University of Applied Sciences
HOCHSCHULE EMDEN·LEER

Fachbereich Technik
Abteilung Elektrotechnik und Informatik

Bachelorarbeit

Erstellung eines prototypischen Systems zur Extraktion und semantischen Einordnung von Texten aus Rastergrafiken im RGB Farbraum auf Basis neuronaler Netze am Beispiel einer Liste von Lebensmittelinhaltsstoffen

Christian Kitte
(Matrikelnummer xxxxxxx)

Erstprüfer	:	Dipl. Inform. Andreas Wilkens
Zweitprüfer	:	Prof. Dr.-Ing. Lars U. Jänchen
Eingereicht am	:	15. September 2019

E R K L Ä R U N G

Soweit meine Rechte berührt sind, erkläre ich mich einverstanden, dass die vorliegende Arbeit Angehörigen der Hochschule Emden/Leer für Studium / Lehre / Forschung uneingeschränkt zugänglich gemacht werden kann.

E I D E S S T A T T L I C H E V E R S I C H E R U N G

Ich, der Unterzeichnende, erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit selbständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Alle Quellenangaben und Zitate sind richtig und vollständig wiedergegeben und in den jeweiligen Kapiteln und im Literaturverzeichnis wiedergegeben. Die vorliegende Arbeit wurde nicht in dieser oder einer ähnlichen Form ganz oder in Teilen zur Erlangung eines akademischen Abschlussgrades oder einer anderen Prüfungsleistung eingereicht.

Mir ist bekannt, dass falsche Angaben im Zusammenhang mit dieser Erklärung strafrechtlich verfolgt werden können.

_____

Ort, Datum, Unterschrift

Danksagung und Widmung

Als ich im Winter 2014 zum ersten Mal mit den Gedanken spielte, mir einen Traum zu erfüllen und neben meiner Arbeit ein Studium zu beginnen, hätte ich nicht gedacht, dass ich heute, fünf Jahre später, erfolgreich am Ende meines Studiums stehe.

Ich möchte die Gelegenheit nutzen, allen, die meinen Entschluss begrüßten und mich darin bestärkten, zu danken. Allen voran gehört hierzu meine Mutter, welche dies nicht mehr erlebte. Meine Lebensgefährtin dafür, dass sie mir immer den Rücken freihielt und ich mich so ganz auf das Studium konzentrieren konnte sowie meinen Arbeitskollegen und Zimmergenossen bei der Arbeit, der immer wieder half, meine Fehler in der Logik und der Rechtsschreibung zu beheben.

Dank gebührt auch dem OMI Team, welches immer da war, um bei Fragen und Problemen zu helfen, den Dozenten, für Ihre Arbeit und Unterstützung beim Aneignen des Lernstoffes und letztlich den diese Arbeit betreuenden Dozenten für Ihr konstruktives Feedback und den Freiraum bei der Umsetzung.

Themen, die herausfordern mag ich. Es ist nie sicher, wo sie einen hinführen und was man aus ihnen lernen kann. Als ich mir das Thema dieser Arbeit aussuchte, dachte ich, mit meinem aktuellen Wissen gut auszukommen. Dies stellte sich als Irrtum heraus. Vielmehr war es so, dass ich erst mit dieser Arbeit zum ersten Mal den Eindruck hatte, wirklich ein Gefühl für die Materie als Ganzes zu bekommen.

Maschinelles Lernen und künstliche Intelligenz gehören mit Sicherheit zu den herausfordernden Themen, welche einen den Einstieg nicht leicht machen. Es beginnt damit, viel zu lesen und mit jedem Mal, wo man glaubt, etwas verstanden zu haben, gibt es etwas Neues, was man zunächst für sich erschließen und verstanden haben muss, um weiter voran gehen zu können.

Diese Arbeit ist auch der Versuch, hier etwas Hilfe zu leisten und einiges klarer erscheinen zu lassen.

Osnabrück, den 15.09.2019

Abstract

Die vorliegende Arbeit behandelt die Konzeption und Umsetzung eines Systems zur Extraktion von Text aus Rastergrafiken im RGB Farbraum, sowie dessen semantischer Abgleich anhand einer erstellten Datenbasis im JSON-Format. Es wird ein einfaches, prototypisches Framework entwickelt, welches den einfachen Einsatz und Wechsel externer neuronaler Netze für die Textdetektion und -erkennung unterstützt. Der entstandene Code steht, wenn nicht anders angegeben unter der MIT-Lizenz.

Im Rahmen der Umsetzung wird die Einbindung und der Einsatz zweier frei verfügbarer Netze, eines EAST-Modells für die Detektion sowie eines CRNN für die Texterkennung, demonstriert. Sie kann als Vorlage für das Einbinden weiterer Netze genutzt werden. Zum Einsatz kommen hierbei die Frameworks TensorFlow und Keras.

Testbilder, sowie eine darauf basierende, musterhafte Evaluierung des Systems stellen eine Basis für ein eigenes Vorgehen beim Test des Gesamtsystems mit wechselnden Netzen zur Verfügung.

Abstract[1]

This thesis deals with the conception and implementation of a system for the extraction of text from raster graphics in RGB colour space, as well as its semantic comparison based on a created database in JSON format. A simple, prototypical framework is developed, which supports the simple use and change of external neural networks for text detection and recognition. Unless otherwise stated, the resulting code is licensed under the MIT license.

Within the framework of the implementation, the integration and use of two freely available networks, an EAST model for detection and a CRNN for text recognition, will be demonstrated. It can be used as a template for the integration of further networks. The frameworks TensorFlow and Keras are used here.

Test images and a sample evaluation of the system based on them provide a basis for a separate procedure for testing the entire system with changing networks.

[1] Übersetzung durch www.deepl.com

Inhaltsverzeichnis

1 Einleitung

Ein Großteil der weltweit vorhandenen Bilder wurde ohne die Absicht aufgenommen, sie später weiter zu verarbeiten. Dies vor allem, wenn man die zunehmende Zahl an Bewegtbilder hinzunimmt. Diese Daten sind mitnichten nur privater Natur, sondern haben häufig einen ernsten Hintergrund. Als Beispiel seien Röntgenaufnahmen in der Medizin oder die Objekterkennung im Bereich des autonomen Fahrens genannt. Eine besondere Form von möglichen Objekten sind Texte und die darin enthaltenen Informationen.

Während als Text codierte Daten sehr einfach einer maschinellen Auswertung zugeführt werden können, gestaltet sich dies bei Textelementen eines Bildes nicht so einfach. Grundsätzlich ist hier der Text ein Teil des Bildes und in diesem eingebettet. Der Text ist somit nicht als Text codiert, sondern Teil einer Rastergrafik. Er besteht aus einer Menge von nicht verbundenen Pixeln ohne klare Lokalisierung oder Begrenzung.

Um die darin enthaltene Information zu erschließen, muss der eingebettete Text zunächst gefunden und im Anschluss als Text interpretiert werden. Erst im Anschluss daran liegt der ursprüngliche Text als ein als Text codiertes Datum vor und kann der weiteren Verarbeitung zugeführt werden.

Sofern Text in einer definierten Umgebung, wie beispielsweise einer eingescannten Seite auftaucht, existieren einfache und etablierte Verfahren zur Texterkennung (OCR - optical character recognition), welche nicht zwangsläufig auf neuronale Netze aufbauen. Im Kontext aufgenommener Bilder wird hier auch von focused scene text gesprochen, dessen Vorteil die Vorhersehbarkeit der Struktur ist.

Obwohl diese Verfahren in wohldefinierten Umgebungen sehr gute Ergebnisse liefern, haben sie Probleme, wenn es sich um incidental scene text handelt. Dessen Vorkommen ist nicht vorhersehbar und wenig bis gar nicht strukturiert. Hier zeigen sich neuronale Netze als außerordentlich leistungsfähige Lösungen.

In dieser Arbeit soll auf Basis neuronaler Netze ein System umgesetzt werden, welches fähig ist, diese Art von Text zu erkennen, zu extrahieren und mit Hilfe einer Datenbasis einzuordnen.

Bei der Umsetzung dieser Arbeit erwies sich vor allem arXiv[2] als Archiv wissenschaftlicher Schriften als eine gute Quelle, um einen tieferen Einblick in die Materie sowie den aktuellsten Forschungen zu erhalten.

[2] Siehe auch https://arxiv.org der Cornell University. (letzter Abruf am 25.08.2019)

2 Grundlagen

Zum besseren Verständnis dieser Arbeit ist es sinnvoll, sich zunächst mit den dahinter liegenden Grundlagen zu beschäftigen.

Ziel dieses Kapitels ist hierbei nicht die erschöpfende Einführung in die zugrundeliegenden Wissensgebiete. Dies kann uns soll im Rahmen dieser Arbeit nicht geleistet werden. Vielmehr soll ein Gefühl hierfür vermittelt werden.

Nach einer kurzen Einführung in die Systematik der Lebensmittelinhaltsstoffe, werden Rastergrafiken und der RGB-Farbraum eingeführt. Ein kurzer Überblick über die Grundlagen neuronaler Netze schließt sich an. Abgeschlossen wird dies Kapitel mit einer kurzen Beschreibung der verwendeten Programmierumgebung.

2.1 Lebensmittelinhaltsstoffe

Einen umfassenden Überblick über alle derzeit in der Lebensmittelindustrie verwendeten Stoffe zu geben, würde auch im Ansatz den Rahmen und den Fokus dieser Arbeit bei weitem überschreiten. Daher soll hier im Folgenden die grundlegende Systematik skizziert werden.

Die für die Lebensmittelindustrie relevanten Gesetze, Vorschriften und Normen sind zum einen auf EU-Basis, zum anderen auf nationaler Ebene geregelt. Eine besondere Rolle nimmt hierbei die Verordnung (EG) Nr. 178/2002[3] ein. In ihr wird das grundsätzliche und allgemeine Lebensmittelrecht formuliert. Als europäische Verordnung ist sie direkt verbindlich.

„Im Sinne des Artikels 2 der Basis-VO [gemeint ist hier die o.g. Verordnung, d.Verf.] sind ‚Lebensmittel' alle Stoffe oder Erzeugnisse, die dazu bestimmt sind oder von denen nach vernünftigem Ermessen erwartet werden kann, dass sie in verarbeitetem, teilweise verarbeitetem oder unverarbeitetem Zustand von Menschen aufgenommen werden." (Frede 2010, S. 311) Dies schließt somit auch die aus verarbeitungstechnischen Gründen hinzugegebenen Stoffe mit ein.

Laut (Wikipedia-Autoren 15.05.2019) zählen aktuell 361 Stoffe zur Gruppe der Lebensmittelzusatzstoffe, wobei 28 über keine E-Nummer verfügen und 7 nicht mehr zugelassen sind.

[3] Siehe auch https://eur-lex.europa.eu/legal-content/DE/TXT/?uri=CELEX:32002R0178. (letzter Abruf am 15.05.2019)

2.1.1 Überblick über die Systematik verwendeter Stoffe

Nach Frede lassen sich die in Lebensmitteln verwendeten Stoffe grundsätzlich in Inhaltsstoffe, Zusatzstoffe sowie Kontaminanten unterteilen, wie auch die folgende Abbildung 1 auf Seite 3 verdeutlicht.

Zu sehen ist, dass die primären Inhaltsstoffe den für den Stoffwechsel relevanten Stoffen zugordnet werden. Sie „[…] dienen dem Aufbau von Körpersubstanz und der Energiegewinnung.“ (Frede 2010, S. 314) Sie sind somit grundlegend und essenziell für den Menschen. Hierzu gehören die Mineralstoffe, Proteine, Fette, Kohlenhydrate, Vitamine sowie Wasser.

Zu den sekundären Stoffen zählten häufig die für den Genusswert eines Lebensmittels wichtigen Stoffe. Frede differenziert hierbei die Genussmittel und schreibt hierzu sinngemäß, dass bei bestimmten Gruppen der Genusswert überwiege. „Genussmittel stellen somit eine bestimmte Kategorie dar, die nicht der täglichen Nahrungsaufnahme bedürfen.“ (Frede 2010, S. 312)

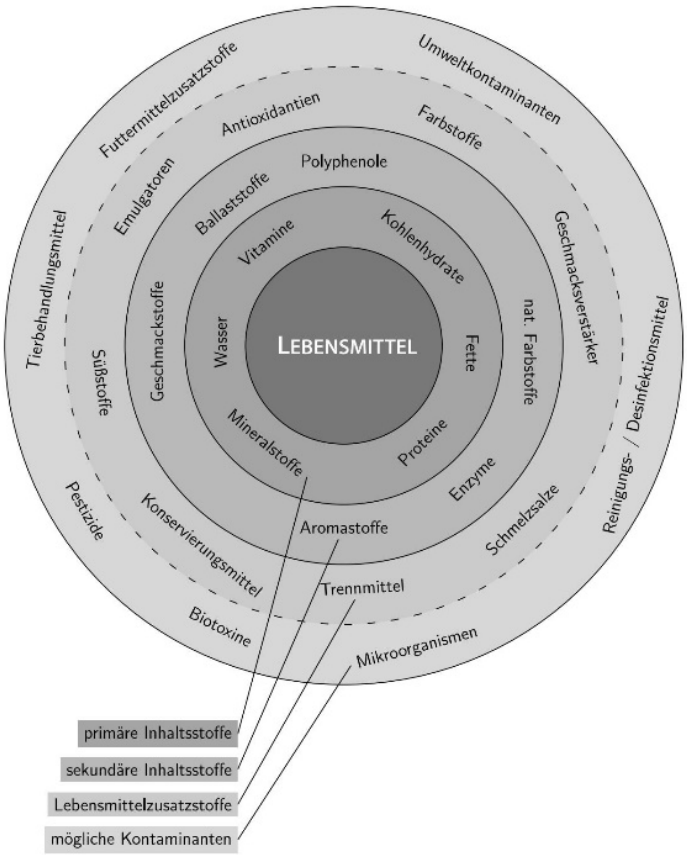

Abbildung 1: Systematik der Inhaltsstoffe in Lebensmitteln (Frede 2010, S. 312)

2.1.1.1 Zusatzstoffe

Von den Inhaltsstoffen zu trennen sind die Zusatzstoffe. Laut (Frede 2010, S. 334) ist ein Zusatzstoff definiert als „[…] ein Stoff mit oder ohne Nährwert, der in der Regel weder selbst als Lebensmittel verzehrt noch als charakteristische Lebensmittelzutat verwendet wird und einem Lebensmittel aus technologischen Gründen bei der Herstellung, Verarbeitung, Zubereitung, Behandlung, Verpackung, Beförderung oder Lagerung zugesetzt wird, wodurch er selbst oder seine Nebenprodukte (mittelbar oder unmittelbar) zu einem Bestandteil des Lebensmittels werden oder werden können." Somit sollen sie die Eigenschaften eines Lebensmittels beeinflussen und werden selbst zu einem Bestandteil.

Im Weiteren führt er aus, dass Ihre Genehmigung von der technischen Notwendigkeit sowie der medizinischen Unbedenklichkeit für den Menschen abhänge. Auch eine Irreführung dürfe nicht stattfinden.

Bis zum Beginn der industriellen Herstellung von Lebensmittel, wurden nur wenige und natürliche Stoffe für die Verarbeitung von Lebensmitteln verwendet. Erst die fabrikmäßige Herstellung großer Mengen von Nahrung führte zum vermehrten und in Teilen notwendigen Einsatz von Zusätzen bei der Verarbeitung. (vergl. hierzu Frede 2010, S. 333)

2.1.1.2 Kontaminanten

Nach (Frede 2010, S. 328) sind „Kontaminanten […] Stoffe, die unbeabsichtigt in ein Lebensmittel gelangen oder dort vorhanden sind und zu einer Kontamination führen." Hierbei grenzt er weitergehend Rückstände als Reste von Stoffen, die im Rahmen der Verarbeitung bewusst hinzugefügt worden sind, gegenüber den Kontaminanten als unerwünschte Stoffe ab.

Hierbei reicht bereits die Möglichkeit für die Entstehung einer Gefahr für den Konsumenten. So schreibt Frede: „Die Stoffe, die absichtlich oder unabsichtlich zu einer Gefahr werden können, werden als ‚mögliche Kontaminanten' bezeichnet." (Frede 2010, S. 314)

Kontaminanten können über die gesamte Verarbeitungskette, angefangen vom Anbau oder der Aufzucht, über die Verarbeitung bis hin zur Lagerung beim Kunden entstehen.

2.2 Rastergrafiken und der RGB Farbraum

Digitale Bilder werden umgangssprachlich häufig einfach anhand Ihres Speicherformates bezeichnet. Wegen dessen weiten Verbreitung ist beispielsweise die Beschreibung eines Bildes als „Bild im JPEG Format"[4] oder einfach nur als „JPEG" ein gebräuchlicher Begriff. Der Farbraum des Bildes bleibt hierbei zumeist unberücksichtigt.

[4] Hierbei handelt es sich um ein verbreitetes Format zur Speicherung von Bildern, das auf eine Spezifikation der Joint Photographic Experts Group (https://jpeg.org) zurück geht. (letzter Abruf am 24.08.2019)

Wesentlich grundlegender und wichtiger ist jedoch Ihr Aufbau in Form einer Matrix. Dieser führt dann auch zu deren Bezeichnung als Rastergrafik. Erst dieser Aufbau in Verbindung mit Farbangaben innerhalb eines definierten Farbraums macht sie für die maschinelle Verarbeitung interessant und verfügbar. Ein weit verbreiteter Farbraum ist hierbei der RGB-Farbraum. Auf beides soll im Folgenden kurz eingegangen werden.

2.2.1 Rastergrafiken

„Um Bilder mit Computersystemen verarbeiten zu können, müssen sie in Datenformate umgesetzt werden, die rechnerkompatibel sind. Diese Umsetzung heißt *Digitalisierung.*“ (Nischwitz et al. 2011, S. 27)

Hierbei wird im Rahmen einer Abtastung ein Bild in Bildpunkte unterteilt und diesen Punkten im Rahmen einer Quantisierung ein oder mehrere Messwerte hinzugefügt. Somit entscheidet die Qualität der zuvor genannten Operationen direkt proportional über die Menge an später verfügbaren Informationen.

Die Bezeichnung Rastergrafik leitet sich hiervon ab. Im Gegensatz zur Vektorgrafik kann eine Rastergrafik als Matrix einzelner Bildpunkte aufgefasst werden. Die Werte der Bildpunkt entsprechen den Werten der Matrix und entsprechen den Intensitäten eines Farbraums oder eines Grauwertes. Ein mit einer Digitalkamera aufgenommenes Bild entspricht aufgrund der Technik und seines Aufbaus einer Rastergrafik.

Das folgende Bild verdeutlicht dies:

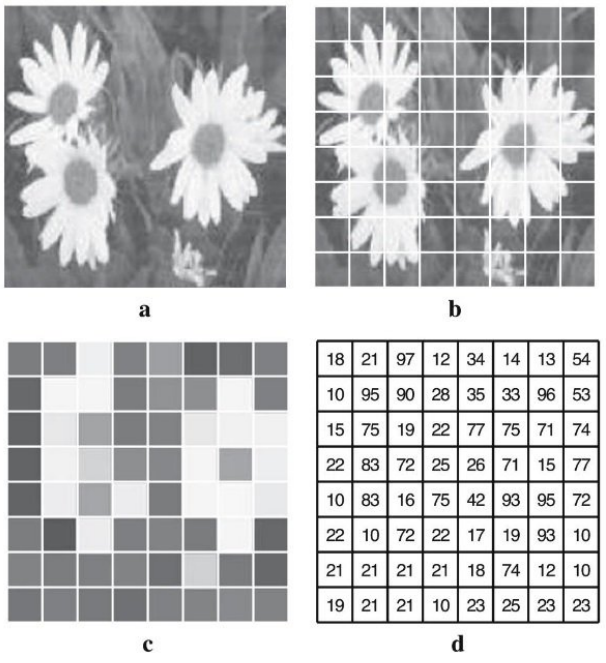

Abbildung 2: Prinzip Digitalisierung (Erhardt 2008, S. 80)

Zu sehen ist ein Bild (a), sowie ein darauf gelegtes Raster (b). Die folgenden Bilder zeigen einen Durchschnittswert (c) sowie dessen Codierung als Matrix (d). Die Werte geben den Mittelwert der Intensität an. Ein anderes Vorgehen könnte für jedes Raster die Werte einzelner Farbanteile wie im folgenden Abschnitt beschrieben hinzufügen. Somit würde man je Bildpunkt drei Werte erhalten.

Um ein Rasterbild anzeigen zu können, müssen die zu den einzelnen Pixeln zugordneten Werte ausgelesen und ihrer Bedeutung entsprechend zur Anzeige gebracht werden.

2.2.2 Der RGB-Farbraum und Graustufen

Der RGB-Farbraum begründet sich in der Dreifarbentheorie. Laut (Schiele 2012, S. 29) beschreibt sie „[…] einen Farbraum mit drei Grundfarben, den Primärvalenzen, die den Ecken eines Dreiecks zugeordnet sind […]. Alle anderen Farben werden als Linearkombination aus den Grundfarben bestimmt."

Dies verdeutlicht die folgende Darstellung, in welcher jede RGB-Farbe anhand dreier Ortsvektoren dargestellt werden kann:

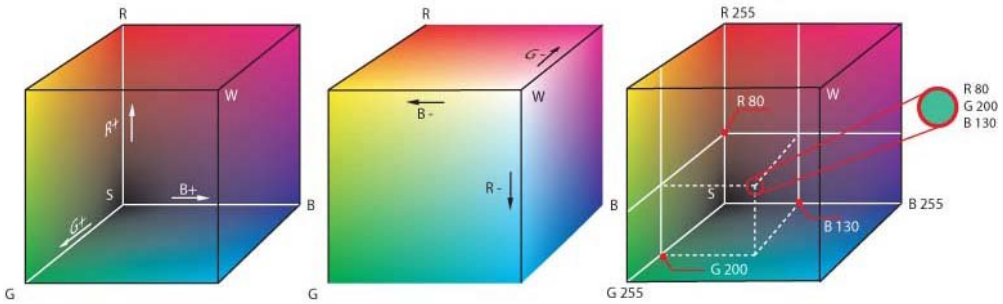

Abbildung 3: RGB – Farbwürfel (Wikipedia-Autoren 14.11.2004)

Handelt es sich um ein Bild im RGB-Farbraum, entfallen somit auf jeden Punkt drei Werte, die den Koeffizienten dieser Linearkombination entsprechen. Somit kann eine Rastergrafik im RGB-Farbraum als eine dreidimensionale Matrix aufgefasst werden. In diesem Zusammenhang wird auch von einem R-, G- und B-Kanal des Bildes gesprochen.

Bei RGB handelt es sich um ein additives Farbmodell. Werden alle Farben in Ihrer höchsten Intensität verwendet, so ist das Ergebnis, wie auch oben zu sehen, immer weiß. Sind hingegen alle Farbanteile gleich stark vertreten, so ist das Ergebnis ein Grau mit einer von der Intensität abhängenden Helligkeit.

Wird auf die Darstellung von Farben zu Gunsten von Graustufen verzichtet, so reicht das Vorhalten nur eines Wertes, welcher der Helligkeit eines Punktes entspricht. In diesem Fall kann die Rastergrafik als eine zweidimensionale Matrix aufgefasst werden.

Unabhängig von der ursprünglichen Qualität der Quantisierung bestimmt die Größe des verfügbaren Zahlenbereiches die Darstellung. So können mit 8-Bit lediglich 255 unterschiedliche Werte dargestellt werden.

2.2.3 Faltung von Bildern

Die Faltung von Bildern wird häufig als deren Filterung bezeichnet mit dem Ziel, Eigenschaften wie Schärfe oder Weichheit zu beeinflussen. Daneben können mit diesem Verfahren auch weitere Aspekte als Ziel dienen. Als Beispiel sei hier die Kantendetektion genannt.

Im der folgenden Abbildung 4 wird das prinzipielle Vorgehen bei der Faltung verdeutlicht. Ein Filterkern wird von links nach rechts und von oben nach unten über die Pixel des Eingangsbildes geführt. Anhand der Werte des Filterkerns bzw. der Faltungsmaske sowie der unter dem Kern liegenden Pixel des Eingangsbildes entstehen als Ergebnis neue Werte. Hierbei sind bei der Faltung für die Bildbearbeitung die Werte für einen Filtervorgang fest vorgegeben und ändern sich in dessen Verlauf nicht.

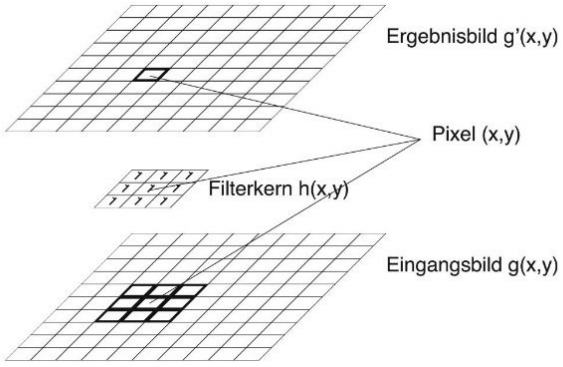

Abbildung 4: Prinzip der Faltung (Erhardt 2008, S. 154)

Da bei dieser Operation nicht nur ein isolierter Bildpunkt betrachtet wird, spricht man von einer Operation im Ortsraum.

2.3 Neuronale Netze

Das im Rahmen dieser Arbeit vorgestellte System baut auf der Verwendung neuronaler Netze auf. Daher sollen hier einige grundlegende Aspekte und Methoden kurz erläutert werden. Eine erschöpfende Auskunft kann und soll jedoch im Rahmen dieser Arbeit auch hier nicht geleistet werden.

Auf weiterführende Aspekte, die erst im Laufe der Bearbeitung an Bedeutung gewinnen, wird im weiteren Verlauf an geeigneter Stelle kurz eingegangen.

2.3.1 Allgemeiner Aufbau

Die Bezeichnung Neuronale Netze rührt aus der Ähnlichkeit zum Aufbau eines biologischen Gehirns her. Im Kern bestehen sie aus einer Eingabe- sowie einer Ausgabeschicht (engl. input, output layer). Dazwischen existiert mindestens eine nicht direkt zugängliche, versteckte Schicht (engl. hidden layer). Handelt es sich um mehr als eine versteckte Schicht, so wird von einem deep learning neural net gesprochen. Daneben existieren Netzstrukturen ohne verstreckte Schichten, auf welche hier nicht näher eingegangen werden soll.

Die Schichten (engl. Layer) bestehen aus einzelnen Knoten (Neuronen, engl. perceptrons), welche über ihre gewichteten Eingänge mit den Knoten der vorhergehenden Schichten verbunden sind. Über ihre Ausgabe sind sie mit den Knoten der folgenden Schicht verbunden.

