Michael Rabko, BSc

1x1 Trainer mit Handschrifterkennung

Masterarbeit

zur Erlangung des akademischen Grades

Diplom-Ingenieur

Masterstudium Softwareentwicklung-Wirtschaft

eingereicht an der

Technischen Universität Graz

Betreuer

Priv.-Doz. Dipl-Ing. Dr.techn. Martin Ebner

Institut für Interactive Systems and Data Science

Graz, September 2017

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Graz, _____ _____

Date Signature

Eidesstattliche Erklärung[1]

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Graz, am _____ _____

Datum Unterschrift

[1] Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008; Genehmigung des Senates am 1.12.2008

Danksagung

Zuerst möchte ich mich bei Priv.-Doz. Dipl.-Ing. Dr.techn. Martin Ebner für die Betreuung, den hilfreichen Ratschlägen und seiner herzlichen Unterstützung bedanken. Weiters bedanke ich mich bei Dipl.-Ing. Behnam Taraghi, der mir während der Implementierungsphase immer mit Rat zur Seite stand.

Meiner Freundin Kerstin danke ich von tiefstem Herzen für ihren starken emotionalen Rückhalt und dass sie stets ein offenes Ohr für meine Sorgen und Probleme hatte. Ebenfalls bedanke ich mich bei meinen Freunden Tobias und Maximilian.

Abschließend gebührt der größte Dank meiner Familie, vor allem meinen Eltern Melitta und Reinhard. Ohne sie wäre ich nicht da, wo ich jetzt bin. Ich liebe Euch!

Kurzfassung

Computer und mobile Geräte sind in unserem Alltag allgegenwärtig präsent, ob in der Arbeit, im privaten Bereich oder mittlerweile auch in Schulen. Sie werden zur Unterstützung für vielseitige Tätigkeiten eingesetzt, sowie zum Beispiel für Lernapplikationen. Die Interaktion dieser Applikationen mit einem Computer basiert dabei oftmals nur auf vordefinierte Eingabemöglichkeiten, obwohl ein Touchscreen das direkte Schreiben mit der Hand oder einem Stift auf einem Bildschirm ermöglicht. Der erste Teil der Arbeit behandelt die Handschrifterkennung von elektronischen Geräten und erläutert, welche Methoden in der heutigen Zeit dafür existieren und wie diese in einem Computerprogramm eingesetzt werden können. Durch dieses erlangte Wissen wurde im zweiten Teil eine mobile Lernapplikation entwickelt, mit dem Kinder ihre Einmaleins-Kenntnisse verbessern können. Das Ziel dieser Masterarbeit ist, mit der implementierten Applikation herauszufinden, wie Kinder auf die interaktive Eingabe mit ihrer Handschrift reagieren. Diesbezüglich wurde in der nachfolgenden Evaluierung die Bedienbarkeit der Applikation mit einer Schulklasse der Volksschule Graz-Hirten überprüft. Das Ergebnis des durchgeführten Usability-Tests zeigte, dass die befragten Kinder das Schreiben mit dem Finger im Großen und Ganzen als sehr positiv wahrnehmen.

Abstract

Nowadays, computers and mobile devices play a huge role in our daily routines; they are used at work, for private purposes and even at school. Moreover, they are used as support for different kinds of activities and task, like for example, learning applications. The interaction of these applications with a computer is based on predefined input methods, whereas a touchscreen facilitates a direct input via hand writing by using a finger or a pen. The first part of this master thesis deals with handwriting recognition of electronical devices. In addition, the variety of handwriting recognition methods which can be found nowadays will be addressed and furthermore it will be explained how these methods can be applied on a computer program. The second part of this master thesis bases on the theoretical first part of this master thesis and deals with the invention of a mobile learning application which is supposed to facilitate children's learning of simple multiplication. The aim of this master thesis is to collect the data of children's experiences using interactive hand writing on mobile devices. In order to gain this data, a school class of the school "Graz-Hirten" was tested and afterwards for evaluational purposes interviewed. The results of these usability tests have shown that children perceived handwriting via finger on screen as quite positive.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Auflistungsverzeichnis

Abkürzungsverzeichnis

PDA	Personal Digital Assistant
NN	Neuronales Netz
KNN	Künstliches Neuronales Netz
CNN	Convolutional Neural Network
LA	Learning Analytics
SOAP	Simple Object Access Protocol
HTTP	Hypertext Transfer Protocol
SDK	Software Development Kit
XML	Erweiterbaren Auszeichnungssprache
RPC	Remote Procedure Call

1. Einleitung

Eine Handschrift ist eine sehr individuelle Fähigkeit und dient zur Kommunikation von Gefühlen und Meinungen. Jeder Mensch hat seine persönliche Art und Weise, Buchstaben oder Zeichen zu schreiben. Obwohl dadurch die Handschrift jedes Menschen unterschiedlich ist, können Menschen diese erkennen und verstehen. Selbst Kinder in einem Alter von sieben Jahren können auf zerknülltem Papier eine große, kleine oder verdrehte handgeschriebene Schrift lesen (Jain et al., 2000).

