Dokumentation: Erkennung von handgeschriebenen Wörtern

Kilian Aaron Brinkner

Maschinelles Sehen WS 23/24

Idee

In diesem Projekt beschäftigte ich mich mit der Erkennung von handgeschriebenen Wörtern auf Bildern. Optical Character Recognition ist seit einigen Jahren schon Teil der Forschung und verbessert sich ständig weiter. Beispiele sind Google Translate, das über die Kamera Text erkennt und es on-the-fly übersetzt oder auch PDF-Programme, die Text in den Dokumenten automatisch erkennen. Doch dies unterscheidet sich von meiner Idee, denn hier betrachte ich Handschrift und keine Druckschrift. Dies erschwert die Erkennung enorm, da jede Person eine andere Handschrift besitzt und somit keine einheitlichen Buchstaben vorhanden sind. Ein Modell müsste dann die Charakteristiken eines Buchstaben erfassen können, auch wenn dieser nicht perfekt geschrieben ist.

Wie man in der Forschung an die Erkennung herangeht, wird hier beschrieben:

„This paradigm shift has been sparked due to adaption of cluster computing and GPUs and better performance by deep learning architectures [21], which includes Recurrent Neural Networks (RNN), Convolutional Neural Network (CNN), Long Short-Term Memory (LSTM) networks etc.“ (Memon et al., 2020)

Es wird gesagt, das ein Teil der verbesserten Erkennung der letzten Jahre die Verwendung von deep learning Architekturen sei. Zum einen Recurrent Neural Networks oder Long Short-Term Memory Networks vor allem für die Erkennung von Wörtern und Sätzen, während Convolutional Neural Networks für die Erkennung von einzelnen Buchstaben geeignet ist.

Diesen Ansatz mit Convolutional Neural Networks verwende ich in meinem Projekt. Dazu nutze ich den “NIST Handprinted Forms and Characters” Datensatz, der tausende Bilder zu jedem Buchstaben enthält. Verfügbar sind die Daten hier: https://research.aimultiple.com/ocr-accuracy/

Datenaufbereitung

Bevor ich die Bilder in einem Netzwerk trainiere, müssen die Daten bereinigt und vereinheitlicht werden. Denn wenn die Formate der Bilder gleich sind, sollte das Modell sich besser auf die Unterschiede der einzelnen Buchstaben fokussieren können.

Zunächst wird das Bild verkleinert, da dadurch die Ladezeit enorm verkürzt wird. Allerdings wird es hier, wie sich später herausstellt, Probleme mit den Architekturen geben, da diese teilweise mit größeren Bildern arbeiten. Danach wird das verkleinerte Bild zugeschnitten auf die schwarze Fläche, also so, dass der Buchstabe den kompletten Platz einnimmt und keine unnötige weiße Fläche zu sehen ist. Anschließend wird daraus ein quadratisches Bild erstellt, damit die Formate aller Bilder gleich sind, egal ob der Buchstabe nun länglich ist wie ein „g“ oder breit wie ein „m“.

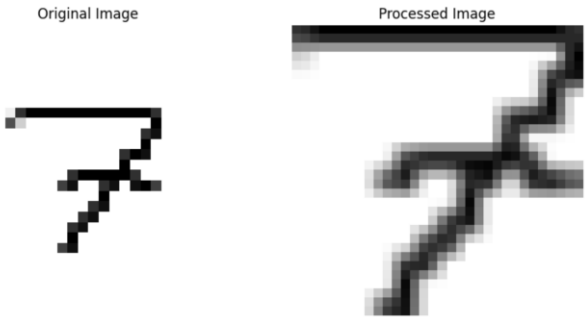


Abbildung 1: Aufbereiten der Bilder

Ergebnisse

Diese Bilder wurden anschließend in ein Convolutional Neural Network gegeben, das darauf trainiert, Ähnlichkeiten zwischen den unterschiedlichen Schreibweisen eines Buchstaben zu erkennen und später diese Buchstaben vorhersagen zu können.

Da jedoch die Anzahl pro vorhandenen Buchstaben im Datensatz variierte, wurden nur jeweils maximal 5000 Bilder pro Zeichen genommen. Teilweise waren auch mehr Bilder pro Zeichen vorhanden, doch teilweise auch weniger. Um dann eine Überrepräsentation zu vermeiden, wurde dieses Limit von 5000 gesetzt, damit jeder Character etwa gleich häufig vertreten ist.

Zunächst wurde das Modell auf Basis der LeNet5-Architektur mit 32x32 Pixeln trainiert.

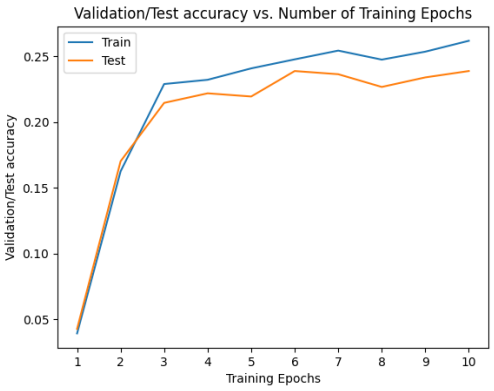


Abbildung 2: Accuracy von LeNet5

Abbildung 3: Loss von LeNet5

Dieses Modell zeigt auf den ersten Blick immer bessere Ergebnisse mit jeder Epoche auf, jedoch ist das Gesamtergebniss nicht zufriedenstellend. Die Genauigkeit liegt nach 10 Epochen bei gerade einmal 25%. Dies ist an sich nicht schlecht, allerdings können andere Modelle besser abschneiden. Auch der Loss ist mit 3.9 noch deutlich zu hoch. Man sieht, dass das Model zu unsicher mit den Vorhersagen ist.

Dann wurde ein neues Modell auf Basis von AlexNet mit einer Eingabe von 32x32 Pixeln trainiert.

