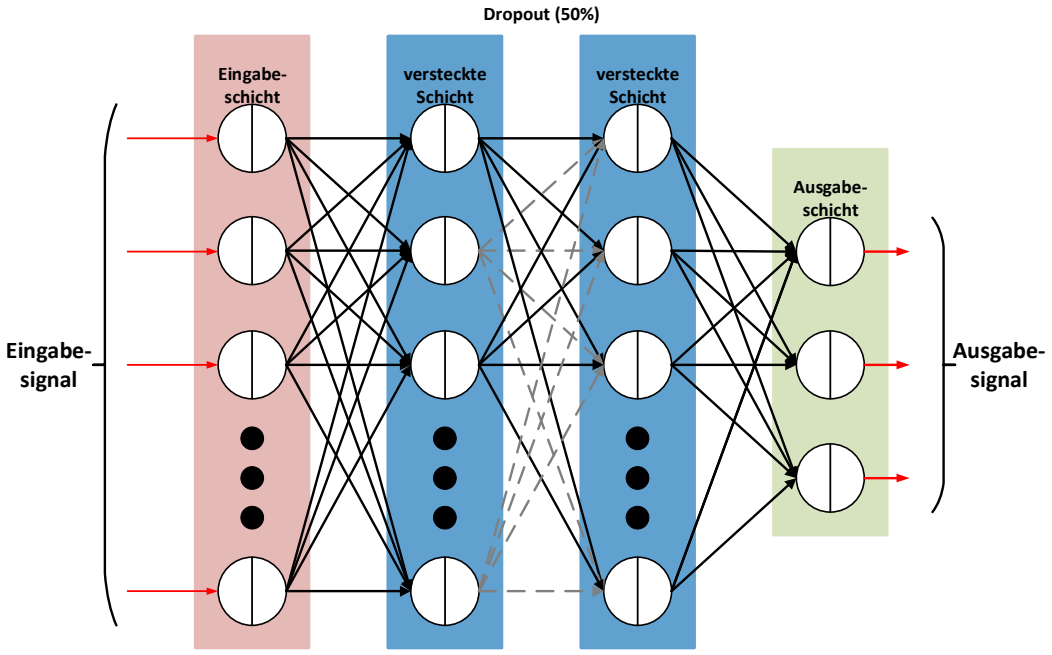

Bild 1: Multilayer Perceptron mit optionalen Dropout-Layer zwischen der ersten und zweiten versteckten Schicht

4 Anwendungsbeispiel in der Echtzeit: Das Produktionssystem

Ausgangslage für diese Untersuchung ist ein flexibles Produktionssystem. Es besteht aus einem Eingangs- und einem Ausgangslager für Werkstücke (Tags). Die Tags werden über einen Deltaroboter aus dem Eingangslager entnommen und können mit einem Transportsystem zu zwei Bearbeitungsmaschinen oder einem Kamerasystem transportiert werden (Bild 2). Das Kamerasystem verfügt über eine LED-Ringleuchte, um den Fremdlichteinfluss zu reduzieren. Die Maschinensteuerung, die Steuerung von Roboter und Transportsystem sowie die Bildverarbeitung werden auf unterschiedlichen Tasks aber in der gleichen Laufzeitumgebung ausgeführt. Die Zykluszeit der Task für den Deltaroboter bzw. für das Transportsystem beträgt dabei nur 250 µs.
Die Zielsetzung für das Produktionssystem ist die Entnahme eines individuellen Werkstücks aus dem Eingangslager, der Transport zum Kamerasystem, die Bildklassifikation mit Hilfe eines MLP und die Ablage des Werkstücks in unterschiedlichen Bereichen des Ausgangslager – je nach Klassifikationsergebnis. Dabei soll die Ausführung des MLPs innerhalb der Laufzeitumgebung so schnell wie möglich