Eine Maschine kann eine Handschrift dagegen nicht verstehen. Die Eingabe über eine Tastatur ist für eine Maschine verständlich. Durch vordefinierte, beziehungsweise vorprogrammierte Eingabemöglichkeiten ist der Maschine zu jedem Zeitpunkt klar, wie sie die Eingabe verarbeiten soll. Mit einer Handschrift stößt die Maschine allerdings an ihre Grenzen. Die Maschine weiß nicht, wie sie die handgeschriebenen Zeichen verarbeiten und interpretieren soll.

Die vorliegende Arbeit befasst sich mit diesem Thema der automatischen Erkennung einer Handschrift und den Einsatz dieser Technik in Mathematik-Lernapplikationen. Zu diesem Zweck wird für diese Masterarbeit eine Lernapplikation für Volksschulkinder entwickelt und evaluiert. Die entwickelte Applikation soll die Kinder dabei unterstützen, die Multiplikationstabelle zu erlernen. Die Applikation basiert auf dem bereits bestehenden *1x1 Trainer* der Technischen Universität Graz.

In der Arbeit soll mit der implementierten Lernapplikation dargelegt werden, wie die Kinder auf diese neue Bedienbarkeit reagieren und wie sich das auf die Usability der Applikation auswirkt. Es soll weiters festgestellt werden, ob der intuitive Einsatz einer Handschrifterkennung sich gegenüber einer herkömmlichen Ziffernblockeingabe bewährt.

Die Arbeit teilt sich in sechs Kapitel auf. Zunächst wird in Kapitel 2 eine kurze Einführung in das Thema *Learning Analytics* gegeben und dargelegt, welche Möglichkeiten für die Erkennung einer Handschrift existieren und wie eine Handschrifterkennung aufgebaut ist. In Kapitel 3 werden verschiedene Lernapplikationen vorgestellt, die über eine Handschrifterkennung verfügen. Aufbauend auf Kapitel 2 werden in Kapitel 4 die Umsetzung und die dafür verwendeten Technologien der Lernapplikation beschrieben. Das fünfte Kapitel beschäftigt sich mit der Evaluation der entwickelten Applikation. Abschließend werden in Kapitel 6 die gewonnenen Erkenntnisse zusammengefasst und mögliche Verbesserungsvorschläge für die Applikation gegeben.

2. Grundlagen

2.1. Handschrifterkennung

Die Handschrifterkennung ist ein Teilbereich der Mustererkennung. Die Mustererkennung versucht einer Maschine die Eigenschaft „beizubringen“, um eine Handschrift verarbeiten zu können. Sie soll ihre Umgebung wahrnehmen, Muster erkennen und daraus entsprechende Resultate oder Entscheidungen treffen (Jain et al., 2000).

In diesem Bereich der automatisierten Handschrifterkennung wird schon seit über 50 Jahren geforscht (Simon, 1992). Bereits in den 60er Jahren wurden Ansätze zur Erkennung einer Handschrift sowie von handgeschriebenen Zahlen gezeigt (Eden, 1962; Genchi et al., 1968). Mittlerweile wurden schon mehrere Ansätze veröffentlicht, die gute Ergebnisse bei der Erkennung einer Handschrift liefern (Fischer et al., 2013; Keysers et al., 2017).

Die Erkennung einer Handschrift wird zwischen „Online“ und „Offline“ unterschieden (Plamondon und Srihari, 2000).

2.1.1. Online-Erkennung

In der Online-Erkennung wird die Handschrift, während sie geschrieben wird, erkannt. Die Erkennung nutzt dafür die physische Verbindung zum Eingabegerät. Diese physische Verbindung kann zum Beispiel mittels einem elektronischem Eingabestift, kurz Stylus, oder einem Finger erreicht werden. Die Berührung des Eingabegerätes liefert wichtige Informationen, die für die simultane Erkennung benötigt werden. Solche Informationen können die Geschwindigkeit, mit der geschrieben wurde, oder die Berührungspunkte über die verfasste Zeit sein. So kann das Eingabegerät zu jedem Zeitpunkt den Berührungspunkt durch eine x- und y-Koordinate feststellen. Auch die Beschleunigung, wie manche Striche gezeichnet werden, kann für eine Online-Erkennung eine bedeutungsvolle Information sein.

In Tappert et al. (1990) werden die Vor- und Nachteile einer Online-Erkennung gut dargestellt. Ein großer Vorteil ist unter anderem die Interaktivität. Die Anwenderin und der Anwender können bei einem Schreibfehler sofort reagieren und den Fehler ausbessern. Ein weiterer Vorteil sind die Informationen, die durch die physische Verbindung gewonnen werden, da sie bei der Erkennung helfen. Andererseits ist gerade die diese Verbindung der größte Nachteil in einer Online-Handschrifterkennung.