Ausgehend von einem Eingangswert berechnet ein neuronales Netz einen Ausgabewert. Erfolgt die Signalweitergabe hierbei nur in Richtung der Ausgabeschicht, so spricht man von einem feedforward neural network (FNN). Erfolgt die Weitergabe in beide Richtungen, so handelt es sich um ein recurrent neural network (RNN). Kommt es im Rahmen der Verarbeitung zu einer Faltung, handelt es sich um ein convolutional neural network (CNN).

2.3.2 Aktivierung – Verlustfunktion - Optimierung

Für die Entwicklung eines Modells sind mindestens drei wichtige Entscheidungen zu treffen. Auf der Ebene der Layer ist dies die Art der Aktivierung seiner Neuronen. Auf der Ebene des Modells die Verlustfunktion sowie die zu verwendende Optimierungsfunktion.

2.3.2.1 Aktivierungsfunktion

Die Aktivierungsfunktion ist eine dem Neuron vorgeschaltete Funktion, welche über der gewichteten Summe der anliegenden Signale angewendet wird.

$$x_i = f\left(\sum_{j=1}^{n} w_{ij} x_j\right)$$

Formel 1: Aktivierungsfunktion (Ertel 2016, S. 269)

Im einfachsten Fall handelt es sich um eine lineare Funktion, bei der die Ausgabe proportional zur Eingabe ist.

$$f(x) = x$$

Formel 2: lineare Funktion (Ertel 2016, S. 269)

Dies kann laut (Ertel 2016, S. 269) problematisch sein, da auf Grund einer fehlenden Beschränkung die Funktionswerte im Laufe der Zeit über alle Grenzen wachsen können.

Eine andere Möglichkeit ist der Einsatz einer Stufenfunktion, welche den Ausgabewert zwischen zwei möglichen Werten beschränkt und somit besonders für binäre Entscheidungen sinnvoll ist. Als Schwelle dient hierbei der vorgegebene Wert θ.

$$x_i = \begin{cases} 0 & \text{falls } \sum_{j=1}^{n} w_{ij} x_j < \Theta \\ 1 & \text{sonst} \end{cases}$$

Formel 3: Schwellwertfunktion (Ertel 2016, S. 269)

Sinngemäß führt laut (Ertel 2016, S. 269f.) jedoch der Einsatz einer Schwellwertfunktion bei stetigen Funktionen in einem nicht binären Kontext zu einer nicht gewollten Unstetigkeit der Ausgabe. Dem kann durch den Einsatz einer Sigmoidfunktion entgegengewirkt werden. Diese verhält sich um den Schwellenwert herum fast linear, nähert sich jedoch asymptotisch für sehr kleine Werte 0 und für große Werte 1. Über den Parameter T lässt sich die so erreichte Glättung variieren.

$$f(x) = \frac{1}{1 + e^{-\frac{x-\Theta}{T}}}$$

Formel 4: Sigmoidfunktion (Ertel 2016, S. 270)

Neben den hier vorgestellten Grundformen existieren eine Reihe weiterer Aktivierungsfunktionen, welche sich jedoch zumeist auf die hier angesprochenen Grundformen zurückführen lassen. Eine davon ist der lineare Gleichrichter (Rectified Nonlinear Unit, ReLU). Nach (Alice Zheng 2019, S. 150) ist ein „[…] linearer Gleichrichter […] eine einfache Abwandlung einer linearen Funktion, wobei der negative Teil zu null gesetzt wird."

$$\mathrm{ReLU}(x) = \max(0, x)$$

Formel 5: Lineare Gleichrichter (Rectified Nonlinear Unit, ReLU) (Alice Zheng 2019, S. 151)

2.3.2.2 Verlustfunktion

Im Kontext eines neuronalen Netzes ist der auftretende Verlust der Unterschied zwischen dem errechneten und dem im Rahmen eines Trainings vorgegebenen Wertes. Je

kleiner dieser Verlust ist, umso besser ist die Genauigkeit des verwendeten Modells. Dies bedeutet, dass die Verlustfunktion als solche nicht zwingend einer Gruppe vorgegeben Funktionen entnommen werden muss, sondern auch manuell und einzelfallorientiert definiert werden kann.

Laut (Hope et al. 2018, S. 41) gibt es „[…] keine ideale Verlustfunktion, und die Wahl der geeignetsten ist oft ebenso sehr eine Kunst wie eine Wissenschaft." Trotz dessen, so wird im Folgenden sinngemäß weiter ausgeführt, sind der mittlere quadratische Fehler (engl. mean squared error, MSE) und die Kreuzentropie die aktuell gebräuchlichsten Verfahren.

Im Fall von MSE kann dies an der natürlichen Nachvollziehbarkeit liegen. So betont (Hope et al. 2018, S. 42), dass sie über eine „[…] intuitive Interpretation [verfüge, d.Verf.] – sie minimiert den mittleren quadratischen Abstand zwischen einem Messwert und dem vom Modell vorhergesagten Wert."

$$L(y, \hat{y}) = \frac{1}{n}\Sigma_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Formel 6: Mittlere Quadratische Fehler (mean squared error, MSE) (Hope et al. 2018, S. 42)

Ohne an dieser Stelle auf die konkrete Herleitung einzugehen, wird bei „[..] der Kreuzvalidierung (engl. cross validation) versucht […], die Modellkomplexität so zu optimieren, dass der Klassifikations- oder Approximationsfehler auf einer beim Lernen unbekannten Testdatenmenge minimal wird.„ (Ertel 2016, S. 232)

$$H(p,q) = -\Sigma_x p(x) \log q(x)$$

Formel 7: Kreuzentropie (cross validation) (Hope et al. 2018, S. 42)

2.3.2.3 Optimierung

Als letzte wichtige Entscheidung soll an dieser Stelle auf die Optimierung eingegangen werden. Im Rahmen der Optimierung wird versucht, den auftretenden Verlust möglichst effektiv zu begrenzen. Dies geschieht in der Regel durch ein Anpassen der Gewichte an den Eingängen der Neuronen.

Das am häufigsten zum Einsatz kommende Verfahren ist hierbei nach (Hope et al. 2018, S. 43) das Gradientenabstiegsverfahren (engl. gradient descent)[5], bei dem letztlich ein

[5] Unter https://data-science-blog.com/blog/2019/01/13/training-eines-neurons-mit-dem-gradientenverfahren findet sich hierzu ein guter Einstieg. (letzter Abruf am 24.08.2019)

lokales Minimum einer multivariaten Verlustfunktion ermittelt wird. Hierbei entsprechen die Variablen der zugrunde liegenden Funktion den Gewichten des Modells.

„Die Gradientenabstiegs-Algorithmen funktionieren gut für hochkomplizierte Netzarchitekturen und eignen sich daher für eine Vielzahl verschiedener Probleme. Insbesondere ist es neuerdings möglich, diese Gradienten mithilfe massiv paralleler Systeme zu berechnen, sodass der Ansatz gut mit der Anzahl von Dimensionen skaliert." (Hope et al. 2018, S. 43)

Neben der Berechnung aller Trainingsdaten innerhalb eines kompletten Durchlaufs (engl. epoche) ist eine weit verbreitete Technik, zufällig gezogene Stichproben (Teildatensätze, batches) zu verwenden und diese stattdessen zu berechnen. Die Vorteile sind hierbei eine beschleunigte Berechnung sowie die grundsätzliche Möglichkeit, auf diese Art bessere lokale Minima zu finden. Dies Vorgehen wird nach (Hope et al. 2018, S. 43) auch als „[…] stochastische[r, d.Verf.] Gradientenabstieg (SGA) […]" bezeichnet.

In der Praxis existieren darüber hinaus weitere Verfahren[6], welche zum Teil schneller, zum Teil genauer arbeiten. Diese hier im Detail aufzuführen würde jedoch den Umfang dieser Arbeit überschreiten.

Das Training eines neuronalen Netzes kann somit als eine Abfolge dauernder Optimierung durch die Ermittlung des Verlustes und der Anpassung von Gewichten aufgefasst werden. Hierbei muss darauf geachtet werden, dass es in diesem Verlauf nicht zu einem Overfitting kommt, bei dem die verwendeten Testdaten quasi auswendig gelernt werden, die Genauigkeit bei unbekannten Daten sich jedoch verschlechtert.

2.3.3 Backpropagation

Für das Training neuronaler Netze ist zumeist eine Backpropagation notwendig. Hierbei handelt es sich um ein Verfahren, bei dem der am Ausgang in Form eines Verlustes anliegende Fehler auf die Knoten der vorhergehenden Schichten abgebildet – backpropagiert – wird.[7]

Wie in der folgenden Abbildung 5: Prinzip Backpropagation auf Seite 12 skizziert, wird hierzu der am Ausgang anliegende Fehler anteilig zu den aktuellen Gewichten rekursiv auf die vorliegenden Schichten zurückgeführt. Der Fehler eines einzelnen Knotens ergibt sich somit anhand der seinen Ausgängen zugeordneten Einzelfehler.

[6] Unter https://github.com/wassname/viz_torch_optim findet sich ein animierter Vergleich verschiedener Verfahren zur Optimierung. (letzter Abruf am 24.08.2019)

[7] In den Abschnitten 2.3.4.5, Seite 15 sowie 2.3.4.6, Seite 16 wird hierauf bei der Behandlung rekurrenter Netze sowie von LSTMs vertiefend eingegangen.

In der Abbildung dies am Beispiel des Neurons $e_{hidden\,3}$ (Mitte oben) zu sehen, der anteilig zu den Gewichten w_{11} und w_{12} verändert wird.

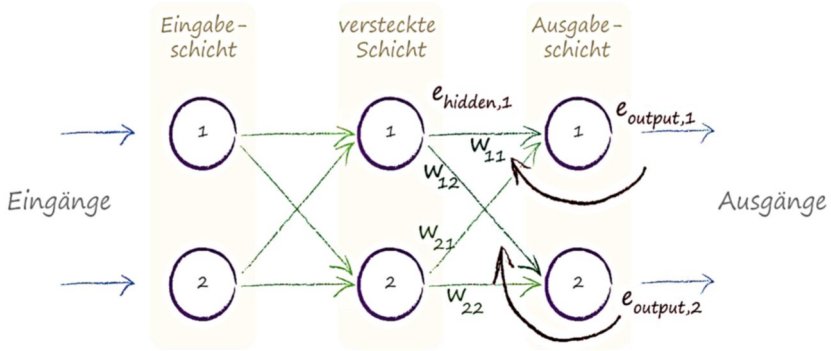

Abbildung 5: Prinzip Backpropagation (Rashid 2017, S. 65)

Der so ermittelte Fehler in Bezug zu einem Knoten kann im Folgenden genutzt werden, um dessen aktuelles Gewicht neu zu justieren.

Somit teilt sich der Backpropagation Algorithmus in die drei Stufen Berechnung eines Ausgabewertes anhand der aktuellen Werte, Ermittlung der Fehler durch Rückführung und abschließend die Anpassung der verwendeten Gewichte.

Neben dem hier vorgestellten Verfahren kommen beim maschinellen Lernen auch weitere Verfahren zum Einsatz, auf welche hier jedoch nicht weiter eingegangen wird.

2.3.4 Häufig vorkommende Layer

Im Folgenden soll kurz auf einige wichtige Arten von Layern sowie deren Bedeutung im Zusammenspiel mit einem neuronalen Netz eingegangen werden.

2.3.4.1 Fully Connected Layer (Dense Layer)

Eine Schicht innerhalb eines neuronalen Netzes, bei der alle Knoten der Schicht mit allen Knoten der vorhergehenden Schicht verbunden sind und umgekehrt, wird als ein fully connected layer oder auch dense layer bezeichnet. Somit handelt es sich hierbei um eine sehr grundlegende und einfache Struktur.

2.3.4.2 Dropout Layer

Dropout bezeichnet in Verbindung mit dem Training neuronaler Netze ein Verfahren, bei dem im Training zufallsgesteuert und wechselnd einzelne Eingänge auf null gesetzt werden.

Dies hat zur Folge, dass ein Modell eine Lösung finden muss, bei der einzelne und vielleicht dominierende Eingaben gerade nicht immer vorhanden sind. Diese so trainierten

Modelle kommen mit den in realen Szenarien immer vorkommenden Variationen besser zurecht.

Dropouts werden in Keras als eigene Schicht innerhalb eines Modells umgesetzt.

2.3.4.3 Convolutional Layer

In einem convolutional layer wird nach (Dörn 2018, S. 130) „[…] die Eingabe der vorherigen Schicht mit einem Faltungskern gefaltet. Die Werte des Faltungskerns sind die zu lernbaren [sic] Parameter des Netzes."

Der technische Ablauf ist hierbei analog zur Faltung einer Rastergrafik und soll hier nochmals beispielhaft anhand der folgenden Abbildung dargestellt werden:

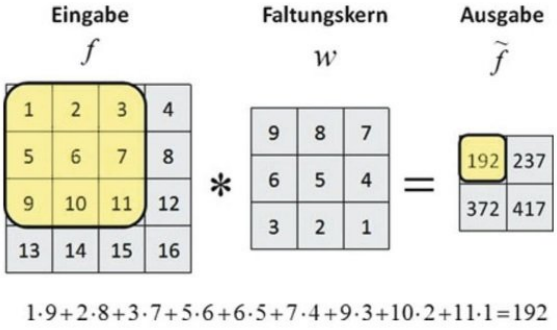

Abbildung 6: Beispiel Faltung (Dörn 2018, S. 130)

In der Abbildung ist zu sehen, wie die ursprüngliche Matrix (links) anhand des Filters (Mitte) zu einer $2x2$ Matrix (rechts) reduziert wird. Hierbei wird der Ausgabewert anhand der folgenden Formel bestimmt:

$$\widetilde{f}(x,y) = \sum_{i=1}^{r}\sum_{j=1}^{r} f(x+i, y+j)\cdot w(i,j), \quad 1 \le x, y \le o.$$

Formel 8: Faltung (Dörn 2018, S. 130)

Im Vergleich zur Anwendung bei Rastergrafiken ergeben sich zwei Unterschiede. Zum einen sind die Werte des Filterkerns nicht statisch, sondern entsprechen gerade den anzulernenden Gewichten, zum anderen kann für eine Schicht mehr als ein Filterkern definiert werden.

„Die Ausgabebilder der Faltungsschicht bestimmen sich dann als die Summe der einzelnen Faltungsbilder. Diese Ausgabebilder werden auch als Merkmalskarten (Feature Maps) bezeichnet.“ (Dörn 2018, S. 130)[8]

Dem Modell wird so die Intuition eines Bildes, zum Beispiel seiner Kanten, gegeben. Hierbei kann sich beispielsweise ein Filterkern auf horizontale, ein anderer auf vertikale Linien „spezialisieren“.

Häufig kommt eine Kombination von convolutional, pooling (siehe Abschnitt 2.3.4.4) sowie einen abschließenden dense layer, wie in der folgenden Abbildung 7 beispielhaft zu sehen ist, zum Einsatz:

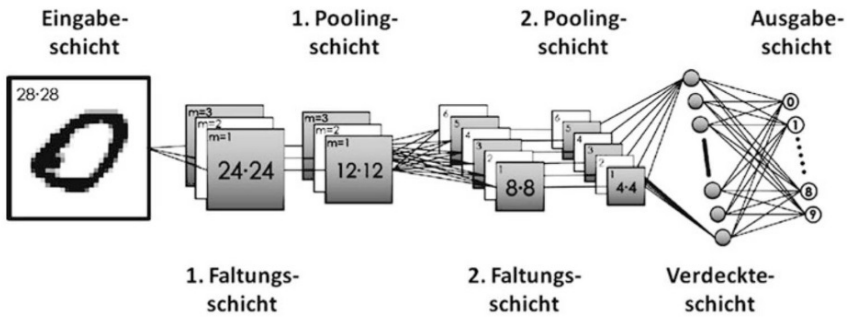

Abbildung 7: CNN Beispiel (Dörn 2018, S. 132)

Man erkennt das von links nach rechts aus zwei Pooling Layern und einem Dense Layer aufgebaute Netz. Hierbei werden die resultierenden Feature Maps kleiner, deren Dimension jedoch größer.

2.3.4.4 Pooling Layer

„Üblicherweise wendet man auf die Ausgabedaten von Konvolutionsschichten ein Pooling an. Technisch wird beim *Pooling* die Datenmenge durch eine lokale Aggregationsfunktion verringert, meist innerhalb einer Merkmalskarte.“ (Hope et al. 2018, S. 56)

Weiter schreibt (Hope et al. 2018, S. 56), dass ein „[…] eher theoretisches Argument für den Einsatz von Pooling ist, dass die berechneten Merkmale unempfindlich gegenüber kleinen Verschiebungen des Bilds sein sollten.“

Die Grundidee ist somit, dem Netz eine Intuition darüber zu vermitteln, dass bestimmte Feature variieren können. So ist die Pixelgenaue Position einer Kante in der Regel weniger wichtig als das Vorhandensein der Kante als solche.

[8] Unter http://scs.ryerson.ca/~aharley/vis findet sich hierzu eine interessante Animation zur Arbeitsweise von Faltungsnetzen. (letzter Abruf am 24.08.2019)

Zwei verbreitete Formen sind das Mean-Pooling und das Max-Pooling. Das folgende Bild zeigt beispielhaft das Prinzip dieses Layer anhand eines Max-Pooling, bei dem aus einem definierten Bereich das jeweilige Maximum übernommen wird.

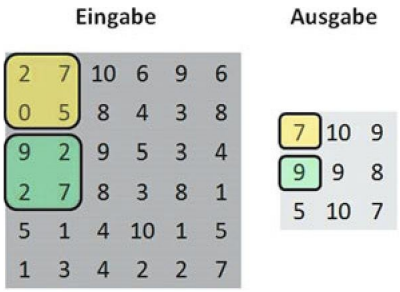

Abbildung 8: Prinzip Max-Pooling (Dörn 2018, S. 131)

Deutlich ist die durch das Pooling durchgeführte Reduzierung der Daten (rechts) gegenüber der zugrunde liegenden Eingabeschicht (links) zu erkennen. Gleichzeitig stellen immer weniger Neuronen die essenziellen Features des Eingabebildes dar.

Bei der Verwendung eines Mean-Pooling würde entsprechend statt des maximalen Wertes der jeweilige Durchschnitt verwendet.

2.3.4.5 Recurrent neural network (RNN)

Rückgekoppelte oder rekurrente Netze wurden bereits kurz zu Beginn dieses Abschnitts angesprochen und sollen auf Grund ihrer Bedeutung hier nochmals aufgegriffen werden.

Bei dieser Art von Netzen verfügt der Knoten neben dem Wert zu einem Zeitpunkt t auch über den Wert zum Zeitpunkt $t-1$. Dieser Zweite Wert wird bei der Abarbeitung berücksichtigt und verleiht dem Netz eine Intuition über einen zeitlichen Ablauf. Das folgende Bild verdeutlicht die hierbei möglichen Rückkopplungen:

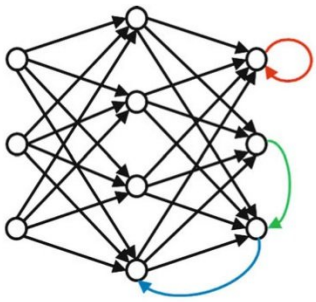

Abbildung 9: Prinzip RNN (Dörn 2018, S. 127)
Rückkopplungen können in direkte (rot), indirekte (blau) sowie seitliche Rückkopplungen (grün) unterschieden werden.

(Dörn 2018, S. 126) beschreibt dies wie folgt: „Mit diesen Rückkopplungen kann ein Netz mit einem dynamischen Verhalten und einem Gedächtnis ausgestattet werden.

Rekurrente Netze liefern in der Praxis sehr gute Resultate zur Erkennung von Sequenzmustern, wie beispielsweise bei der Handschrift oder Spracherkennung.“

2.3.4.6 Long Short-Term Memory Units (LSTM)

Um der ungewollten Beeinflussung der Fehler bei rekurrenten Netzen zu entgehen[9], setzt man intelligente Knoten ein, welche Werte direkt speichern, aber auch vergessen können. Der auftretende Fehler wird also nicht mehr einfach nur unüberwacht zurück propagiert.

(Hope et al. 2018, S. 89) schreibt hierzu, dass sich LSTM „[…] von einfachen RNNs darin [unterscheidet, d.Verf.], dass es über spezielle *Gedächtnismechanismen* verfügt, mit denen die rekurrenten Zellen Information besser über längere Zeiträume speichern und dadurch langfristige Abhängigkeiten besser als einfache RNNs abbilden können.“

Die folgende Abbildung zeigt den schematischen Aufbau einer solchen Einheit:

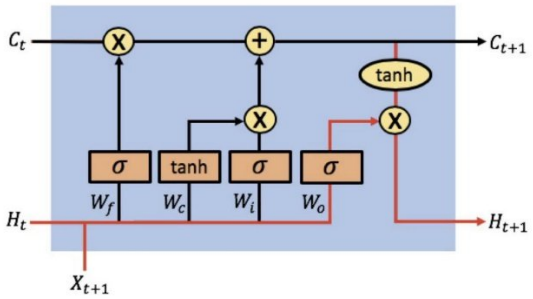

Abbildung 10: Zelle eines LSTM zur Zeit t (Arbel 2018)

In der obigen Abbildung führen die Ein- und Ausgänge C (oben links und rechts) das anliegende Signal. Die Signale an H und X (unten links und rechts) bestimmen das Verhalten des LSTM.

Hierzu schreibt (Arbel 2018): „The LSTM has three gates that update and control the cell states, these are the forget gate, input gate and output gate. The gates use hyperbolic tangent and sigmoid activation functions. […] The forget gate controls what information in the cell state to forget […] ”.[10]

2.3.5 Exploding / Vanashing - Gradient-Problem

Ein Problem nicht nur rekurrenter Netze ist die ungewollte starke Vergrößerung (engl. exploding) bzw. Verkleinerung (engl. vanashing) eines Fehlers bei der Backpropagation

[9] Siehe hierzu 2.3.5 - Exploding / Vanashing - Gradient-Problem, Seite 16

[10] Die LSTM verfügt über drei Gates, die die Zellzustände aktualisieren und steuern, dies sind das Vergessensgate, das Eingangsgate und das Ausgangsgate. Die Gates verwenden hyperbolische Tangenten- und Sigmoid-Aktivierungsfunktionen. […] Das Vergessensgate steuert, welche Informationen im Zellenzustand zu vergessen sind […]. (Übersetzung durch www.deepl.com)

entweder auf vorgelagerte Knoten oder aber auf frühere Zeitabschnitte des Knotens im Falle rückgekoppelter Netze. Ursächlich hierfür ist die wiederholte Multiplikation sehr großer oder sehr kleiner Werte.

Im ersten Fall schreibt (Alese 2018) hierzu: „A solution to fix this is to apply gradient clipping; which places a predefined threshold on the gradients to prevent it from getting too large, and by doing this it doesn't change the direction of the gradients it only change its length."[11]

Im Fall des vanashing gradient bietet sich laut (Wang 2019) als einfachste Lösung der Einsatz von ReLU[12] als Aktivierungsfunktion an: „The simplest solution is to use other activation functions, such as ReLU, which doesn't cause a small derivative."[13]

Bei rekurrenten Netzen kann das Problem mit Hilfe der im Abschnitt 2.3.4.6 besprochenen Long Short-Term Memory Units umgangen werden.

2.4 Entwicklungsumgebung

Um die in dieser Arbeit gegebene Aufgabenstellung umsetzen zu können, bedarf es einer hierzu geeigneten Entwicklungsumgebung. Kenntnisse in der verwendeten Programmiersprache Python sowie gängiger Bibliotheken werden hierbei vorausgesetzt.

Im Folgenden sollen daher das Frameworks TensorFlow[14] als zentrale Komponenten sowie dessen präferierte Sprache Keras[15] kurz erläutert werden.

2.4.1 TensorFlow

Nimmt man die in (Hope et al. 2018, S. 6) gegebene Umschreibung, so ist „TensorFlow […] ein Software-Framework für numerische Berechnungen, die auf Datenflussgraphen beruhen. Es ist aber vor allem als Schnittstelle zum Beschreiben und Implementieren maschineller Lernalgorithmen gedacht, vor allem von Deep-Learning-Netzen." Neben TensorFlow existieren hierbei weitere Frameworks, mit deren Hilfe ebenso neuronale Netze entwickelt, trainiert und weitergegeben werden können.

Abbildung 11: Symbol TensorFlow (TensorFlow)

[11] Eine Lösung, um dies zu beheben, ist das Anwenden von Gradienten-Clipping, das einen vordefinierten Schwellenwert auf die Gradienten setzt, um zu verhindern, dass sie zu groß werden, und dadurch die Richtung der Gradienten nicht ändert, sondern nur ihre Länge. (Übersetzung durch www.deepl.com)
[12] siehe 2.3.2.1 - Aktivierungsfunktion, Seite 8
[13] Die einfachste Lösung ist die Verwendung anderer Aktivierungsfunktionen, wie z.B. ReLU, die keine kleine Ableitung verursacht. (Übersetzung durch www.deepl.com)
[14] Die Seite von TensorFlow findet sich unter https://www.tensorflow.org. (letzter Abruf am 26.08.2019)
[15] Die Seite von Keras findet sich unter https://keras.io. (letzter Abruf am 26.08.2019)

Die Verwendung von Graphen zur Abbildung und Umsetzung hat den großen Vorteil, dass vor der eigentlichen Ausführung alle Rechenschritte bekannt sind und somit die Möglichkeit einer Optimierung gegeben ist.

Der Name TensorFlow leitet sich hierbei von den Tensoren ab, welche im Laufe der Verarbeitung bildlich gesprochen durch den Berechnungsgraphen hindurchfließen, wie in der folgenden Abbildung verdeutlicht wird.

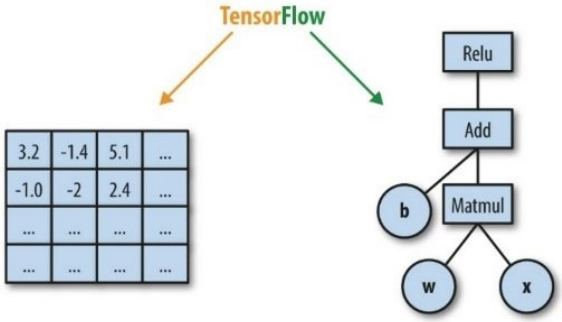

Abbildung 12: Prinzip TensorFlow (Hope et al. 2018, S. 5)

Laut (Hope et al. 2018, S. 6) ist hierbei die „[…] Kernkomponente von TensorFlow […] in C++ geschrieben und besitzt zwei primäre Schnittstellen zu Hochsprachen, in denen sich Berechnungsgraphen beschreiben und ausführen lassen." Weiter wird im Folgenden sinngemäß ausgeführt, dass die umfangreichste Schnittstelle in Python geschrieben sei, die in C++ geschriebene jedoch vor allem einen Zugriff auf niederer Ebene anbiete.