durchgeführt werden, damit das Klassifikationsergebnis vor dem nächsten Sollwert für die Bewegungssteuerung des Deltaroboters bzw. des Transportsystems vorliegt.
Das Anwendungsbeispiel wird für die Erkennung handgeschriebener Ziffern zwischen 0 und 9 getestet. Die zu klassifizierenden Ziffern werden von einer beliebigen Person auf die verschiedenen Werkstücke (Tags) geschrieben und unsortiert in ein Eingangslager gelegt. Die Tags werden zum Kamerasystem transportiert. Das jeweilige Bild wird vom MLP eingelesen und das Klassifikationsergebnis wird innerhalb der echtzeitfähigen SPS-Laufzeitumgebung berechnet. Abhängig von diesem Ergebnis werden die Ziffern 0 bis 4 in den oberen Bereich des Ausgangslagers und die Ziffern 5 bis 9 in den unteren Bereich des Lagers ausgegeben. Ein Video zu diesem Demonstrator ist in [9] zu sehen.
Das Anwendungsbeispiel „Ziffernklassifikation“ eignet sich, um das Verhalten bei Produkten mit einer hohen Variation innerhalb der Produktklasse zu untersuchen. Eine herkömmliche Bildverarbeitung ohne Verwendung des Maschinellen Lernens kann diese 10 Produktklassen nicht unterscheiden.
In einem weiteren Anwendungsbeispiel werden vier verschiedene Formen (Kreis, Dreieck, Quadrat, Stern) klassifiziert [10]. Sie werden entweder direkt auf die Werkstücke (Tags) gezeichnet oder auf schwarze Selbstklebefolie gezeichnet, ausgeschnitten und auf die Tags geklebt. In diesem Anwendungsfall kann das Objekt während des Transports seine Rotation ändern. Ein Video dieses Anwendungsbeispiels steht zur Verfügung [11].

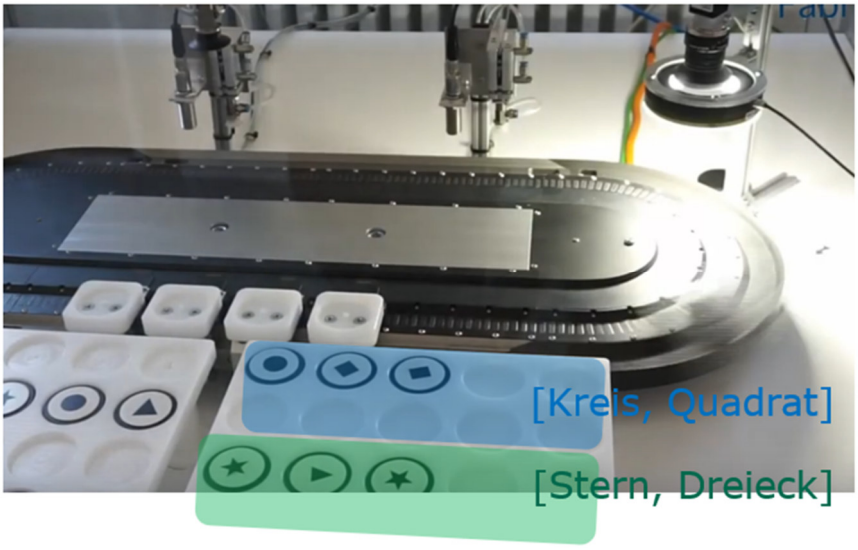

Bild 2: Anwendungsbeispiel zum Einsatz Maschinellen Lernens für individualisierte Produktion

5 Umsetzung

5.1 Sammeln von Daten

Das Training des neuronalen Netzes für das erste Anwendungsbeispiel wird mit dem MNIST-Datensatz [12] und für das zweite Beispiel mit dem four-shapes-Datensatz [13] durchgeführt. Der vorhandene Datensatz kann im nächsten Schritt mit Daten aus dem Produktionssystem erweitert werden, um erneut ein Training durchzuführen, das auf bessere Ergebnisse führt. Sobald genügend eigene Bilder aufgenommen sind, kann das Training komplett auf Daten aus dem Produktionssystem basieren.

5.2 Datenaufbereitung

Die 8-bit Graustufenbilder, die im Produktionssystem aufgenommen werden, müssen vorverarbeitet werden. Die Schritte bestehen aus Bildsegmentierung, optionaler Filterung des verrauschten Hintergrunds, Skalierung und Umwandlung in einen Vektor. Diese Funktionen, einschließlich der Erfassung des Bildes, müssen von einer Bildverarbeitungsbibliothek bereitgestellt werden.
Zunächst ist es wichtig, das aufgenommene Bild auf den relevanten Bildausschnitt zu reduzieren (Festlegung der Region of Interest). Dieses Segment ist in Bild 3 rot hervorgehoben.