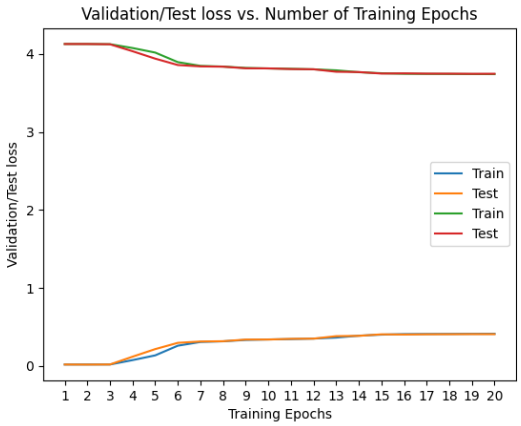


Abbildung 4: Accuracy (untere Kurven) und Loss (obere Kurven) mit AlexNet

Auf der Abbildung sind Loss und Accuracy in einer Abbildung zu sehen. Die Accuracy liegt nach 20 Epochen etwas über bei 0.4. Das ist zwar eine Verbesserung im Gegensatz zu LeNet5, allerdings ist dadurch jeder zweite Buchstabe immernoch falsch vorhergesagt. Der Loss ist auch wieder zu hoch bei fast 4, wodurch man sieht, dass das Modell nicht gut auf neue Daten generalisiert und daher viele Fehler auf den Trainings- und Testdaten macht.

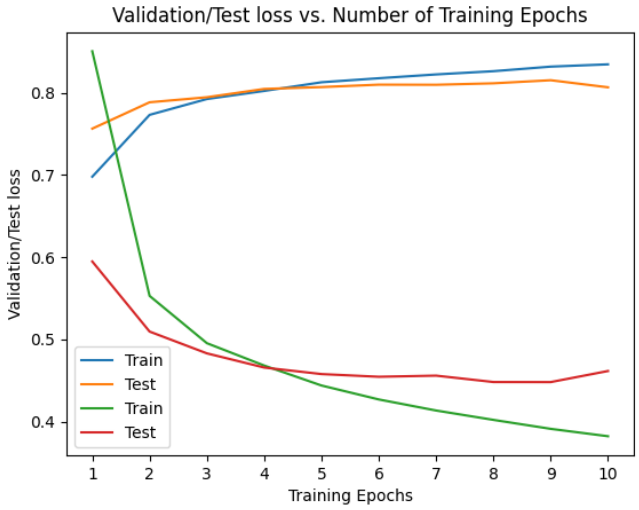


Abbildung 5: Accuracy (obere Kurven) und Loss (untere Kurven) bei ResNet50

Das ResNet50 macht schon einen deutlichen Unterschied. Der Loss liegt bei unter 0.5 der Testdaten und die Accuracy bei über 0.8. Diese Architektur konnte also eine enorme Verbesserung gegenüber der vorigen Modelle zeigen. Allerdings ist der Input hier auch ein 32x32 Pixel-Bild, wodurch einige Layer wegfallen mussten und nicht die komplette Architektur ausgenutzt werden konnte.

Deshalb wurde zuletzt das ResNeXt50 mit einer Eingabe von 128x128 Pixeln genutzt. Dadurch musste kein Layer wegfallen und die komplette Architektur konnte genutzt werden.

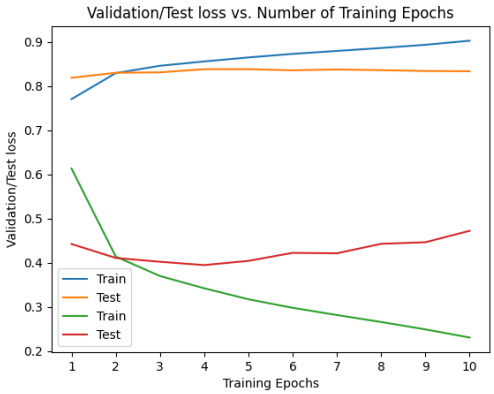


Abbildung 6: Accuracy (obere Kurven) und Loss (untere Kurven) bei ResNeXt50

Die Accuracy liegt bei 0.91, was das beste Ergebnis bisher ist. Doch es ist ein Abstand zwischen der Accuracy der Trainingsdaten und der Testdaten vorhanden. Das könnte bedeueten, dass das Modell zwar gut auf die vorhandenen Daten trainiert, jedoch die Kenntnisse schlechter auf neue Daten übertragen kann. Dasselbe sieht man beim Loss: Der Loss ist auch niedriger, jedoch sieht man eine Steigung der Kurve bei den Testdaten. Sie deutet darauf hin, dass hier Overfitting stattfindet. Das Modell lernt die Trainingsdaten zu genau und passt sich ihnen leicht an, anstatt allgemeine Muster zu erfassen. Möglicherweise ist dies auf eine zu komplexe Modellarchitektur zurückzuführen.

Trotz des Overfittings wurde im weiteren mit dem ResNeXt50 gearbeitet.

Beispiele

Abbildung 7: Anwendung des Modells auf Buchstaben

Es wurden viele gute Vorhersagen der Character gemacht. Das Modell erkennt die meisten Buchstaben und Zahlen ohne Probleme, doch es kann zu Verwechslungen kommen, wie hier zu sehen:

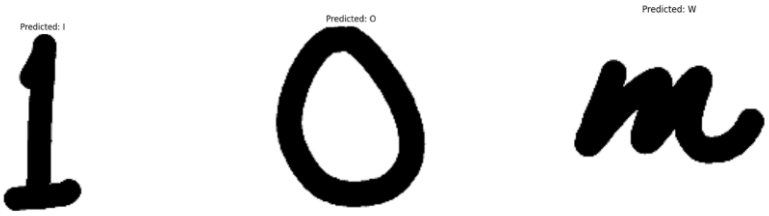


Abbildung 8: Verwechslungen bei der Vorhersage

Es gibt immer wieder Zeichen, die verwechselt werden. Zum Beispiel eine Eins, die als „l“, eine Null, die als „o“ oder ein „m“ als „w“ erkannt wird. Einige wenige Zeichen sind tatsächlich eindeutig falsch vorhergesagt. Allerdings gibt es viele Verwechslungen, die auch ein Mensch nicht eindeutig zuordnen könnte. Gleichzeitig tritt das Problem auf, dass das Modell Groß- und Kleinbuchstaben hin und wieder verwechselt. Allerdings halten sich die Fehler noch in Grenzen, denn die meisten Zeichen können gut erkannt werden.