Daneben verfügt TensorFlow nach eigenen Worten neben Python über ein breites System an Unterstützung für verschiedene Umgebungen wie JavaScript, Swift oder iOS. (vergl. hierzu TensorFlow).

Aktuell[16] ist der Wechsel hin zur Version 2 des Frameworks weit fortgeschritten. In dessen Verlauf werden die Bibliotheken stark überarbeitet, was als Folge eine Anpassung von Anwendungen notwendig machen wird.

2.4.2 Keras

Keras beschreibt sich selbst als „[…] a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation."[17] (Keras 11.02.2019)

[16] Mai 2019

[17] …eine High-Level-API für neuronale Netzwerke, die in Python geschrieben wurde und auf Tensor-Flow, CNTK oder Theano laufen kann. Es wurde mit dem Ziel entwickelt, schnelle Experimente zu ermöglichen. (Übersetzung durch www.deepl.com)

Abbildung 13: Symbol Keras (Keras 11.02.2019)

Somit unterstützt Keras als eine von wenigen Bibliotheken nicht nur TensorFlow, sondern auch weitere Frameworks im Bereich der neuronalen Netze und ermöglicht so die Wiederverwendung von Code auf verschiedenen Plattformen. „Dies ist möglich, weil Keras das Backend vollständig abstrahiert; Keras besitzt eine eigene Datenstruktur für Graphen, die Berechnungsgraphen verwaltet und sich um die Kommunikation mit TensorFlow kümmert.“ (Hope et al. 2018, S. 135)

Die Definition und der Umgang mit Modellen erfolgt in Keras nativ in Python ohne die Notwenigkeit, Modelle zunächst deklarativ zu definieren. Hierdurch werden eine schnelle und saubere Implementierung, Training und Nutzung von Prototypen und finalen Modellen möglich.

Die Verbindung zwischen Keras und TensorFlow ist hierbei so stark, dass Keras offiziell auf den Seiten von TensorFlow als eine der high-level APIs genannt wird. (vergl. hierzu TensorFlow)

Im Rahmen der bereits zuvor angesprochenen Version 2 des Frameworks wird dessen Integration in TensorFlow nochmals weiter vorangetrieben. (vergl. hierzu Sandeep Gupta, Josh Gordon, Karmel Allison)

3 Anforderungen

Nachdem im vorhergehenden Kapitel auf die notwendigen Grundlagen eingegangen wurde, sollen nun die Anforderungen an das zu erstellende System näher betrachtet werden.

3.1 Funktionsumfang des Systems

Ausgehend von einem Eingabebild wird ein zweites Bild ausgegeben, welches die enthaltenen Lebensmittelinhaltsstoffe, im folgenden auch Inhaltsstoffe oder einfach Stoffe genannt, kennzeichnet.

Inhaltsstoffe im Sinne dieser Anwendung sind im Ursprungsbild enthaltene Texte, welche auf einen hinterlegten Inhaltsstoff verweisen. Die Zuordnung der Stoffe erfolgt hierbei anhand einer fest definierten Datenbasis von Lebensmittelinhaltsstoffen und erhebt nicht den Anspruch auf Vollständigkeit. Als Kriterium gelten primär die für einen Stoff hinterlegten Schlüsselwörter.

Als Inhaltsstoff erkannter Text wird mit einer roten, sonstigen Text mit einer grünen Umrandung versehen. Diese Rahmen werden im Folgenden als bounding box bezeichnet. Zusätzlich wird die E-Nummer des Inhaltsstoffes annotiert.

3.2 Anforderungen an die Inhaltsstoffe

Um die zuvor definierte Aufgabenstellung umzusetzen, müssen im Vorfeld alle in Betracht kommenden Inhaltsstoffe, deren alternativen Bezeichnungen sowie die weiteren, zu haltenden Informationen zusammengestellt, aufbereitet und verfügbar gemacht werden.

Um die Identifikation einzelner Stoffe zu ermöglichen, müssen diese zudem in einer abfrage- und erweiterbaren Struktur vorliegen. Die Einteilung der gefundenen Texte eines Bildes in Inhaltsstoffe und sonstigen Texte ist von der Verfügbarkeit innerhalb dieser Datenstruktur abhängig.

Um dieses Ziel zu erreichen, ist die Schaffung einer einfachen und erweiterbaren Möglichkeit zur Haltung der Inhaltsstoffe notwendig. Diese soll selbst möglichst einfach und nativ sein. Somit entfallen proprietäre und aufwändige System wie beispielsweise MySQL oder Access zugunsten einer textbasierten Lösung.

3.3 Anforderungen an die Eingangsdaten

Die Eingangsdaten liegen gemäß der Aufgabenstellung als Rastergrafik im RGB Farbraum vor. Das Speicherformat der Bilder ist hierbei sekundär. Teile des Bildes können Schlüsselwörter von Inhaltsstoffen enthalten. Text innerhalb einer natürlichen

Umgebung wird häufig allgemein als scene text bezeichnet und im Folgenden in dieser Bedeutung verwendet.

In einem realen Szenario würde solch ein Bild das Ergebnis einer ad hoc Aufnahme sein. Im Rahmen dieser Arbeit steht es als Datei eines Verzeichnisses zur Verfügung und wird dem zu erstellenden System als Array zur Verfügung gestellt. Der Prozess wird hierzu manuell zur Ausführung gebracht.

Aufgrund der Aufgabenstellung werden nur Bilder in einem geeigneten Format und Farbraum erwartet, so dass hier keine Konvertierungen vorzusehen sind. Da auch in einem real existierenden System in der Regel die möglichen Variationen sehr eingeschränkt sind, kommt dies einem echten Szenario trotz dessen sehr nahe.

Die zu verarbeitenden Bilder können in einer beliebigen Größe vorliegen. Eine notwendige Größenänderung muss daher systemintern und ohne eine Interaktion von Seiten des Nutzers vorgesehen werden.

3.4 Anforderungen an das neuronale Netz

Die hier vorgestellte Lösung basiert auf den Einsatz eines neuronalen Netzes, welches zur Lösung der Anforderung eingebunden und gegebenenfalls angepasst und trainiert werden muss.

Die Auswertung soll möglichst unabhängig von der Formatierung und Anordnung des Textes im Bild sein. Auch der Hintergrund des Bildes soll sich nach Möglichkeit nicht negativ auf die Erkennungsrate auswirken. Somit muss das neuronale Netz in der Lage sein, Text in einer in hohem Maße unstrukturierten Form zu erkennen.

Diese spezielle Form der Texterkennung in Bildern wird manchmal auch als *Reading Text in the Wild* umschrieben[18].

Zusammenfassend ergeben sich als Aufgaben:

1. Erkennen relevanter Regionen mit Text, im Folgenden als Region of Interest oder kurz RoI bezeichnet
2. grundlegende Klassifizierung dieser Bereiche als Text bzw. kein Text
3. Erkennen und Extraktion der Textbereiche in Form von Wörtern

[18] Siehe hierzu als Beispiel Soullard et al. (23.01.2019)

3.5 Verwendung des Ergebnisses

Das Ergebnis der Verarbeitung ist ein um relevante Informationen erweitertes, neues und eigenständiges Bild. Zur Umsetzung werden die von einem neuronalen Netz gelieferten Vorhersagen genutzt.

Diese liegen in Form einer Auflistung von Textbereichen mit ihren jeweiligen Koordinaten vor und können so zur Vorhersage (engl. prediction) des enthaltenen Textes genutzt werden. Die Koordinaten definieren hierbei einen Rahmen (engl. bounding box) um den Text.

Im ersten Schritt werden die gefundenen Begriffe mit den definierten Inhaltsstoffen abgeglichen. Kann ein Begriff nicht zugeordnet werden, so handelt es sich mutmaßlich um keinen relevanten Begriff. Nicht relevante Begriffe werden verworfen. Im zweiten Schritt werden die verbleibenden, identifizierten Inhaltstoffe wie in Abschnitt 3.1 beschrieben ausgegeben.

Der hier dargestellte Prozess zur Verwendung der Ergebnisse kann konzeptionell als eigenständiger Prozess angesehen werden, auch wenn er eng mit dem neuronalen Netz verwoben werden kann und eine Trennung nicht immer sinnvoll ist.

3.6 Ausführungsumgebung

Die hier erstellte Lösung würde im realen Einsatz anhand seiner API angesprochen und innerhalb eines anderen Systems, gegebenenfalls nach leichten Anpassungen, als Bibliothek fungieren.

Im Rahmen dieser Arbeit wird das umgebende System durch manuell auszuführenden Code simuliert. Somit kann die entstehende Lösung innerhalb einer Entwicklungsumgebung direkt genutzt werden. Hierdurch wird der Zugang zum Programm für Interessierte wesentlich einfacher und intuitiver gestaltet, ohne sich zu weit von einem möglichen Produktivcode zu entfernen.

Als Sprache kommt Python in Verbindung mit etablierten Bibliotheken wie OpenCV, NumPy oder Pandas zum Einsatz. Besonders OpenCV[19] bietet mit seiner Unterstützung für eine Vielzahl an Grafikformaten eine komfortable Art des Zugriffs. An der Basis wird der Einsatz von TensorFlow und Keras präferiert. Die High Level API von Keras hat den Vorteil, dass bei der Programmierung nicht direkt mit den Tensoren und Objekten von TensorFlow interagiert werden muss. Zudem eröffnet Keras einen einfachen Zugang zu verschiedenen, vortrainierten Netzen.

[19] Die Seite von OpenCV findet sich unter https://opencv.org/. (letzter Abruf am 26.08.2019)

3.7 Zusammenfassung

Ausgehend von einer allgemeinen Beschreibung des geplanten Umfangs wurden in diesem Kapitel die grundsätzlichen Anforderungen an ein resultierendes System angesprochen und differenziert.

Als grundsätzlich Aufgaben können genannt werden:

1. Vorverarbeitung des Eingabebildes
2. Lokalisierung und Erkennung vorhandener Wörter
3. Abgleich und visuelle Kennzeichnung der Inhaltsstoffe in einem neuen Bild

Die folgende Abbildung verdeutlicht nochmals die Aufgaben sowie die Anforderungen an das hier zu erstellende System:

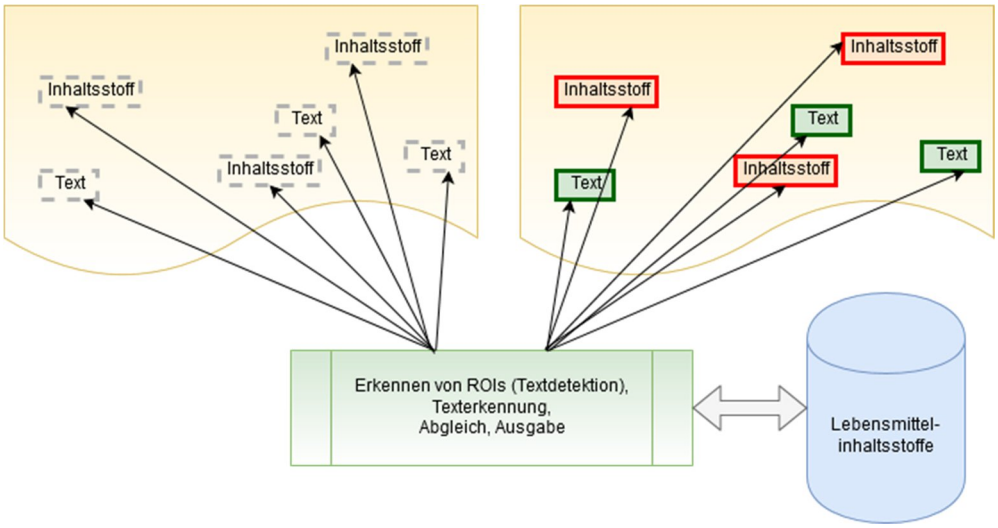

Abbildung 14: Übersicht der Anforderungen

Auf der linken Seite ist ein neutrales Eingabebild mit Scene Text zu sehen. Aufgabe des Systems ist die Erkennung und Auswertung der Texte. Hierzu nutzt es eine Datenbasis relevanter Inhaltsstoffe, hier blau eingetragen. Auf der rechten Seite ist das resultierende Ausgabebild zu sehen. Rote Rahmen kennzeichnen Inhaltsstoffe, grüne sonstigen Text.

Die Ausarbeitung eines ersten konkreten Lösungsansatzes für die genannten Aufgaben ist Inhalt des im folgenden Kapitel vorgestellten Konzeptes.

4 Konzeption

Nachdem die grundsätzlichen Anforderungen an das zu erstellende System im vorhergehenden Kapitel definiert worden sind, soll im Rahmen der Konzeption eine tragfähige Lösung zur Umsetzung entwickelt werden.

Um diese Aufgabe erfolgreich umzusetzen, soll zu Beginn auf einige Begrifflichkeiten und Grundlagen eingegangen werden, bevor das Problem der Texterkennung selbst betrachtet wird.

Anschließend beschäftigt sich der Abschnitt Netzstruktur mit der Auswahl einer für die Aufgabe geeigneten Netzstruktur. Hierzu werden einige aktuelle Netze kurz vorgestellt und auf Ihre Verwendbarkeit untersucht.

Im Abschnitt Datenhaltung wird ein Konzept zur Haltung der Lebensmittelinhaltsstoffe erarbeitet sowie eine programmtechnische Lösung vorgestellt.

Abschließend wird im umfangreichsten Teil das Konzept einer Ende-zu-Ende Lösung der hier behandelten Aufgabe entwickelt und vorgestellt.

4.1 Maschinelles Sehen

Sehr allgemein formuliert, beschäftigt sich das maschinelle Sehen (engl. computer vision) mit der Extraktion von Daten aus Bildern. Der Einsatzbereich reicht von einfachen Vermessungen über Detektion bis hin zum Versuch „Verständnis“ für ein Bild zu generieren. Als Beispiel sei hier die automatisierte Verschlagwortung von Bildern genannt.

Im Folgenden soll zur Verdeutlichung auf einige grundlegende Begrifflichkeiten mit Bezug zu dieser Arbeit kurz eingegangen werden.

4.1.1 Image Recognition vs. Image Processing

Aufgabe der Bildbearbeitung (engl. image processing) ist die Bearbeitung eines vorhandenen Bildes mit dem Ziel ein neues, bearbeitetes Bild zu erzeugen. Ziele dieser Bearbeitung können als Beispiel die Hervorhebung und Verdeutlichung von Details wie Kanten sein. Hierfür ist ein Verständnis des Bildes nicht erforderlich.

Die Bilderkennung (engl. image recognition) nutzt die Methoden des image processing, verfolgt jedoch das Ziel, den Inhalt eines Bildes zu erfassen und zu verstehen. Handelt es sich bei den zu erkennenden Objekten um Texte, so handelt es sich um eine Texterkennung (engl. text recognition).

4.1.2 Incidental vs. Focused Scene Text

Texte innerhalb von Bildern können in zwei disjunkte Gruppen, den zufälligen sowie den fokussierten Texten innerhalb eines Bildes, unterteilt werden.

In (Robust Reading Competition) findet sich hierzu eine treffende Umschreibung: „Incidental scene text refers to text that appears in the scene without the user having taken any specific prior action to cause its appearance or improve its positioning / quality in the frame. While focused scene text is the expected input for applications such as translation on demand, incidental scene text covers another wide range of applications linked to wearable cameras or massive urban captures where the capture is difficult or undesirable to control."[20]

Somit ist eine wichtige Eigenschaft von incidental scene text, dass der enthaltene Text keinerlei Vorverarbeitung in Form einer Positionierung oder ähnlichem erfährt.

4.1.3 Klassifikation

Die Klassifikation (engl. classification) ist die Vorhersage einer Klasse für ein gegebenes Objekt. Handelt es sich hierbei um genau zwei mögliche Klassen, so spricht man von einer binären Klassifikation.

4.1.4 Lokalisierung

Im Rahmen der Lokalisierung (engl. localization) wird versucht, den Ort zu bestimmen, an dem ein zu klassifizierendes Objekt anzutreffen ist. Auch für die Lokalisierung muss ein Modell zuvor trainiert werden. Im Rahmen des Trainings werden Bilder mit den zu identifizierenden Klassen und einer dazu gehörigen bounding box trainiert. Im Kontext des Trainings wird diese als ground truth bezeichnet.

4.1.5 Detektion

Die Detektion (engl. detection) beinhaltet die Klassifikation und die Lokalisierung von Objekten in einem Bild. Die Detektion unterscheidet jedoch darüber hinaus gehend die einzelnen Instanzen einer gleichen Klasse von Objekten.

4.1.6 Segmentierung

Die Segmentierung (engl. segmentation) unterscheidet sich zu den zuvor genannten Arten der Selektion grundlegend in der Form, dass sie auf Pixelebene arbeitet. Sie ist insofern als genauer anzusehen, als die zuvor genannten Verfahren.

[20] Zufälliger Szenentext bezieht sich auf Text, der in der Szene erscheint, ohne dass der Benutzer zuvor eine bestimmte Maßnahme ergriffen hat, um sein Aussehen zu verändern oder seine Positionierung / Qualität im Rahmen zu verbessern. Während fokussierter Szenentext der erwartete Input für Anwendungen wie die Übersetzung auf Abruf ist, deckt der zufällige Szenentext einen weiteren großen Anwendungsbereich ab, der mit tragbaren Kameras oder massiven Stadtaufnahmen verbunden ist, bei denen die Erfassung schwierig oder unerwünscht zu steuern ist. (Übersetzung durch www.deepl.com)

Bei der segmantic segmantation werden unterschiedliche Klassen erkannt, wohingegen bei der instance segmantation einzelne Instanzen von Klassen unterschieden werden.

4.1.7 Zusammenfassung

Anhand der untenstehenden Abbildung werden die zuvor genannten Stufen der Erkennung nochmals prägnant zusammengefasst:

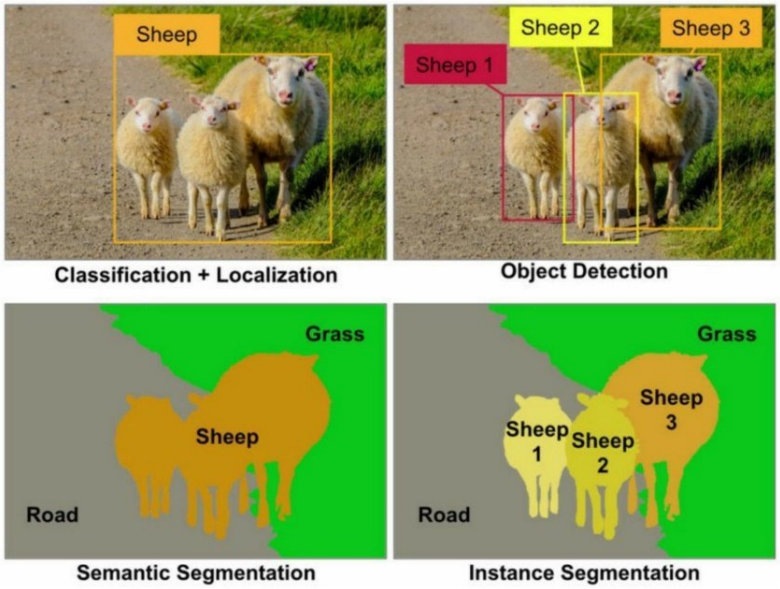

Abbildung 15: Arten der Selektion (Parmar 2018)

Deutlich werden in der Abbildung die Unterschiede zwischen der reinen Klassifizierung und Verortung (oben links) sowie der ergänzenden Unterscheidung gleicher Klassen (oben rechts), erkennbar. Dies gilt entsprechend für die Segmentierung (unten).

Aus den zuvor gemachten Ausführungen ergibt sich sehr deutlich, dass es sich bei der in dieser Arbeit vorliegenden Aufgabenstellung um die Detektion und dem Erkennen von Text (text detection and recognition) innerhalb eines Bildes handelt. Eine genauere Betrachtung führt zu der Beurteilung der betreffenden Texte als incidental scene texts.

Um diese erfolgreich umzusetzen, muss das hier zu entwickelnde System in der Lage sein, einzelne Wortvorkommen klassifiziere und lokalisieren zu können. Einzelne Vorkommen können hierbei mehrmals auftauchen. Somit reicht es nicht aus, nur das Vorkommen selbst zu bestimmen. Vielmehr muss das System in der Lage sein, einzelne Instanzen der gleichen Klasse zu unterscheiden und zu verorten. Eine Segmentierung ist jedoch zur Lösung der Aufgabe nicht erforderlich.

4.2 Textdetektion und Texterkennung

Die Auswertung von Texten aus Bildern ist ein bereits in die Jahre gekommenes Problem, für welches eine Reihe von Lösungen in Form von OCR Software (engl. optical

character recognition) existieren. Besonders für strukturierte und klar erkennbare Textkörper sind die hier vorhandenen Lösungen sehr effizient und verlässlich. Sie arbeiten auf der Ebene der einzelnen Buchstaben und nutzten Eigenschaften des Trägers wie Hintergrund, Fonts oder Kantendetektion.

Das Erkennen von Texten innerhalb natürlicher Umgebungen stellt diese Art der Erkennung vor nicht lösbare Probleme und führen zu einer schlechten Performanz. Gründe hierfür sind die fehlende Struktur sowie problematische Hintergründe.

Um diese Problematik zu lösen, wurden im Laufe der Zeit verschiedene Herangehensweisen entwickelt, wobei sich neuronale Netze als sehr erfolgreich erwiesen. Nach wie vor ist die Entwicklung hier sehr agil. Die hierzu gehörende Disziplin wird als text recognition bezeichnet. Im Rahmen dieser Arbeit steht dieser Begriff synonym für Texterkennung. Sie legt den Fokus letztlich auf die Erkennung des Wortes, statt des einzelnen Buchstabens.

Sofern Texte innerhalb eines größeren Kontextes aus einem Bild extrahiert werden, ist der erste Schritt auf dem Weg zur Texterkennung die Lokalisierung von Regionen mit möglichen Textvorkommen, die Textdetektion oder text detection. Diese Regionen werden als Region of Interest oder kurz RoI bezeichnet und können in beliebiger Anzahl an jeder Stelle sowie in jeder Größe und Ausrichtung innerhalb des Bildes existieren. In einem zweiten Schritt wird geprüft, ob die entsprechende Region Text enthält und dieser vorhergesagt.

Ein triviales Verfahren hierfür ist als sliding window bekannt. Hierbei wird ein Fenster systematisch über das anliegende Bild verschoben und der dort enthaltene Ausschnitt jeweils mit Hilfe eines externen CNNs klassifiziert und gegebenenfalls der weiteren Verarbeitung zugeführt. Mit einer mittleren Laufzeit $\theta(n^2)$ ist der Algorithmus jedoch sehr langsam und für Echtzeitanwendungen nicht geeignet, jedoch sind auch Laufzeiten von $\theta(1)$ möglich[21].

Mit modernen Netzen wie EAST, SSD oder YOLO existieren jedoch auch Lösungen, welche alle in einem Bild vorhandenen Objekte in nur einem Durchlauf erfassen und im Falle der letztgenannten auch klassifizieren können. Auf geeignete neuronale Netze wird im folgenden Abschnitt 4.3 weiter eingegangen.

[21] Siehe auch https://storage.googleapis.com/pub-tools-public-publication-data/pdf/34843.pdf. (letzter Abruf am 25.08.2019)

4.3 Netzstruktur

Die hier behandelte Umsetzung der Aufgabenstellung erfolgt auf Basis eines neuronalen Netzes, welches hierdurch eine zentrale Rolle einnimmt und auf den gesamten Lösungsweg einwirkt.

Im Rahmen dieses Abschnitts ist daher vor allen anderen Fragen zu klären, welches der möglichen Netze zur Umsetzung der Aufgabe Anwendung findet.

Da eine umfassende Untersuchung aller in Frage kommenden Netze den Rahmen dieser Arbeit bei weitem überschreitet, werden an dieser Stelle nur zwei mögliche Systeme zur Lösung dieser Aufgabe vorgestellt und verglichen.

Zuvor werden jedoch noch einige wichtige Konzepte kurz eingeführt, um ein tieferes Verständnis zu vermitteln.

4.3.1 Building vs. Using

Ein wichtiges Merkmal zur Beurteilung eines Systems ist die Art und Weise, in der ein neuronales Netz eingesetzt wird.

Besonders zu Beginn des Einsatzes neuronaler Netze wurden diese gezielt zur Lösung bestimmter und definierter Problemstellungen entwickelt und genutzt. Häufig waren dies einfache CNNs oder RNNs. Sie verfügten über eine geringe Anzahl an Layern und konnten verhältnismäßig einfach für ihre spezielle Aufgabenstellung trainiert werden. Das Building in Form der Konzeption und Umsetzung stand hierbei im Fokus.

Mit der fortschreitenden Entwicklung auf diesem Gebiet wurden die verwendeten Netze ebenso wie die zu lösenden Probleme komplexer. Diesen Fortschritt beförderte auch die zunehmende Verfügbarkeit von Rechenleistung, freien Datasets sowie frei zugänglichen, wissenschaftlichen Arbeiten[22] in diesen Bereich.

Moderne, oft auch als state-of-the-art bezeichnete, Netze nutzen häufig neuronale Netze als Komponente und erweitern diese. Hierbei kann aktuell auf eine Vielzahl, auch vortrainierter, Netze mit unterschiedlichen Architekturen zurückgegriffen werden.

Somit kann festgestellt werden, dass sich der Fokus hin zu einem Using verlagert hat. Ein Beispiel hierfür sind die hier vorgestellten EAST und SSD Netze, welche nicht nur auf andere Netze aufbauen, sondern hierbei auch eine breite Variation erlauben.

4.3.2 Training neuronaler Netze

Ein wichtiger Punkt bei der Konzeption und dem Einsatz eines neuronalen Netzes ist die Möglichkeit des Trainings des verwendeten Netzes. Namentlich muss evaluiert werden,

[22] Eine wichtige Quelle hierfür ist die Seite https://arxiv.org der Cornell University. (letzter Abruf am 24.08.2019)

inwieweit vortrainierte Modelle zum Einsatz kommen und gegebenenfalls im Rahmen weiterer Maßnahmen optimiert werden können. Als aufwendige, aber grundsätzlich immer mögliche Option bietet sich das auf ein Problem bezogene vollständige Training an. Im Folgenden sollen einige wichtige Begrifflichkeiten kurz erläutert werden.

