Prof. Dr.-Ing. Reinhard Kaschuba
Labor für Bildverarbeitung, Labor für Photonik
Simulation Optischer Systeme
Wintersemester 2023/2024

Hochschule Bielefeld
University of Applied Sciences and Arts

Echtzeiterkennung von Gebärden mit Convolutional Neural Networks

Linus Langenkamp, Reinhard Kaschuba
Hochschule Bielefeld, Fachbereich Ingenieurwissenschaften und Mathematik

Kurzfassung:

Gegenstand dieser Abhandlung ist das Trainieren eines Convolutional Neural Networks (CNN) zur Klassifikation eines Bilddatensatzes von Gebärdenzeichen sowie das Erläutern der prinzipiellen Funktionsweise von CNNs. Hierbei wird der vollständige Workflow - vom Erstellen der Trainingsdaten bis hin zum Verwenden des trainierten Modells in einer Applikation - dargestellt. Das resultierende Programm bietet die Möglichkeit, die Gebärdenzeichen in Echtzeit zu erlernen.

Problemstellung:

Das deutsche Fingeralphabet ist die in Deutschland gebräuchliche Sprache zum Buchstabieren von Namen und Wörtern mithilfe von Gebärdenzeichen. Jeder Buchstabe des Alphabetes entspricht hierbei einem spezifischen Handzeichen. Diese Arbeit beschäftigt sich damit ein neuronales Netz, welches eine Echtzeiterkennung des Fingeralphabetes ermöglicht, zu trainieren. Darauf aufbauend ist das Ziel dieser Arbeit eine Lernapplikation zu entwickeln, die das Lernen der Sprache erleichtert. Diese Zielsetzung impliziert, dass das neuronale Netz eine hohe Zuverlässigkeit beziehungsweise Genauigkeit aufweisen muss, sodass der Nutzer oder die Nutzerin die Sprache korrekt erlernt. Um dieses Ziel zu erreichen, werden die Trainingsdaten mit einem einzigen spezifischen Versuchsaufbau generiert. Es handelt sich somit nicht um eine generische Lösung. Ferner steht die generelle Umsetzung einer derartigen Lernapp im Vordergrund. In weiteren Untersuchungen kann dann eine Skalierung des CNNs vorgenommen werden.

Versuchsaufbau und Datenbeschaffung:

Die zum Training des neuronalen Netzes notwendigen Bilddaten werden hierzu über ein *Python*-Skript selbst erstellt. Bei der verwendeten Kamera handelt es sich um eine *Allied Vision Alvium 1800 U-811c* mit 8,1 Megapixel. Diese Kamera findet Verwendung, da sie die Möglichkeit bietet, die Belichtungszeit exakt einzustellen, was bei herkömmlichen Webcams nicht der Fall ist. Somit kann ein mögliches 50Hz Flackern von LEDs herausgefiltert werden. Dieser Vorteil ist von besonderer Bedeutung, da der selbstgeschriebene Hintergrundentferner einen Hintergrund benötigt, bei dem sich die Pixel über die Zeit nur geringfügig ändern. Um die Trainingsdaten für das CNN zu generieren, werden Videos der einzelnen Gebärden aufgenommen, in die einzelnen Frames zerlegt, transformiert und schließlich auf der Festplatte abgelegt. Während des Videos bewegt sich die Hand in alle Richtungen, da die Klassifikation der Gebärden unabhängig von der Position der Hand ist. Insgesamt ergibt sich ein Datensatz mit 54060 verschiedenen Bildern der Handzeichen. Beispiele hierfür sind:

Abbildung 1. Buchstaben A, M, R, U, Y und Q des deutschen Fingeralphabetes

Convolutional Neural Networks:

Bevor konkret auf das Training und die Anwendungen von CNNs eingegangen wird, sollen die Vorteile gegenüber herkömmlichen neuronalen Netzen in der Bildverarbeitung hervorgehoben und die prinzipielle Funktionsweise dargestellt werden.
Wird eine Bildklassifikation mit einem klassischen fully connected Neural Network durchgeführt, resultiert zwischen zwei beliebigen Layern eine Anzahl an Gewichten, die in etwa dem Produkt der Neuronenanzahl im Layer i und Layer i+1 entspricht. Das Netz weist somit eine enorme Anzahl an Freiheitsgraden auf und berücksichtigt nicht die Position der Pixel zueinander. Hier kommt das Konzept des Filters aus der Bildverarbeitung ins Spiel. Filter können mithilfe von Faltungsoperationen Kanten oder spezifische Details im Bild hervorheben. Ist I ein zweidimensionales Bild und K ein zweidimensionaler Filter, ergibt sich die diskrete Faltung (en. Convolution) von I und K als:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n)$$