Auswertung der Erkennung von Buchstaben

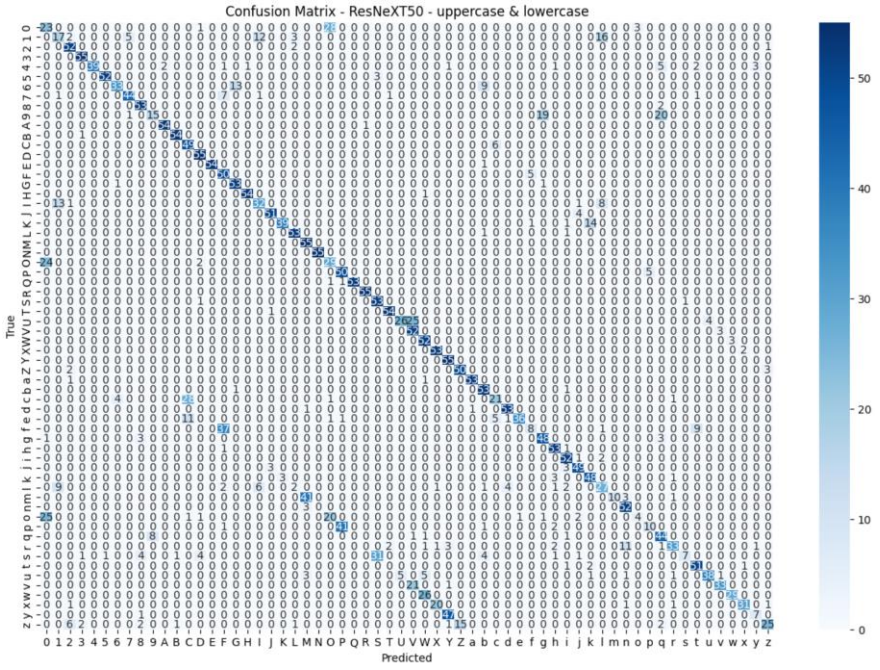


Abbildung 9: Confusion Matrix meines Modells

Bei der Konfusionsmatrix ist der Diagonalen zu erkennen, dass die meisten Zeichen – sowohl Buchstaben als auch Zahlen – sehr gut erkannt werden. Es gibt einige Verwechslungen, wie zuvor beschrieben, bei „o“, „0“, „i“, „1“ usw. Teilweise hätte auch Menschen Schwierigkeiten damit, diese zu unterscheiden. Denn hier werden nur die einzelnen Zeichen gezeigt, wodurch hier nicht aus dem Kontext gelesen werden könnte, um welches Zeichen es sich tatsächlich handelt.

Es ist aber auch zu sehen, dass es einige Verwechslungen hinsichtlich Groß- und Kleinschreibung gibt. Man erkennt nämlich eine zweite, etwas schwächere Diagonale. Diese zeigt, dass viele Kleinbuchstaben als Großbuchstaben identifiziert wurden. Allerdings sei auch hier gesagt, dass eine Unterscheidung von zum Beispiel einem kleinen und einem großen „P“ oder „Z“ schwierig ohne Kontext ist. Tatsächlich ist die umgedrehte Variante weniger häufig vertreten. Großbuchstaben werden selten als Kleinbuchstaben erkannt. Denn man sieht kaum eine Diagonale in der oberen rechten Ecke, die das sonst vermuten lassen würde.

Es gibt auch andere Metriken, mit denen sich das Modell untersuchen lässt:

	precision	recall	f1-score	support
0	0.32	0.42	0.36	55
1	0.42	0.31	0.36	55
2	0.80	0.95	0.87	55
3	0.93	1.00	0.96	55
4	1.00	0.71	0.83	55
5	0.98	0.95	0.96	55
6	0.87	0.60	0.71	55
7	0.88	0.80	0.84	55
8	0.84	0.96	0.90	55
9	0.65	0.27	0.38	55
A	0.96	0.98	0.97	55
B	0.96	0.98	0.97	55
C	0.55	0.89	0.68	55
D	0.86	1.00	0.92	55
E	1.00	0.98	0.99	55
F	0.51	0.91	0.65	55
G	0.79	0.96	0.87	55
H	0.98	0.98	0.98	55
I	0.63	0.58	0.60	55
J	0.93	0.93	0.93	55
K	0.93	0.71	0.80	55
L	0.87	0.96	0.91	55
M	0.53	1.00	0.70	55
N	1.00	1.00	1.00	55
O	0.36	0.53	0.43	55
P	0.54	0.91	0.68	55
Q	1.00	0.96	0.98	55
R	0.98	1.00	0.99	55
S	0.61	0.96	0.75	55
T	0.95	0.98	0.96	55
U	0.84	0.47	0.60	55
V	0.53	0.95	0.68	55
W	0.60	0.95	0.73	55
X	0.71	0.96	0.82	55
Y	0.51	1.00	0.67	55
Z	0.77	0.91	0.83	55
a	0.98	0.96	0.97	55
b	0.76	0.96	0.85	55
c	0.66	0.38	0.48	55
d	0.91	0.96	0.94	55
e	0.97	0.65	0.78	55
f	0.57	0.15	0.23	55
g	0.70	0.87	0.77	55
h	0.84	0.96	0.90	55
i	0.83	0.95	0.88	55
j	0.84	0.89	0.87	55
k	0.74	0.87	0.80	55
l	0.50	0.49	0.50	55
m	1.00	0.18	0.31	55
n	0.76	0.95	0.85	55
o	0.57	0.07	0.13	55
p	0.67	0.18	0.29	55
q	0.57	0.80	0.67	55
r	0.87	0.60	0.71	55
s	0.88	0.13	0.22	55
t	0.81	0.93	0.86	55
u	0.90	0.69	0.78	55
v	0.89	0.60	0.72	55
w	0.91	0.53	0.67	55
x	0.91	0.56	0.70	55
y	0.64	0.13	0.21	55
z	0.83	0.45	0.59	55
accuracy			0.75	3410
macro avg	0.78	0.75	0.73	3410
weighted avg	0.78	0.75	0.73	3410