4.3.2.1 Transfer Learning / Feature Extractor / Finetuning

"Transfer learning is a machine learning method where a model developed for a task is reused as the starting point for a model on a second task."[23] (Jason Brownlee 2017)

Weiter wird sinngemäß ausgeführt, dass in der Praxis zwei unterschiedliche Herangehensweisen unterschieden werden können:

1. Ein fertig vortrainiertes Modell dient als Basis. Hierzu werden dessen Struktur und Gewichte übernommen, dessen letzte Schichten jedoch ersetzt und gegebenenfalls weitere hinzugefügt. Anschließend erfolgt ein weiterführendes Training zur Lösung der eigentlichen Aufgaben. Die Gewichte des Basismodells werden hierbei entweder mit angepasst oder bleiben unverändert.
2. In einem ersten Schritt wird ein Modell für ein grundlegendes Problem entwickelt und trainiert. Im Anschluss daran wird ein weiteres Modell für die eigentliche Problemstellung entwickelt. Dieses übernimmt als Eingabe die Ausgaben des ersteren Modells. Es wird somit mit zwei getrennten Modellen gearbeitet

Allgemein werden die hierzu genutzten Methoden als Transfer Learning und Finetuning bezeichnet. Die folgende Abbildung skizziert dies Vorgehen:

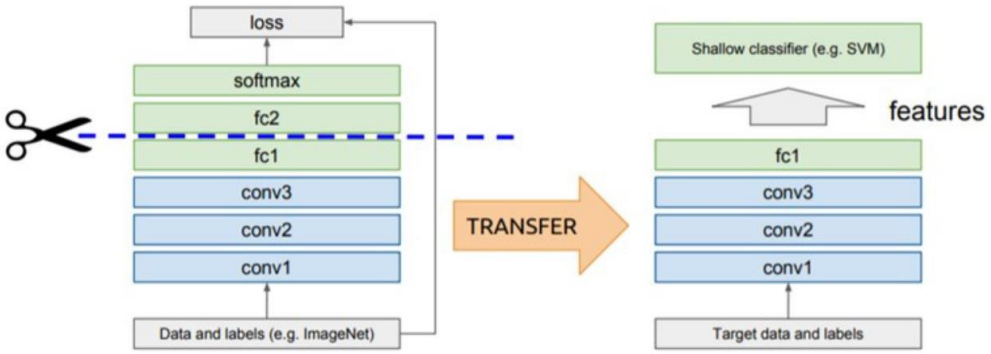

Abbildung 16: Prinzip des Transfer Learning (Sarkar 2018)

In der Abbildung ist links ein fertig trainiertes Modell zu sehen, von welchem die obersten Schichten, die der abschließenden Klassifizierung dienen, entfernt werden. Das verbleibende Model dient mit seinen trainierten Gewichten einem zweiten Modell rechts als

[23] Transferlernen ist eine maschinelle Lernmethode, bei der ein für eine Aufgabe entwickeltes Modell als Ausgangspunkt für ein Modell für eine zweite Aufgabe wiederverwendet wird. (Übersetzung durch www.deepl.com)

feature extractor. Dieses wird lediglich um wenige Layer ergänzt und in einem weiterführenden Training, dem finetuning, auf eine Aufgabe vorbereitet.

Hierzu schreibt (Sarkar 2018) auf Towards Data Science: „The key idea here is to just leverage the pre-trained model's weighted layers to extract features but not to update the weights of the model's layers during training with new data for the new task." [24]

Finetuning ist somit letztlich ein ergänzendes Training eines kleinen Netzes auf Basis der Zieldaten einer neuen Anwendung. Als Eingabe dienen hierbei die Features des darunter liegenden Netzes. Als Alternative hierzu kann zudem auch das gesamte Netz an den zur Spezialisierung verwendeten Daten weiter trainiert werden, was jedoch entsprechende Ressourcen voraussetzt. Die hierzu verwendeten Trainingsdaten werden als Datasets bezeichnet.

4.3.2.2 Competitions und Datasets

Viele der heute verfügbaren Netze entstammen der Forschung und wurden im Rahmen von etablierten Wettbewerben (engl. competitions) anhand standardisierter und etablierter Datasets trainiert und getestet. Die hierzu erforderlichen Ressourcen an Daten, Zeit und Rechenleistung lassen sich im normalen Alltag nur schwer bis gar nicht umsetzen. Nicht selten handelt es sich bei den Teilnehmern um Teams renommierter Wissenschaftler im Umfeld einer wissenschaftlichen Einrichtung.

Diese Wettbewerbe ermöglichen es, den Fortschritt und die Leistungsfähigkeit neuer Netze anhand von standardisierten Datenbeständen (engl. datasets) und Aufgaben (engl. tasks) vergleichbar zu bewerten. Die so entstehenden, trainierten neuronalen Netze sind häufig frei verfügbar und dienen als Basis eines transfer learnings.

Beispiele[25] hierfür sind:

- kaggle (https://www.kaggle.com)
  Sammlung aktueller Datasets zu verschiedenen Themen sowie eigene Wettbewerbe
- COCO (http://cocodataset.org)
  eine von Microsoft initiierte umfangreiche Datensammlung mit verschiedenen Schwerpunkten sowie eigene Wettbewerbe

[24] Die Grundidee dabei ist, die gewichteten Schichten des vortrainierten Modells einfach zu nutzen, um Merkmale zu extrahieren, aber nicht, die Gewichte der Schichten des Modells während des Trainings mit neuen Daten für die neue Aufgabe zu aktualisieren. (Übersetzung durch www.deepl.com)

[25] Alle zu den Beispielen angegebenen Seiten wurden letztmalig am 24.08.2019 abgerufen. Infos zu den genannten Abkürzungen finden sich am Ende.

- ImageNet (http://www.image-net.org)
  eine umfangreiche Sammlung von über 14 Mio. Bilder mit 1000 Klassen, darunter rund 1 Mio. mit bounding box annotiert, sowie eigene Wettbewerbe[26]
- Robust Reading Competition (https://rrc.cvc.uab.es)
  Sammlung größerer Datasets zu verschiedenen Themen im Bereich Texterkennung sowie eigene Wettbewerbe
- MNIST database (http://yann.lecun.com/exdb/mnist)
  umfangreiche, aber nicht mehr als aktuell angesehene Gruppe von Datasets zum Thema Schrifterkennung

Besonders das Abschneiden bei Wettbewerben von COCO und ImageNet wird regelmäßig zur Klassifizierung verschiedener Netztypen herangezogen. ImageNet bewertet hierbei in den Kategorien Top-5 und Top-1 Treffer. Im ersten Fall gilt als Erfolg, wenn eine der fünf höchsten Wahrscheinlichkeiten übereinstimmt. Für den zweiten Fall muss das Ergebnis genau vorhergesagt werden.

Die im Rahmen dieser Wettbewerbe trainierten Modelle sind ebenso wie die Datasets selbst als Ganzes, oder aber in Form ihrer vortrainierten Gewichte öffentlich verfügbar. Häufig werden Quellen für vortrainierte Modelle auch als model zoo bezeichnet.

Für Datasets existieren keine universellen Vorgaben an den Aufbau oder das Format. Grundsätzlich enthalten sie jedoch eine als X bezeichnete Eingabe und eine als Lable bezeichnete Ausgabe. Der Aufbau richtet sich hierbei an den Task, für welchen das Dataset Trainingsdaten liefert. So umfassen die Trainingsdaten im Bereich der Objekterkennung neben den Objekten auch deren Position in Form von bounding boxen. Die in den Labeln enthaltene korrekte Zuordnung eines Objektes zu einer bounding box wird als ground truth bezeichnet.

4.3.2.3 Datasets und Trainingsdaten

Wie zuvor ausgeführt, werden häufig etablierte Datasets zum Training eines Modells verwendet oder aber in Form von vortrainierten Modellen genutzt. Daneben kommen eigene Daten in Bezug zur Problemstellung zum Einsatz. Hierbei handelt es sich entweder um reale aufbereitete Daten oder aber um mit Hilfe eines Programms künstlich generierte Daten.

Ein Einsatzbereich generierter Daten ist die Erstellung von synthetischen Texten, umgangssprachlich auch als SynText bezeichnet. Hierbei werden Bilder mit Texten über

[26] ImageNet Large Scale Visual Recognition Competition (ILSVRC)

ein Programm in immer neuen Kombinationen zusammengeführt und so eine umfangreiche Datenbasis generiert.

Im Bereich des maschinellen Sehens ergibt sich mit Hilfe der Augmentation eine weitere Möglichkeit, Daten effizienter zu nutzen. Hierbei werden anhand der vorhandenen Bilddaten verschiedene Variationen generiert und dem Training zugeführt.

Im Allgemeinen wird das eingesetzte Dataset, wie im folgenden Bild zu sehen, für seinen Einsatz im Verhältnis 1:3 aufgeteilt.

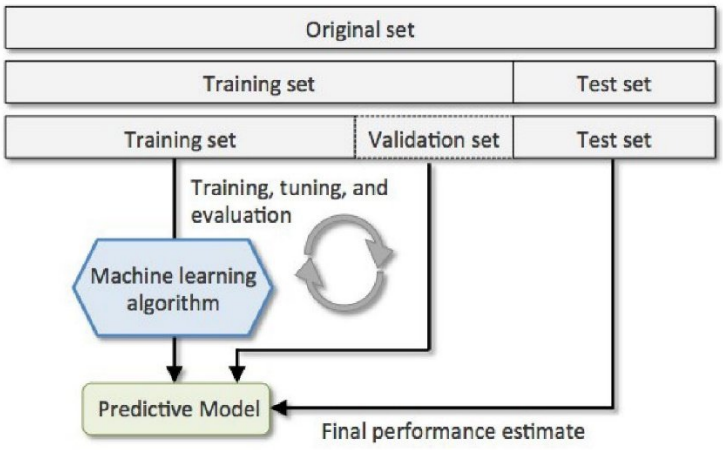

Abbildung 17: Struktur Dataset (Gonfalonieri 2019)

Hierbei dient der größte Teil des Datasets (Training set, Mitte links) dem Training des Netzes. Hierzu wird dieses in zwei weitere Teile unterteilt. Das größere dient dem Training (Training set, unten links), das andere der Validierung im Training (Validation set, unten Mitte).

Die verbliebenen Daten (Test set, Mitte rechts) dienen der Ermittlung der Genauigkeit des Netzes nach dem Training. Hierbei ist es wichtig, dass es sich um für das Netz bis dahin unbekannte Daten handelt.

4.3.3 Multibox Detektoren

Bereits im Abschnitt 4.2 - Textdetektion und Texterkennung wurde kurz auf das Verfahren zur Detektion von Objekten eingegangen. Zum besseren Verständnis der in den folgenden Abschnitten vorgestellten Lösungsansätzen, wird an dieser Stelle das Prinzip der Multibox Detektoren kurz weiter vorgestellt.

Es handelt sich hierbei um Systeme auf Basis neuronaler Netze, die in der Lage sind, in nur einem Durchlauf alle möglichen Regionen eines Bildes zu untersuchen und je nach Modell zu klassifizieren. Zur Erinnerung geschah dies mit der sliding windows Methode durch wiederholte Verarbeitung der einzelnen Regionen.

Basis des Systems ist die Tatsache, dass man die zuvor genannten Regionen eines Bildes auch als Ergebnis einzelner Filterungen eines Filterkerns interpretieren kann, der pixelweise über eine Bild geführt wird. Dies kann mittels einer Filterschicht umgesetzt werden. Die Lage der jeweiligen Schichten wirkt sich hierbei auf die Größe der betrachteten Region im Bild aus. Die gesamte Auswertung geschieht durch die Auswertung der Ausgaben der eingesetzten Schichten. Dies kann auf verschiedenen Wegen geschehen.

4.3.4 SSD + VGGNet

Ein mögliches System zu Lösung der hier vorliegenden Aufgabenstellung ist ein SSD auf Basis eines VGG Netzes. Beide Bestandteile sollen hier kurz vorgestellt werden.

4.3.4.1 VGGNet – Visual Geometry Group (2014)

Das VGG Netz ist das letzte Netz, bei dessen Architektur auf eine rein lineare Anordnung der Schichten gesetzt wurde. Hierdurch eignet es sich sehr gut als Basis neuer Lösungen und verfügt zudem mit weniger als acht Prozent Fehlerrate[27] über eine hohe Genauigkeit.

Die folgende Abbildung verdeutlicht den Aufbau von links nach rechts:

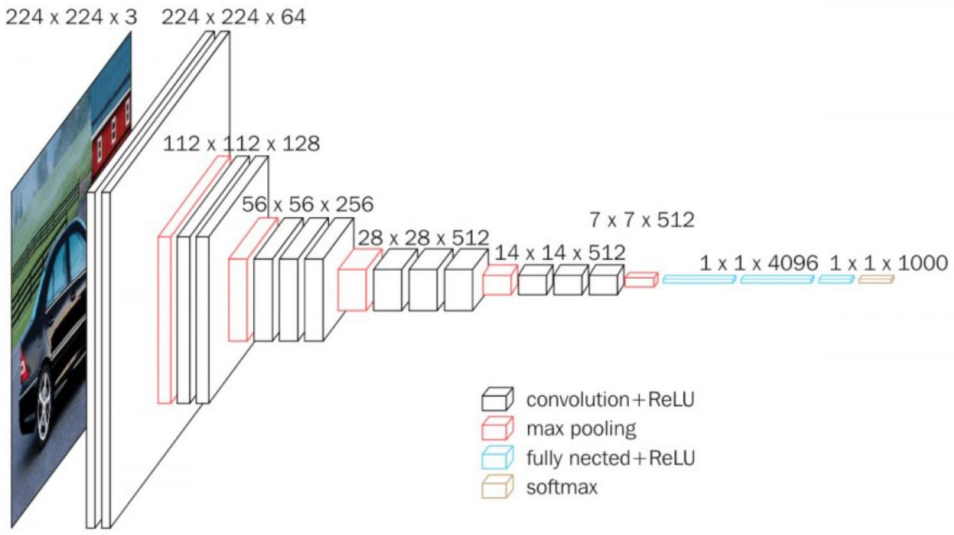

Abbildung 18: Struktur VGG-16 Netz (Muneeb ul Hassan 2018)

Auf Faltungsschichten (schwarz) folgt jeweils ein Pooling (rot) und am Ende (rechts) drei Dense Layer (blau), sowie ein Softmax (braun) als Ausgabe. Die 16 verweist auf das Vorhandensein von 16 Faltungs und Dense Layern.

[27] Siehe hierzu auch https://medium.com/coinmonks/paper-review-of-vggnet-1st-runner-up-of-ilsvlc-2014-image-classification-d02355543a11. (letzter Abruf am 20.08.2019)

4.3.4.2 SSD – Single Shot MultiBox Detector (2016)

Bei einem SSD handelt es sich nicht um ein reines neuronales Netz, sondern um ein System, welches in seiner ursprünglichen Form auf ein VGG-16 Netz als feature extractor aufbaut, dieses jedoch erweitert und dessen Ausgabe nachbereitet.

Durch die parallele Auswertung der feature maps mehrerer Ebenen und den Einsatz fest vorgegebener bounding boxes wird die Detektion mehrerer Objekte in einem Arbeitsablauf geleistet. Diesen Umstand verdankt das Model seinen Namen.

Durch die Verwendung verschiedener, fest vorgegebener Boxen je betrachtetem Ausschnitt und deren parallelen Verarbeitung können Objekte verschiedene Formate erkannt werden. Durch seinen Aufbau wird die anfängliche Eingabe von 300 x 300 auf 38 x 38 herunter gebrochen. Hierdurch zeigen sich schwächen bei kleineren Objekten. Dies Verhalten kann zum Teil durch höhere Auflösungen kompensiert werden (vergl. hierzu Hui 2018).

Die folgende Abbildung zeigt den Aufbau eines SSD mit einem VGG-16 Netz als feature extractor:

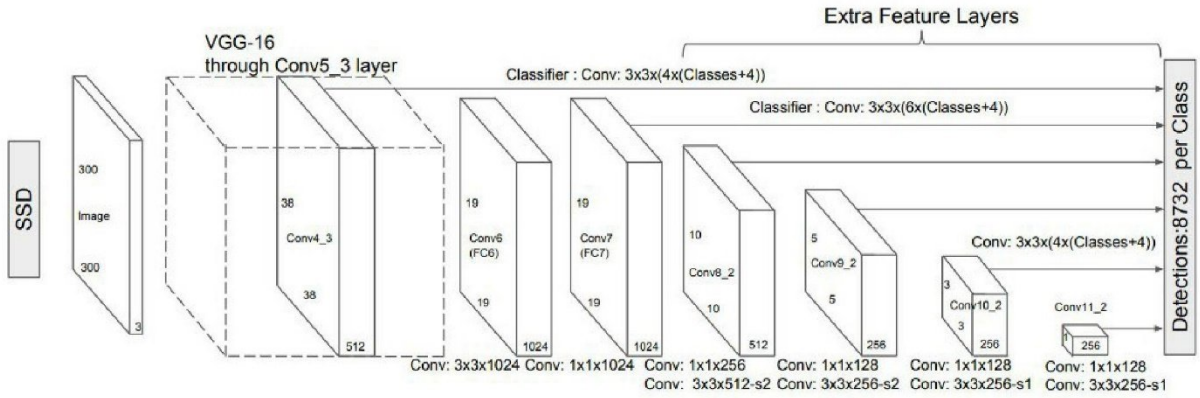

Abbildung 19: Struktur SSD mit VGG-16 Netz (Hui 2018)

Wie in der Abbildung zu erkennen ist, folgen auf dem VGG-16 Netz (links) eine Reihe weiterer Schichten. Deren Anzahl variiert nach konkreter Ausprägung des SSD. Jede Schicht führt ihre Ausgabe, hier durch horizontale Linien oben rechts dargestellt, zur abschließenden Auswertung (rechts).

4.3.4.3 Vor- und Nachteile des Einsatzes

Grundsätzlich ist die Verwendung eines SSD Netzes für die hier umzusetzende Aufgabe möglich und stellt auf Grund seines Aufbaus de facto eine Ende-zu-Ende Lösung dar. Über geeignete Quellen sind Objekterkennung trainierte Gewichte sowie Referenzimplementierungen für verschiedenen Konfigurationen verfügbar. Als Datasets kommen hier vor allem COCO und ImageNet zum Einsatz.

Die Verwendung nur eines Netzes zur Lösung des Problems ist auf den ersten Blick ein sehr homogener und effizienter Lösungsansatz für das vorliegende Problem. Drei wichtige Faktoren sprechen jedoch dem Einsatz dieses Netzes entgegen:

1. Es existieren nur sehr wenige und geeignete Modelle, die speziell für die Texterkennung trainiert wurden. Nach dem aktuellen Kenntnisstand des Autors handelt es sich hierbei fast ausnahmslos um Englisch. Dies führt zu einer schlechten Performanz bei der Verwendung mit deutscher Sprache, was ein vollständiges oder zumindest weiter gehendes Training erforderlich macht.
2. Die Anzahl möglicher Kandidaten wird nochmals durch den Umstand verringert, dass nur sehr wenige Referenzimplementierungen im Umfeld von Keras und TensorFlow existieren. Da das vortrainierte Modell jedoch später mit der konkreten Implementierung harmonisieren muss, folgt hieraus ein akuter Mangel an geeigneten Alternativen.
3. Ein vollständiges Training für ein SSD ist sehr aufwendig, aber auch das reine fine tuning gestaltet sich aufwendig. Grundsätzlich kann nur das gesamte Modell trainiert werden, jedoch nicht separat einzelne Fähigkeiten, wie beispielsweise die Detektion.

4.3.5 EAST + CRNN

Eine zweite Möglichkeit zur Umsetzung einer Lösung der anstehenden Aufgabe ist die Kombination eines EAST sowie eines CRNN Netzes. Im Gegensatz zu der im vorhergehenden Abschnitt vorgestellten Lösung, kommen bei diesem Ansatz zwei unabhängige Netze zum Einsatz und erst deren Zusammenspiel führt zu einer Ende-zu-Ende Lösung.

Beide Systeme sollen im Folgenden in der Reihenfolge ihres Einsatzes innerhalb einer möglichen Lösung vorgestellt werden.

4.3.5.1 EAST – An Efficient and Accurate Scene Text Detector (2017)

Ähnlich wie bei dem zuvor aufgeführten SSD können auch bei diesem Detektor in nur einen Durchgang mehrere Objekte detektiert werden. Im Gegensatz zu den vorgenannten ist dieses Modell jedoch auf die Detektion von Text spezialisiert.

Die Detektion erfolgt hierbei wie bei einem SSD durch die Auswertung einzelner Layer und den Umstand, dass hierdurch systematisch das gesamte Bild in nur einen Durchgang ausgewertet werden kann. Ein vortrainiertes Modell dient als feature extractor, dessen Ergebnisse der weiteren Verarbeitung zugeführt werden.

Nach (Zhou et al. 10.07.2017, S. 4323) wird im originalen Paper als feature extractor allgemein ein am ImageNet vortrainiertes Faltungsnetz voraus gesetzt. Als mögliche Variationen wurden dort neben einem VGG Netz auch PVANET[28] untersucht.

Die folgende Abbildung verdeutlich die Struktur des EAST Netzes anhand eines PVANET als feature extractor:

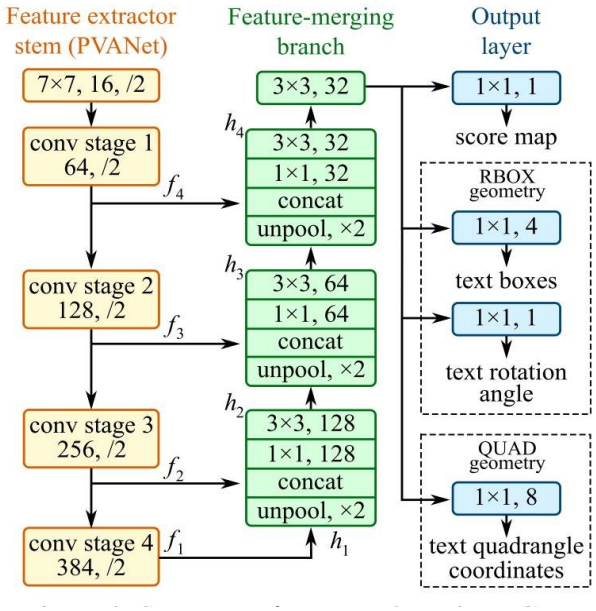

Abbildung 20: Struktur eines EAST Netzwerkes (Zhou et al. 10.07.2017, S. 4323)

Auf dem Bild ist links der Aufbau des zugrunde liegenden Netzes (PVANET, gelb) als reines Faltungsnetz zu erkennen. Die hier gewonnen Features werden mit Hilfe der mittleren Netzstruktur (grün) zusammengefasst und zusätzlich gefaltet. Dies entspricht der Untersuchung einzelner Regionen des zugrunde liegenden Bildes. Als letztes werden die so gewonnenen Features im rechten Block (blau) ausgewertet. Deutlich ist die Ähnlichkeit mit der Struktur eines SSD Netzes zu erkennen.

4.3.5.2 CRNN - Convolutional-Recurrent Neural Network (2015)

Bei einem CRNN werden die Fähigkeiten eines Faltungsnetzes mit denen eines rekurrenten Netzwerkes vereint. Hierbei dient das Faltungsnetz als feature extractor, wohingegen das rekurrente Netz die zeitliche Abfolge der features berücksichtigt. Hierdurch eignen sie sich sehr gut für die Erkennung von Texten.

Ein in einem Bild eingebetteter Text kann als eine zeitliche Folge von Features aufgefasst werden, wobei bestimmte Sequenzen einzelnen Buchstaben respektive Wörtern entsprechen. Ziel des Trainings ist somit die Zuordnung definierter Sequenzen zu

[28] Das Netz wird in Kim et al. (30.09.2016) ausführlich vorgestellt und ist hier für das Verständnis nicht weiter relevant.

Vorhersagen in Form von Buchstaben und Wörtern. Dies geschieht grundsätzlich durch eine Anpassung der entsprechenden Gewichte, erfordert jedoch eine weitergehende Bearbeitung.

Die folgende Abbildung verdeutlicht den Aufbau eines CRNN zur Erkennung von Text innerhalb eines Bildes:

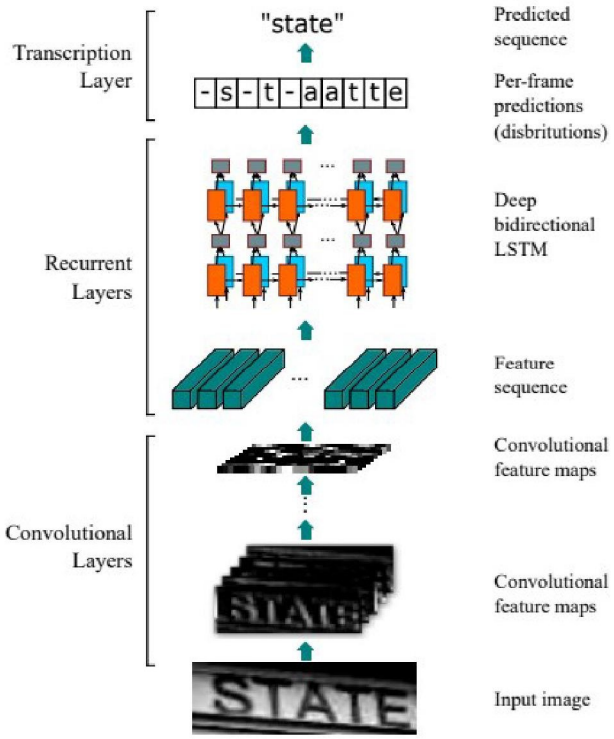

Abbildung 21: Struktur eines CRNN zur Texterkennung (Shperber 2019)

In der Abbildung ist zu erkennen, wie zu Anfang (im Bild unten) mit Hilfe eines Faltungsnetzes die Merkmale des Bildes extrahiert werden und als eine fortlaufende Sequenz an eine rekurrente Schicht in Form bidirektionaler LSTMs (Mitte) weitergegeben wird. Durch die Verwendung bidirektionaler LSTM können Verzweigungen zu beiden Richtungen der Zeitachse erfasst werden.

Am Schluss (oben) steht eine Vorhersage verschiedener Buchstaben, welche jedoch noch nicht fehlerfrei ist. Dies wird erst durch den abschließenden Transcription Layer geleistet. Das hierzu benötigte Vorgehen wird im folgenden Abschnitt thematisiert.

4.3.5.2.1 CTC – Connectionist Temporal Classification

Ein Faltungsnetz extrahiert aus einem gegebenen Netz Features, welche dazu genutzt werden, Vorhersagen über den Inhalt eines Bildes zu machen. Um nun Objekte lokalisieren und klassifizieren zu können, werden verschiedene Ausschnitte (RoIs) untersucht

und klassifiziert. Der Ausschnitt mit der höchsten Wahrscheinlichkeit wird als Vorkommen des Objektes verortet.