Je nach Wahl der Parameter des Filters kann ein unterschiedlicher Effekt erzielt werden. Visuell lässt sich die Faltung des gesamten Bildes mit einem Filter oder Kernel wie folgt darstellen:

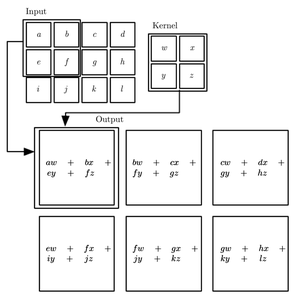

Abbildung 2. Schematische Darstellung der Faltung [1]

Der Kernel bewegt sich hierbei von links nach rechts und von oben nach unten über das Bild. Bei CNNs werden nun keine festen Filter, wie in der Bildverarbeitung, verwendet. Ferner werden dem Netz alle Parameter des Filters überlassen, sodass das CNN selbst erlernt, welche Features oder Kanten des Bildes wichtig sind.

Um die Dimension des Bildes weiter zu reduzieren, wird zudem das Max Pooling verwendet. Hierbei zerteilt man das Bild in kleine Quadrate und übernimmt in die nächste Schicht nur das Maximum aus jedem Quadrat:

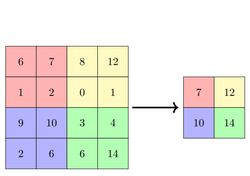

Abbildung 3. Beispiel für Max Pooling

Des Weiteren ist es notwendig, eine nichtlineare Aktivierungsfunktion zu verwenden. Bei CNNs handelt es sich hierbei häufig um die ReLU-Funktion ReLU(x)=max{0,x}. Die ReLU-Funktion sorgt dafür, dass alle Details mit negativem Wert auf Null gesetzt werden. Dadurch werden nur spezifische Features des Bildes betrachtet. [1]

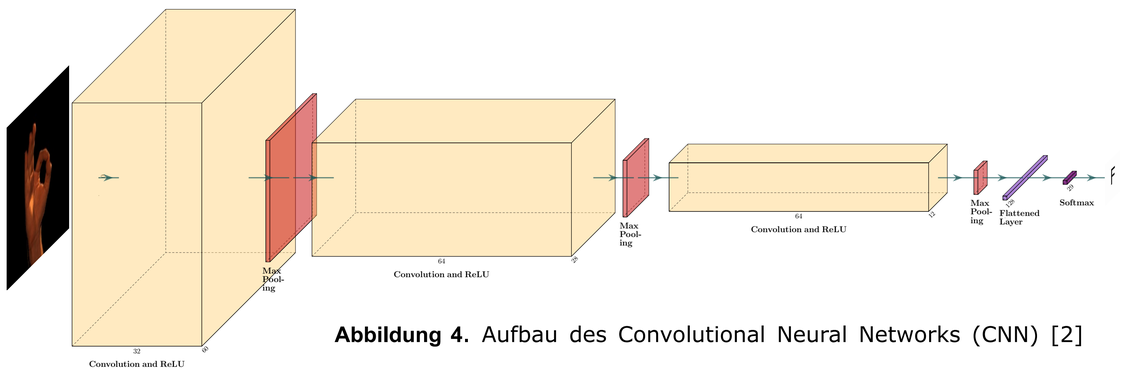

Abbildung 4. Aufbau des Convolutional Neural Networks (CNN) [2]

Die Hintereinanderschaltung von Convolutional Layer, ReLU und Max Pooling ist im Wesentlichen der Grundbaustein von CNNs. Auch für das selbsterstellte Convolutional Neural Network wurde diese Architektur verwendet. Zunächst wird das Bild eingelesen, dann durchläuft es dreimal eine Kette von Faltung, ReLU-Aktivierung und Max Pooling, wobei die Dimension sukzessive reduziert wird. Danach werden die extrahierten Features ausgeflacht und in eine klassische Schicht überführt. Jedes dieser Neuronen ist dann mit jedem Neuron der finalen Schicht verbunden. Daraufhin wird eine Softmax-Funktion angewendet. Mithilfe dieser kann der Output des Netzes für jede Klasse als Wahrscheinlichkeit des entsprechenden Zeichens interpretiert werden.