2.1.1.1. Anwendungsbereich

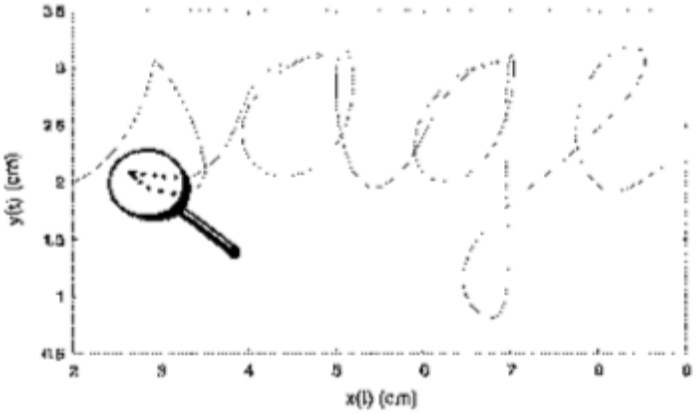
Die Online-Handschrifterkennung wird in vielen technischen Bereichen genutzt. Ein bekanntes Einsatzgebiet sind stiftbasierte Computer, wie ein Personal Digital Assistant (PDA). Die Signaturüberprüfung ist ein weiterer Bereich, wo die Online-Handschrifterkennung ihre Anwendung findet. Dabei wird mittels einer zuvor gespeicherten Unterschrift überprüft, ob die Person wirklich diese Person ist, für die sie sich ausgibt (Impedovo und Pirlo, 2008).

Abbildung 2.1.: Ein Personal Digital Assistant mit einem Eingabestift. Quelle: Tapia und Rojas, 2007

2.1.2. Offline-Erkennung

Der große Unterschied der Offline-Erkennung zur Online-Erkennung ist, dass die Daten nach dem Schreiben erst zu einem viel späteren Zeitpunkt verarbeitet werden. Erst vor der Erkennung wird die Handschrift mit Hilfe eines Scanners oder einer Kamera der Maschine zugänglich gemacht. Das daraus resultierende Bild kann danach von einer Maschine verarbeitet werden. Die Offline-Erkennung nutzt keine physische Verbindung zum Eingabegerät. Sie stellt dadurch im Gegensatz zur Online-Erkennung eine schwierigere Aufgabe dar. Die Abbildung 2.2 zeigt den Unterschied zwischen den beiden Erkennungsarten.

(a) Offline



(b) Online

Abbildung 2.2.: Die Abbildungen zeigen den Unterschied der vorhandenen Informationen zwischen einer Offline- und Online-Erkennung. Bei einer Offline-Erkennung (a) stehen der Erkennungsmethodik nur die Pixelwerte eines eingescannten Bildes zur Verfügung. Eine Online-Erkennung (b) verfügt demgegenüber zusätzlich über die zeitliche Information der Berührungskoordinaten (x,y). Quelle: Plamondon und Srihari, 2000

2.1.2.1. Anwendungsbereich

Die Offline-Erkennung findet in vielen Bereichen ihre Anwendung. Sie wird beispielsweise im Bankwesen und im Postwesen eingesetzt. Das Postwesen nutzt die Erkennung von Adressen und Namen für die automatische Sortie-

rung von Briefen. Im Bankwesen wird sie bei einer Überweisung verwendet, um die Überweisungsmenge, Kontodaten und die benötigte Unterschrift aus einem Überweisungscheck zu extrahieren (Priya et al., 2016). Die Unterschrift wird anschließend durch eine Signaturüberprüfung verifiziert. Die Signaturüberprüfung kann auch mit einer Offline-Erkennung durchgeführt werden (Roy et al., 2017).

2.1.3. Ablauf einer Handschrifterkennung

In diesem Abschnitt wird der Ablauf einer Handschrifterkennung näher dargestellt. Die Erkennung bezieht sich auf eine Offline-Erkennung. Die angesprochenen Techniken in diesem Kapitel können genauso für eine Online-Erkennung eingesetzt werden. Beide Erkennungsvarianten teilen sich weitestgehend dieselben Ansätze und Problemstellungen.

Das Ziel einer Handschrifterkennung ist aus den vorhandenen Bilddaten den Text zu extrahieren, der auf dem Bild ersichtlich ist. Der Ablauf einer Offline-Handschrifterkennung kann in fünf wesentlichen Schritten unterteilt werden. Natürlich können manche der unten angeführten Schritte verschmelzen oder nicht ausgeführt werden. Solch ein Verhalten hängt von der Erkennungsmethodik ab (Arica und Yarman-Vural, 2001).