Bei der Erkennung von Text ergibt sich das Problem, das ein Buchstabe nicht genau einem Ausschnitt zuordbar ist. Zudem erstrecken sich Wörter selbst über mehrere solcher Ausschnitte. Es kann zu Lücken in der Vorhersage kommen, die nicht relevant sind und gewollten Lücken, die das Ende eines Wortes kennzeichnen. Die folgende Abbildung verdeutlicht dies:

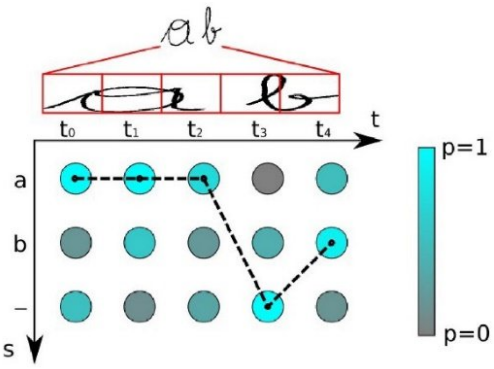

Abbildung 22: Ausgabematrix als Ergebnis einer Faltung (Scheidl 2018)

In der Ausgabematrix unten im Bild ist anhand des Pfades zu sehen, dass an drei aufeinander folgenden Punkten ein „a“ vorhergesagt wird, gefolgt von einer Lücke und den Buchstaben „b“. Dies ist eine Folge davon, dass im Eingabebild der Schriftzug „ab“ über eine Sequenz von vier Zeiteinheiten auftaucht. Zeiteinheiten folgen dem Verlauf der verwendeten Filterkerns von links nach rechts sowie von oben nach unten.

Eine Lösung für dieses Problem bietet der CTC[29] Loss, eine spezielle Verlustfunktion. Das nachfolgende Bild verdeutlicht den prinzipiellen Ablauf:

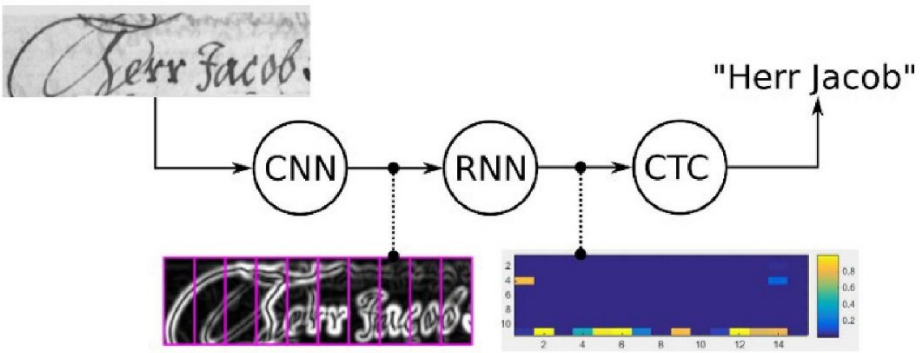

Abbildung 23: Prinzip der Texterkennung mittels CRNN und CTC (Scheidl 2018)

[29] Siehe auch https://www.cs.toronto.edu/~graves/icml_2006.pdf. (letzter Abruf am 21.08.2019)

In dem Bild ist zu erkennen, wie über eine Faltungsnetz (CNN) eine Reihe von 10 Features ermittelt werden (unten links). Für jedes Feature wird nun auch der vorhergehende und nachfolgende Zustand mit Hilfe eines bidirektionalen rekurrenten Netzes untersucht (RNN). Anschließend wird der CTC Loss optimiert (CTC) und das Ergebnis ausgegeben.

Hierbei wird sinngemäß laut (N 2017) zum einen die höchste Wahrscheinlichkeit eines Pfads innerhalb einer Sequenz für ein bestimmtes Label berücksichtigt, zum anderen wird bidirektional nicht nur der aktuelle, sondern auch der vorhergehende und nachfolgende Zeitabschnitt untersucht.

4.3.5.3 Vor- und Nachteile des Einsatzes

Wie zuvor spricht grundsätzlich nichts gegen die Verwendung einer Kombination von EAST mit einem nachgelagerten CRNN. Auch diese Lösung würde letztlich ein Ende-zu-Ende Netz umsetzen. Für beide Netze existieren geeignete Referenzimplementierungen sowie geeignete, vortrainierte Gewichte. Als Datasets kommen auch hier vor allem COCO und ImageNet zum Einsatz.

Vier wichtige Faktoren sprechen für den Einsatz dieser Kombination von Netzen:

1. EAST Netze sind Spezialisten für die Detektion von Texten und reagieren weniger empfindlich auf unterschiedliche Sprachen, da nicht der Sinn eines Textes im Fokus liegt, sondern die Buchstaben. In ersten Tests waren die Erkennungsraten ohne ein fine tuning bereits erstaunlich gut.
2. CRNN Netze dienen der Extraktion von Texten. Durch ihren Aufbau sind sie in der Lage, Texte mit variabler Länge zu erkennen. Durch ihren Fokus auf den Buchstaben, statt eines Wortes, sind sie im praktischen Einsatz sehr unabhängig von der Trainingssprache. Auch hier waren erste Tests bereits ohne ein fine tuning bemerkenswert gut.
3. Die eingesetzten Netzwerke lassen sich getrennt voneinander auswählen und fine tunen, wobei der Aufwand wesentlich geringer ist, als bei einem SSD.
4. Für beide Netze existieren gute Referenzimplementierungen, welche die ursprünglich in Caffe umgesetzten Netze nach Keras/TensorFlow portieren und zudem vortrainierte Modelle verfügbar machen. Da viele der Implementierungen unter der GNU oder MIT lizensiert sind, können Sie in Übereinstimmung mit den Vorgaben der Lizenz genutzt und angepasst werden.

Gegen den Einsatz der hier vorgestellten Lösung steht vor allem die notwendige Orchestrierung zweier separater Netze. Diese fordert einen deutlich höheren Aufwand als dies bei dem Einsatz eines SSD der Fall wäre.

4.3.6 Entscheidung

Im voran gegangenen Abschnitt wurde das als Basis dienende neuronale Netz thematisiert. Im Mittelpunkt standen hier zum einen ein SSD Netz, zum anderen eine Kombination von EAST und CRNN.

Ob die Verwendung nur eines Netzes oder aber die zweier Netzte die elegantere Lösung darstellt, liegt zu einem großen Teil an dem jeweiligen Betrachter und kann hier nicht abschließend behandelt werden. Für beide Lösungen existieren geeignete Referenzimplementierungen und zumindest grundsätzlich vortrainierte Gewichte.

Betrachtet man jedoch die genannten Vor- und Nachteile beider Lösungen, so überwiegen die Vorteile der Lösung auf Basis eines CRNN. Speziell die Trennung von Detektion und Erkennung bietet Raum für eine bessere Aufteilung der Komponenten und ermöglicht somit einen modularen Aufbau des Systems. Gleichzeitig ermöglicht es die Optimierung nur jeweils eines Netzes bis hin zu dessen kompletten Ersetzung.

Hierbei steht die Verwendung bereits vortrainierter Gewichte einer kompletten Überarbeitung oder dem Einsatz eigener Ansätze bei der Gestaltung eines Netzes entgegen, da das trainierende Netzt und die daraus resultierenden Gewichte strukturell zusammenpassen müssen.

Aus den dargelegten Gründen wird im Rahmen dieser Arbeit für die Umsetzung der Einsatz einer Kombination von EAST und CRNN als externe Komponenten favorisiert. Bei Schaffung einer geeigneten Umgebung ergeben sich für einen Austausch sowohl mit einem anderen, externen System als auch mit einer eigenen Lösung zu einem späteren Zeitpunkt keine Probleme.

4.4 Datenhaltung

Die grundlegenden Anforderungen an die Datenhaltung wurden bereits zuvor im Abschnitt 3.2 - Anforderungen an die Inhaltsstoffe, Seite 20 thematisiert und werden daher an dieser Stelle nicht nochmals ausgeführt.

Grundsätzlich kommt eine Vielzahl an Eigenschaften in Betracht, welche sich für eine tiefere Beschreibung eines Inhaltsstoffes eignen und nützliche Informationen zu einem gefundenen Stoff anbieten. Gleichzeitig steht deren Aufnahme nicht automatisch für eine spätere Nutzung zur Erweiterung des Eingangsbildes, sondern stellt einen Vorrat an verwendbaren Informationen dar.

Der zentrale Zweck der Datenhaltung ist die Zuordnung eines extrahierten Textes zu einem Eintrag innerhalb der Datenhaltung und damit zu einer eindeutigen ID, alle anderen Funktionen sind sekundär.

Konkret werden die folgenden Eigenschaften[30] im Rahmen dieser Arbeit berücksichtigt werden:

- ID ➔ ein eindeutiger Bezeichner als Zahl
- E-Nr ➔ die E-Nummer eines Stoffes
- Stoff ➔ die Bezeichnung/-en
- Bemerkung ➔ allgemeine Bemerkungen
- Anmerkung ➔ weitere Anmerkungen wie Maximalwerte etc.
- Einordnung ➔ eine Einordnung des Stoffes (Default: neutral)
- Suchwörter ➔ weitere Suchwörter für den Stoff

Anhand der gegebenen Informationen wird eine interne Suchliste erstellt. Die Suchbegriffe hierzu werden hierbei dynamisch aus der E-Nummer, dem Namen sowie optional angegebenen Suchwörtern erzeugt. Sie stellen jeweils einen Schlüssel dar, der auf die eindeutige ID eines Stoffes verweist. Die folgende Abbildung gibt einen Überblick über diese Struktur:

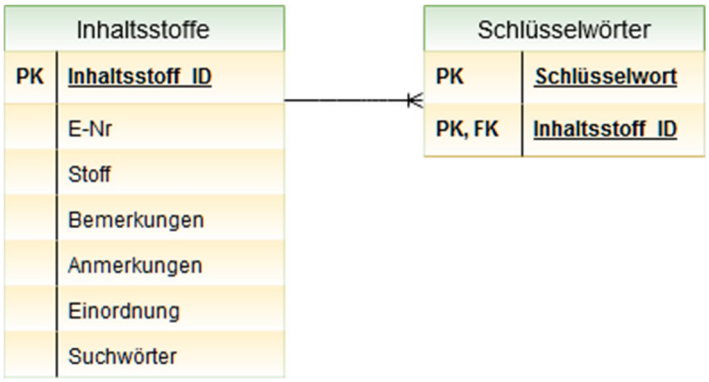

Abbildung 24: Struktur der Datenhaltung für die Inhaltsstoffe

In der Abbildung ist links der Datensatz eines einzelnen Inhaltsstoffes mit seinen Feldern zu sehen. Er wird durch einen eindeutigen, numerischen Wert (PK) identifiziert. Zu jedem Inhaltsstoff gehört eine Gruppe von Daten (rechts), welche auf ihn verweisen (PK, FK) und de facto eine 1:n Beziehung darstellen. Die Auflistung der Schlüsselwörter dient hierbei der Suche nach einem Inhaltsstoff über ein Schlüsselwort, wie oben beschrieben.

Wie bereits im Abschnitt Anforderungen an die Inhaltsstoffe gefordert, wird die Datenhaltung in einem textbasierten Format umgesetzt. Aufgrund der geplanten Nutzung stehen neben einem möglichst nativen und menschenlesbaren Format und Zugriff zwei weitere Aspekte im Vordergrund.

[30] Zum besseren Verständnis werden hier die deutschen Bezeichnungen verwendet. Im Rahmen der Programmierung kommen hingegen die englischsprachigen Bezeichnungen zum Einsatz.

Der erste Aspekt betrifft die Zugriffsmöglichkeiten und Performanz der Datenhaltung. Diese sollte die Möglichkeit einer strukturierten Ablage anbieten. Da ausschließlich lesend auf die Daten zugegriffen wird, ist diese Eigenschaft besonders zu beachten und sollte schnell und möglichst effizient sein.

Der zweite Aspekt betrifft die Unterstützung der im Rahmen dieser Umsetzung angewendeten Sprache Python. Jede verwendete Lösung soll im besten Fall nativ unterstützt werden, oder aber über installierte Bibliotheken verwendbar sein.

4.4.1 Mögliche Datenformate

Als mögliche Kandidaten zur Umsetzung der Datenhaltung bieten sich die etablierten Standards JSON, YAML sowie XML an. Ein Format wie CSV scheidet hingegen aufgrund der fehlenden Möglichkeit, Strukturen effizient abzubilden, aus.

4.4.1.1 JSON - JavaScript Object Notation

JSON[31] besticht durch seine sehr einfache und lineare Struktur. Sie basiert auf einer normalen Textdatei. Somit ist zur Erzeugung oder dem Auslesen keine separate Syntax oder Umgebung erforderlich. Zur Strukturierung und Typisierung werden lediglich sechs Strukturzeichen, sowie drei reservierte Wörter genutzt.

JSON ist in sehr vielen Sprachen entweder direkt oder über Erweiterungen verfügbar. Innerhalb von Python ist es ein natives Format und innerhalb der Standardbibliothek verfügbar. Hierbei entspricht die oberste Objektstruktur von JSON direkt einem Dictionary in Python.

4.4.1.2 YAML - YAML Ain't Markup Language

Ebenso wie zuvor JSON basiert YAML[32] auf einer einfachen Textdatei ohne die Notwendigkeit einer besonderen Umgebung. Hierbei ist laut (Wikipedia-Autoren 25.06.2019) JSON ab der Version 1.2 eine echte Untermenge von YAML und baut auf der Überzeugung auf, dass jede Datenstruktur über assoziative Listen, Arrays sowie Skalaren beschrieben werden kann.

Die Entwickler des Formats sehen im Vergleich zu JSON den Vorteil von YAML hauptsächlich in der Lesbarkeit auf Kosten einer komplexeren Erstellung und schreiben: “In contrast, YAML’s foremost design goals are human readability and support for serializing

[31] Siehe auch http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf. (letzter Abruf am 16.08.2019)
[32] Siehe auch https://yaml.org/spec/1.2/spec.html. (letzter Abruf am 16.08.2019)

arbitrary native data structures. Thus, YAML allows for extremely readable files, but is more complex to generate and parse."[33] (Oren Ben-Kiki, Clark Evans, Ingy döt Net 2009)

Für YAML existieren eine Vielzahl an Sprachbibliotheken, so auch für Python. Damit ist ein Zugriff über Python möglich, jedoch nur über eine kapselnde Bibliothek.

4.4.1.3 XML – Extensible Markup Language

Der größte Unterschied von XML zu den zuvor behandelten Formaten liegt in ihrer Definition als Markup Language. So beinhaltet eine XML[34] Datei neben den eigentlichen Daten eine durch eine feste Syntax definierte Beschreibung der Daten. Hierdurch kann XML per se als komplexer eingestuft werden, auch wenn seine Serialisierung auf Basis von Textdateien im Klartext erfolgt.

Durch den vorgegebenen Aufbau können selbst komplizierte Strukturen klar erfasst und validiert werden. Durch die Verwendung von Transformationsdateien können als XML vorliegende Daten auf sehr unterschiedliche Art zur Ausgabe gebracht werden. Im Gegenzug wird deren Erzeugung und Auslesen deutlich aufwändiger.

Für XML existieren in den meisten Sprachen nativ oder in Form von Bibliotheken die Möglichkeit der Verarbeitung. Innerhalb von Python ist hierzu eine externe Bibliothek notwendig. Somit erfolgt der Zugriff nur indirekt und gekapselt.

4.4.1.4 Wahl des Datenformats

Auf Basis der grundlegenden Anforderung, wie in Abschnitt Anforderungen an die Inhaltsstoffe besprochen, kamen die Formate JSON, YAML sowie XML in die engere Auswahl.

Als wichtigste Kriterien wurden deren Einfachheit, Lesbarkeit und ein möglichst nativer Zugang priorisiert. Aufgrund seiner Komplexität scheidet hierbei XML aus, obwohl es eine sehr mächtige Sprache ist.

JSON hat gegenüber YAML zwei große Vorteile. Zum einen ist es ein in Python nativ verständliches Format und Teil der Standardbibliothek, zum anderen ist es ein leicht zu erstellendes Format. Die sich hierbei in Punkto Verständlichkeit ergebenen Defizite für die menschliche Lesbarkeit haben hier aufgrund der einfachen Datenstruktur nur ein geringes Gewicht.

[33] Im Gegensatz dazu sind die wichtigsten Designziele von YAML die menschliche Lesbarkeit und die Unterstützung bei der Serialisierung beliebiger nativer Datenstrukturen. YAML ermöglicht somit extrem lesbare Dateien, ist aber komplexer zu generieren und zu analysieren. (Übersetzung durch www.deepl.com)

[34] Siehe auch https://www.w3.org/TR/xml. (letzter Abruf am 16.08.2019)

Somit wird innerhalb der konkreten Umsetzung die Datenhaltung in einem JSON-Format erfolgen. Der Zugriff kann somit mit Python eigenen Mitteln umgesetzt werden.

4.4.2 Programmtechnische Umsetzung

Die programmtechnische Umsetzung der Datenhaltung erfolgt in einem eigenen Modul, welches über eine einzelne Schnittstelle mit nur einer Funktion angesprochen wird. Dies ermöglicht eine lockere Bindung und vermeidet eine zu enge Verflechtung mit dem restlichen System.

Als Kompensation für das etwas schlechter zu bearbeitende JSON-Format wird die Konvertierung einer in einem definierten Format vorliegenden Exceldatei als Teil der Komponente angeboten. Somit kann die Konfiguration der Inhaltsstoffe sowohl innerhalb einer Exceldatei als auch einer JSON-Datei erfolgen. Die folgende Abbildung zeigt den grundsätzlichen Aufbau der Komponente:

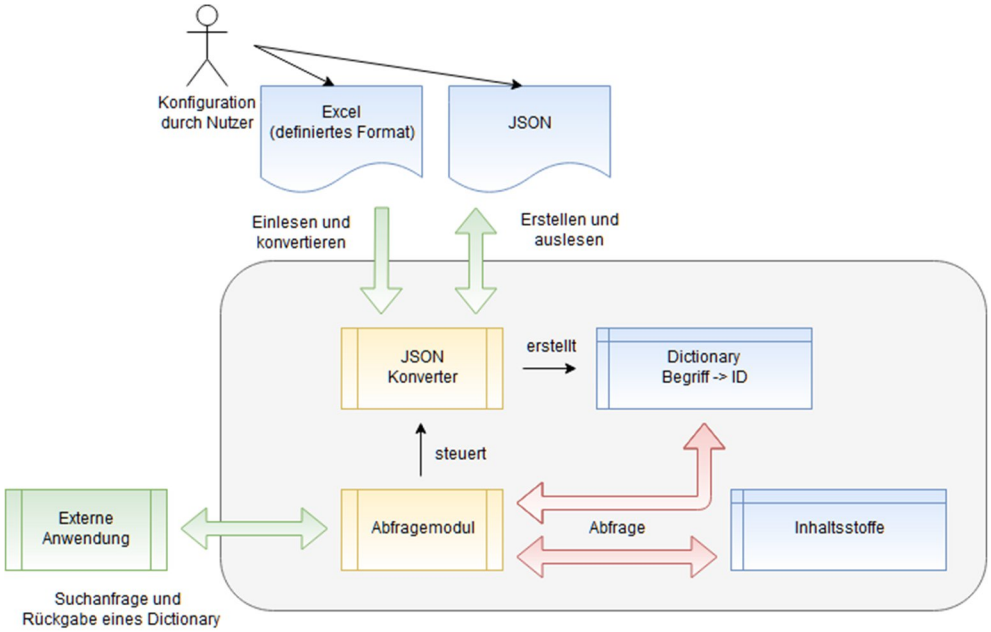

Abbildung 25: Programmtechnische Umsetzung der Datenhaltung

Ein Abfragemodul steuert im Innern das Zusammenspiel des Programmblocks. Über einen JSON-Konverter kann entweder das Einlesen einer Exceldatei und deren anschließende Konvertierung und Speicherung in das JSON-Format, als auch das direkte Auslesen einer vorliegenden JSON-Datei initiiert werden.

Auf Basis der vorliegenden JSON-Datei wird zum einen ein Dictionary zur Suche, zum anderen eine Liste der vorliegenden Inhaltsstoffen generiert. Dieses dient als Datenbasis bei der Rückgabe von Daten. Somit werden die Excel- und JSON-Datei nur initial genutzt.

Es existiert lediglich eine kleine Schnittstelle als API zum Rest der Anwendung. Über diese kann mittels eines Suchwortes eine Suche initiiert oder anhand einer ID Eigenschaften eines Elements zurückgegeben werden.

4.5 Textextraktion und visuelle Umsetzung

Die im Rahmen des Kapitels 3 - Anforderungen, Seite 20 aufgestellten Anforderungen an das umzusetzende System, führt zu den nachfolgend aufgelisteten Aufgaben, auf die im Folgenden detaillierter eingegangen werden soll:

- Vorverarbeitung der Eingabedaten
- Detektion vorhandener Textelemente
- Extraktion darin enthaltener Begriffe
- Abgleich der Begriffe mit der Datenbasis
- Visualisierung der positiv erkannten Wörter

4.5.1 Vorverarbeitung der Eingabedaten

Neuronale Netze haben abhängig von ihrer Konzeption sowohl für das Training als auch für die Vorhersage definierte Anforderungen an ihre Eingaben. Eingaben für neuronale Netze erfolgen immer in numerischer Form und im Falle von TensorFlow häufig in Form mehrdimensionaler Numpy Arrays mit einer definierten Shape.

Aus der Perspektive von Python handelt es sich bei einer Shape um ein Tupel von Ganzzahlen, die jeweils für eine Dimension stehen. So hat ein Bild mit der Weite w und der Höhe h im RGB Farbraum die Shape (w, h, 3). Die 3 steht hierbei für die Farbanteile Rot (R), Gelb (G) und Blau (B). Da ein Netzwerk grundsätzlich mehr als nur eine Eingabe verarbeitet, werden diese Angaben um die Anzahl der Elemente a ergänzt. Somit hat die Eingabe eines Bildes im RGB Format die Shape (a, w, h, 3).

Aus dem zuvor ausgeführten Umstand folgt, dass grundsätzlich jede Bilddatei verarbeitet werden kann, sofern deren Größe und Farbraum angepasst und sie in ein Numpy Array konvertiert werden kann. Dies ist bei Rastergrafiken grundsätzlich der Fall. Beide Begriffe sind somit im Rahmen dieser Arbeit äquivalent. Eine eventuelle Umwandlung des Farbraumes stellt hierbei die größere Anforderung dar, wohingegen bei der Größe häufig ein einfaches Vergrößern oder Verkleinern auf die Zielgröße ausreicht.

Bei den in dieser Arbeit genutzten Eingabedaten wird auf Grund der Vorgaben davon ausgegangen, dass es sich um Rastergrafiken im RGB Format handelt. Dies ermöglicht im Folgenden eine einfachere Weiterverarbeitung ohne die Notwendigkeit, den Farbraum umzuwandeln. Die folgende Abbildung verdeutlicht diese Schritte:

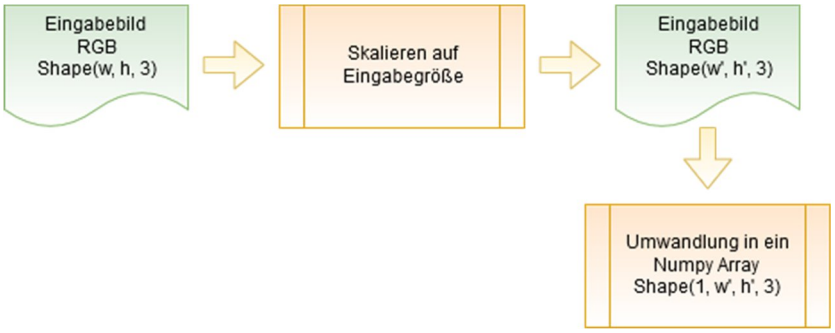

Abbildung 26: Vorverarbeitung des Eingabebildes

Zu sehen ist, wie links ein Eingabebild der Verarbeitung zugeführt wird. Nachdem es zunächst auf eine gültige Größe skaliert wurde, wird es in ein Numpy Array umgewandelt.

In Anlehnung an die in Abschnitt 4.1.2 - Incidental vs. Focused Scene Text, Seite 25 erläuterten Begrifflichkeiten, handelt es sich bei den zu erwartenden Eingaben um Bilder mit incidental scene text. Per Definition handelt es sich somit um Textdarstellungen in einer nicht vorherbestimmbaren Umgebung.

Dies führt dazu, dass Maßnahmen, welche bei bestimmten Bildern zu einer Verbesserung der Ergebnisse führen würden, für andere genau das Gegenteil bewirken könnten. Beispiele solcher Maßnahmen sind Filter zur Kontrasterhöhung oder der Verstärkung existierender Kanten. Sie können somit nur bedingt zielführend eingesetzt werden, da die Art der vorliegenden Bilder nicht im Voraus bekannt ist.

Ausgehend von dieser Feststellung wird im Rahmen dieser Arbeit auf Maßnahmen zur Optimierung der Eingangsbilder verzichtet.

4.5.2 Detektion vorhandener Textelemente

Nach der Aufbereitung werden, die nun in Form eines Arrays vorliegenden Bilder zur Detektion von Texten an ein EAST Netzwerk übergeben. Als Ergebnis wird eine Auflistung potenzieller Textregionen zurückgegeben.

Diese liegen in Form von Koordinaten vor, welche als bounding boxen Bildbereiche im ursprünglichen Bild definieren. Im Folgenden ist dieser Vorgang nochmals visuell dargestellt:

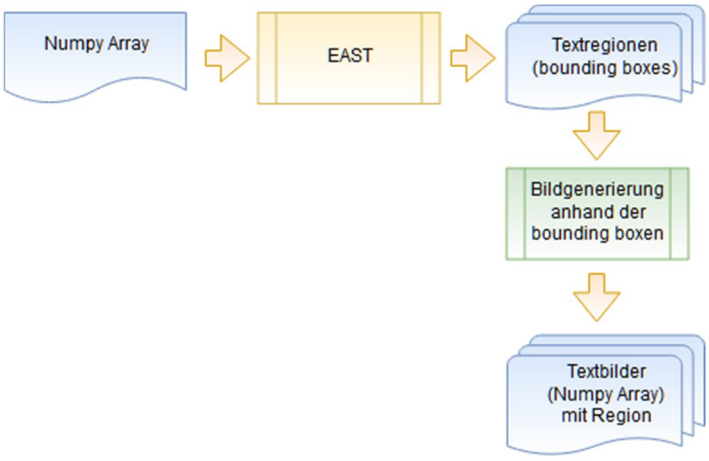

Abbildung 27: Textdetektion und Erstellung einzelner Textbilder

Oben links ist die Zuführung eines einzelnen Bildes in Form eines Arrays zum Netzwerk dargestellt. Die hieraus resultierenden Textregionen rechts werden im Anschluss weiterverarbeitet. Das Ergebnis ist eine Gruppe einzelner Textbildern mit jeweils einem Wort im Kontext seiner Verortung.

