www.javaspektrum.de

Ausgabe 6/2018
D € 7,90 · A € 8,90 · SFR 14,30

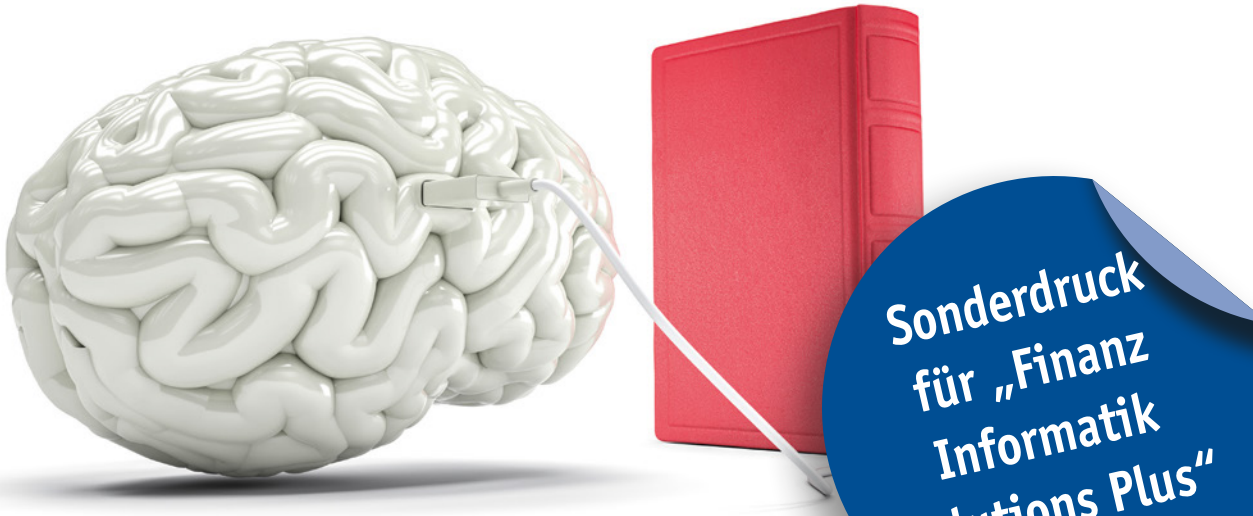
4 194156 107907 06

JavaSPEKTRUM

Magazin für professionelle Entwicklung und Integration von Enterprise-Systemen

Cognitive Computing

Mit dem Computer auf Augenhöhe?



Sonderdruck für „Finanz Informatik Solutions Plus"

finanz informatik solutions plus

Interview

Die Data Scientists Marc Hilbert und Yury Dzerin über KI-Forschungsprojekte für den Rennsport im VW Data Lab