Abbildung 10: Precision, Recall und f1-Score meines Modells

Die Precision sagt aus, wie viele der erkannten Zeichen tatsächlich richtig waren. 0.78 zeigt, dass das Modell die Zeichen überwiegend erkannt hat. Es gibt hierbei schlechtere Werte, wie beim „O“, das gerade einmal einen Wert von 0.36 hat. Da sieht man wieder das Problem, dass die Vorhersage oft auf eine Null oder ein kleines „o“ trifft anstatt einem großen „O“. Allerdings gibt es auch perfekt vorhergesagte Zeichen wie die „4“ oder „E“ mit einem Wert von 1.0. Gerade die sind charakteristisch eindeutig, wodurch das Modell keine Fehler gemacht hat bei der Vorhersage der beiden Zeichen. Allerdings ist bei der „4“ der Recall deutlich geringer bei 0.71, was bedeutet, dass teilweise eine „4“ vorhergesagt wurde, obwohl dies ein anderes Zeichen war.

Anhand des auch einigermaßen hohen Recalls von 0.75 kann man trotzdem schließen, dass es nur wenige falsch negative Vorhersagen im Vergleich zu den wahren positiven Vorhersagen gibt.

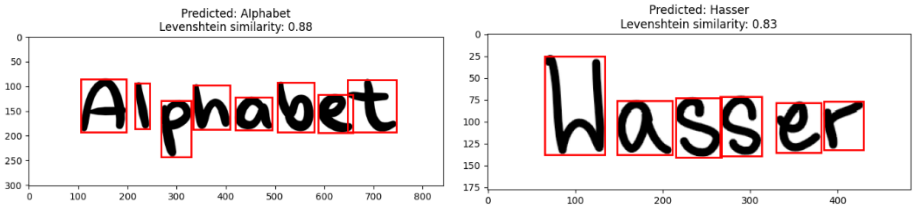
Der f1-score, der sich auf die beiden Werte bezieht, ist dementsprechend auch gleich hoch. Natürlich geht es noch besser, da immerhin 25 % falsch vorhergesagt wurden. Allerdings sieht man anhand der Werte, dass es eben die Schwierigkeiten mit doppeldeutigen Zeichen gibt, bei denen auch der Mensch ohne Kontext Schwierigkeiten hätte, diese zu erkennen. Wenn aber ein Zeichen eindeutig ist, wie zum Beispiel das „E“, dann wird dies auch in der Regel richtig erkannt.

Auswertung der Erkennung von Wörtern

Nun ist die Grundlage dafür geschaffen, Wörter zu erkennen. Mein Modell kann nämlich einzelne Zeichen erkennen. Bei der weiteren Berechnung wird davon ausgegangen, dass ein Zeichen eine zusammenhängende schwarze Fläche ist. Diese werden ausgeschnitten und das Modell auf die einzelnen Buchstaben angewandt.

Abbildung 11: Aufteilung der Wörter in Buchstaben

Das Ausschneiden der Buchstaben erfolgt über einen eigens implementierten Algorithmus, der alle zusammenhängenden dunklen Flächen erkennt und entsprechend wie zu Beginn beschrieben aufbereitet mit Zuschneiden, Verkleinern und das Bild Quadrieren. Der Algorithmus schneidet also die einzelnen Buchstaben aus und lässt das Modell entscheiden, um welchen Buchstaben es sich handelt. Daraus setzt sich dann ein Wort zusammen.