Vorverarbeitung:

Um Invarianz gegenüber dem Hintergrund zu erzielen, wird selbiger entfernt. Dieser Prozess wird sowohl für die Trainingsdaten als auch bei späteren Liveanalysen durchgeführt.

Für die ersten 24 Frames, während die Kamera läuft, werden Referenzbilder des Hintergrunds aufgezeichnet. Danach wird das arithmetische Mittel dieser Bilder bestimmt und als Referenzhintergrund gesetzt. Geht ein Livebild ein, so wird für jeden Pixel des Bildes überprüft, ob es mindestens einen RGB-Farbwert gibt, sodass die Differenz von Livebild und Referenz größer als 12 ist. Ist dies der Fall, so wird der Pixel aus dem Livebild in einer Maske auf 1 und andernfalls auf 0 gesetzt. Um zusätzlich Rauschen zu verringern, werden noch zwei Faltungen auf die Maske angewendet. Danach wird die Maske mit dem Livebild punktweise multipliziert und zurückgegeben.

Eine weitere Transformation, welche vorab durchgeführt wird, ist die Skalierung auf eine geringe Auflösung. In diesem Fall werden die Bilder der Gebärdenzeichen von der Dimension 200 x 200 x 3 auf 64 x 64 x 3 verkleinert. In vielen Fällen werden so fast keine wesentlichen Informationen entfernt und die Dimensionalität des Problems nimmt ab, was in einem erheblich schnelleren Training resultiert.

Training:

Mithilfe der erstellten Trainingsdaten und dem CNN (Abbildung 4), welches in dem *Python*-Paket *Keras* realisiert ist, kann das Training durchgeführt werden. Hierzu wird der Datensatz der Gebärdenzeichen in 1/3 Validierung und 2/3 Training unterteilt. Der Trainingsanteil wird dazu benutzt das CNN sukzessive zu trainieren, wobei die Validierungsdaten dazu dienen, den Fehler des Netzes mit unabhängigen Daten zu bestimmten. Bei dem verwendeten Optimierer handelt es sich um *adam* und die verwendete Verlustfunktion ist der *crossentropy loss*. Verringert sich der Fehler während des Trainings zwei Epochen in Folge nicht (sog. early stopping), so wird das Training vorzeitig abgebrochen und das beste Netz zurückgegeben.

Die Abbildungen 6 und 7 zeigen die Resultate des Trainings auf. Es ist zu erkennen, dass der Fehler des Netzes sukzessive abnimmt und nach 40 Epochen bei 10e-9 liegt. Zusätzlich zeigt insbesondere die Confusion Matrix auf, dass nur sehr wenige Fehler bei der Klassifikation der Gebärden geschehen. Dies ist zum einen durch die Qualität des Netzes, zum anderen aber auch durch die Art der Datenbeschaffung zu begründen, da in einer Videoaufnahme verschiedene Frames hintereinander sehr ähnlich zueinander sind. Letztendlich kann nur eine Echtzeitanalyse mit vollständig unabhängigen Daten die Aussagekraft dieser Ergebnisse validieren.

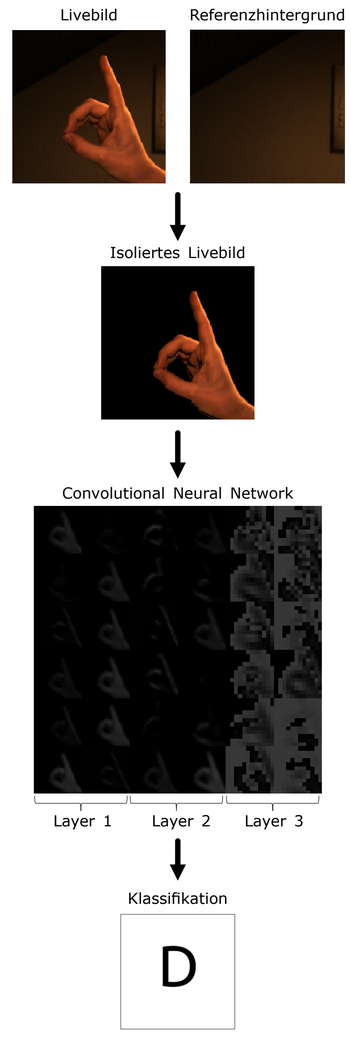

Abbildung 5. Beispielhafter Workflow des CNNs mit Vorverarbeitung für den Buchstaben D