2.1.3.1. Vorverarbeitung

Der erste Schritt in einer Handschrifterkennung ist die Vorverarbeitung der Erkennungsdaten. Die Vorverarbeitung wird vor der Erkennungsmethode (Klassifizierung) durchgeführt. Ihre Aufgabe ist es, die Eingabedaten aufzubereiten, damit sie leichter und vor allem korrekt erkannt werden. Die Aufbereitung der Daten ist in der Offline-Erkennung ein sehr wichtiger

Schritt, weil der Erkennungsmethodik keine zusätzlichen Informationen der Handschrift zur Verfügung stehen.

Im Offline-Ansatz liegt, wie oben beschrieben, die zu erkennende Handschrift nur als Bilddatei vor. Die Vorverarbeitungsphase und die restlichen Schritte arbeiten mit den Pixelwerten, die die Bilddatei repräsentieren. Es existieren viele Verfahren zur Vorverabreitung der Daten. Die geläufigsten Verfahren sind das Schwellenwertverfahren, das Rauschunterdrückungsverfahren und die Normalisation der Daten (Plamondon und Srihari, 2000; Arica und Yarman-Vural, 2001).

Schwellenwertverfahren

Eine Handschrift kann auf einem eingescannten Bild sehr blass erscheinen und deshalb die Erkennung beeinträchtigen. Aus diesem Grund wird in der Vorverarbeitungsphase das Schwellenwertverfahren eingesetzt. Das Verfahren erstellt aus dem ursprünglichen Bild ein Binärbild. Die Handschrift hebt sich durch diesen Vorgang vom Hintergrund ab (Otsu, 1979). (siehe Abbildung 2.3) Das Schwellenwertverfahren wird in die Bereiche globales und lokales Schwellenwertverfahren aufgeteilt. Beim globalen Verfahren gilt der definierte Schwellenwert für das gesamte Bild. Währenddessen wird beim lokalen Verfahren für jeden Pixel ein über die lokale Informationen berechneter Schwellenwert eingesetzt (Sahoo et al., 1988).

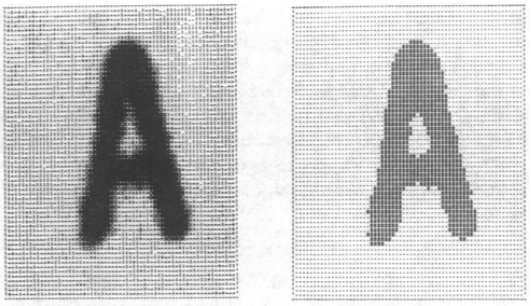


Abbildung 2.3.: Das linke Bild wird mit einem Schwellenwertverfahren bearbeitet. Das rechte Bild zeigt das Ergebnis. Quelle: Otsu, 1979

Rauschunterdrückungsverfahren

Das Eingangsbild kann aufgrund des Scanvorgangs möglicherweise falschfarbige Pixel enthalten. Diese falschfarbigen Pixel werden auch als Bildrauschen bezeichnet. Sie können die Handschrift unbeabsichtigt verändern, was sich zum Beispiel durch Lücken in einer Linie bemerkbar machen kann. Das Bildrauschen hat einen negativen Einfluss auf das Erkennungsergebnis. Das Rauschunterdrückungsverfahren versucht diese Problematik zu beheben (Buades et al., 2005). (siehe Abbildung 2.4)

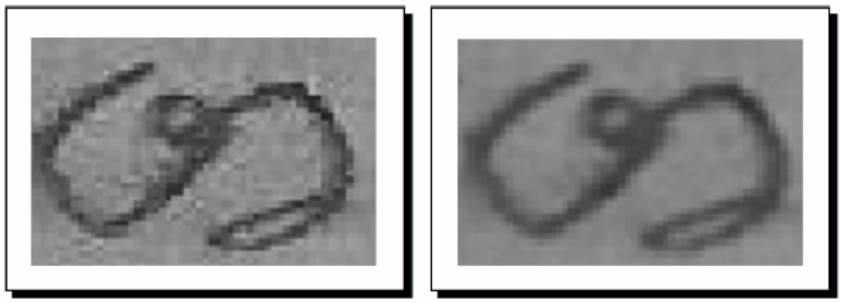


Abbildung 2.4.: Die Abbildung zeigt das Ausgangsbild und das Ergebnis eines Rauschunterdrückungsverfahren. Quelle: Surinta und Chamchong, 2008

Normalisierung

In diesem Verfahren wird die Handschrift normalisiert. Das Ziel der Normalisierung ist die Handschrift auf eine Einheit zu bringen. Alle Zeichen haben danach, wie in Abbildung 2.5 gezeigt wird, die gleiche Höhe. Sie können somit leicher von der Erkennungsmethode verarbeitet werden. Eine weitere Aufgabe der Normalisierung ist die Angleichung einer Handschrift. Sie kann entweder aus schief geschriebenen Buchstaben bestehen oder der gesamte Text wurde schräg geschrieben. Die Normalisierung versucht diese auftretenden Variationen zu beheben (Senior und Robinson, 1998; Vinciarelli und Luettin, 2001).