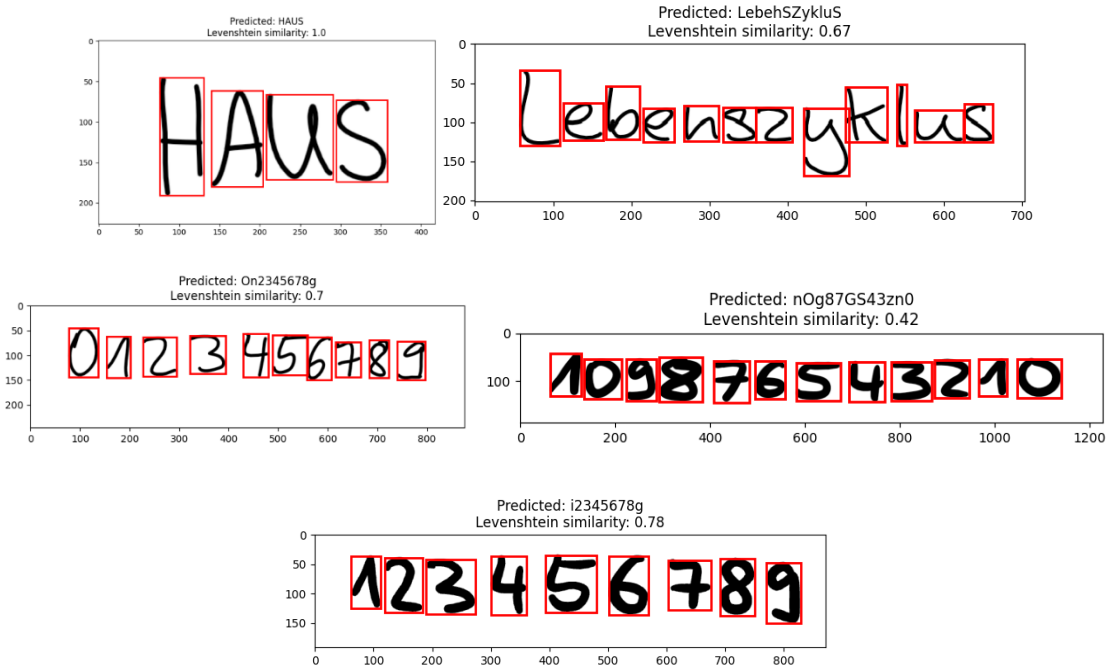


Abbildung 12-18: Erkennung von Wörtern mit meinem Modell

Zudem prüft der Algorithmus nach i-Punkten. Diese sind eigentlich eine zusammenhängende Fläche, weswegen diese als einzelner Buchstabe gewertet werden würde. Doch wenn die Fläche einen bestimmten Schwellenwert unterschreitet, gilt die Fläche als i-Punkt und es wird der dazugehörende Strich gesucht. Dabei wird horizontal gesehen, die nächste Fläche gesucht, die dann mit dem i-Punkt zusammengeführt wird. Somit entsteht aus dem i-Strich und aus dem i-Punkt einzelner Buchstabe. Doch hier stellt sich die Frage, inwiefern dies mit anderen Punkten von „ä“, „ö“ und „ü“ umsetzbar ist. Grundsätzlich ist auch anzumerken, dass dieser Algorithmus nur selbst implementiert ist und nicht das neurale Netzwerk „i“ als ganzes Zeichen erkennt, wie es optimalerweise gewünscht wäre.

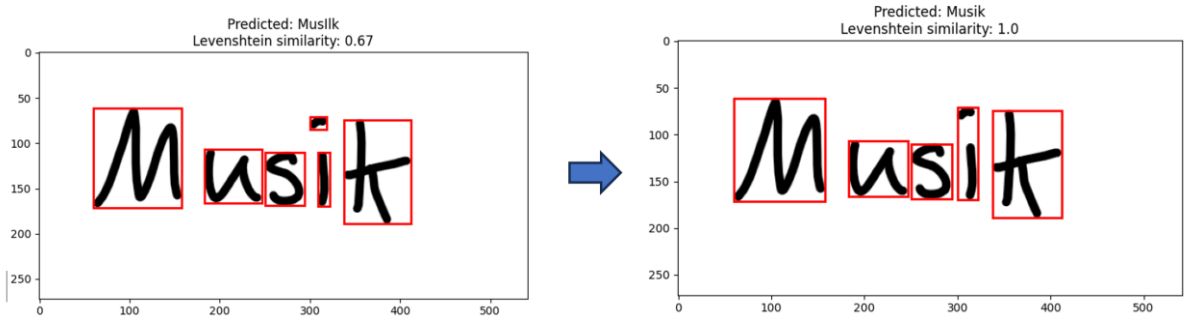


Abbildung 19: Zusammenfassen von i-Punkt und i-Strich

An den Bildern ist die Levenshtein-Distanz angezeigt. Diese misst die Ähnlichkeit zwischen zwei Zeichenketten. Definiert wird sie so:

Die Levenshtein-Distanz ist die „minimale Anzahl der notwendigen Änderungen, um zwei Zeichenketten aneinander anzugleichen“ (Luber 2022).

Es wird dabei berechnet, wie viele Operationen wie Einfügen, Löschen und Ersetzen nötig sind, um von einem Wort in ein anderes zu kommen. Je höher der Wert, desto ähnlicher sind sich zwei Wörter.

Es ist zu sehen, dass die Werte dafür stark schwanken. Teilweise kann ein falscher Buchstabe dafür sorgen, dass der Wert drastisch sinkt. Allerdings sieht man auch, dass einige vorhergesagte Wörter sehr ähnlich dem tatsächlichen Wort sind mit einer Distanz von 0.8, 0.9 oder sogar 1.0. Außerdem können hier die typischen Verwechslungen wieder beobachtet werden: eine „9“, die als „g“ erkannt wird etwa.

Die Levenshtein-Distanz aller meiner vorgesagten Test-Zeichenketten beträgt im arithmetischen Mittel 0.73. Das bedeutet, die erkannten Wörter und Zahlen entsprechen einer Ähnlichkeit von 0.73 den tatsächlichen Zeichenketten. Der Wert kann natürlich noch besser sein, allerdings sieht man auch mit bloßem Auge, dass die Wörter ziemlich gut getroffen sind. Teilweise sind es einzelne Zeichen, die den Schnitt nach unten ziehen, aber im Grunde werden Großteile der Wörter gut erkannt.