Handschriftenerkennung mit neuronalen Netzen

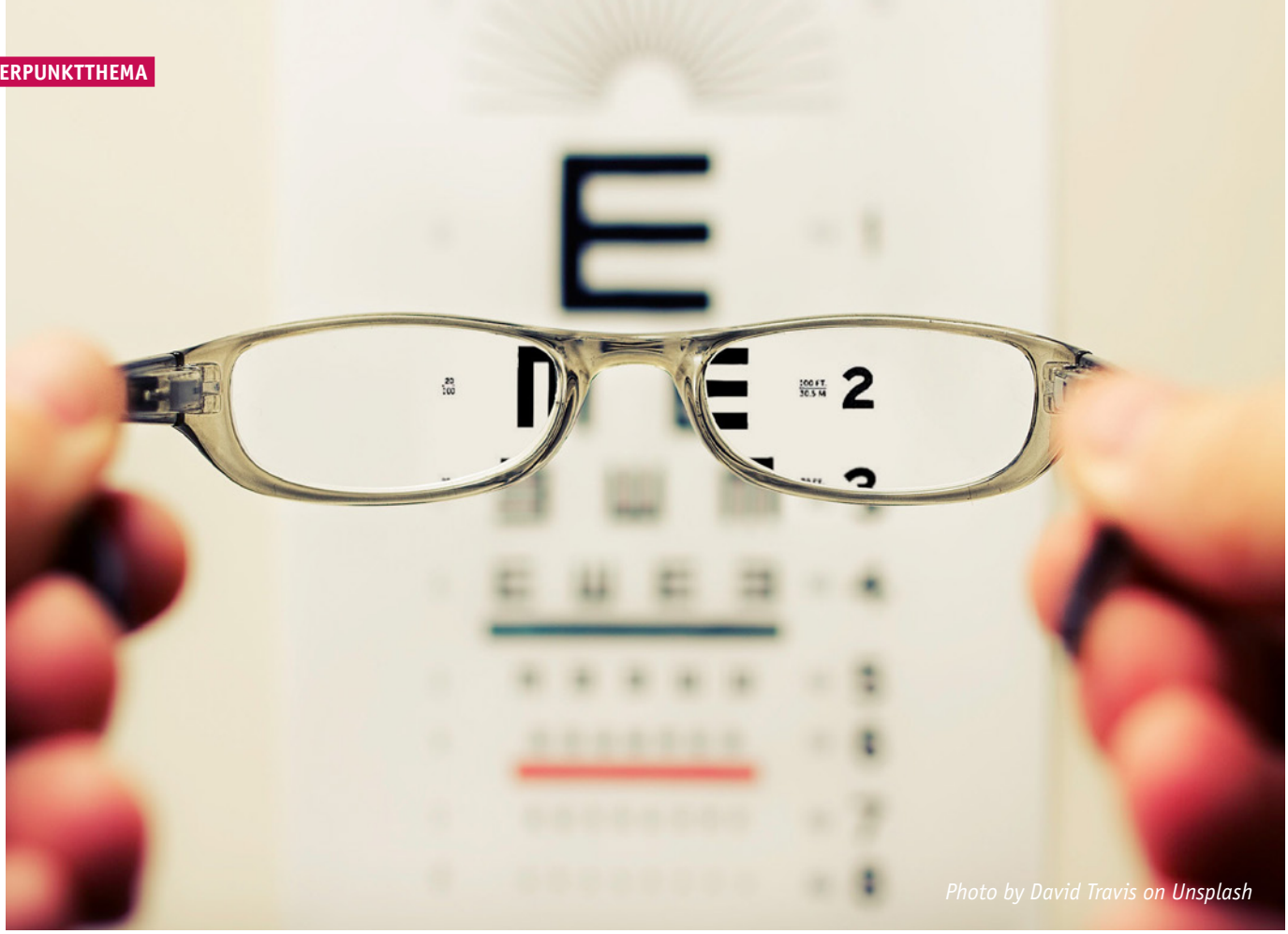
Fachthemen

Objektreferenzen in Java

Immerwährende Immunität für Java-Code

Cognitive Computing – Mit dem Computer auf Augenhöhe?

G30325



So gut wie das Auge?

Handschriftenerkennung mit Neuronalen Netzen

Ralf-Thomas Pietsch

Die Umsetzung regulatorischer Anforderungen in der Finanzwirtschaft stellt Firmen manchmal vor Herausforderungen, die mit KI-Technologien gemeistert werden können. So auch bei der Finanz Informatik Solutions Plus, deren Entwickler das automatische Erkennen von handschriftlich erfassten Steuer-ID-Nummern durch Einsatz eines Convolutional Neuronal Networks realisierten. Diese Lösung ergab sich erst nach längerem Suchen, Testen und Dimensionieren unterschiedlicher Neuronaler Netzwerke. Die gefundene Lösung erkennt Handschriften fast so gut wie das menschliche Auge.

Wer schon einmal auf einem Tablet-PC versucht hat, seine Handschrift durch eine App erkennen zu lassen, weiß, wie akkurat geschrieben werden muss, um ein einigermaßen zufriedenstellendes Ergebnis zu erhalten. Trotz größerer Anstrengungen scheinen Fehlerraten von fünf bis zehn Prozent durchaus keine Seltenheit. Eine Negativquote, die jeden Entwickler zu Recht am Erfolg seiner Arbeit zweifeln lässt, wenn er eine Lösung entwickelt, mit der beliebige Handschriften erkannt und die ausgelesenen Daten automatisch gespeichert werden sollen.