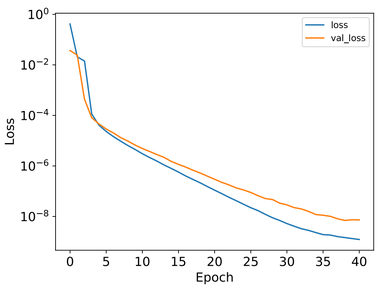

Abbildung 6. Fehler der Testdaten (Logarithmische Skalierung)

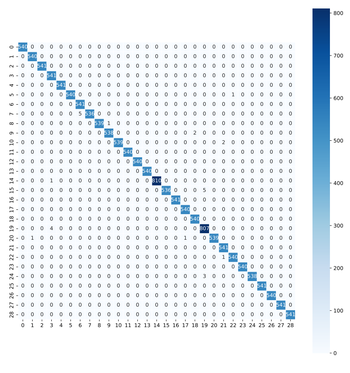

Abbildung 7. Confusion Matrix des trainierten Modells mit den Testdaten

Echtzeitanalyse und Lernapplikation:

Das trainierte Convolutional Neural Network wird abschließend in eine Lernapplikation eingebettet. Die App, die in Python geschrieben ist, hat insgesamt 3 Modi. Im ersten Modus werden einzelne Buchstaben zufällig vorgegeben und ein zugehöriges Piktogramm angezeigt. Der Nutzer beziehungsweise die Nutzerin muss das entsprechende Gebärdenzeichen für eine bestimmte Haltezeit in die Kamera zeigen. Diese Haltezeit kann per Slider beliebig zwischen 0 und 3 Sekunden verschoben werden. Erkennt das CNN den Buchstaben für alle Frames während dieser Haltezeit, wird ein neuer Buchstabe generiert. Der zweite Modus, welcher in Abbildung 8 dargestellt ist, funktioniert analog zum ersten, wobei nun ganze Wörter buchstabiert werden müssen. Zudem wird jeweils die Evaluation des CNN angezeigt, sodass die lernende Person indirekt Feedback über die Handhaltung bekommt. Im dritten Modus werden wie in Abbildung 5, die einzelnen Schritte des Prozesses visuell dargestellt. Insbesondere ist es möglich, einen Blick auf die Faltungen im CNN zu werfen.

Fazit:

Insgesamt funktioniert die Echtzeit-Lernapplikation mit der Hand, durch die das Netz trainiert wurde, sehr gut. In fast allen Fällen erkennt das CNN die gezeigte Gebärde. Auch bei unbekannten Händen wird der Buchstabe meist erkannt. Zudem ist die Lernapplikation äußerst performant, da in Echtzeit nur wenige Berechnungen, wie die Extraktion des Hintergrunds und die Evaluation des Netzes durchgeführt werden müssen. Zusätzlich kann in die Lernapplikation ein weiterer Modus zur eigenständigen Eingabe von Buchstaben implementiert werden. Alles in allem resultiert ein effizientes Programm mit eingebettetem neuronalem Netz, welches durch eine größere Menge von Trainingsdaten, insbesondere verschiedener Handformen und -größen, eine noch bessere Generalisierung aufweisen kann.

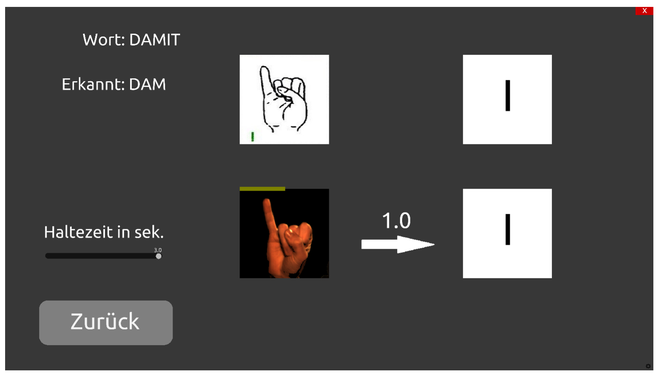

Abbildung 8. Lernapp mit Echtzeitklassifikation des deutschen Fingeralphabetes

Quellen:
[1] **Goodfellow et al. 2016** https://www.deeplearningbook.org/
[2] **Haris Iqbal** https://github.com/HarisIqbal88/PlotNeuralNet
[3] **Landesverband Bayern der Gehörlosen e. V.**

Selbsterstellte Inhalte:
[4] https://github.com/linuslangenkamp/Optische-Systeme
[5] https://www.youtube.com/watch?v=eP4siesBkNM