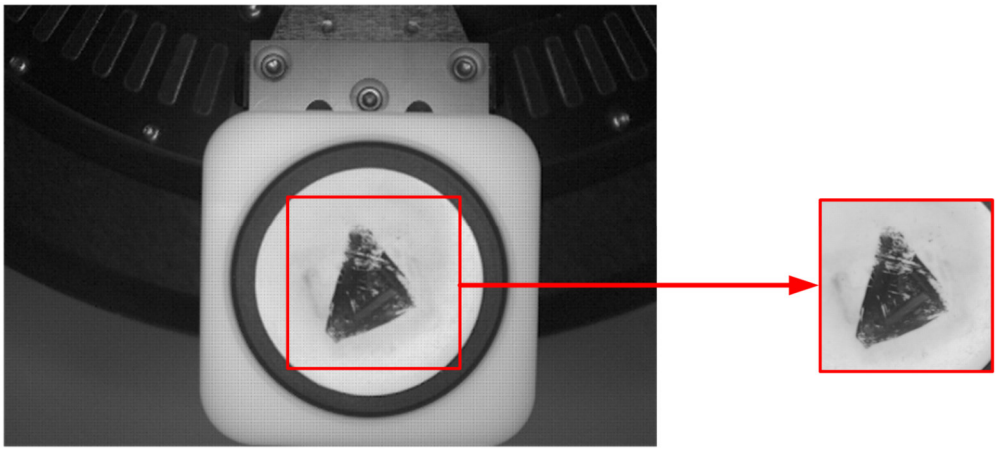

Bild 3: Festlegung der Region of Interest

Eine zweidimensionale Gaußsche Filterung mit integrierter linearer Histogrammspreizung kann verwendet werden, um die Ziffer bzw. die Form vom Rauschen zu trennen. Diese Filterung ist optional und wird nur für das erste Anwendungsbeispiel „Ziffernklassifikation“ durchgeführt.
In beiden Anwendungsbeispielen werden für das Training Bilder mit einer Datengröße von 28x28 Pixeln verwendet. Die Bilder vom original four-shapes-Datensatz besitzen eine Datengröße von 200x200 Pixeln und werden zur Vergleichbarkeit der beiden Anwendungsbeispiele auf eine einheitliche Datengröße von 28x28 Pixel verkleinert. Es ist jedoch auch möglich, eine höhere Auflösung zu verwenden, die insbesondere im zweiten Anwendungsbeispiel zu besseren Klassifikationsergebnissen führen kann.
Die Größe des Bildsegments muss dieselbe Datengröße wie die Trainingsdaten besitzen. Zu diesem Zweck wird die bilineare Filterung gewählt, die das Bildsegment durch eine lineare Interpolation auf die entsprechende Pixelanzahl verkleinert, wie in Bild 4 gezeigt.

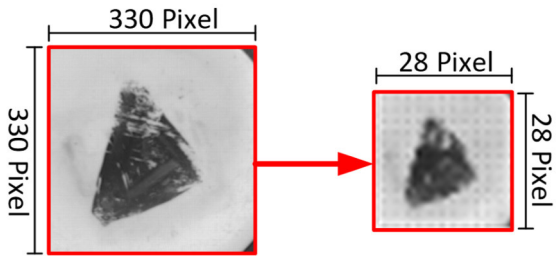

Bild 4: Skalierung des Bildsegmentes (ohne zweidimensionale Filterung)

Das MLP erwartet einen Eingangsvektor. Daher muss das Bild in einen Vektor der Länge 784 umgewandelt werden. Der Eingangs- und Ausgangsvektor des MLPs ist vom Datentyp LREAL oder REAL.

Zusätzlich zur Umwandlung der Bilder in Vektoren werden die Werte der Pixel auf einen Bereich zwischen 0 und 1 normalisiert, indem alle Werte durch 255 dividiert werden.