Dass die Entwickler der Finanz Informatik Solutions Plus (FI-SP) es im Endeffekt geschafft haben, handschriftliche Ziffern beliebi-

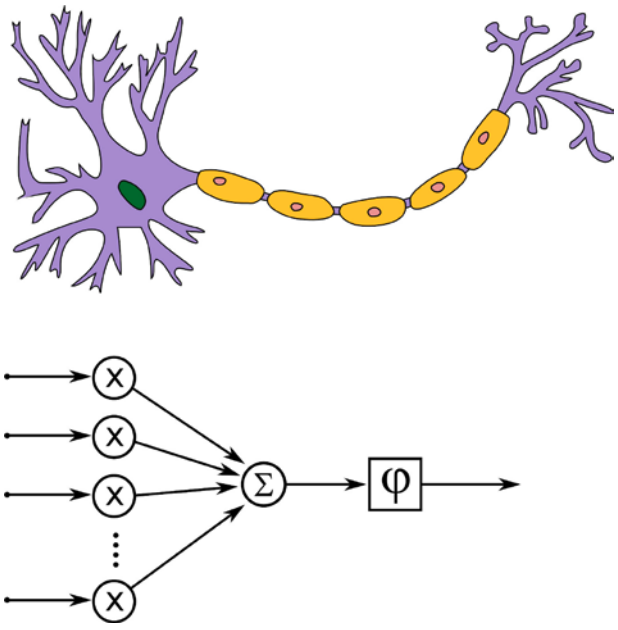


Abb. 1: Vorbild für Neuronale Netze sind Neuronen (oben, Zeichnung des Neurons von Quasar Jarosz, Lizenz CC-BY-SA-3.0). Die Funktion der Dentriten (links oben) wird durch die gewichtete Summe nachgebildet (links unten). Das Axion (mitte oben) durch die Aktivierungsfunktion

Dr. Ralf-Thomas Pietsch ist Softwarearchitekt und -entwickler bei der Finanz Informatik Solutions Plus GmbH. Seine persönlichen Schwerpunkte liegen beim Design verteilter Systeme und der Modellierung von Arbeitsabläufen sowie dem Informationsdesign und der User Experience. Neben Java beschäftigt er sich auch mit Programmiersprachen abseits des Mainstreams und schaut gerne mal über den Tellerrand.
E-Mail: ralf-thomas.pietsch@f-i-solutions-plus.de

ger Verfasser mit einer Genauigkeit von mehr als 98 Prozent automatisiert zu erkennen, war ein lehrreiches Stück Arbeit, in dem die Softwareingenieure die Möglichkeiten und Grenzen unterschiedlicher Neuronaler Netzwerke kennengelernt sowie Datensätze auf ihre Eignung hin verprobt und auch die für diesen Zweck geeigneten Datensätze erstellt haben.

Das richtige Netz gefischt

Wer in Deutschland einen Kredit in Höhe von 12.000 Euro und mehr in Anspruch nimmt, muss seit dem 1. Januar 2018 seinem Kreditgeber seine Steueridentifikationsnummer (Steuer-ID) mitteilen. Das gilt für neue wie für bestehende Kredite. Diese Erweiterung der regulatorischen Anforderungen an Banken und Sparkassen bedeutet unter anderem, dass die Institute die Steuer-IDs ihrer Bestandskunden anfordern und nachträglich erfassen müssen. Die notwendigen Formulare dafür sind schnell erstellt und versendet. Die Stammdaten sind alle vorhanden und die Kästen für das Eintragen der 11-stelligen Steuer-ID vom Marketing schnell „gemalt".

Die Herausforderung beginnt, wenn die Formulare per Fax, E-Mail und Post handschriftlich ausgefüllt wieder zurückkommen. Spätestens dann stellt sich die Frage, wie sich die Daten automatisiert erfassen und die relevanten Informationen erkennen lassen. Nach einer Recherche scheiden marktgängige Lösungen aus, denn sie erkennen nur Druckbuchstaben. Infrage kommen vielmehr Lösungen zur Mustererkennung, zum Beispiel Neuronale Netze.

Im Falle des konkreten Projektes musste es eine Lösung sein, die nahtlos in die bestehende Java-Anwendung integriert werden kann. Verschiedene Prototypen auf Basis eines einfachen Feed-Forward-Netzwerks [Wiki-c] und auch auf Basis der ausführlich dokumentierten Java-Bibliothek SNIPE führten nicht zum gewünschten Ziel. Daher entschieden sich die Entwickler für den Einsatz eines Convolutional Neural Networks (CNN). Solche komplexeren Netze wurden für das Verarbeiten von Bildern entwickelt. Sie sind einfachen Neuronalen Netzwerken in puncto Mustererkennung überlegen und können trainierte Muster auch dann noch erkennen, wenn sich deren Position innerhalb des Eingangsbilds verschiebt.