4.5.3 Extraktion darin enthaltener Begriffe

Die Extraktion von Texten aus Bildern wird als text recognition oder als optical character recognition (OCR) bezeichnet. Im Rahmen dieses Prozesses wird für jede zuvor gefundene Textregion der darin enthaltene Text vorhergesagt. Hierzu werden die im vorhergehenden Schritt erstellten Textbilder einem CRNN zugeführt. Da diese bereits als Numpy Array vorliegen, entfällt die weitere Aufbereitung.

Als Ergebnis erhalten wir hierdurch eine Auflistung von Begriffen mit einer Verortung in Form der Koordinaten der zugehörigen bounding box. Die nachfolgende Abbildung verdeutlicht diesen Schritt:

Abbildung 28: Extraktion der Texte aus Textbildern

Links ist die Eingabe in Form verorteter Numpy Arrays zu sehen, rechts eine Auflistung vorhergesagter Texte, ebenfalls mit Angabe eines Ortes.

4.5.4 Abgleich der Begriffe mit der Datenbasis

Nachdem die Liste aller in einem Eingangsbild vorhergesagten Wörter mit ihrer Verortung vorliegt, können diese mit den im System hinterlegten Inhaltsstoffen abgeglichen werden. Die hierzu notwendige Komponente und deren Umsetzung wurden bereits

ausführlich unter 4.4.2 - Programmtechnische Umsetzung, Seite 44 vorgestellt und sollen hier nicht wiederholt werden.

Der durchgeführte Abgleich führt zu einer Aufteilung der vorhergesagten Wörter in zwei Listen. Dies sind zum einen Wörter ohne einen Match sowie erfolgreich abgeglichene Wörter. Diese werden um Eigenschaften des Stoffes, hier als Dictionary bezeichnet, erweitert. Die folgende Abbildung visualisiert dies Vorgehen:

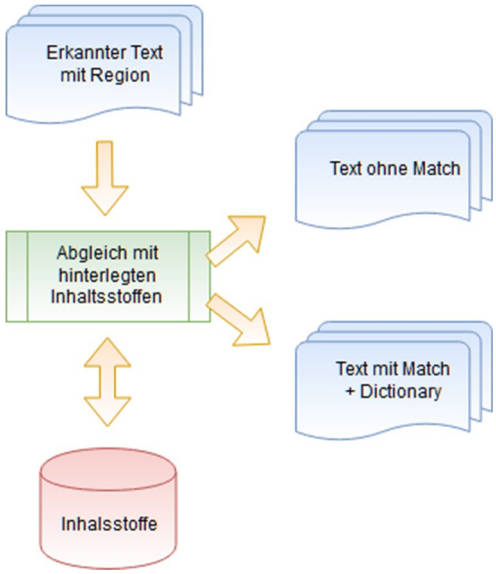

Abbildung 29: Abgleich der gefundenen Texte mit den hinterlegten Inhaltsstoffen

Links oben wird der gesamte erkannte Text als Liste von Einzelworten mit zugehöriger Region mit den hinterlegten Stoffen abgeglichen (hier grün dargestellt). Als Ergebnis des Prozesses entstehen zwei Listen (rechts) von Einträgen mit dazugehörigen Regionen. Die Liste mit den erfolgreich abgeglichenen Stoffen wird darüber hinaus mit den Eigenschaften des Stoffes erweitert.

4.5.5 Visualisierung der positiv erkannten Wörter

In einem abschließenden Schritt wird das Eingangsbild um das Ergebnis der Verarbeitung erweitert und als neues Bild im JPEG-Format unter Verwendung des RGB Farbraums abgelegt. Hierbei werden gefundene Wörter wie folgt annotiert:

- Identifizierte Inhaltsstoffe: rote bounding box und Angabe der ID des Inhaltsstoffes.
- Sonstige gefundene Wörter: grüne bounding box

Die folgende Abbildung verdeutlicht auch diesen Schritt:

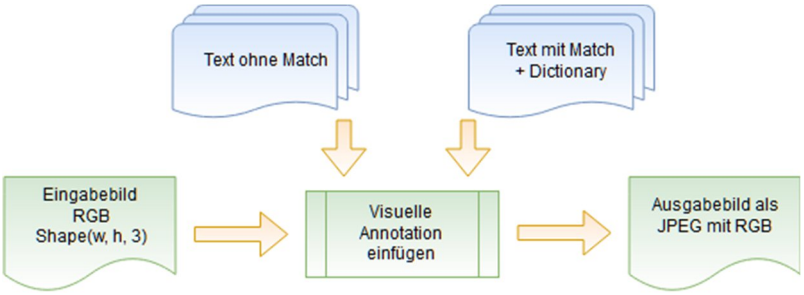

Abbildung 30: Einfügen der visuellen Annotation

Zu sehen ist, wie das ursprüngliche Eingangsbild (links) unter zu Hilfenahme der zuvor generierten Listen (blau) erweitert und im Anschluss als JPEG im RGB Format serialisiert wird.

4.5.6 Zusammenfassung

In diesem Abschnitt wurden das notwendige Vorgehen skizziert, um aus einem gegebenen Bild die darin enthaltenen Texte zu erkennen, anhand einer hinterlegten Datenbasis abzugleichen sowie das resultierende Ergebnis visuell darzustellen.

Auf Basis der im Abschnitt 4.3 - Netzstruktur, Seite 28 geführten Diskussion werden hierfür zwei getrennte und externe, neuronale Netze in Form eines EAST sowie eines CRNN verwendet um die Detektion und Erkennung umzusetzen.

Zusammen mit den vor- und nachgelagerten Arbeiten wird so eine text recognition pipeline erstellt, die zu einem Ende-zu-Ende System zur Lösung der in dieser Arbeit gegebenen Aufgabenstellung führt.

In der folgenden Abbildung wird abschließend nochmal das Zusammenspiel der einzelnen Komponenten und Prozesse zur Lösung des Gesamtproblems dargestellt. Man erkennt von oben links beginnend den Datenfluss der Verarbeitung, wie er zuvor ausführlich behandelt wurde.

In der Mitte rechts sind die beiden in dieser Arbeit verwendeten neuronalen Netze und deren Teilschritte bei der Verarbeitung zu erkennen. Den Abschluss bilden die Prozesse zur visuellen Darstellung der Ergebnisse unten links sowie die Ausgabe des so erzeugten Bildes.

Für eine detailliertere Ausführung wird an dieser Stelle verzichtet und auf die jeweiligen Abschnitte des Kapitels Konzeption verwiesen.

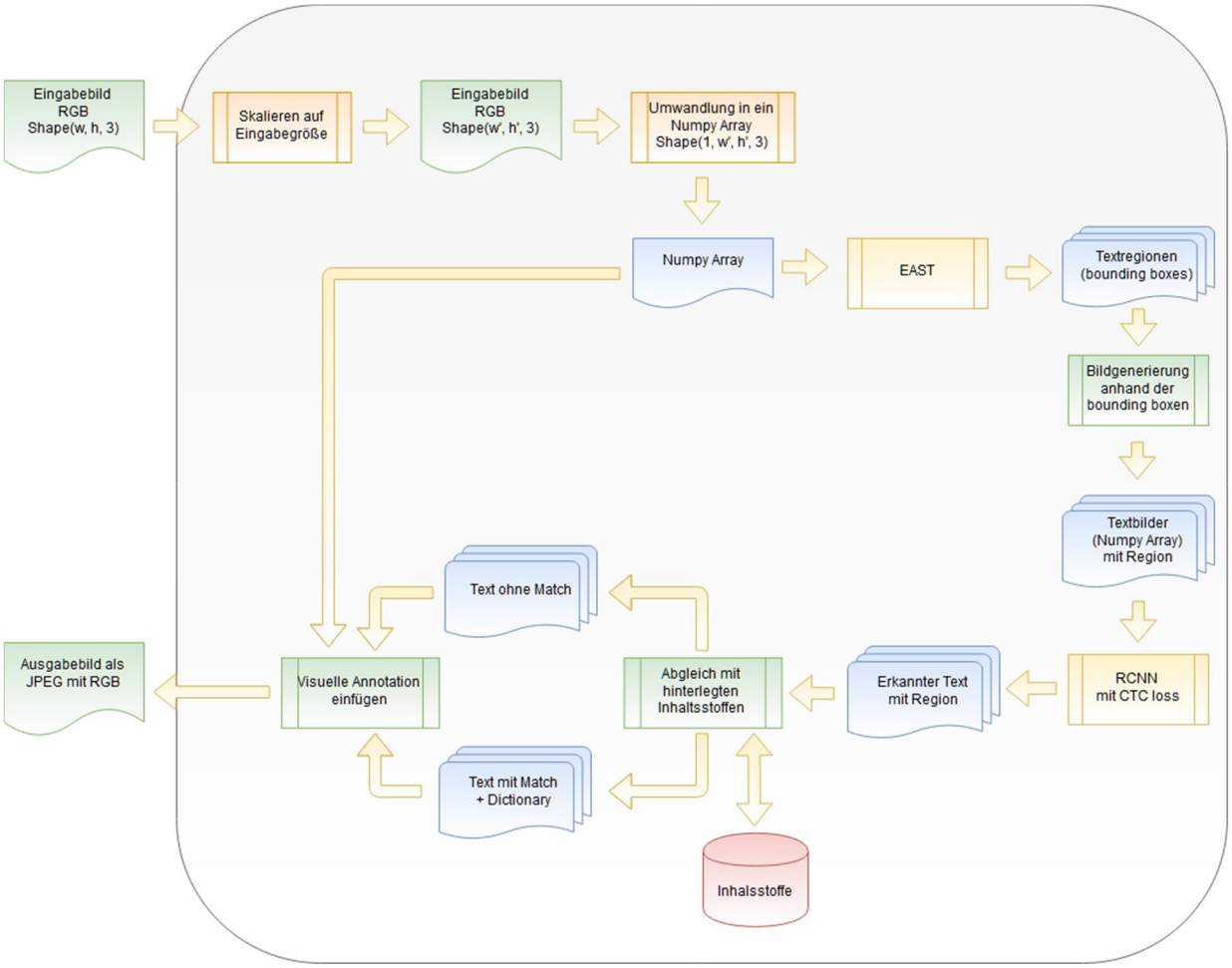

Abbildung 31: Ansicht des Gesamtsystems

4.6 Zwischenfazit

Zu Anfang des Kapitels Konzeption wurde auf einige wichtige Begrifflichkeiten kurz eingegangen, um ein tieferes Verständnis der Thematik an sich und der Problematik der Texterkennung im Besonderen zu vermitteln.

Die folgenden Abschnitte beschäftigten sich mit dem einzusetzenden neuronalen Netz sowie der Umsetzung der Datenhaltung. Hier wurde deutlich, dass eine Kombination zweier Netze als externe Komponenten eine gute Option darstellt. Zum Einsatz kamen ein EAST-Netz sowie ein CRNN. Als Format für die Datenhaltung wurde JSON favorisiert. Weitere Details wie die Programmiersprache oder die eingesetzten Bibliotheken wurden bereits zuvor festgelegt.

Abgeschlossen wurde das Kapitel mit der Konzeption des hier umzusetzenden Systems als Ganzes. Im folgenden Kapitel soll nun das konzipierte System realisiert werden.

5 Realisierung

Dieses Kapitel beschäftigt sich mit der Umsetzung des zuvor ausgearbeiteten Konzeptes. Anders als zuvor geht es hier somit weniger um den logischen Fluss der Daten, sondern vielmehr um die Aufteilung und das Zusammenspiel der einzelnen Komponenten, um dieses Ziel zu erreichen.

Zudem wird kurz auf die bei der Umsetzung aufgetretenen Probleme sowie deren Lösung eingegangen. Sofern auf konkreten Code verwiesen wird, wird dies durch die Nutzung einer eigens hierfür `vorgesehenen Formatierung` verdeutlicht. Ebenso werden Dateinamen **`besonders hervorgehoben`**.

Die dieser Arbeit zugrunde liegende praktische Umsetzung nutzt ein EAST Modell zur Textdetektion sowie ein CRNN zur Texterkennung und setzt somit das zuvor ausgearbeitete Konzept um. Sofern nicht ausdrücklich darauf hingewiesen wird, beziehen sich die folgenden Ausführungen auf die im Abschnitt 5.8 - Aktuelle Musterimplementierung anhand von East und CRNN, Seite 61 vorgestellte Umsetzung.

5.1 Python und Objektorientierung

Obwohl Python die Objektorientierung unterstützt und intern rein objektorientiert arbeitet, gibt es Unterschiede zu „normalen“ objektorientierten Sprachen wie C# oder Java.

So findet man im reinen Python beispielsweise keine Definition für virtuelle, generische oder private Methoden. Eine private Methode in Python wird als Beispiel über die Konvention definiert, diese mit einem Unterstrich zu beginnen.

Um die Sprache dahingehend zu erweitern, existieren Bibliotheken wie OOP Extension[35], welche objektorientierte Eigenschaften über Annotationen verfügbar machen. Diese würden sich jedoch negativ auf die Weitergabe und Lesbarkeit auswirken. Daher werden sie hier nicht verwendet.

5.2 Zusätze zu Demonstrationszwecken

In einem realen Einsatz würde das hier vorliegende System zu einem bestimmten Zeitpunkt starten und anschließend bereit sein, um mit Hilfe seiner Eingabemethoden ein Bild übergeben zu bekommen, es zu bearbeiten und anschließend wieder zurück zu geben. Dieses Einsatzszenario bedarf jedoch eines umgebenden Systems, welches hier nicht existiert.

[35] Die Seite des Pakets findet sich unter https://pypi.org/project/oop-ext. (letzter Abruf am 24.08.2019)

Um dennoch die Funktion in einem realistischen Scenario demonstrieren zu können, wurde dem Projekt die Datei **`demo.py`** hinzugefügt, welche als „Nutzer“ des Systems agiert. Hierzu wurde der Konfiguration zusätzliche Angaben zum Ein- und Ausgabeverzeichnis hinzugefügt.

Für die Evaluierung der vorliegenden Lösung wurden dem Projekt zudem Testbilder hinzugefügt. Gesteuert werden die durchgeführten Tests durch die Datei **`evaluation.py`**. Kapitel 6 - Evaluierung, Seite 63 thematisiert dies vertiefend.

5.3 Die Verzeichnisstruktur

Die aktuelle Verzeichnisstruktur[36] stellt sich alphabetisch wie folgt dar, ist jedoch durch Änderung der zugrunde liegenden Konfiguration zu beeinflussen:

- ocr_system – Hauptverzeichnis der umgesetzten Lösung
- annotation_constants – Verzeichnis mit Konstanten zur Visualisierung
- bridges –Übergang zu den eingesetzten, externen Modellen
- data – Datenbasis mit den Daten zum Abgleich der gefundenen Texte
- evaluation – Verzeichnis mit Testbildern und -ausgaben
- income – Verzeichnis zur Eingabe der zu verarbeitenden Bilder
- license – Verzeichnis für die Lizenz
- outcome – Verzeichnis zur Ausgabe der verarbeiteten Bilder

5.4 Zentrale Konfiguration

Der Aufbau der Anwendung wird mit Hilfe einer zentralen Konfiguration gesteuert, welche in der Datei **`constant.py`** festgelegt ist. Primär handelt es sich hierbei um Pfad- und Dateiangaben. Im Detail sind dies in der Reihenfolge des Eintrags:

- INPUT_DIR –definiert das Eingabeverzeichnis
- OUTPUT_DIR – definiert das Ausgabeverzeichnis
- EVALUATION_DIR – definiert das Arbeitsverzeichnis für die Evaluierung
- DATABASE_JSON – definiert die JSON-Datei der Datenbasis
- DATABASE_EXCEL – definiert die optionale Exceldatei der Datenbasis
- BRIDGES_JSON – definiert die JSON-Konfiguration der Bridges

5.5 Übersicht der Komponenten

Nachdem auf die Konfiguration und den Verzeichnissen der Anwendung eingegangen wurde, sollen nun auf die restlichen Komponenten des Systems näher betrachtet werden.

[36] Im Anhang findet sich eine ausführliche Auflistung der Verzeichnisstruktur

Insgesamt verfügt das hier vorgestellte System, abgesehen von Bridges[37], im Kern über fünf einzelne Komponenten, welche im Zusammenspiel zu einer Lösung führen. Diese sind im Einzelnen in der Reihenfolge ihrer Wichtigkeit:

- Scanner
- Detector
- Recognizer
- Ingrediens
- BoundingBoxImageHandler

Das Zusammenspiel dieser Komponenten wird in der folgenden Abbildung verdeutlicht:

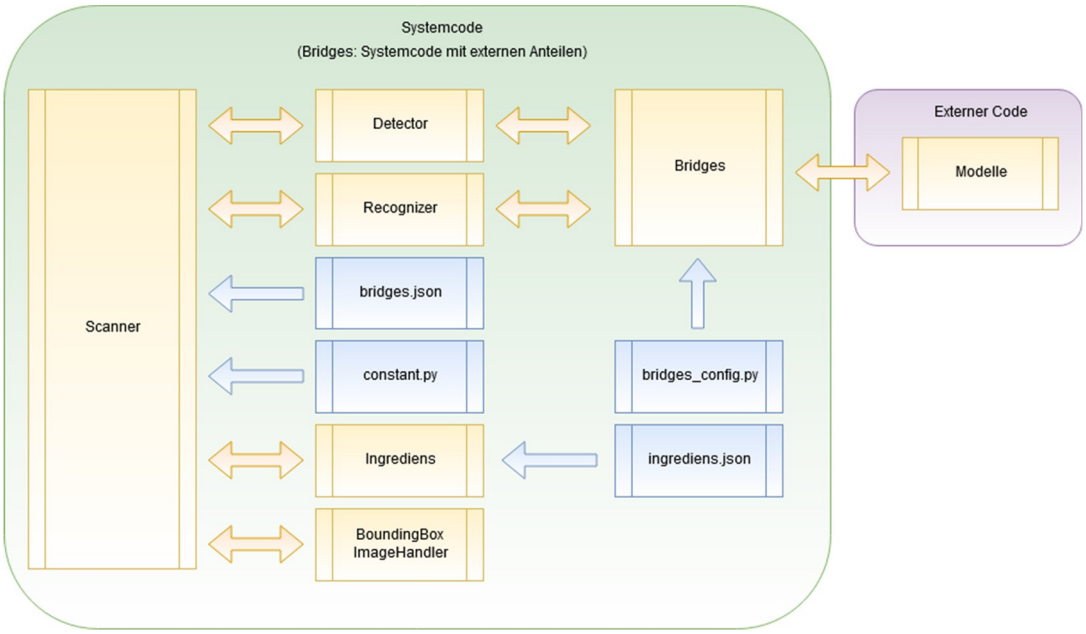

Abbildung 32: Übersicht Gesamtsystem

In der Abbildung ist zu sehen, wie die Klasse Scanner die übrigen Klassen nutzt und auf welchen Wegen das System konfiguriert wird, hier in blau eingetragen.

Bevor auf die oben rechts zu sehende Komponente Bridges und ihr Verhältnis zu externem Code, hier violett hinterlegt, näher eingegangen wird, sollen zunächst die restlichen Komponenten des Systems, hier in grün hinterlegt, betrachtet werden.

5.5.1 Klasse Scanner

Die Klasse Scanner ist die zentrale Klasse des Systems und verantwortlich für die korrekte Verarbeitung des jeweiligen Bildes. Die wichtigste Methode ist `scann(…)`, welche als Parameter eine Rastergrafik in Form eines Numpy Arrays übergeben bekommt.

[37] Auf Bridges wird im nachfolgenden Abschnitt 5.6 - Einbindung von Modellen, Seite 56 näher eingegangen.

Mit Hilfe einer Instanz der Klasse Detektor werden mögliche Textbereiche zunächst identifiziert und im Anschluss mit Hilfe einer Instanz der Klasse Recognizer in einen Text überführt.

Die so erkannten Wortvorkommen werden abschließend über eine Instanz der Klasse Ingrediens mit den zugrunde liegenden Lebensmittelinhaltsstoffen abgeglichen und gegebenenfalls visuell hervorgehoben.

Die Klasse BoundingBoxImageHandler stellt hierbei für den gesamten Workflow die notwendigen Funktionalitäten zur Bearbeitung der Bilder zur Verfügung.

Über die Methode `auto_scann(…)` ist es möglich, eine im Dateisystem liegende Datei direkt zu bearbeiten. Hierfür werden der Methode die Pfade zur Ein- und Ausgabedatei übergeben. In diesem Fall erfolgen das Laden und Speichern durch die Klasse Scanner.

5.5.2 Klasse Detector

Bei der Klasse Detektor handelt es sich um ein abstraktes Objekt, welches Textbereiche in einem Bild erkennen kann, im Folgenden auch als Textdetektor bezeichnet. Sie kennt nur die Methode `scann(…)`, welche ein Bild in Form eines Numpy Arrays entgegennimmt und eine Liste von Boxen zurückgibt. Eine Box ist hierbei definiert als eine Auflistung von Punkten, die ein Polygon beschreiben.

Intern wird in Abhängigkeit der in **`bridges.json`** gehaltenen Konfiguration eine Bridgeklasse[38] geladen, über der die Anforderung an ein externes neuronale Netz weiter geleitet wird. Somit dient die Klasse Detektor auch der Entkopplung des Systems von den konkret zum Einsatz kommenden Modell.

Die Instanziierung erfolgt über die Klassenmethode `instance(…)`, welcher der Pfad zur maßgeblichen Konfigurationsdatei **`bridges.json`** übergeben wird.

5.5.3 Klasse Recognizer

Bei der Klasse Recognizer handelt es sich um ein abstraktes Objekt, welches Text in einem Bild erkennen und interpretieren kann, im Folgenden auch als Texterkenner bezeichnet. Sie verfügt wie die Klasse Detector nur über die Methode `scann(…)`, welche wiederum ein Bild in Form eines Numpy Arrays entgegennimmt. Als Rückgabe wird ein String übergeben, der den in dem Bild erkannten Text darstellt.

Wie schon zuvor dient auch diese Klasse der Entkopplung des Systems von dem konkret zur Anwendung kommenden externen Netz und verwendet hierzu die Instanz einer

[38] Auf Bridges wird im nachfolgenden Abschnitt 5.6 - Einbindung von Modellen, Seite 56 näher eingegangen.

Bridgeklasse. Auch hier erfolgt die Steuerung über **`bridges.json`**. Die Instanziierung erfolgt analog zur Klasse Detector.

5.5.4 Klasse Ingrediens

Aufgabe der Klasse Ingrediens ist die Kapselung der im JSON-Format vorliegenden Datenbasis der Inhaltsstoffe. Neben Methoden zum manuellen Erstellen der Datenbasis stellt sie mit der Klassenmethode `convert(…)` Funktionalität zur Verfügung, um die Datenbasis anhand eines Excelfiles zu erstellen. Hierzu werden der Pfad der zugrunde liegenden Excel- sowie der zu erstellenden JSON-Datei übergeben.

Die Einbeziehung eines Excelfiles als Basis ist hierbei ein Entgegenkommen in Richtung der Usability, da dies Format etwas übersichtlicher ist. Grundsätzlich ist lediglich eine JSON-Datei in einer gültigen Formatierung notwendig.

Die Instanziierung erfolgt über die Methode `instance(…)`, welcher der Pfad der zugrunde liegenden JSON-Datei sowie ein boolesche Parameter usePatch übergeben wird. Über die Steuerungsfunktion des Parameters wird im Abschnitt 5.7.4 - Möglichkeiten zur Verbesserung, Seite 60 eingegangen.

Über die Methode `contains()` kann abgefragt werden, ob der übergebene String vorhanden ist. Als Rückgabe werden ein boolescher Wert sowie eine Zahl zurückgegeben. Der Wert gibt Auskunft, ob der String vorhanden ist, die Zahl repräsentiert im Erfolgsfall die ID, ansonsten eine -1. Die Rückgabe einer ID ermöglicht den Zugriff auf das jeweilige Element und seinen Eigenschaften.

5.5.5 Klasse BoundingBoxImageHandler

Die Klasse stellt mit ihren statischen Methoden Hilfsfunktionalität zur Bearbeitung von Bildern unter anderen im Zusammenhang mit Boxen zur Verfügung. Wie zuvor ist auch hier eine Box als eine Auflistung von Punkten, die ein Polygon beschreiben, zu verstehen. Übergebene Bilder werden als Numpy Arrays betrachtet und bearbeitet.

Wichtige Funktionen sind hierbei unter anderen die Erzeugung von Detailbildern anhand von Boxen mit Hilfe von `get_subimage(…)` sowie die Ausgabe von Texten im Kontext einer Box mit Hilfe der Methode `put_text(…)`.

Die Klasse nutzt hierbei im starken Maße die Fähigkeiten von OpenCV[39] und ist im Gegenzug von dessen Fähigkeiten abhängig. Eine sich hieraus ergebene Besonderheit ist die Unfähigkeit, Umlaute auszugeben. Daher werden diese im auszugebenen Text durch ihre Entsprechungen ersetzt.

[39] Die Seite des Paketes findet sich unter https://pypi.org/project/opencv-python. (letzter Abruf am 24.08.2019)

5.6 Einbindung von Modellen

Im Rahmen dieser Arbeit stehen die Begriffe Modell und neuronales Netz gleichwertig für ein neuronales Netz sowie sämtlichen, für dessen Betrieb notwendigen Code.

Wie schon zuvor im Abschnitt 4.3 - Netzstruktur, Seite 28 ausgeführt, wird im Rahmen dieser Arbeit der Einsatz eines EAST Modells zur Textdetektion sowie eines CRNN zur Texterkennung als externe Komponenten präferiert. Darüber hinaus soll die Einbindung externer Netze offen gestaltet sein und so einen einfachen Wechsel ermöglichen.

In der Praxis gestaltet sich ein solcher Wechsel in der Hauptsache aus zwei Gründen komplexer:

- frei verfügbare und vortrainierte Modelle sind sehr unterschiedlich aufgebaut
- durch den unterschiedlichen Aufbau ergeben sich große Unterschiede in deren Ansteuerung, der Datenübergabe sowie den ausgegebenen Ergebnissen

Durch die genannten Umstände ist es nicht möglich, ein einzelnes Interface zu entwickeln, welches direkt alle Modelle abdeckt. Eine zu starke Anpassung der verwendeten Modelle widerspricht jedoch dem Ziel eines einfach durchzuführenden Austausches sowie der einfachen Verwendung externer Lösungen. Zudem besteht die Gefahr, dass deren vortrainierte Gewichte nicht mehr nutzbar sind.