5.3 Trainieren des Modells

Das Training wird mit Keras bzw. MATLABs "Neural Networks Pattern Recognition" durchgeführt.
Im Falle von Keras kann das Training in Google Colab stattfinden. Die bereitgestellte Rechenleistung ist sehr groß, und das Training kann auf einem Grafikprozessor erfolgen. Dies hat den Vorteil, dass die benötigte Trainingszeit im Vergleich zum Training auf einem Standard-PC sehr kurz ist. Außerdem können sehr viele Daten gleichzeitig in das Netz geladen werden.
Bei Verwendung von MATLAB kann das Training über die Cloud-Lösung "MATLAB Online" erfolgen. Offline-Trainingsoptionen sind für Keras und MATLAB ebenfalls verfügbar.
Das Training für die Ziffernklassifikation in Keras bzw. MATLAB basiert auf dem gleichen Modell. Es besteht aus einer Eingabeschicht mit 784 Neuronen und einer Ausgabeschicht mit 10 Neuronen. Zwischen der Eingabe- und der Ausgabeschicht werden zwei versteckte Schichten mit je 90 Neuronen verwendet. Für die versteckten Schichten wird die Aktivierungsfunktion "sigmoid" und für die Ausgabeschicht die Aktivierungsfunktion "softmax" verwendet.
Die Netzwerkarchitektur für die Klassifikation der Formen besitzt einen ähnlichen Aufbau. Die Eingabeschicht besteht aus 784 Neuronen. Anschließend folgen zwei versteckte Schichten mit 512 und 256 Neuronen. Bei beiden versteckten Schichten wird die Aktivierungsfunktion „sigmoid“ verwendet. Die beiden versteckten Schichten besitzen außerdem eine Dropoutrate von 70%. Die Ausgabeschicht besteht aus vier Neuronen mit der Aktivierungsfunktion „softmax“.
Bei Keras wird ADAM als Optimierer gewählt, bei MATLAB ist es der Scaled Conjugate Gradient Algorithmus. In beiden Fällen wird die kategoriale Kreuzentropie als Verlustfunktion verwendet.
Durch die Verwendung von Dropout-Layern können die Ergebnisse für die Klassifikation der Formen im zweiten Anwendungsbeispiel deutlich verbessert werden. Dropout-Layer werden aktuell von MATLABs "Neural Networks Pattern Recognition" nicht unterstützt. Somit wird das Training für das zweite Anwendungsbeispiel in Keras durchgeführt.
Der Original-Datensatz [13] für das Anwendungsbeispiel zur Klassifikation der Formen wird erweitert, in dem der Datensatz erneut geladen und jedes Bild mit Rauschen überlagert wird. Der Datensatz besteht damit aus insgesamt 26308 Bildern. Für das Trainieren über 150 Epochen mit Keras benötigt Google-Colab bei einer „Batch Size“ von 512 insgesamt 82,1 Sekunden.

5.4 Einsatz des Modells in der industriellen Steuerung

Der wesentliche Schritt für den Einsatz des neuronalen Netzes ist seine Umwandlung in maschinenlesbaren Code mit Hilfe eines Compilers. Das Produktionssystem läuft auf einem Industrie-PC. Es beherbergt die Steuerung der Maschine selbst, die Steuerung des Transportsystems, die Steuerung des Deltaroboters, die Bildverarbeitung und das neuronale Netz. Im Folgenden werden drei Optionen für die Implementierung eines trainierten neuronalen Netzes in eine SPS-Laufzeitumgebung vorgestellt.

- Option 1: Nach dem Training eines MLPs mit MATLAB wird das neuronale Netz in einen Simulink-Block mit einem Eingangsvektor und einem Ausgangsvektor exportiert. In einem nächsten Schritt kann der PLC-Coder von Simulink einen Funktionsbaustein für Codesys, B&R Automation Studio, Siemens TIA Portal, Rexroth Indraworks, Rockwell Studio oder Beckhoff TwinCAT 3 generieren. Im Fall des Produktionssystems wird ein Funktionsbaustein für TwinCAT 3 generiert.
- Option 2: MATLAB wird für das Training des MLPs verwendet. Das neuronale Netz wird in einen Simulink-Block exportiert, wie in Option 1 geschehen. Im nächsten Schritt wird das TwinCAT-Target für Simulink verwendet, um das neuronale Netz in C++-Code zu konvertieren und eine ausführbare TwinCAT-Laufzeitdatei zu erstellen, die in die TwinCAT-Lösung integriert werden kann.