Möglich ist dies durch die besondere Funktionsweise dieser komplexen Netzwerke. Der Aufbau von CNNs ist biologisch motiviert und basiert auf Forschungsergebnissen, die bis in die 1950er Jahre zurückreichen. Im Gegensatz zu Feed-Forward-Netzwerken „betrachten" Convolutional Layer den Input mittels einer mathematischen Faltungsoperation (engl. Convolution), die im Endeffekt eine spezielle zweidimensional gewichtete Summe des Eingangssignals darstellt. Der Input wird in Form einer Matrix verarbeitet, sodass auch Bilder als Input verwendet werden können. Dabei können neben der Breite und der Höhe auch Farbkanäle analysiert werden.

Im Übrigen besteht ein CNN aus Filtern und Aggregations-Layern, die in mehreren Schichten hintereinander aufgebaut sind. Ein

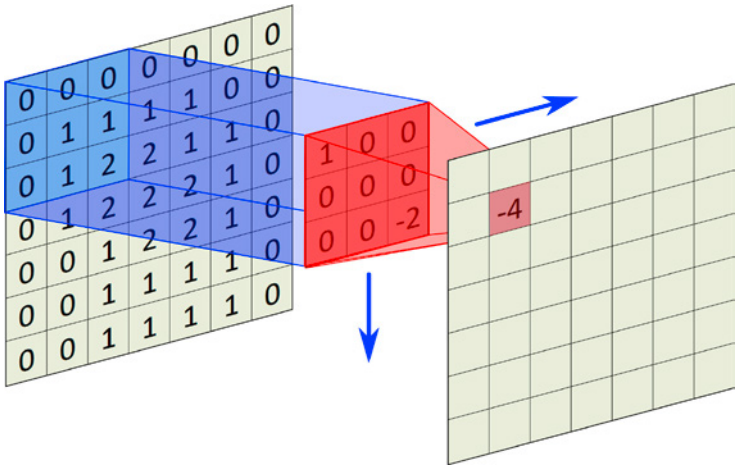


Abb. 2: Die Berechnung der Ausgangswerte in einem Convolutional Layer erfolgt durch Verschiebung der Gewichtungsfaktoren über das Eingangssignal

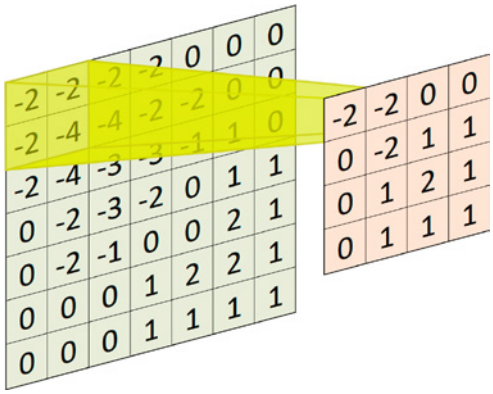


Abb. 3: Durch „SubSampling" wird die Auflösung reduziert. Im dargestellten Beispiel wird jeweils der Maximalwert aus einem 2x2 Quadrat ausgewählt

Convolutional Layer enthält hierbei gleichzeitig zahlreiche Matrizen, die mehrere Ausgangswerte liefern, die von den folgenden Layern weiterverarbeitet werden. Die Matrix, welche die Gewichte enthält, wird zur Berechnung des Ausgangssignals schrittweise über das gesamte Eingangssignal bewegt. So wird für jede Position ein Ausgangswert berechnet (s. Abb. 2). Die berechneten Werte werden danach noch in einem SubSampling-Layer zusammengefasst, der die „Auflösung" verringert und beispielsweise den maximalen Wert auswählt (s. Abb. 3).