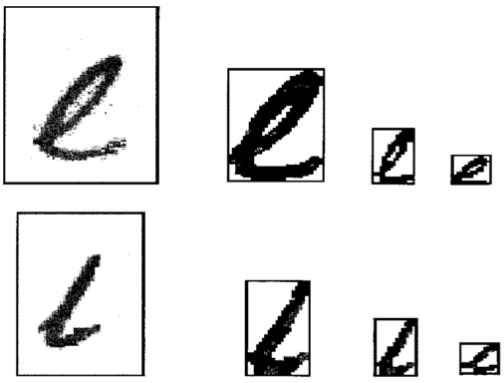


Abbildung 2.5.: Die Abbildung zeigt die Normalisierung der Buchstaben „e“ und „l“ . Quelle: Arica und Yarman-Vural, 2001

2.1.3.2. Segmentation

Das Eingangsbild wird durch die oben genannten Verfahren und Methoden aufbereitet. Der nächste Schritt in der Erkennung einer Handschrift ist die Segmentierung. Die Segmentierung versucht die Schrift in kleine sinnvolle Einheiten zu unterteilen (siehe Abbildung 2.6). Sie gliedert sich in externe und interne Segmentierung. Bei der externen Segmentierung wird die

Handschrift in Sätze, Zeilen oder Wörter unterteilt. In der internen Segmentierung wird dagegen versucht, die einzelnen Buchstaben oder Symbole aus der Eingabe heraus zu trennen (Arica und Yarman-Vural, 2001).

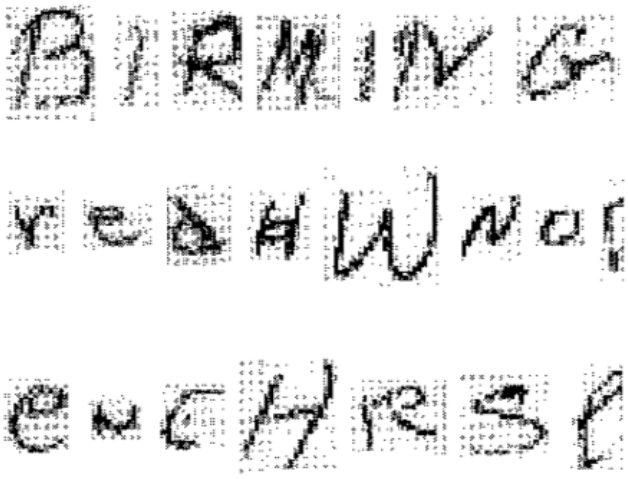


Abbildung 2.6.: Ergebnis einer internen Segmentation. Quelle: Plamondon und Srihari, 2000

2.1.3.3. Extrahierung von Merkmalen

Nachdem das Bild vorverarbeitet und die Handschrift daraus in sinnvolle Einheiten unterteilt wurde, ist der nächste Schritt die Merkmalsextraktion. Die Merkmalsextraktion ist für die Klassifikation von großer Bedeutung, denn je besser die definierten Merkmale das Symbol beschreibt, umso genauer kann die Erkennungsmethode das Objekt einer Klasse zuordnen. In der Phase der Merkmalsextraktion ergeben sich zwei Problemstellungen, die Merkmalsauswahl und die eigentliche Merkmalsextraktion. Die Merkmalsauswahl ist, wie der Name vermuten lässt, die Auswahl von passenden Merkmalen, die die verschiedenen Symbole am besten für die gewählte Klassifizierungsmethode unterscheidbar macht und damit eine hohe Klassifizierungsgenauigkeit erreicht. Merkmale, die bei einer Erkennungsmethode

gute Ergebnisse liefern, müssen nicht zwangsmäßig auch bei einer anderen Methode gute Resultate erzielen (Trier et al., 1996).

In der Merkmalsextraktion einer Handschrift gibt es zwei Arten von Merkmalen, statistische und strukturelle Merkmale. Bei den statistischen Merkmalen wird das Bild durch eine statistische Verteilung beschrieben. Zu den statistischen Merkmalen gehören beispielsweise mathematische Abbildungen (Funktionen) oder die Pixeldichte. Die strukturellen Merkmale basieren dagegen auf geometrischen und topologischen Eigenschaften der Symbole. Zu diesen Eigenschaften zählen zum Beispiel Striche, Endpunkte oder der Umriss eines Symbols. Mit den strukturellen Merkmalen kann im Gegensatz zu den statistischen Merkmalen das Originalbild aus den Merkmalen wiederhergestellt werden (Mohamad et al., 2015).