Vergleich

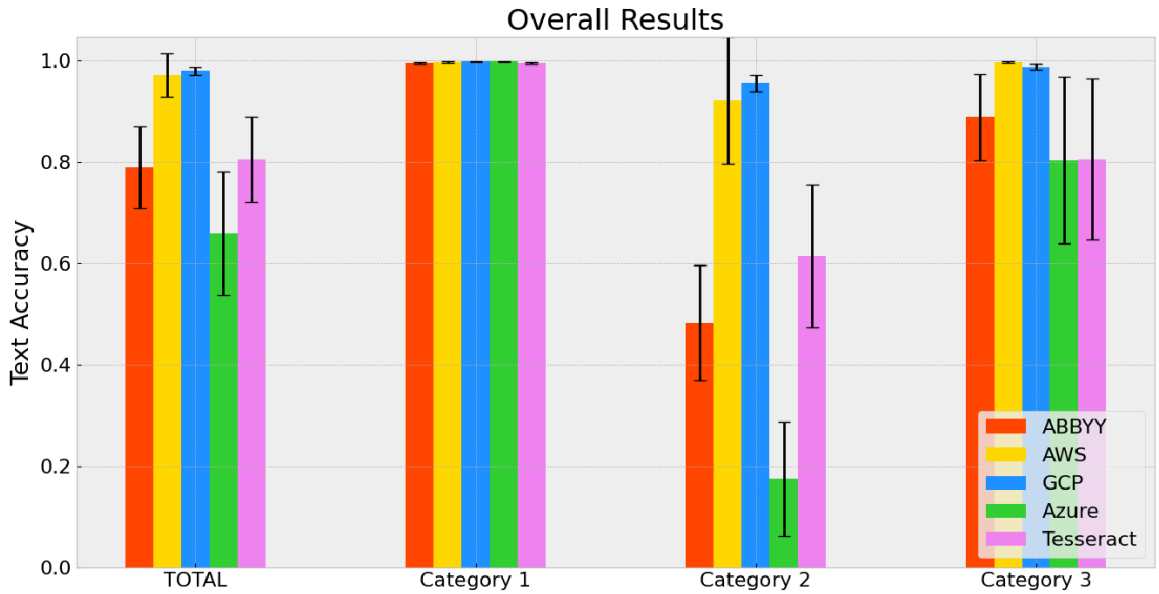


Abbildung 20: Vergleich heutiger Technologien (Dilgemani 2024)

Kategorie 1: Web Page Screenshots, die Text enthalten

Kategorie 2: Handwriting

Kategorie 3: Quittungen, Rechnungen und gescannte Verträge

Die Abbildung zeigt heutige state-of-the-art Technologien zur Optical Character Recognition. Alle Modelle funktionieren perfekt auf Web Page Screenshots. Fotos und Scans von gedruckten Buchstaben funktionieren auch ziemlich gut. Dies ist relativ einfach zu erkennen, da gedruckte Buchstaben immer gleich aussehen. Im Gegensatz zu handgeschriebenen Wörtern. Diese scheinen diese Modelle nicht immer gut zu erkennen. Das Problem ist, wie in der Einleitung beschrieben, dass Handschrift immer anders aussieht, selbst wenn dieselbe Person das Wort zweimal schreibt. Diese Schwierigkeit scheinen heutige state-of-the-art Modelle noch nicht zu 100% lösen zu können, wenngleich GCP und AWS relativ gut dabei sind.

In der Studie wurde zur Berechnung der Ähnlichkeit die Kosinus-Distanz verwendet. Die gebräuchliche Levenshtein-Variante wurde abgelehnt, da sie die Position der Texte mit einbeziehen würde. Da dies bei meinem Projekt keine Rolle spielt, habe ich die Levenshtein-Funktion verwendet. Auch wenn der Vergleich mit zwei unterschiedlichen Varianten stattfindet, würde mein Modell im Vergleich in der Kategorie 2 mit einer Ähnlichkeit von 0.73 im oberen Mittelfeld dieser Technologien spielen. Allerdings sei hier nochmals erwähnt, dass in der Studie eine andere Ähnlichkeitsberechnung genutzt wurde und die dortigen OCR-Technologien auch in die andere Kategorien fallen, während mein Modell lediglich in der Lage ist, Handschrift zu erkennen.

Tesseract

Im Folgenden wird Tesseract von Google verwendet, um dessen Ergebnisse auf meine Testdaten zu überprüfen.