Diesen Ansatz verfolgten die Entwickler in dem konkreten Projekt. Sie implementierten zunächst eine Lösung auf Basis der nativ in Java implementierten CNN-Bibliothek, JavaCNN [GitH-a]. Die Bibliothek steht unter der MIT-Lizenz [MITLic], womit auch einem produktiven Einsatz nichts entgegensteht. Fast 100 Forks sowie die 90 Sterne auf dem Entwicklerportal github.com zeugten davon, dass es sich um ein aktiv genutztes CNN handelt.

Die Einstiegshürden beim Einsatz dieser Bibliothek sind sehr gering, sodass sehr schnell mit der Implementierung begonnen werden konnte. Nachdem einige Anpassungen am Quellcode vorgenommen wurden, um die Bibliothek in den Build-Prozess integrieren zu können, wich mit den ersten Tests die anfängliche Begeisterung recht schnell. Als klar wurde, dass die zunächst gewählte Netzwerkkonfiguration zu einfach war und für eine höhere Erkennungsrate mehr Layer benötigt wurden, entpuppte sich JavaCNN hinsichtlich der Konfiguration der Netzstruktur als wenig flexibel. Trotz sorgfältiger Auswahl der Koeffizienten führte das

Hinzufügen von Layern immer wieder zu Laufzeitfehlern. Damit war die Anzahl der einsetzbaren Convolutional Layer praktisch auf zwei limitiert und eine Steigerung der Komplexität und Tiefe des Netzes nicht möglich. In der Praxis kamen die Netze auf Basis von JavaCNN nicht über eine Erkennungsrate von 85 Prozent hinaus. Eindeutig zu wenig für den geplanten Einsatzzweck. Dennoch waren erste erfolgreiche Schritte mit CNNs getan.

Größere Flexibilität mit Deeplearning4J

Auf der Suche nach einer Java-Bibliothek, die eine größere Flexibilität bei der Konfiguration der Netzwerkstruktur erlaubt, stießen die Entwickler schließlich auf die plattformübergreifende Java-Programmbibliothek für Künstliche Intelligenz (KI), Deeplearning4J [DL4J]. Die Bibliothek stellt zwar beim Einsatz einige Hürden auf und setzt vor allem tiefer gehende Kenntnisse über die Konfigurationsmöglichkeiten von CNNs voraus, bietet aber die Möglichkeit, die Netzwerk-Layer freier und umfangreicher zu konfigurieren.

Mit den inzwischen gewonnenen Erfahrungen über die Konfiguration von CNNs wurde das Netz Schritt für Schritt so dimensioniert, dass es optimale Ergebnisse liefert. Vielversprechend war, dass bereits die erste evaluierte Netzstruktur eine Erkennungsrate von rund 88 Prozent ohne weitere Optimierungen der Trainingsdatensätze erbrachte. Dieses CNN versprach auf Basis der bis dahin gemachten Erfahrungen ein ausreichend hohes Erfolgspotenzial, um mit einer erneuten Erweiterung der Testdatensätze sowie kontinuierlicher Trainings mit schrittweiser Optimierung der Netzstruktur eine automatisierte hohe Güte in der Handschriftenerkennung realisieren zu können.