2.1.3.4. Training und Erkennung

Der nächste Schritt nach der Merkmalsextraktion ist die Erkennung der Handschrift. In dieser Phase wird versucht, ein unbekanntes Symbol einer vordefinierten Symbolklasse zuzuordnen. Dieser Vorgang wird auch als Klassifizierung bezeichnet. Arica und Yarman-Vural (2001) listen vier Vorgehensweisen zur Klassifizierung von Zeichen auf:

Template-Matching

Das Template-Matching ist eine Bildverarbeitungstechnik, die aus einem Teilbild ein gesamtes Bild identifizieren oder klassifizieren kann. Das Verfahren benötigt zur Erkennung eine Vorlage von jeder definierten Klasse (Zeichen). Die Zuordnung basiert auf einer Ähnlichkeitsberechnung, wie zum Beispiel der euklidischen Distanz. Ein neu zu klassifizierendes Teilbild wird mit jeder Vorlage verglichen und auf deren Ähnlichkeit überprüft. Das Teilbild wird dann der Klasse, mit der sie die höchste Ähnlichkeit aufweist,

zugeteilt. Dieses Verfahren hat jedoch mit Einschränkungen zu kämpfen. Bildrauschen und ähnlich geschriebene Symbole können die Erkennungsrate negativ beeinflussen (Tubbs, 1989; Trier et al., 1996).

Statistische Methoden

Die nächste Methode, um ein Zeichen zu klassifizieren, ist das statistische Verfahren. In einem statistischen Verfahren werden alle Zeichen durch einen Merkmalsvektor repräsentiert. Der Merkmalsvektor besteht aus den zuvor extrahierten Merkmalen (siehe Merkmalsextraktion). Der statistische Ansatz besteht aus einem Trainingsteil und einem Klassifizierungsteil. In der Trainingsphase wird eine Entscheidungsgrenze definiert, die mithilfe einer Wahrscheinlichkeitsverteilung antrainiert oder direkt angegeben werden kann. Mit der Entscheidungsgrenze kann jedes Zeichen eindeutig einer Klasse zugeordnet werden. Eine weitere Möglichkeit eine Entscheidungsgrenze zu definieren, ist die Angabe einer parametrischen Darstellung. Diese Darstellung muss zuerst antrainiert werden, um die bestmögliche Entscheidungsgrenze zu finden. Nach der Definition einer Entscheidnungsgrenze kann ein Symbol klassifiziert werden. Die Klassenzugehörigkeit wird vom definierten Merkmalsvektor und der Entscheidungsgrenze bestimmt (Jain et al., 2000).

Strukturelle Methoden

Die strukturelle Methode, auch syntaktische Methode genannt, nutzt anstatt den Merkmalsvektoren die strukturelle Information zur Klassifizierung. Es existieren mehrere Ansätze für die Repräsentation dieser strukturellen Information. Ein Ansatz dafür, ist die Information in einer hierarchischen, baum-ähnlichen Beschreibung abzubilden (siehe Abbildung 2.7). Ein weiterer Ansatz für die Repräsentation der Information mit einem syntaktischen Verfahren ist ein relationaler Graph. Die Zuordnung eines Symbols zu ei-

ner Klasse basiert auf der strukturellen Ähnlichkeit. Die Symbole werden dafür in kleinere Sub-Symbole (Muster) aufgeteilt. Ein Symbol kann so einfacher einer Klasse zugeordnet werden. Die einfachsten Muster werden als Primitive bezeichnet. Die Beziehung zwischen den aufgeteilten Mustern (Primitiven) und den unterschiedlichen Symbolen ist durch logische und mathematische Operationen definiert. Die Klassifizierung wird durch eine Syntax-Analyse bestimmt. Die Syntax-Analyse überprüft, wie sich die strukturelle Beschreibung eines unbekannten Symbols mit der von einer definierten Klasse unterscheidet. Das Symbol wird danach der Klasse mit der größten Ähnlichkeit zugeordnet (Albus et al., 2012).