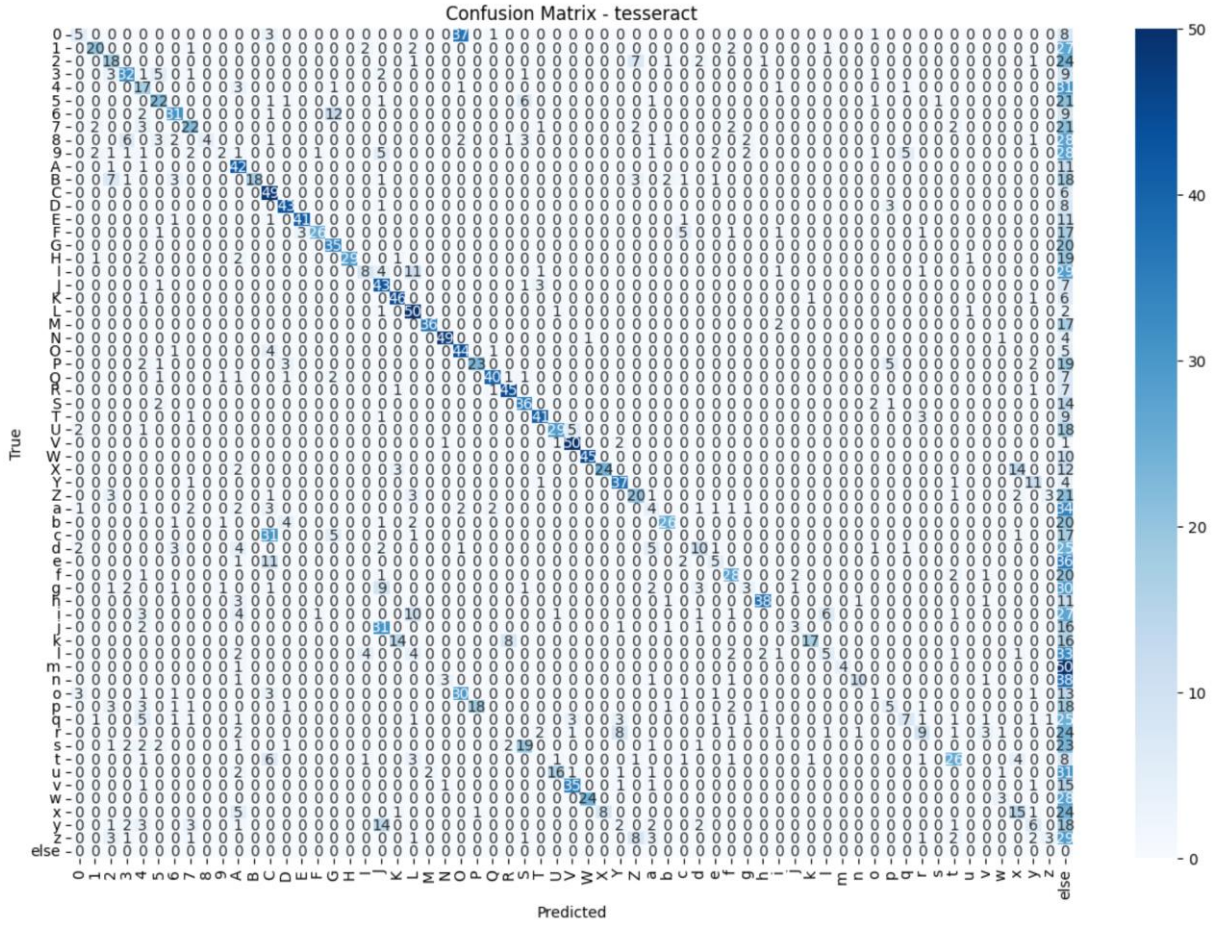


Abbildung 21: Confusion Matrix mit Tesseract

Bei der Konfusionsmatrix von Tesseract auf dieselben Daten ist eine starke Diagonale zu sehen, die auf viele richtig geschätzte Zeichen deutet. Besonders die Zahlen 0 bis 9 und die Großbuchstaben werden richtig erkannt. Kleine Buchstaben werden häufiger als die jeweiligen Großbuchstaben erkannt. Was aber auffällt, ist die letzte Spalte. Eine große Anzahl an Zeichen wurde als etwas komplett anderes erkannt. Teilweise als ein noch komplett unbeachtetes Zeichen wie „%“, „&“ oder „]“, das mein Modell nicht berücksichtigt. Tesseract erkennt teilweise auch mehrere Zeichen, obwohl nur ein Zeichen auf dem Bild zu sehen ist.

	precision	recall	f1-score	support
0	0.38	0.09	0.15	55
1	0.77	0.36	0.49	55
2	0.43	0.33	0.37	55
3	0.68	0.58	0.63	55
4	0.31	0.31	0.31	55
5	0.58	0.40	0.47	55
6	0.67	0.56	0.61	55
7	0.61	0.40	0.48	55
8	1.00	0.07	0.14	55
9	0.40	0.04	0.07	55
A	0.52	0.76	0.62	55
B	1.00	0.33	0.49	55
C	0.42	0.89	0.57	55
D	0.80	0.78	0.79	55
E	0.93	0.75	0.83	55
F	0.93	0.47	0.63	55
G	0.64	0.64	0.64	55
H	1.00	0.53	0.69	55
I	0.53	0.15	0.23	55
J	0.37	0.78	0.50	55
K	0.70	0.84	0.76	55
L	0.56	0.91	0.69	55
M	0.95	0.65	0.77	55
N	0.91	0.89	0.90	55
O	0.38	0.80	0.51	55
P	0.55	0.42	0.47	55
Q	0.89	0.73	0.80	55
R	0.79	0.82	0.80	55
S	0.52	0.65	0.58	55
T	0.84	0.75	0.79	55
U	0.59	0.53	0.56	55
V	0.53	0.91	0.67	55
W	0.64	0.82	0.72	55
X	0.75	0.44	0.55	55
Y	0.67	0.67	0.67	55
Z	0.49	0.36	0.42	55
a	0.17	0.07	0.10	55
b	0.79	0.47	0.59	55
c	0.00	0.00	0.00	55
d	0.48	0.18	0.26	55
e	0.42	0.09	0.15	55
f	0.00	0.00	0.00	0
g	0.67	0.51	0.58	55
h	0.33	0.05	0.09	55
i	0.90	0.69	0.78	55
j	0.00	0.00	0.00	55
k	0.50	0.05	0.10	55
l	0.89	0.31	0.46	55
m	0.38	0.09	0.15	55
n	1.00	0.07	0.14	55
o	0.83	0.18	0.30	55
p	0.12	0.02	0.03	55
q	0.36	0.09	0.14	55
r	0.50	0.13	0.20	55
s	0.50	0.16	0.25	55
t	0.00	0.00	0.00	55
u	0.67	0.47	0.55	55
v	0.00	0.00	0.00	55
w	0.00	0.00	0.00	55
x	0.50	0.05	0.10	55
y	0.41	0.27	0.33	55
z	0.21	0.11	0.14	55
else	0.43	0.05	0.10	55
accuracy			0.40	3410
macro avg	0.55	0.39	0.41	3410
weighted avg	0.56	0.40	0.42	3410

Abbildung 22: Precision, Recall und F1-Score bei Tesseract

Eine Precision von 0.55 zeigt, dass recht wenige Zeichen richtig erkannt wurden. Auch der relativ niedrige Recall von 0.39, zeigt, dass viele Zeichen übersehen wurden. Zwar gibt es auch hier hohe Werte wie bei „D“ oder „K“, jedoch liegen diese Werte insgesamt unter denen meines Modells. Insbesondere spielt das Problem eine Rolle, dass Tesseract oftmals mehrere Zeichen in einem Bild erkennt.