– Option 3: Das Training des MLPs wird mit Keras durchgeführt und als ONNX-Datei exportiert. Mit dem ML Model Manager der TwinCAT-Bibliothek TF3810 [14] wird sie in eine Binärdatei umgewandelt. Diese Datei wird geladen und in der TwinCAT-Laufzeit mit Hilfe von Funktionsbausteinen der TF3810-Bibliothek ausgeführt.

Die Produktionsanlage wird mit einer TwinCAT-Laufzeit betrieben, die auf einem Industrie-PC C6920 von Beckhoff mit einem Intel®-Celeron®-T3100-1,9-GHz-2-Core-Prozessor gehostet wird. Die verwendete Laufzeitumgebung ist TwinCAT 3.1 Build 4024[6].

6 Ergebnisse

Die drei Optionen werden in der oben beschriebenen SPS-Laufzeitumgebung verglichen. Insgesamt wird jedes Netzwerk jeder Option einhundertmal ausgeführt. Gemessen wird die Zeitspanne von der Bereitstellung der Eingaben bis zum Empfang der Ausgaben des neuronalen Netzes.
Die durchschnittliche / maximale / minimale Ausführungszeit für das neuronale Netz der Ziffernklassifikation von Option 1 beträgt 217 µs, 221 µs und 214 µs. Die durchschnittliche / maximale / minimale Ausführungszeit von Option 2 beträgt 99 µs, 103 µs und 98 µs. Die durchschnittliche / maximale / minimale Ausführungszeit von Option 3 beträgt 54 µs, 59 µs und 53 µs.
Eine weitere Reduzierung der Ausführungszeit der Netzwerke kann durch die Veränderung des Datentyps im neuronalen Netz erreicht werden. Wird der Datentyp von LREAL auf REAL verändert, so verringert sich die durchschnittliche Ausführungszeit bei der Option 1 auf 156 µs und bei der Option 3 auf 39 µs. Bei der Veränderung des Datentyps sind keine signifikanten Abweichungen der Ergebnisse der Netzwerke im Vergleich zum Datentyp LREAL festgestellt worden. Eine Änderung des Datentyps von LREAL auf REAL ist bei der Option 2 nicht möglich gewesen.
Für das zweite Anwendungsbeispiel zur Identifikation der vier Formen beträgt die durchschnittliche Ausführungszeit 275 µs unter Verwendung des Datentyps REAL.

7 Diskussion

Für die erste Option sind folgende Punkte zu nennen:

1. Das Arbeiten mit MATLAB /Simulink wird oft als intuitiv empfunden. Der Export des neuronalen Netzes mit dem PLC-Coder ist einfach.
2. Der Export aus Simulink über den PLC-Coder ist für alle gängigen SPS-Entwicklungsumgebungen möglich.
3. Die PLC-Coder Toolbox ist nicht im Standardumfang von MATLAB enthalten und erhöht die MATLAB-Lizenzkosten.
4. Der Compiler ist ein generischer Code-Translator und ist nicht für neuronale Netze optimiert.
5. Das neuronale Netz benötigt eine durchschnittliche Ausführungszeit von 217 µs. Es ist damit etwa viermal so langsam wie die schnellste Option (3).

Für die zweite Option gilt folgendes:

1. Der Export des neuronalen Netzes mit Simulink Coder erzeugt plattformunabhängigen C/C++ Code. Es wird jedoch ein herstellerspezifisches Target benötigt, um das neuronale Netz in die SPS-Laufzeitumgebung einzubinden.

[6] Es ist zu betonen, dass die folgenden Ergebnisse für eine SPS-Laufzeitumgebung gelten, die auf einem Intel-Multicore-Prozessor beheimatet ist. Es gibt auch Möglichkeiten, die neuronalen Netze auf separaten Prozessoren auszuführen. Siemens bietet z. B. das Erweiterungsmodul TM-NPU [15] für die S7-1500 an, auf dem ein neuronales Netz ausgeführt werden kann.