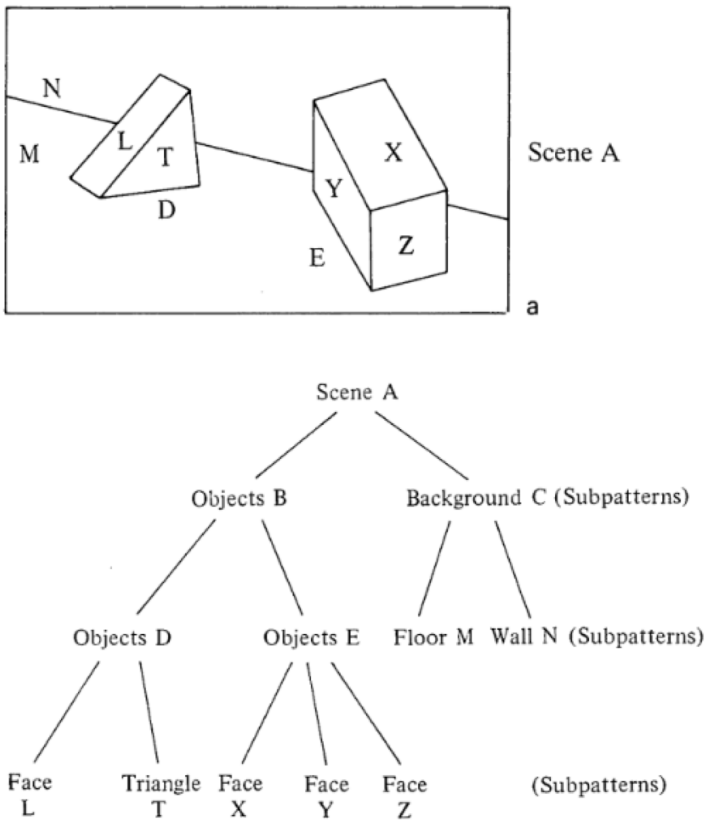


Abbildung 2.7.: Hierarchische, baum-ähnliche Beschreibung der Informationen von der Ansicht A. Quelle: Albus et al., 2012

Neuronale Netze

Das neuronale Netz beruht auf einem biologischen Vorbild. Es besitzt die grobe Struktur und Arbeitweise eines menschlichen Gehirns. Sie werden auch als Künstliche Neuronale Netze (KNN) bezeichnet. Dr. Robert Hecht-Nielsen definiert in »Neural Networks Primer, Part I« ein KNN so:

> „...a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs."

Ein KNN besteht, wie ein Gehirn, aus mehreren Neuronen, die für die Verarbeitung von Informationen zuständig sind. In einem neuronalen Netz werden diese Neuronen als Verarbeitungseinheit, Knoten oder Einheit bezeichnet. Jedes Neuron arbeitet eigenständig und führt seine Berechnungen parallel neben anderen Neuronen durch. Ein einzelnes Neuron beeinflusst somit das gesamte Ergebnis des neuronalen Netzes. Eine Verarbeitungseinheit kann mehrere Eingangssignale, aber nur ein Ausgangssignal besitzen. Sie besitzt außerdem einen kleinen Speicher, um den Wert, der sogenannten Gewichtung, zu speichern. Dieser gespeicherte Wert verändert das Eingangssignal. Die Übertragungsfunktion berechnet aus allen Eingangssignalen und der Gewichtung die Netzeingabe. Die Aktivierungsfunktion ermittelt danach das Ausgangssignal, welches an alle verbundenen Einheiten gesendet wird. Die Abbildung 2.8 zeigt den Aufbau einer Verwaltungseinheit.

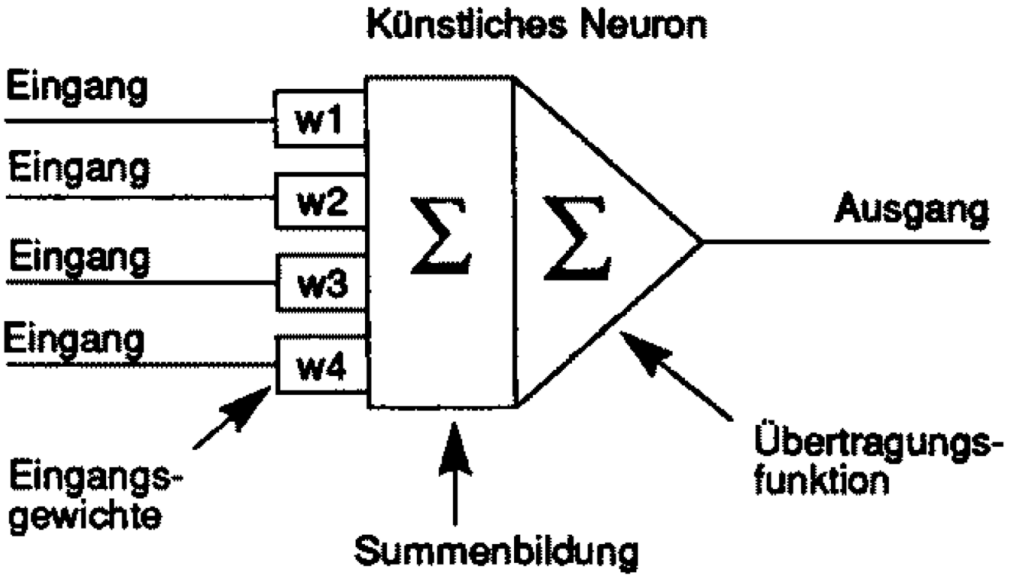


Abbildung 2.8.: Die Abbildung zeigt den Aufbau eines künstlichen Neurons. (Quelle: https://goo.gl/iXRC7h, letzter Aufruf: 25.08.2017)

Die Verarbeitungseinheiten werden je nach Netzwerkmodell in einer Eingabeschicht, einer verdeckte Schicht oder einer Ausgabeschicht aufgeteilt. (siehe Abbildung 2.9)