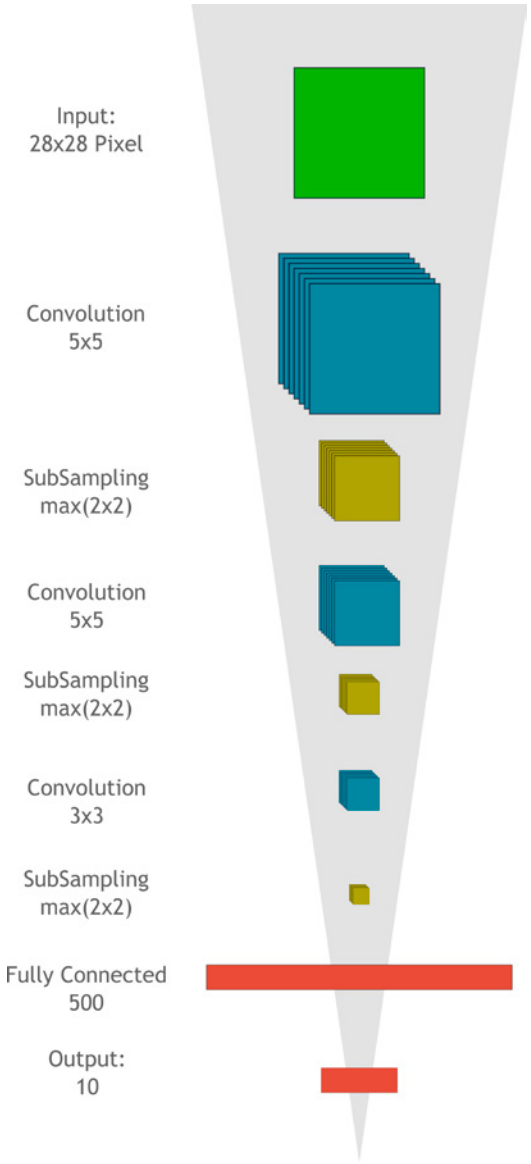


Abb. 4: Schematischer Aufbau eines der eingesetzten Netzwerke. Die erkannten Merkmale werden im Verlauf des Netzes von oben nach unten immer konkreter. Im Output-Layer gibt es schließlich bei einem der Ausgänge den größten Ausschlag

Training, Fehler, Back Propagation

Neuronale Netze werden gewöhnlich mittels sogenannter Back Propagation trainiert. Hierbei wird das Netz nacheinander mit zahlreichen Trainingsdaten versorgt. Die Abweichung an den Ausgängen des Netzes zwischen erwarteten und tatsächlichen Werten wird als Fehler klassifiziert. Erkennt ein Netzwerk eine Ziffer „1“ als Ziffer „8“, wird der Fehler vom Ausgang zurück zum Eingang durch alle Layer des Netzes nachverfolgt. Die Fehlerentwicklung verteilt sich dabei auf einzelne Koeffizienten. Diese Koeffizienten werden entsprechend korrigiert. Dabei werden Koeffizienten, die stark an einem falschen Ausgangssignal beteiligt sind, stärker geändert als Koeffizienten, die im konkreten Fall weniger Einfluss auf den Fehler des Ausgangssignals haben.

Je komplexer eine Netzwerkstruktur ist, desto mehr Trainingsdatensätze sind auch notwendig, damit der Back Propagation-Algorithmus die Koeffizienten des Netzes auch korrekt einstellen kann. Ein ausreichend großer Trainingsdatensatz ist daher von entscheidender Bedeutung. Eine zu komplex gewählte Netzwerkstruktur verbraucht beim Training unnötige Ressourcen und bietet keine höhere Erkennungsleistung. Daher ist auch Fingerspitzengefühl beim Auswählen und Anpassen der Netzwerkstruktur gefragt.

Die Back Propagation ist zeitaufwendig. Erst durch zahlreiche Testdaten und durch wiederholte Ausführung „lernt“ das Netz mit der Zeit anhand der Trainingsdaten Ziffern zu erkennen. Im vorgestellten Fall wird das Netz darauf trainiert, dass genau an einem der zehn Ausgänge ein großer Wert im Sinne einer hohen Erkennungsrate anliegt. Jeder der zehn Ausgänge ist hierbei genau einer der zehn Dezimalziffern zugeordnet.

Nach mehreren Iterationen und Anpassungen an der Netzwerkstruktur bestand das Netz schließlich aus insgesamt neun Layern:

- einem Input-Layer mit 784 Eingängen, entsprechend 28x28 Pixel der Eingangsbilder,
- drei Convolutional Layern mit unterschiedlichen Dimensionen (5x5, 5x5, 3x3),
- die jeweils von einem Sub-Sampling-Layer gefolgt werden.

Abgeschlossen wird das Netz durch einen Fully Connected Layer mit 500 Neuronen und einen Output-Layer mit zehn Ausgängen – für jede Ziffer einen (s. Abb. 4).

Die in jedem Convolutional Layer erkannten Merkmale werden hierbei immer konkreter. Während der erste Layer sich auf die Erkennung von einfachen geometrischen Mustern, wie horizontale, vertikale und diagonale Farbverläufe, einstellt, werden im nächsten Layer beispielsweise Kreissegmente und Bögen erkannt. Der abschließende Fully Connected Layer dient zur Gewichtung der Merkmale, die vom letzten Convolutional Layer erkannt wurden und die für die Erkennung einer konkreten Ziffer notwendig sind.