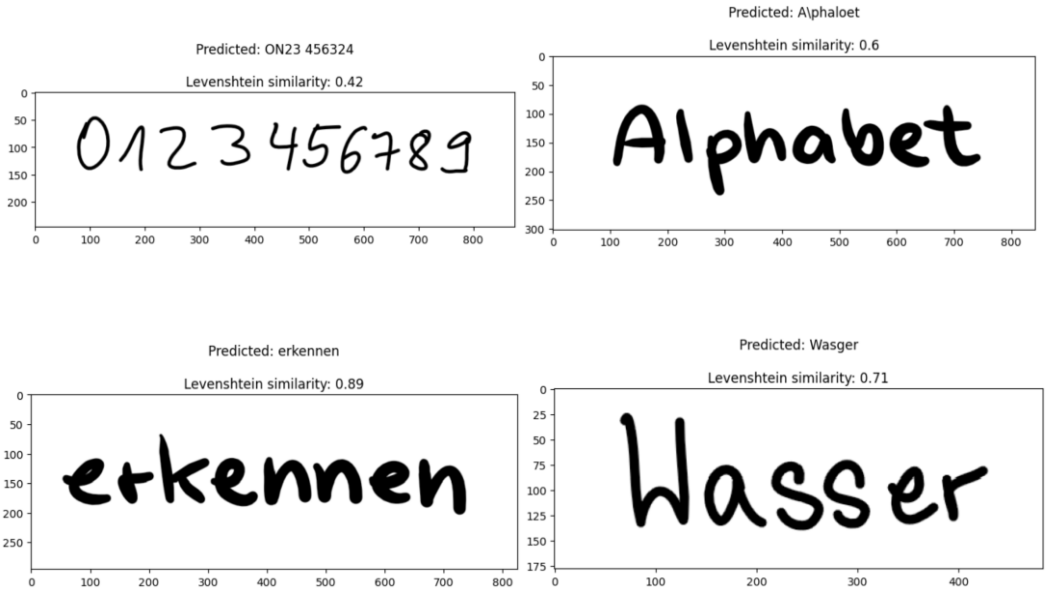


Abbildung 23-26: Erkennung von Wörtern mit Tesseract

In den Beispielwörtern sieht man, dass Tesseract manchmal dieselben Schwierigkeiten hat wie mein Modell („ON“ statt „01“). Doch im Grunde erkennt Tesseract die Wörter auch gut. Scheinbar kann Tesseract besser mit kompletten Wörtern umgehen als mit einzelnen Zeichen.

Variante	Precision	Recall	F1-Score	Levenshtein
Tesseract	0.55	0.39	0.41	0.7
ResNeXt50	0.78	0.75	0.73	0.73

Im direkten Vergleich zwischen beiden Modellen ist zu sehen, dass mein Modell deutlich besser mit einzelnen Buchstaben umgehen kann als Tesseract. Tesseract kommt bei der Erkennung von Zeichenketten schon an mein Modell heran. Jedoch muss gesagt sein, dass mein Modell nur die vortrainierten Zeichen beherrscht und bei weitem noch nicht alle Zeichen. Auch kann mein Modell keine Schreibschrift erkennen.

Fazit

Mein Modell kann einzelne Zeichen recht gut erkennen. Zwar gibt es einige Verwechslungen, doch im Prinzip kann es diese erkennen. Mit passenden Algorithmen, um Buchstaben und Zahlen aus Zeichenketten auszuschneiden, kann das Modell auch komplette Wörter erkennen. Nach der Levenshtein-Distanz sind die vorhergesagten Wörter zu 0.73 ähnlich zu den tatsächlichen Wörtern. Dies ist ein hoher Wert, der sicher noch verbessert werden kann.

Um aber auch Schreibschrift erkennen zu können, wäre ein anderer Ansatz mit Objekterkennung notwendig. Dabei müssten die Buchstaben vom Neuralen Netzwerk in einem Bild automatisch erkannt werden, ohne dass dazu ein selbst geschriebener Algorithmus notwendig ist. Denn mein vorgestelltes Modell beherrscht nur getrennt geschriebene Zeichen, wodurch auch Probleme mit Punkten über Buchstaben entstehen (i, ä ,ö, ü) oder sogar Sonderzeichen.

Es wurde gezeigt, dass das ResNeXt50 unter den vorgestellten Architekturen am besten funktioniert. Hierbei müssen aber auch hochauflösende Bilder trainiert werden, damit die komplette Architektur ausgenutzt werden kann. Es konnten Hinweise darauf gefunden werden, dass das Modell etwas schlechter auf neue Daten generalisieren konnte und mehr die Trainingsbilder lernt. Dennoch konnten relativ gute Ergebnisse erzielt werden.

Letztendlich konnte das Ziel erreicht werden, Wörter zu erkennen. Es wurde ein Ansatz mit einem Convolutional Neural Network vorgestellt, das einzelne Buchstaben erkennt. Teilweise sogar besser als Tesseract. Jedoch ist das Modell beschränkt auf Screenshots von getrennt geschriebenen Zeichen. Also lässt sich sagen, dass mein Modell bei diesem einen bestimmten, aber limitierten Anwendungsfeld funktioniert.

Literatur

Jamshed Memon, Maira Sami, Rizwan Ahmed Khan, and Mueen Uddin (2020): Handwritten Optical Character Recognition (OCR): A Comprehensive Systematic Literature Review (SLR) - online in Internet: https://ieeexplore.ieee.org/document/9151144 [2023-01-12].

Stefan Luber (2022): Was ist die Levenshtein-Distanz?. – online in Internet: https://www.bigdata-insider.de/was-ist-die-levenshtein-distanz-a-502eb7581b6879ee988a93b214821689/ [2024-01-31].

Cem Dilgemani (2024): OCR in 2024: Benchmarking Text Extraction/Capture Accuracy. – online in Internet: https://research.aimultiple.com/ocr-accuracy/ [2024-01-23].