2. Die Integration des neuronalen Netzes in die SPS-Laufzeitumgebung mittels herstellerspezifischer Targets ist auf eine Teilmenge von Automatisierungsanbietern beschränkt. Sie ist oft nur für sogenannte Soft-SPSen möglich. Bei klassischen hardwarebasierten SPSen ist dies nicht möglich.
3. Der Simulink Coder in Verbindung mit der TE1400 von TwinCAT ist im Vergleich zum SPS-Coder kostengünstig.
4. Der Coder verwendet einen Code-Translator, der für ein breites Spektrum von Aufgaben geeignet ist und daher nicht für neuronale Netze optimiert ist.
5. Das neuronale Netz benötigt eine durchschnittliche Ausführungszeit von 99 µs. Es ist damit etwa doppelt so langsam wie die schnellste Option (3).

Bei der Verwendung von MATLAB für das Training des neuronalen Netzes (erste und zweite Option) muss zwischen flachen neuronalen Netzen und tiefen neuronalen Netzen unterschieden werden, da hierfür unterschiedliche MATLAB-APIs (die jedoch in derselben Toolbox verfügbar sind) verwendet werden. Ein fließender Übergang zwischen diesen beiden APIs ist nicht möglich. So werden beispielsweise Dropout-Schichten in den flachen neuronalen Netzen von MATLAB nicht unterstützt.
Die dritte Methode unter Verwendung von Keras und TwinCAT 3 Machine Learning Inference Engine TE3810 hat die folgenden Eigenschaften:

1. Keras ist kostenlos und ein weit verbreitetes ML-Framework in der industriellen Umgebung. Zwischen den verschiedenen Toolboxen gibt es keine Unterschiede in Bezug auf Training, Modellstruktur und Funktionsumfang. Die Verwendung von Dropout-Layern ist möglich.
2. Ein Import des neuronalen Netzes ist über das Standardformat ONNX [7] möglich und kann in eine Binärdatei umgewandelt werden. Somit ist ein Know-how-Schutz möglich.
3. Verschiedene neuronale Netze können während der Ausführung der SPS-Runtime geladen werden.
4. Es ist möglich, Fehlerzustände beim Laden und Ausführen des Netzes abzufragen.
5. Der verwendete Compiler ist für neuronale Netze optimiert.
6. Das neuronale Netz benötigt eine durchschnittliche Ausführungszeit von nur 54 µs, das ist etwa doppelt so schnell wie die zweite Option und etwa viermal so schnell wie die erste Option.
7. Es ist auch möglich, den Datentyp REAL statt LREAL für den Eingangs- bzw. Ausgangsvektor des MLP zu verwenden. In diesem Fall beträgt die Ausführungszeit nur 39 µs statt 54 µs.

Die Abweichung zwischen den Ausführungszeiten der beiden Anwendungsbeispiele lässt sich damit erklären, dass das Modell für den four-shape-Datensatz mehr als fünfmal so viele zu erlernende Parameter besitzt. Somit steigt die Rechenzeit für die Klassifizierung.

8 Fazit

Die Ausführung von neuronalen Netzen zur Bildanalyse wird in Sub-Millisekunden demonstriert. Im besten Fall werden die Klassifizierungsergebnisse auf einem Standard-Industrie-PC in nur 39 µs berechnet. Übliche Bewegungssteuerungsaufgaben werden typischerweise in 1 - 2 ms ausgeführt. In der obigen Anwendung ist die Bewegungssteuerungsaufgabe für das Transportsystem und den Deltaroboter sogar 250 µs schnell. Das heißt, der nächste Sollwert für die Bewegungssteuerung muss innerhalb der nächsten Zykluszeit von nur 250 µs vorliegen. Es wird gezeigt, dass selbst in dieser anspruchsvollen Echtzeitumgebung ausreichend Zeit bleibt, um die Bilder zu klassifizieren und die einzelnen Produkte in Abhängigkeit vom Klassifizierungsergebnis zu bewegen.
Der wichtigste Schritt zum Einsatz des neuronalen Netzes ist seine Umwandlung in maschinenlesbaren Code mit Hilfe eines Compilers. Der Compiler der TwinCAT 3 Neural Network Inference Engine TF3810 ist der effizienteste, da er für neuronale Netze optimiert ist. Darüber hinaus lassen sich damit während der Ausführung der SPS-Laufzeitumgebung verschiedene neuronale Netze laden. Dies bietet den Vorteil, dass individualisierte Produktionssysteme bei einem Wechsel des neuronalen Netzes nicht angehalten, neu konfiguriert und gestartet werden müssen.