In der späteren Praxis zeigte sich Deeplearning4J dann noch einmal von einer ganz unerwarteten Seite: Kurz vor der ersten Produktivnahme stellte sich heraus, dass der Ansatz, in der Programmbibliothek möglichst viel nativen Code für zahlreiche Plattformen bereitzustellen, einen hohen Speicherbedarf erfordert. Die FI-SP benötigt aber nur den Windows-Code zur Entwicklung und den Linux-Code für alle anderen Systeme einschließlich des Produktivsystems. Daher wurden durch Konfiguration des Java-Build-Tools Maven die überflüssigen Code-Binaries entfernt, was die ausgelieferte WAR-Datei wieder auf eine akzeptable Größenordnung brachte.

Der Datenschatz für die Trainings

Neben der Auswahl und Strukturierung eines geeigneten Netzes kommt dem Einsatz eines geeigneten Trainingsdatensatzes eine entscheidende Rolle zu. In der Literatur findet sich sehr schnell, dass für die Handschriftenerkennung der MNIST-Datensatz [MNIST] weit verbreitet ist. Erste Tests zeigen aber, dass die US-amerikanischen MNIST-Daten in Deutschland wenig hilfreich sind. Denn die Schreibweisen für die Ziffer „1“ und in vielen Fällen auch für die Ziffer „7“ unterscheiden sich erheblich von der in Deutschland üblichen Schreibweise (s. Abb. 5).

Alternative Datensätze mit einer europäischen Schreibweise sind nicht verfügbar. Da für komplexere Netzwerkstrukturen ein ausreichend großer Trainingsdatensatz notwendig ist, musste eine Alternative zu MNIST gefunden werden. Die Entwickler fanden einen recht naheliegenden Ansatz, um einen geeigneten Datensatz für das Training und die Tests des Netzes zu generieren: Sie baten kurzerhand ihre Kollegen aus dem Projekt um Schriftproben. Diese zeigten sich mal wieder als ein Team, das schnell und pragmatisch Lösungen findet, und lieferten ihre Schriftproben an das Projekt. Die Ziffern wurden zunächst noch halbautomatisch aus den Formularen separiert. Im Laufe der Entwicklung wurden Schriftproben von weiteren Kollegen aus dem ganzen Unternehmen geliefert und die Separation der Ziffern aus den Formularen automatisiert.

So konnte in relativ kurzer Zeit ein Test- und Trainingsdatensatz generiert werden, der parallel zur Weiterentwicklung der Netzwerkstruktur stetig anwuchs. Ein Teil dieses Datensatzes wurde von der FI-SP freigegeben und kann für eigene Experimente genutzt werden [GitH-b]. Im Laufe der Entwicklung konnten so auch immer komplexere Netzwerkstrukturen vernünftig trainiert werden.

Damit haben die Entwickler zeitnah einen großen Datensatz an Ziffern in europäischer Schreibweise aufgebaut, mit dem das Netz trainiert und getestet werden konnte. Dabei wurden auch erste Ausschnitte aus anonymisierten Schriftproben der Live-Daten einbezogen. Mit diesem Datensatz konnte dann direkt getestet werden, wie die Formular-Daten optimal extrahiert werden. Denn die Formulare wurden mit Kästchen versendet, in die die Ziffern der Steuer-ID eingetragen waren. Der Vorteil war, dass jede ID in der korrekten Schreibweise zurückgesendet wurde. Der Nachteil war, dass die Ränder der Kästchen für die Erkennung eine zusätzliche Herausforderung darstellten. Wurden für das erste Training und Testen die Ziffern noch aus den Formularen manuell separiert und extrahiert, musste dafür im weiteren Projektverlauf eine geeignete Automationslösung für die Vorverarbeitung gefunden werden.

Die Softwareexperten entwickelten einen Algorithmus, der in mehreren Schritten die Ziffern aus den Kästchen extrahiert. Zunächst führt der Algorithmus mittels Gauß-Filters [Wiki-b] eine Weichzeichnung des Eingangsbildes durch. Damit wird das oft stark auftretende Rauschen bei unterschiedlichen Quellen gedämpft. Im nächsten Schritt werden mittels des Canny-Edge-Algorithmus [Wiki-a] die Kanten des Formulars extrahiert. In einem weiteren Bearbeitungsschritt wird das Bild ausgehend von den Eckpunkten einheitlich eingefärbt. Das Resultat ist ein Bild, in dem die Kästchen des Formulars eindeutig hervortreten. Durch die Berechnung eines horizontalen und vertikalen Histogramms können in diesem Bild die Positionen der Kästchen nun sehr genau bestimmt werden. Das ist die Voraussetzung, um die Ziffern anschließend extrahieren zu können.