Aus diesem Grunde wurde für das hier vorliegende System das Konzept der Bridge eingeführt, welches im Folgenden näher beschrieben wird.

5.6.1 Bridge

Eine Bridge im Sinne dieser Arbeit ist eine Klasse, welche als Fassade zu einem hinterlegten Modell wirkt. Es existieren zwei Typen dieser Klasse:

- Bridge zu einem Textdetektor
- Bridge zu einem Texterkenner

Zu jeder Zeit ist nur eine Bridge je Typ innerhalb des Systems aktiv. Die Konfiguration hierzu erfolgt mit Hilfe der in **`constant.py`** mit dem Schlüssel BRIDGES_JSON festgelegten JSON-Datei und den darin enthaltenen vier Schlüsseln:

- detector_module — Python Datei der Bridge für den Detektor
- detector_class — Name der zugehörigen Python Klasse
- recognizer_module — Python Datei der Bridge für die Erkennung
- recognizer_class — Name der zughörigen Python Klasse

Jede als Bridge eingesetzte Klasse verfügt über einen parameterlosen Konstruktor sowie einer Methode `scann(…)`, welcher ein Bild zur Verarbeitung übergeben wird. Die Rückgabe dieser Funktion richtet sich nach deren Einsatz.

Wird sie als Bridge zu einer Klasse Detector verwendet, so gibt sie eine Liste von Boxen zurück. Jede Box beinhaltet eine Liste von Punkten, die einen Polygon beschreiben und einen Ort mit Text innerhalb des übergebenen Bildes identifizieren. Wird sie als Bridge einer Klasse Recognizer verwendet, so gibt sie den aus dem übergebenen Bild vorhergesagten Text zurück.

Für jedes vom System verwendete Modell muss eine eigene Bridge implementiert werden. In Richtung des Systems müssen die zuvor genannten Voraussetzungen erfüllt werden. Intern ist die Bridge für drei Aufgaben verantwortlich:

- Initialisierung des zugrunde liegenden Modells und das Laden vortrainierter Gewichte
- Weitergabe der Systemanfragen in einer für das jeweilige Modell verständlichen Art und Weise
- Rückgabe und Anpassen der Ausgabe des Modells an die vom System erwartete Art und Weise

Hierbei kann zum einen Code des verwendeten Modells selbst zum Einsatz kommen, zum anderen kann dies vollständig durch eigene Implementierungen umgesetzt werden. Für alle Bridgeklassen existiert mit der Datei **`bridges_config.py`** eine gemeinsame Konfigurationsdatei, in welcher änderbare Einstellungen hinterlegt werden können. Die folgende Abbildung verdeutlicht die genannten Zusammenhänge:

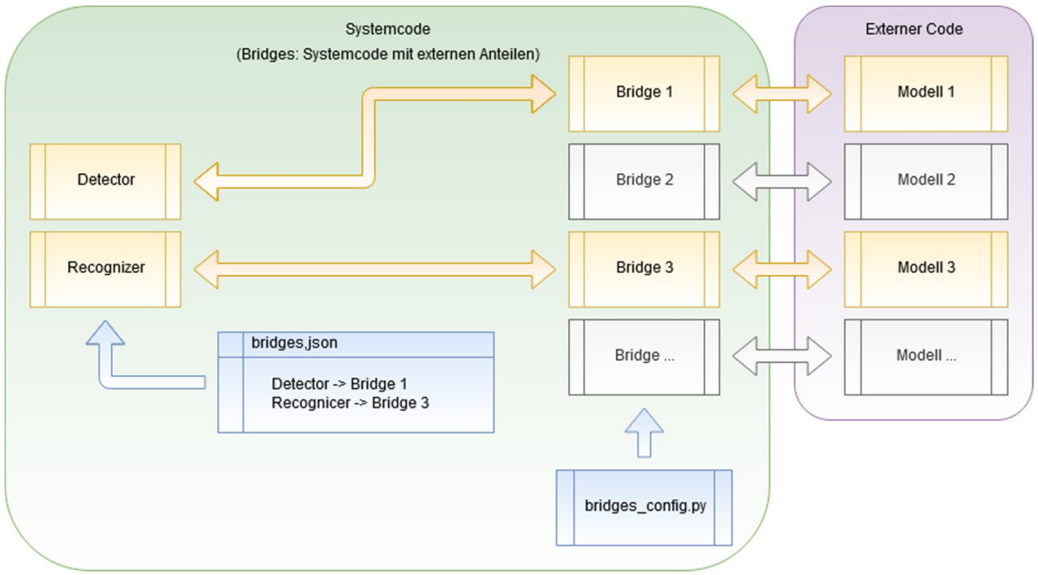

Abbildung 33: Konzept der Bridges

Klar sind die dem System zugehörigen Komponenten, hier grün hinterlegt, zu erkennen. Dem entgegen stehen die externen Modelle, hier mit violettem Hintergrund. Die Verbindung wird über Bridges hergestellt, die Steuerung erfolgt über Konfigurationsdateien (blau). In dem dargestellten Fall würde das Modell 1 als Detektor, Modell 3 als Texterkenner fungieren.

Als Namenskonvention wird in dieser Arbeit die Kennzeichnung Bridges mit der Endung „_bridge“ favorisiert. Andere Benamungen sind auf Grund der freien Konfiguration ebenso möglich.

5.6.2 Modelle

Im Rahmen des hier vorgestellten Systems werden Modelle grundsätzlich als externe Komponenten betrachtet. Dies gilt auch für den Fall, dass selbst erstellte und trainierte neuronale Netz zum Einsatz kommen.

Der Speicherort der Modelle befindet sich in dem Unterordner models des Verzeichnisses bridges. Jedes hinterlegte Modell wird hierbei wiederum in einen eigenen Unterordner, im Folgenden als Modellordner bezeichnet, abgelegt. Jeder Modellordner verfügt über zwei Unterordner.

Im Unterordner pretrained finden sich vortrainierte Gewichte, serialisierte Modelle und Modellstrukturen, sofern sie vorhanden sind. Im Unterordner license befindet sich die jeweiligen Lizenzinformationen des Modells.

Da die Verwendung vollständig über die zugehörige Bridge erfolgt, sind die zuvor angegebenen Benamungen lediglich als ein Vorschlag für eine sinnvolle Unterteilung anzusehen. Praktisch kann hiervon abgewichen werden.

Im Abschnitt 5.8 - Aktuelle Musterimplementierung anhand von East und CRNN, Seite 61 wird auf die im Rahmen dieser Arbeit verwendeten externen Modelle näher eingegangen. Hierbei wurde der Code dieser Modelle bis auf die zur Funktion notwendigen Bestandteile verkleinert, um sie übersichtlicher zu gestalten.

5.7 Problematik der semantischen Zuordnung

Im Zuge der vorliegenden Umsetzung wurden eine Reihe von Problemen identifiziert, welche eine korrekte semantischen Zuordnung erschwerten. Diese sollen hier kurz thematisiert werden.

Die semantische Zuordnung ist ein Zusammenspiel der Klassen Detector, Recognizer sowie Ingrediens. Der Erfolg oder Misserfolg bei der Erkennung gefundener Wörter hängt somit davon ab, wie gut diese Klassen zusammenarbeiten. Zusammenfassend

konnten hier die folgenden Probleme beobachtet werden, auf die im Anschluss eingegangen wird:

- nicht korrekt erkannte Textgrenzen
- nicht korrekt erkannter Text
- nicht korrekt erkannter Inhaltsstoff

5.7.1 Nicht korrekt erkannte Textgrenzen

Hierbei handelt es sich um ein Problem der Detektion von Text und hängt direkt von dem verwendeten neuronalen Netz ab. Es ist somit direkt vom Training und der konkreten Umsetzung des verwendeten Models sowie der Vor- und Nachbearbeitung der Daten abhängig.

Häufig äußern sich Fehler in Form von abgeschnittenen Wörtern oder dem Nichterkennen von zusammen gehörigen Begriffen. Lücken innerhalb des Begriffes werden hier als Wortgrenzen aufgefasst. Ein Beispiel für diese Art von Problemen zeigt die folgende Abbildung:

Abbildung 34: Nicht korrekt erkannte Textgrenze

Zu sehen ist in der Abbildung ein Bildausschnitt mit dem Wort „Benzoesäure“, welches von dem verwendeten Textdetektor nicht korrekt erkannt und als Folge falsch markiert wurde (grüner Rand). Der vorhergesagte Text war in diesem Fall das Wort „Benzoesaun“, da erschwerend ein nicht erkannter Umlaut hinzukam.

Dies Verhalten ist für die hier bearbeitete Aufgabenstellung problematisch, steht jedoch nicht zwangsläufig für ein schlecht umgesetztes Netz.

5.7.2 Nicht korrekt erkannter Text

Ein weiteres Problem betrifft die Vorhersage des im Bild enthaltenen Textes. Hier konnte beobachtet werden, dass es regelmäßig zu einer Nichterkennung von Umlauten und anderen „typisch“ deutschen Zeichen kommt. Auch werden ursprünglich korrekt erkannte Lücken im Rahmen der nachfolgenden Optimierung als „fehlerhafte Lücken“ entfernt, und somit falsch ausgegeben. Ein Beispiel für dies verhalten zeigt die folgende Abbildung:

Abbildung 35: Nicht korrekt erkannter Text

Das obenstehende Bild wurde von dem in dieser Anwendung verwendeten CRNN Netzwerk als Text „1120“ vorhergesagt. Das Angeschnittene „E“ im Bild wurde als „1“ interpretiert, die Lücke dahinter entfiel ganz.

Besonders die auf Umlaute und Sonderzeichen basierenden Fehler sind dem Training anhand englischsprachiger Texte geschuldet, auf denen vortrainierte Modell in den meisten Fällen basieren. Wird ein Netz nicht anhand von Umlauten trainiert, wird es diese nicht erkennen und die vorgefundenen Punkte als Störungen interpretieren und in Folge ignorieren. Auch diese Eigenschaft steht nicht für eine grundsätzlich schlechte Umsetzung, sondern kann durchaus seine Berechtigung haben.

5.7.3 Nicht korrekt erkannter Inhaltsstoff

Selbst für den Fall, dass eine Textregion sowie der enthaltene Text korrekt vorhergesagt wurden, kann es dazu kommen, dass ein für einen Menschen offensichtlich in die Gruppe der Inhaltsstoffe fallender Begriff nicht als solcher erkannt wird.

Die hierfür verantwortlichen Gründe sind vielfältig und können hier nicht abschließend behandelt werden. Wichtige Gründe sind Unterschiede in der Schreibung sowie die Verwendung von Abkürzungen. Die Verwendung nicht deutscher, sondern alternativer Bezeichnungen kommt erschwerend hinzu. Ein Beispiel hierfür zeigt die folgende Abbildung:

Abbildung 36: Nicht korrekt erkannter Inhaltsstoff

Der gezeigte Text wurde als „e150acr“ vorhergesagt und nicht erkannt. Hierbei handelt es sich im Original um eine Gruppe von Bezeichnern (E 150a, E 150c sowie E 150d), welche jeweils eine bestimmte Art von Zuckerkulör bezeichnen. Selbst bei korrekter Erkennung als „a150acd“ wäre eine Zuordnung auch bei Verwendung erweiterter Schlüsselwörter durch eine einfache Suche nicht zwingend erfolgreich, da aktuell drei unterschiedliche Einträge existieren, welche gleichermaßen korrekt wären.

5.7.4 Möglichkeiten zur Verbesserung

Um den zuvor ausgeführten Problematiken entgegen zu wirken, bieten sich eine Reihe von Möglichkeiten an.

So bietet die hier vorgestellte Lösung mit dem Konzept der Bridge eine grundsätzliche und einfache Möglichkeit, verschiedene Modelle mit sehr wenig Aufwand einbinden zu können. Dies eröffnet zwei grundlegende Lösungswege:

Zum einen kann ein für das hier anstehende Problem besser geeignete Modell gesucht und eigebunden werden. Durch die Teilung von Detektion und Erkennung sowie der geringen Kopplung zur Anwendung ist hier grundsätzlich jede mögliche Kombination bis hin zum Vorhalten multipler Netze für verschiedene Szenarien denkbar. Zum anderen kann ein bereits gut arbeitendes System im Rahmen eines Finetunings speziell an die geforderten Anforderungen angepasst werden.

Eine weitere Möglichkeit zur Optimierung bietet die Klasse Ingrediens. In der einfachsten Form erfolgt die Suche nach einem Inhaltsstoff anhand des Begriffes sowie einer vollständigen Übereinstimmung. Der Erfolg dieser Suche kann verbessert werden, wenn man neben den eigentlichen Namen der Inhaltsstoffe weitere textliche Entsprechungen in Form von Schlüsselwörtern hinzufügt. Hierbei steigt jedoch auch die Gefahr nicht korrekter Zuordnung und die Qualität hängt stark von den gewählten Schlüsselbegriffen ab.

Aktuell werden die folgenden Verfahren angewendet, um die Treffergenauigkeit bei einem Abgleich mit der Datenbank zu erhöhen:

- Suchwörter werden in Kleinschreibung übergeben und nur anhand dessen verglichen
- E-Nummern werden mit und ohne Bindestrich zwischen dem führenden „E“ und der Zahl gesucht, enthaltenen Leerzeichen entfernt.
- in Namen und Suchwörtern enthaltene Umlaute „ä“, „ö“ und „ü“ werden vor dem Vergleich in „a“, „o“ und „u“ umgewandelt. Dies wirkt sich vor allem bei anhand der englischen Sprache trainierten Modellen aus, birgt jedoch auch die Gefahr neuer und anders gelagerter Fehler bei der Erkennung. Dieser Modus muss daher mit der Option `usePatch = True` explizit eingeschaltet werden.

Eine weitere, jedoch nicht implementierte Möglichkeit ist die Suche nach partiellen Übereinstimmungen. Hierdurch können Fehler als Folge nicht exakter Detektion entgegengewirkt werden. Nach Meinung des Verfassers ist es hier jedoch sinnvoller, auf ein anderes Modell zur Texterkennung auszuweichen, oder aber ein vorhandenes weiter zu trainieren.

5.8 Aktuelle Musterimplementierung anhand von East und CRNN

In den voran gegangenen Abschnitten wurde die vorliegende Lösung als ein System beschrieben, welches seine Aufgaben auf Basis unterschiedlicher Netze durchführen kann. Abschließend soll nun auf die hier konkret vorliegende Implementierung kurz eingegangen werden.

Als Detektor wird eine von Jan Zdenek veröffentlichte Implementierung eines EAST Modells[40] verwendet, für den Part der Texterkennung eine ebenfalls von ihm veröffentlichte Implementierung eines CRNN Netzes[41]. Für beide Lösungen sind jeweils vortrainierte Modelle verfügbar.

Die Wahl fiel aus mehreren Gründen auf diese Netze. Zum einen entsprechen sie den drei Grundforderungen dieser Arbeit, namentlich der Nutzung von TensorFlow sowie die Umsetzung in Python und Keras, zum anderen bieten sie den Download vortrainierter Netze an. Ein weiterer wichtiger Grund ist jedoch auch der Aufbau der Lösung an sich. Die Umsetzungen sind jeweils strukturiert, übersichtlich und das Projekt an sich wohl strukturiert.

Die bei dieser Arbeit stattgefundene Reduktion nicht benötigter Codebestandteile stellt sich somit nicht als notwendig dar, sondern als eine optionale Maßnahme zur Erhöhung der Übersichtlichkeit.

5.9 Alternative Implementierung mit OpenCV

Zur Demonstration der Möglichkeit, beliebig weitere Netze innerhalb des vorliegenden Systems einbinden und nutzen zu können, wurde dem System eine alternative Implementierung des EAST Detektor hinzugefügt.

Bei dem Code handelt es sich um ein von Abhishek Singh veröffentlichten Mustercode[42], der die Nutzung der in OpenCV[43] verfügbaren EAST Implementierung zeigt. Die hierzu notwendige Bridge befindet sich in der Datei **east_open_cv_bridge.py**. Die im Modelverzeichnis[44] hinterlegten Gewichte sind Teil des Mustercodes.

Das neue Modell kann einfach durch Anpassung der **`bridges.json`** verfügbar gemacht werden. Hierfür sind lediglich die Einträge für detector_module und detector_class mit den bereits hinterlegten Alternativen wie folgt auszutauschen:

1. detector_module_alternative in detector_module umbenennen
2. detector_class_alternative in detector_class umbenennen

Im Anschluss kann das System wie gewohnt gestartet werden und das neue Modell wird zur Textdetektion verwendet.

[40] Quellcodeverzeichnis Code 01
[41] Quellcodeverzeichnis Code 02
[42] Quellcodeverzeichnis Code 03
[43] Die Seite des Pakets findet sich unter https://pypi.org/project/opencv-python. (letzter Abruf am 24.08.2019)
[44] Der vollständige Pfad lautet bridges/models/east_open_cv/pretrained/frozen_east_text_detection.pb

6 Evaluierung

Nachdem in den zuvor genannten Kapiteln auf die Konzeption und Umsetzung eingegangen wurde, beschäftigt sich dieses Kapitel mit der Evaluierung der vorgestellten Lösung. Zwei Aspekte stehen hierbei im Vordergrund:

- die Abfrage der Datenbasis anhand von Suchbegriffen
- die Qualität der Texterkennung

Mehrfach wurde hervorgehoben, dass die zur Anwendung kommenden Modelle primär als externe Komponenten angesehen werden. Somit definiert sich das vorliegende System grundsätzlich als ein Konsument der durch die Modelle gegebenen Funktionalität. Die Modelle hingegen sind austauschbar.

Nach Ansicht des Verfassers ist es daher wenig zweckmäßig, die verwendeten Modelle einzeln einer direkten Betrachtung zu unterziehen. Die hier zu gewinnenden Informationen können in den meisten Fällen direkt aus der Beschreibung eines Modells entnommen werden. Einen wesentlich wichtigeren Aspekt kommt der Zusammenarbeit der jeweils verwendeten Modelle innerhalb des Gesamtsystems zu. Diese soll daher im Mittelpunkt stehen.

Aus Gründen der Vergleichbarkeit ist es sinnvoll, bei einem Wechsel der verwendeten neuronalen Netze eine möglichst gleichartige und somit vergleichbare Wiederholung sämtlicher Schritte der Prüfung zu ermöglichen. Dies führt zu der Forderung, das System selbst entsprechend zu erweitern.

Als zu prüfende Kernkompetenzen des Systems können die aufgeführten drei Aufgaben benannt werden:

- Abgleich von Begriffen mit der verwendeten Datenbasis
- Detektion und Erkennen von Texten
- Erkennen von Buchstaben und Sonderzeichen

6.1 Generierung von Testbildern

Wie bereits zuvor ausgeführt, sollen die durchgeführten Test möglichst vergleichbar sein. Daher werden zwei Bildvorlagen mit Texten eingeführt. Eine beinhaltet einen einfachen Satz in verschiedenen Ausführungen, die andere Nummern, eine Reihe von Sonderzeichen sowie Umlaute und Variationen einer E-Nummer. Die Textfarbe ist jeweils schwarz. Die folgende Abbildung verdeutlicht dies:

Abbildung 37: Testbilder für Text und Sonderzeichen

Zu sehen ist auf der linken Seite des Bildes, wie ein Satz auf verschiedene Arten dargestellt wird. Im Detail sind verschiedene Größen und Verzerrungen erkennbar wie sie bei Aufnahmen eines Etikettes vorkommen können. Auf der rechten Seite sind neben Zahlen verschiedene Sonderzeichen, Umlaute sowie Variationen der E-Nummer zu erkennen. Kernaspekt hier ist die grundsätzliche Fähigkeit, diese zu erkennen, weshalb auf Variationen verzichtet wurde.

Um nicht scharfe Bilder zu simulieren, wird eine weitere Version der Textvorlage mit einem verschwommenen Rand erzeugt. Diese konnte durch eine Überlagerung des Ursprungsbildes mit einer zweiten, weichgezeichneten Version realisiert werden.

Abbildung 38: Verschwommene Testbilder für Text und Sonderzeichen

Die so entstandenen Bilder dienen in den folgenden Tests als Vorlagen für die Textdetektion und -erkennung.

Um die Besonderheiten eines Scene Textes zu simulieren, wird von jeder Version eine weitere, mit einem Hintergrundbild hinterlegte, Variante eingeführt. Als Hintergrund wurde das folgende Waldbild[45] mit einer Brücke gewählt:

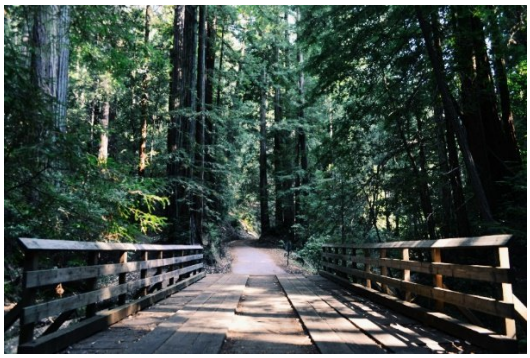

Abbildung 39: Hintergrundbild zur Simulation eines Scene Textes

Deutlich sind in der Abbildung oben eine Vielzahl an Farbnuancen, Schatten, Flächen und Kanten zu erkennen. Sie bedeuten sowohl bei der Detektion als auch der Erkennung eine große Herausforderung.

Zum Test der reinen Texterkennung werden aus den Zeilen der zuvor erzeugten Textbilder neue Einzelbilder generiert. Dies zeigt das folgende Bild beispielhaft:

Abbildung 40: Ein Satz in seinen vier Versionen

Deutlich ist in dem Bild der Schriftzug mit scharfer Kante ganz oben, sowie der Text mit verschwommener Kante darunter zu erkennen. Anschließend wiederholen sich die zwei Versionen mit einem alternativen Hintergrund. Für die verwendete Texterkennung handelt es sich um eine lange Reihe von Buchstaben mit Zwischenräumen. Im besten Fall werden alle Buchstaben inklusive der Leerzeichen zurückgegeben.

Die generierten Bilder sind Teil der hier vorgestellten Lösung und bilden die Basis der im folgenden beschriebenen Verfahren zur Evaluierung des Systems auf Basis der

[45] Quellcodeverzeichnis Code 04

aktuell verwendeten neuronalen Netze. Eine detaillierte Auflistung der Testbilder sowie der Verzeichnisse findet sich im Anhang.

6.2 Abgleich von Begriffen mit der verwendeten Datenbasis

Bei der Evaluierung des korrekten Abgleichs mit der hinterlegten Datenbasis steht die Funktionalität der Klasse Ingrediens im Mittelpunkt. Sie bestimmt maßgeblich die Qualität der vorliegenden Lösung. Letztlich ist zu klären, inwieweit ein Suchbegriff korrekt verarbeitet wird. Die Geschwindigkeit selbst ist jedoch auf Grund der aktuell geringen Datenmenge zu vernachlässigen.

Die Überprüfung findet anhand von definierten Suchbegriffen statt, die nur bei einer korrekten Vorverarbeitung und Anwendung des Suchalgorithmus als wahr zurückgegeben werden. Um dies Ziel zu erfüllen, ist zudem die korrekte Erstellung der internen Suchtabelle innerhalb der Klasse Ingrediens notwendig. Die korrekte Rückgabe als wahr zeigt, dass die Implementierung korrekt erfolgte. Als Kriterium für das Suchwort kommen hierbei die folgenden Vorgaben in Betracht:

- Das Suchwort ist ein als E-Nummer hinterlegter Begriff
- Das Suchwort ist ein Begriff innerhalb der hinterlegten Namen
- Das Suchwort ist eines der hinterlegten Schlüsselwörter

Hierbei muss durch eine Kombination von Groß- und Kleinschreibung sichergestellt sein, dass möglichst alle vorkommenden Besonderheiten abgedeckt sind.

Der Code zur Durchführung befindet sich in der Datei **`evaluation.py`**. Den Kern bildet hierbei eine einfach anpassbare Python Liste von Suchwörtern, welche alle zu prüfenden Begriffe und Variationen enthält. Diese aktuell 40 Einträge[46] umfassende Liste wird der ebenfalls in dieser Datei enthaltenen Funktion `evaluate_database(…)` übergeben.

Innerhalb dieser Methode wird die hinterlegte Datenbasis in der aktuellen Version nach jedem Listeneintrag befragt und das Ergebnis als Text in der Konsole ausgegeben. Hierzu wird die Methode `db_contains(…)` der Klasse Scanner analog zum normalen Ablauf genutzt.

Über die einzelnen Ausgaben können Fehler klar erkannt werden. Ebenfalls sind sofort die Kombinationen ersichtlich, bei der es zu einem Problem kam. Somit ist eine zielgenaue Untersuchung der Fehlerursache möglich. In der folgenden Abbildung ist ein Ausschnitt der originalen Ausgabe widergegeben:

[46] Die zuständige Python Liste ist keywords in der Datei **evaluation.py**

```
E-160e evaluate to True stands for E 160e - Beta-apo-8-Carotinal
E - 160e evaluate to True stands for E 160e - Beta-apo-8-Carotinal
e-160e evaluate to True stands for E 160e - Beta-apo-8-Carotinal
e - 160e evaluate to True stands for E 160e - Beta-apo-8-Carotinal
E160 e evaluate to True stands for E 160e - Beta-apo-8-Carotinal
```

Abbildung 41: Auszug der Kontrolltexte bei der Evaluierung der Datenbasis

Ganz links sind die abgefragten Versionen der E-Nummer E 160e zu sehen, daneben die jeweilige Rückgabe (hier immer True). Ganz rechts ist der Identifikationsstring des Inhaltsstoffes ausgegeben.

Bei der Durchführung des beschriebenen Tests wurde für jeden Suchbegriff ein True zurückgegeben. Dies zeigt, dass die Klasse Ingrediens wie erwartet arbeitet.

6.3 Detektion und Erkennen von Texten

Bei der Evaluierung anhand eines ganzen Bildes mit mehreren Textquellen kommt die Funktion `scann(…)` der Klasse Scanner zum Einsatz und somit auch hier die im regulären Einsatz verwendete Methode.

Gesteuert wird der Test durch die in der Datei **`evaluate.py`** hinterlegten Funktion `evaluate(…)`, der eine Liste mit Testbildern, ein Basisverzeichnis sowie eine Instanz der Klasse Scanner übergeben wird. Die übergebene Liste enthält hierbei für jedes Bild eine Liste mit jeweils zwei Einträgen. Hierbei handelt es sich um einen Datei- sowie einen Verzeichnisnamen.

Der Dateiname bezeichnet eine Bilddatei im Basisverzeichnis, das analog zu einer normalen Verarbeitung betrachtet werden soll, der Verzeichnisname das Unterverzeichnis zum Basisverzeichnis, in dem das bearbeitete Bild abgelegt werden soll. Zuvor wird ein eventuell existierendes Verzeichnis mit diesen Namen gelöscht.