Die beiden Anwendungsbeispiele konzentrieren sich auf Produktionssysteme, deren Produkte in ihrer Beschaffenheit variieren können. Dazu können auch Qualitätseigenschaften gehören. In diesen Fällen kann Maschinelles Lernen (ML) sowohl den Programmieraufwand reduzieren als auch die Effizienz herkömmlicher Bildverarbeitungslösungen verbessern und so flexible und agile Produktionsprozesse ermöglichen.

9 Quellen

[1] Y. Koren, "The Local Factory of the Future for Producing Individualized Products," in The Bridge, National Academy of Engineering, Band 51, Nr. 1, S. 20-26, April 2021.

[2] C. Demant, et. al., Industrielle Bildverarbeitung, 2011, Springer Verlag.

[3] J. Frochte, „Maschinelles Lernen“, 2019, Carl Hanser Verlag München.

[4] Beckhoff, TF7xxx | TwinCAT 3 Vision, Zugegriffen: 28.11.2021. [Online]. Verfügbar unter: https://www.beckhoff.com/en-us/products/automation/twincat/tfxxxx-twincat-3-functions/tf7xxx-tc3-vision

[5] Keras, Zugegriffen 28.11. 2021. [Online]. Verfügbar unter: https://keras.io/

[6] MathWorks, MATLAB, Zugegriffen: 28.11.2021. [Online]. Verfügbar unter: https://de.mathworks.com/products/matlab.html

[7] ONNX Supported Tools, Zugegriffen: 28.11.2021. [Online]. Verfügbar unter: http://onnx.ai/supported-tools

[8] M. Li et al., "The Deep Learning Compiler: A Comprehensive Survey," in IEEE Transactions on Parallel and Distributed Systems, Band 32, Nr. 3, S. 708-727, März 2021, doi:10.1109/TPDS.2020.3030548.

[9] Fachhochschule Kiel, et al., Machine learning for image recognition in the real-time context of industrial control systems, Zugegriffen: 28.11.2021. [Online]. Verfügbar unter: https://www.youtube.com/watch?v=KmPGQ2w_Peg&t=3s

[10] J. Fiedler, “Maschinelles Lernen zur Formerkennung in einem flexiblen Fertigungssystem zur Unterstützung individualisierter Produktion“, Bachelor-Thesis, Fachbereich Informatik und Elektrotechnik, Fachhochschule Kiel, Kiel, 2021.

[11] Fachhochschule Kiel, et al., Real-time object recognition with neural networks for industrial control systems, Zugegriffen: 28.11.2021. [Online]. Verfügbar unter: https://www.youtube.com/watch?v=DXfj88JkX5YY

[12] Y. LeCunn et al., THE MNIST DATABASE of handwritten digits, Zugegriffen: 28.11.2021. [Online]. Verfügbar unter: http://yann.lecun.com/exdb/mnist/

[13] Smeschke, Four Shapes, Zugegriffen: 28.11.2021. [Online]. Verfügbar unter: https://www.kaggle.com/smeschke/four-shapes

[14] Beckhoff, TF3810 | TwinCAT 3 Neural Network Inference Engine, Zugegriffen: 28.11.2021. [Online]. Verfügbar unter: https://www.beckhoff.com/en-us/products/automation/twincat/tfxxxx-twincat-3-functions/tf3xxx-tc3-measurement/tf3810.html

[15] Siemens, SIMATIC S7-1500 TM NPU, Zugegriffen: 28.11.2021. [Online]. Verfügbar unter: https://new.siemens.com/global/de/produkte/automatisierung/systeme/industrie/sps/simatic-s7-1500/simatic-s7-1500-tm-npu.html