Darüber hinaus wurde eine Lösung entwickelt, mit der die Ausrichtung der Formulare korrigiert werden konnte. Dazu wurden die

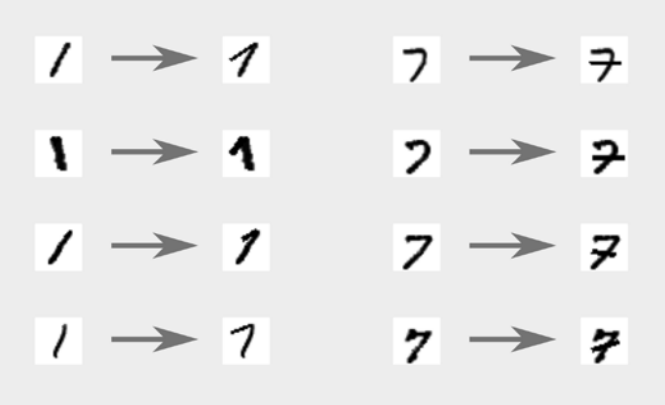


Abb. 5: Durch die Transformation in die in Deutschland übliche Schreibweise wurde der MNIST-Datensatz in vollem Umsatz zum Training einsetzbar

Ziffern zunächst mit Rahmen extrahiert. Erst danach wurde der Rahmen in den Bildausschnitten entfernt. Mit diesen Vorbereitungen konnte das Neuronale Netz automatisch mit Live-Daten aus dem kundenspezifischen Formular versorgt werden. Der entwickelte Algorithmus findet auch in Produktion Verwendung, um die Ziffern aus den Formularen zu extrahieren und dem Neuronalen Netzwerk zur Verfügung zu stellen.

Training des Netzwerks

Um die Erkennungsrate des nicht trainierten CNNs auf eine Zielgröße von 95 und mehr Prozent zu steigern, musste das CNN weiter trainiert werden. Dazu haben die Entwickler den Trainingsdatensatz erweitert. Der Testdatensatz umfasste knapp 60.000 eigene Samples. Darüber hinaus wurde der MNIST-Datensatz durch einen selbstentwickelten Algorithmus an den Ziffern „1“ und „7“ modifiziert. Der Algorithmus ergänzte die für die deutsche Schreibweise fehlenden Querstriche. Damit vergrößerte sich der Testdatensatz um weitere rund 68.000 Ziffern. Weitere 20.000 Trainingsdatensätze lieferten die Formularfeldausschnitte aus den Live-Daten. Letztere brachten immer wieder neue Herausforderungen mit sich, die keiner vorab erahnen konnte: Schiefe Kästchen oder auch Handy-Fotos der Steuer-ID mussten in der Praxis als Grundlage für die Erkennung dienen. Die Trainingsdurchläufe dienten daher nicht nur dazu, das Netzwerk zu trainieren, sondern auch dazu, die Vorverarbeitung zur Extraktion der Ziffern so zu organisieren, dass sie den realen Begebenheiten entsprach. Gleichwohl stand die Entwicklung einer optimalen Struktur des CNNs stets im Vordergrund. Und wurde mit dem Erfolg gekrönt, dass inzwischen hervorragende 98 Prozent der eingelesenen Ziffern richtig erkannt werden.

Literatur und Links

[DL4J] https://deeplearning4j.org/
[GitH-a] https://github.com/ratopi/JavaCNN
[GitH-b] https://github.com/fi-sp/
[MITLic] https://choosealicense.com/licenses/mit/
[MNIST] Y. LeCun, C. Cortes, Ch. J. C. Burges, The MNIST Database of handwritten digits, http://yann.lecun.com/exdb/mnist/
[Wiki-a] https://de.wikipedia.org/wiki/Canny-Algorithmus
[Wiki-b] https://de.wikipedia.org/wiki/Weichzeichnen#GaußFscher_Weichzeichner
[Wiki-c] https://en.wikipedia.org/wiki/Feedforward_neural_network