Der Vorgang wird einmal für Text[47] und einmal für Sonderzeichen[48] in allen unter 6.1 angegebenen Versionen ausgeführt. Durch die Verwendung von Listen kann der Ablauf dieser Tests einfach abgeändert und neuen Bedürfnissen angepasst werden. Die zu untersuchenden Kernfragen sind hierbei:

- Wie akkurat werden Textregionen innerhalb eines Bildes vorhergesagt
- Wie akkurat wird der in einem Textausschnitt enthaltene Text vorhergesagt

[47] Die zuständige Python Liste ist evallist_text in der Datei **evaluation.py**
[48] Die zuständige Python Liste ist evallist_special_text in der Datei **evaluation.py**

Anhand der zuvor genannten Punkte kann die Leistung eines Systems in diesem Bereich beurteilt und Rückschlüsse auf mögliche Verbesserungen gemacht werden.

6.4 Erkennen von Buchstaben und Sonderzeichen

Aufgrund der großen Menge an Einzelbildern wurde für die Erkennung von Buchstaben und Sonderzeichen eine andere Methodik in der Funktion `evaluate_char(…)` implementiert. Im Kern wird hier die Funktion `predict_text(…)` der Klasse Scanner genutzt. Somit wird auch hier analog zur normalen Funktionalität vorgegangen.

Der Funktion `evaluate_char(…)` wird eine Liste mit Namensendungen übergeben. Jede Namensendung steht für eine der in 6.1 definierten Versionen. Weiter wird ein Basisverzeichnis übergeben sowie drei Zahlenwerte. Die ersten zwei bezeichnen einen Zahlenbereich von-bis, die dritte Zahl die Anzahl an Vornullen. Anhand dieser Angaben können nun die Dateinamen für die real im Basisverzeichnis hinterlegten Bilddateien generiert werden.

Der Dateiname ergibt sich aus dem mit Vornullen formatierten Zahlenwert sowie der Angabe der Version inklusive der Endung zur Angabe des Dateiformates. Die Ausgabe erfolgt in einem nach dem formatierten Zahlenwert benannten Unterverzeichnis.

Das Verfahren kann leicht um eigene Bilder und Versionen ergänzt und angepasst werden. Wichtig für eine korrekte Ausführung ist lediglich, dass für jedes Bild alle definierten Versionen in der korrekten Bezeichnung verfügbar sind.

Die im Rahmen von diesem Test zu untersuchenden Kernfragen sind:

- werden alle Buchstaben korrekt erkannt
- werden alle Sonderzeichen korrekt erkannt
- werden Leerzeichen korrekt erkannt

Essenziell ist hierbei, dass die jeweilige Bildkomposition manuell vorgegeben und somit für verschiedene Modelle identisch ist. Im Idealfall erkennt ein Netz jeden Buchstaben inklusive der enthaltenen Leerzeichen. Das Erkennen aller Zeichen unter Wegfall der Leerzeichen ist jedoch ebenso ein als gut anzusehendes Ergebnis, da die Erkennung von Wortgrenzen grundsätzlich in der Verantwortung der Textdetektion gehört.

6.5 Ergebnis für die Detektion und Erkennung von Text

Abschließend soll auf das Ergebnis der zuvor beschriebenen Verfahren zur Evaluierung kurz eingegangen werden. Die bei der Durchführung entstandenen Ergebnisbilder sind wie die Testbilder selbst Bestandteil der hier vorliegenden Lösung. Der jeweilige Speicherort kann den zuvor gemachten Ausführungen entnommen werden.

Es zeigt sich, dass systemisch bei Vorliegen eines nicht weißen Hintergrundes sich sowohl die Erkennung als auch die Detektion von Text rapide verschlechtert bis hin zu keiner einzigen erfolgreichen Vorhersage. Dies gilt sowohl für die Kombination von Detektion und Erkennung als auch der reinen Erkennung. Für Bilder mit schwarzer Schrift und weißem Hintergrund war die Erkennungsrate grundsätzlich besser. Nicht schwarzer Text wurde nicht getestet. Im Folgenden soll nur die zuletzt genannte Gruppe weiter betrachtet werden.

In der Gruppe der Sonderzeichen schnitt die Detektion eher schlecht ab. Die folgende Abbildung zeigt stellvertretend das Ergebnisbild mit scharfen Kanten:

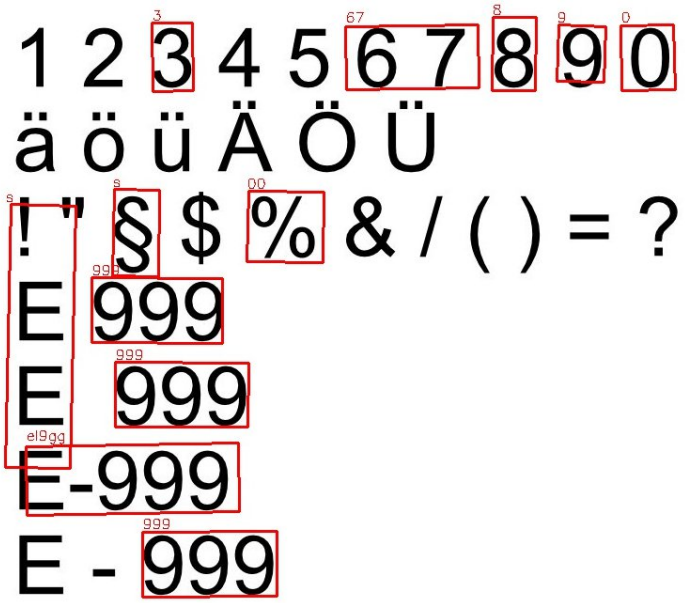

Abbildung 42: Ergebnis Detektion und Erkennung für Sonderzeichen

Deutlich ist zu erkennen, dass viele einzelne Zeichen nicht erkannt oder mehrere Vorkommen als zusammengehörend detektiert werden (roter Rahmen). Im Gegensatz hierzu wurden tatsächlich zusammenhängende Zahlen wie „E-999“ oder „999“ grundsätzlich korrekt detektiert. Zahlen und Buchstaben wurden nach einer korrekten Detektion korrekt erkannt, Sonderzeichen hingegen nicht.

Werden nur die Zahlen, Umlaute und Sonderzeichen wie oben zu sehen zeilenweise direkt einer Texterkennung unterzogen, so zeigt sich, dass Zahlen korrekt erkannt werden. Umlaute werden dem zugrunde liegenden Buchstaben zugeordnet, Sonderzeichen werden nicht erkannt. Dies stützt das zuvor betrachtete Ergebnis. Als Grund hierfür ist das Training anhand eines englischen Zeichensatzes ohne Sonderzeichen zu benennen. Bei der Vorlage mit weichen Kanten zeigt sich in Teilen ein etwas besseres Ergebnis, welches jedoch durch weitere Fehler relativiert wird.

Werden statt Sonderzeichen normale Sätze untersucht, verbessert sich grundsätzlich das erhaltene Ergebnis. Die folgende Abbildung zeigt stellvertretend das Ergebnis einer Textdetektion und -erkennung für Text mit scharfen Kanten:

Abbildung 43: Ergebnis Detektion und Erkennung für normalen Text

In dem Bild ist gut zu erkennen, dass die besten Ergebnisse für horizontal angeordnete Texte ohne Verzerrungen wie beispielsweise in der ersten Zeile erreicht werden. Ein Vergleich mit der Vorlage für weiche Kanten zeigt hier ein gleichartiges Ergebnis. Weiche Kanten haben somit eher wenig Einfluss. Gleiches gilt auch für eine leicht runde Ausrichtung wie sie rechts unten zu sehen ist. Jegliche Art von sonstiger, auch eindimensionaler Verzerrung wie in Zeile zwei, führt jedoch zu einer hohen Fehlerrate sowohl bei der Detektion als auch der Erkennung.

Somit kann als Fazit ausgeführt werden, dass bei Verwendung der aktuellen Konstellation wie im Abschnitt 5.8 beschrieben, die besten Ergebnisse mit Text auf weißen oder neutralen Grund erzielt werden, sofern der Text möglichst horizontal oder maximal leicht gebogen ist. Sonderzeichen oder Leerzeichen werden nicht detektiert und Umlaute werden mit deren zugrunde liegenden Buchstaben vorhergesagt. Gerade der Wegfall von Leerzeichen kann zu Problemen bei der Interpretation von E-Nummern führen.

7 Zusammenfassung

Zum Abschluss dieser Arbeit soll ein kurzes Fazit über die Umsetzung und Lösung der im Titel gegebenen Aufgabenstellung gezogen werden. Im sich anschließenden Ausblick sollen Möglichkeiten zur Weiterentwicklung aufgezeigt werden.

7.1 Fazit

Ziel dieser Arbeit war die Konzeption und Umsetzung einer Lösung, mit deren Hilfe in Rastergrafiken enthaltener Text extrahiert und einer weitergehenden Verwendung zugeführt werden kann. Dies beinhaltet die semantische Zuordnung des extrahierten Textes. Dies Ziel wurde mit der vorliegenden Lösung erreicht.

Der zunächst angedachte Weg der vollständigen Entwicklung und Trainings eines neuronalen Netzes wurde im Verlauf der Konzeption aufgegeben. Die Gründe hierfür sind vielschichtig und wurden im Abschnitt 4.3 - Netzstruktur ausführlich diskutiert.

Stattdessen entstand ein prototypisches System, welches zur Erfüllung der anstehenden Aufgabe die Möglichkeit bietet, verschiedene Netze einfach zu nutzen. Durch die Implementierung grundlegender Methoden zur Evaluierung des Systems, kann die Leistungsfähigkeit hierbei einfach verglichen werden. Durch die Teilung der Kompetenzen in Textdetektion und Texterkennung die jeweils beste Lösung verwendet werden.

Somit liegt mit der vorliegenden Arbeit nicht nur eine Lösung der Aufgabenstellung vor, sondern die Basis eines einfachen Frameworks. Durch den offenen Ansatz kann dieses leicht angepasst und erweitert werden.

7.2 Ausblick

Die im Rahmen der Evaluierung durchgeführten Testläufe unter Verwendung der in Abschnitt 5.8 beschriebenen Musterimplementierung zeigen eine eher mäßige Genauigkeit bei der Vorhersage. Dies betrifft sowohl die Detektion als auch die Erkennung. Die Zuordnung korrekt erkannter Begriffe ist hingegen sehr genau.

Es besteht eine hohe Wahrscheinlichkeit, dass durch den Einsatz anderer Netze oder im Rahmen eines Finetunings der eingesetzten Netze, die Genauigkeit wesentlich gesteigert werden kann. Auch die Einführung zusätzlicher Layer ist denkbar. Dies bietet Raum für eine Fortentwicklung.

Hilfe bieten die Ergebnisse der durchgeführten Evaluierung, welche die Schwächen der aktuellen Netze deutlich aufzeigen.

Darüber hinaus bieten die Vor- und Nachbearbeitung der Daten mögliche Ansatzpunkte zur Verbesserung der Leistung.

Abkürzungsverzeichnis

API ..... *Application Programming Interface*
Basis-VO ..... *Basisverordnung*
C# ..... *C-Sharp*
CNN ..... *Convolutional Neural Network*
CNTK ..... *Microsoft Cognitive Toolkit*
COCO ..... *Common Objects in Context*
CRNN ..... Convolutional Recurrent Neural Network
CSV ..... *Comma-separated values*
CTC ..... *Connectionist Temporal Classification*
EAST ..... *An Efficient And Accurate Scene Text Detector*
EG ..... *Europäische Gemeinschaft*
EU ..... *Europäische Union*
FK ..... Foreign Key
FNN ..... *Feedforward Neural Network*
GNU ..... *General Public License*
ID ..... *Identifikator*
ILSVRC ..... *ImageNet Large Scale Visual Recognition Competition*
iOS ..... *Betriebssystem von Apple*
JPEG ..... *Joint Photographic Experts Group*
JSON ..... JavaScript Object Notation
LSTM ..... *Long Short-Term Memory Units*
MIT ..... *Massachusetts Institute of Technology*
MNIST database ..... *Modified National Institute of Standards and Technology database*
MSE ..... *Mean Squared Error*
OCR ..... *Optical Character Recognition*
OMI ..... *Online Medieninformatik*
OOP ..... *objektorientierte Programmierung*
PK ..... *Primary Key*
PVANET ..... *Deep but Lightweight Neural Networks for Real-time Object Detection*
ReLU ..... *Rectified Nonlinear Unit*
RGB ..... *Rot Grün Blau*
RNN ..... *Recurrent Neural Network*
ROI ..... Region of Interest
SGA ..... *stochastischer Gradientenabstieg*

SSD.....Single Shot MultiBox Detector
VGG.....Visual Geometry Group
XML.....Extensible Markup Language
YAML.....YAML Ain't Markup Language
YOLO.....You Only Look Once

Literatur- und Quellenverzeichnis

Alese E (2018) The curious case of the vanishing & exploding gradient. https://www.facebook.com/medium. https://medium.com/learn-love-ai/the-curious-case-of-the-vanishing-exploding-gradient-bf58ec6822eb. Zugegriffen: 28. Mai 2019

Alice Zheng AC (2019) Merkmalskonstruktion für Machine Learning

Arbel N (2018) How LSTM networks solve the problem of vanishing gradients. https://www.facebook.com/medium. https://medium.com/datadriveninvestor/how-do-lstm-networks-solve-the-problem-of-vanishing-gradients-a6784971a577. Zugegriffen: 01. Juli 2019

Dörn S (2018) Programmieren für Ingenieure und Naturwissenschaftler. Springer Berlin Heidelberg, Berlin, Heidelberg

Erhardt A (2008) Einführung in die Digitale Bildverarbeitung; Grundlagen, Systeme und Anwendungen. Vieweg+Teubner / GWV Fachverlage GmbH Wiesbaden, Wiesbaden

Ertel W (2016) Grundkurs Künstliche Intelligenz. Springer Fachmedien Wiesbaden, Wiesbaden

Frede W (2010) Handbuch für Lebensmittelchemiker. Springer Berlin Heidelberg, Berlin, Heidelberg

Gonfalonieri A (2019) How to Build A Data Set For Your Machine Learning Project. https://www.facebook.com/towardsdatascience. https://towardsdatascience.com/how-to-build-a-data-set-for-your-machine-learning-project-5b3b871881ac. Zugegriffen: 03. Juli 2019

Hope T, Resheff YS, Lieder I (2018) Einführung in TensorFlow; Deep-Learning-Systeme programmieren, trainieren, skalieren und deployen. O'Reilly, Heidelberg

Hui J (2018) SSD object detection: Single Shot MultiBox Detector for real-time processing. https://medium.com/@jonathan_hui/ssd-object-detection-single-shot-multibox-detector-for-real-time-processing-9bd8deac0e06. Zugegriffen: 01. August 2019

Jason Brownlee (2017) A Gentle Introduction to Transfer Learning for Deep Learning. https://www.facebook.com/MachineLearningMastery/. https://machinelearningmastery.com/transfer-learning-for-deep-learning/. Zugegriffen: 29. Mai 2019

Keras (11.02.2019) Home - Keras Documentation. https://keras.io/#you-have-just-found-keras. Zugegriffen: 19. Mai 2019

Kim K-H, Hong S, Roh B, Cheon Y, Park M (30.09.2016) PVANET: Deep but Lightweight Neural Networks for Real-time Object Detection. https://arxiv.org/pdf/1608.08021. Zugegriffen: 26. August 2019

Muneeb ul Hassan (2018) VGG16 - Convolutional Network for Classification and Detection. https://www.facebook.com/neurohive.io/. https://neurohive.io/en/popular-networks/vgg16/. Zugegriffen: 03. Juli 2019

N K (2017) Speech Recognition: You down with CTC? https://gab41.lab41.org/speech-recognition-you-down-with-ctc-8d3b558943f0. Zugegriffen: 31. Juli 2019

Nischwitz A, Fischer M, Haberäcker P, Socher G (2011) Computergrafik und Bildverarbeitung; Band II: Bildverarbeitung. Vieweg+Teubner Verlag, Wiesbaden

Oren Ben-Kiki, Clark Evans, Ingy döt Net (2009) YAML Ain't Markup Language (YAML™) Version 1.2; 3rd Edition, Patched at 2009-10-01. https://yaml.org/spec/1.2/spec.html. Zugegriffen: 30. Juni 2019

Parmar R (2018) Detection and Segmentation through ConvNets – Towards Data Science. https://www.facebook.com/towardsdatascience. https://towardsdatascience.com/detection-and-segmentation-through-convnets-47aa42de27ea. Zugegriffen: 20. Juni 2019

Rashid T (2017) Neuronale Netze selbst programmieren; Ein verständlicher Einstieg mit Python. O'Reilly, Heidelberg

Robust Reading Competition Overview - Incidental Scene Text - ICDAR 2019. https://rrc.cvc.uab.es/?ch=4. Zugegriffen: 03. Juli 2019

Sandeep Gupta, Josh Gordon, Karmel Allison Standardizing on Keras: Guidance on High-level APIs in TensorFlow 2.0. https://medium.com/tensorflow/standardizing-on-keras-guidance-on-high-level-apis-in-tensorflow-2-0-bad2b04c819a. Zugegriffen: 19. Mai 2019

Sarkar D(D) (2018) A Comprehensive Hands-on Guide to Transfer Learning with Real-World Applications in Deep Learning. https://www.facebook.com/towardsdatascience. https://towardsdatascience.com/a-comprehensive-hands-on-guide-to-transfer-learning-with-real-world-applications-in-deep-learning-212bf3b2f27a. Zugegriffen: 20. Juni 2019

Scheidl H (2018) An Intuitive Explanation of Connectionist Temporal Classification. https://towardsdatascience.com/intuitively-understanding-connectionist-temporal-classification-3797e43a86c. Zugegriffen: 31. Juli 2019

Schiele H-G (2012) Computergrafik für Ingenieure. Springer Berlin Heidelberg, Berlin, Heidelberg

Shperber G (2019) A gentle introduction to OCR. https://towardsdatascience.com/a-gentle-introduction-to-ocr-ee1469a201aa. Zugegriffen: 29. Juli 2019

Soullard Y, Ruffino C, Paquet T (23.01.2019) CTCModel: a Keras Model for Connectionist Temporal Classification. https://arxiv.org/pdf/1901.07957. Zugegriffen: 26. August 2019

TensorFlow Introduction to TensorFlow. https://www.tensorflow.org/learn. Zugegriffen: 19. Mai 2019

TensorFlow TensorFlow Guide. https://www.tensorflow.org/guide. Zugegriffen: 19. Mai 2019

Wang C-F (2019) The Vanishing Gradient Problem – Towards Data Science. https://www.facebook.com/towardsdatascience. https://towardsdatascience.com/the-vanishing-gradient-problem-69bf08b15484. Zugegriffen: 28. Mai 2019

Wikipedia-Autoren sV (14.11.2004) Datei:RGB farbwuerfel.jpg. https://de.wikipedia.org/wiki/Datei:RGB_farbwuerfel.jpg. Zugegriffen: 18. Mai 2019

Wikipedia-Autoren sV (15.05.2019) Liste der Lebensmittelzusatzstoffe. https://de.wikipedia.org/w/index.php?oldid=187539415. Zugegriffen: 15. Mai 2019

Wikipedia-Autoren sV (25.06.2019) YAML. https://de.wikipedia.org/w/index.php?oldid=189830079. Zugegriffen: 30. Juni 2019

Zhou X, Yao C, Wen H, Wang Y, Zhou S, He W, Liang J (10.07.2017) EAST: An Efficient and Accurate Scene Text Detector. https://arxiv.org/pdf/1704.03155. Zugegriffen: 26. August 2019

Quellcodeverzeichnis

In der Arbeit kommen im Bereich der Musterimplementierungen externer und nicht vom Verfasser programmierter Quellcode zum Einsatz. Die betreffenden Stellen sind innerhalb des Quellcodes im Rahmen der Dokumentierung gekennzeichnet.

Sofern die gesamte Datei betroffen ist, wird zu Anfang der Datei darauf hingewiesen. Dies betrifft die externen Modelle. Sind nur Teile der Datei betroffen, so findet sich der Vermerk „External code“ zu Anfang der betreffenden Methode. Dies betrifft die Bridgeklassen.

Die folgende Aufstellung listet die Quellen der verwendeten Codebasen im Original:

Code 01
Jan Zdenek, Implementation of EAST scene text detector in Keras
Downlaod: GitHub (Username kurapan) - https://github.com/kurapan/EAST
Letzter Zugriff: 24.08.2019
Lizenz: GNU General Public License v3.0[49]

Code 02
Jan Zdenek, Convolutional recurrent neural network for scene text recognition or OCR in Keras
Download: GitHub (Username kurapan) - https://github.com/kurapan/CRNN
Letzter Zugriff: 24.08.2019
Lizenz: MIT License[50]

Code 03
Abhishek Singh, Text Detection from images using OpenCV
Download: GitHub (Username ZER-0-NE) - https://github.com/ZER-0-NE/EAST-Detector-for-text-detection-using-OpenCV
Letzter Zugriff: 24.08.2019
Lizenz: MIT License

Code 04
Das als Hintergrund für Testbilder dienende Waldbild mit Brücke
Download: Unsplash (Username Amos G) - https://unsplash.com/photos/IrGYJPF0JS0
Letzter Zugriff: 25.08.2019
Lizenz: Frei

[49] https://www.gnu.org/licenses/gpl-3.0.de.html
[50] https://opensource.org/licenses/mit-license.php

Abbildungsverzeichnis

Formelverzeichnis

Anhang

(1) Inhalt des beiliegenden Datenträgers

- ausarbeitung.pdf (*die vorliegende Arbeit*)
- **code**
  - **ocr_system (*Dateien und Ordner des erstelltenSystems*)**
- **externer_code**
  - **code_01 (*Dateien und Ordner gem. Quellcodeverzeichnis*)**
  - **code_02 (*Dateien und Ordner gem. Quellcodeverzeichnis*)**
  - **code_03 (*Dateien und Ordner gem. Quellcodeverzeichnis*)**
  - **code_04 (*Dateien gem. Quellcodeverzeichnis*)**

(2) Dateien und Verzeichnisse des erstellten Systems

Im Folgenden werden die wichtigsten Ordner und Dateien des im Rahmen dieser Arbeit erstellten Systems alphabetisch aufgelistet. Die in Teilen hinzugefügten Nummern verweisen auf die hierauf Bezug nehmenden Kapitel.

- **ocr_system *(Systemordner)***
  - ***annotation_constants (Dateien für die Konfiguration der Annotation**)*
    - eval_annotation_constants.py *(Konfiguration für die Evaluierung)*
    - neg_annotation_constants.py *(*Konfiguration *für nicht gefundene Einträge)*
    - pos_annotation_constants.py *(*Konfiguration *für gefundene Einträge)*
  - **bridges *(Dateien und Ordner zum Einbinden externer Modelle gem. 5.6)***
    - **models *(Ordner für externe Modelle)***
      - **crnn *(Dateien und Ordner für CRNN Modell gem. 5.8)***
      - **east *(Dateien und Ordner für EAST Modell gem. 5.8)***
      - **east_open_cv *(Dateien und Ordner für EAST Modell aus OpenCV)***
    - bridges.json *(JSON-Datei für die Konfiguration der bridges gem. 5.6.1)*
    - bridges_config.py *(Konfiguration der Modelle gem. 5.6.1)*
    - crnn_bridge.py *(Bridge für das Model crnn gem. 5.8)*
    - east_bridge.py *(Bridge für das Model east gem.5.8)*
    - east_open_cv_bridge.py *(Bridge für das Model east_open)*
  - **data (Ordner mit Daten zum Abgleich gefundener Begriffe gem. 4.4)**
    - ingrediens.json *(JSON-Datei mit Inhaltsstoffen)*
    - ingrediens.xlsx *(Excel-Datei mit Inhaltsstoffen als Zusatz)*
  - **evaluation *(Dateien und Ordner für die Evaluierung gem. 6.0)***
    - **chars *(Ordner für Sätze - Erkennung gem. 6.4)***
    - **special_chars *(Ordner für Sonderzeichen - Erkennung gem. 6.4)***
    - **special_text *(Ordner für Sonderzeichen - Detektion/Erkennung gem. 6.5)***
    - **text *(Ordner für Sätze - Detektion/Erkennung gem. 6.5)***
    - forrest.jpg *(Hintergrundbild für die Testbilder)*
    - testfotos.afphoto *(Testbilder als Affinity Projekt)*
  - **income *(Ordner für die Eingaben gem. 5.2)***

- **license** ***(Ordner für Dateien mit den Lizenzangaben für die Anwendung)***
- **outcome** ***(Ordner für die Ausgaben gem. 5.2)***
- bounding_box_image_handler.py *(Hilfsklasse zur Bildbearbeitung und Annotation)*
- constant.py *(definiert Konstanten für das System)*
- demo.py *(simuliert einen Nutzer des Systems gem. 5.2)*
- detector.py *(Klasse detector gem. 5.5.2)*
- evaluation.py *(Code zur Evaluierung des Systems gem. 6.0)*
- ingrediens.py *(Klasse ingrediens gem. 5.5.4)*
- recognizer.py *(Klasse recognizer gem. 5.5.3)*
- scanner.py *(Klasse scanner gem. 5.5.1)*

Die Ordner chars und special_chars enthalten Testbilder für die Evaluierung gemäß Abschnitt 6.4. Die Ordner text und special_text enthalten Bilder für die Evaluierung gemäß Abschnitt 6.5.

Die Ergebnisse der Evaluierung sind in Unterordnern der jeweiligen Ordner zu finden. Für jedes Testbild existiert hierbei ein Ordner mit dem Namen der Testdatei.

Anhand der Dateinamen ist die Art des Bildes zu erkennen. Enthält der Dateinamen den Text „_black_“, so handelt es sich um eine Version mit schwarzer Schrift auf weißen Grund. Enthält er hingegen den Text „_forest_“, so handelt es sich um eine Version mit einem Bildhintergrund.

Bei Dateinamen mit einem Text „_sharp_“ handelt es sich um ein Testbild mit scharfer Schriftkante, bei „_sharp_blur_“ um ein Bild mit einer verschwommenen Schriftkante.

In den Ordnern income und outcome sind zu Demonstrationszwecken die Eingabedatei **test.jpg** sowie die daraus resultierenden Ausgabedateien hinterlegt, wie sie bei Ausführung der Datei **demo.py** entstehen.

Hierbei ist **001.jpg** die annotierte Ausgabedatei. Die restlichen Dateien stehen für alle gefundenen Textvorkommen und sind mit dem bei der Texterkennung vorhergesagten Text benannt. Sie dienen nur der Information und Kontrolle.