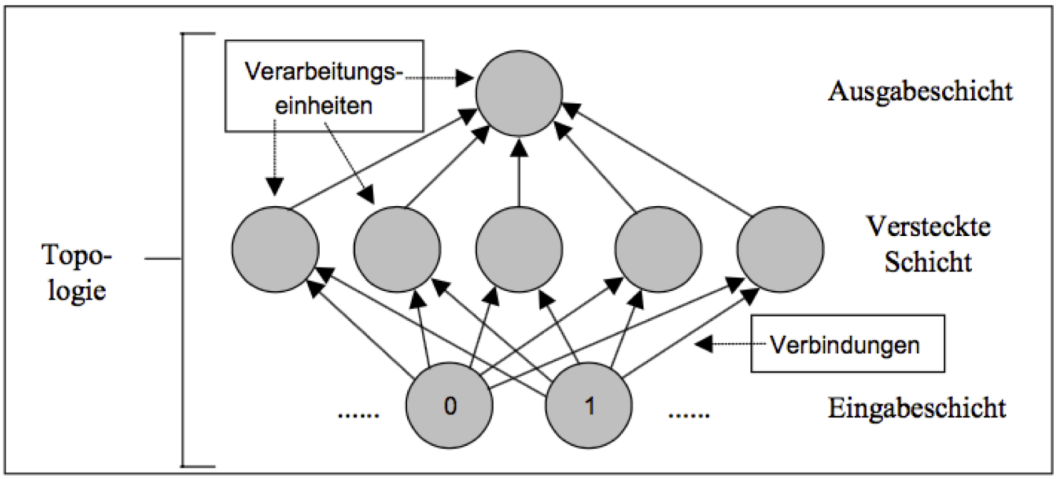


Abbildung 2.9.: Topologie eines neuronalen Netzes. Quelle: Strecker, 1997

Künstliche Neuronale Netze lassen sich aufgrund ihrer Verarbeitungsrichtung in zwei Klassen einteilen (Strecker, 1997):

- **Vorwärtsgerichte Informationsverarbeitung (Feed-Forward)**
  In Feedforward-Netzen werden die Signale an die darauf folgende Schicht weitergegeben. So kann beispielsweise eine Verarbeitungseinheit ihr Ausgangssignal nicht an dieselbe Schicht oder an vorige Schichten senden

- **Rückgekoppelte Informationsverarbeitung (Feed-Backward)**
  Die rückgekoppelte Informationsverabeitung ist unter anderem als rekurrentes oder rückgekoppeltes neuronales Netz bekannt. In einem solchen Netz ist die Verbindung einer Verabreitungseinheit mit sich selbst und jede andere Verbindung zwischen zwei Verarbeitungseinheiten möglich.

Neuronale Netze finden ihre Anwendung in der Handschrifterkennung (LeCun et al., 1998). Doch bevor ein Künstliches Neuronales Netz in der

Lage ist, ein Symbol zu klassifizieren, muss das Netz mittels Testdaten trainiert werden. Das Training ist ein iterativer Prozess und lässt sich in zwei Arten unterteilen (Hecht-Nielsen, 1988; Strecker, 1997; Kruse et al., 2015):

- **Überwachtes Lernen**
  Beim überwachten Lernen liegen zu den Testdaten die dazugehörigen Ausgaben vor. Nach jedem Durchlauf überprüft das neuronale Netz seine Ausgabe mit den vorliegenden korrekten Antworten. Bei einer hohen Fehlerrate ändert das Netz die Gewichtungen der Neuronen und startet einen neuen Durchlauf. Das Training wird erst beendet, wenn eine ansprechende Fehlerrate erreicht wurde.

- **Unüberwachtes Lernen**
  Im Gegensatz zum überwachten Lernen liegen beim unüberwachten Lernen keine Ausgabedaten vor. Das Netz erhält keine Rückmeldung, ob die Symbole korrekt klassifiziert wurden. Deshalb orientiert sich ein KNN auf die extrahierten Merkmale der Symbole, um sich selber ein Feedback zur Klassifizierung zu geben.

2.1.3.5. Nachverarbeitung

Die einzelnen Zeichen einer Handschrift werden in der vorigen Phase erkannt. Nun werden die erkannten Zeichen wieder zu einem Text zusammengefügt. Jedoch wurden im bisherigen Erkennungsvorgang noch keine semantischen oder kontextuellen Informationen berücksichtigt. Die Nachbearbeitung versucht diese Informationen aus dem Ergebnis der Erkennung zu extrahieren. Das Ziel der Nachbearbeitung ist demzufolge, die erkannten Wörter beziehungsweise Buchstaben wieder zusammenzusetzen, sodass sie mit dem eingegebenen Text übereinstimmen (Arica und Yarman-Vural, 2001).

2.2. Learning Analytics

Für den Begriff Learning Analytics (LA) existieren unterschiedliche Definitionen. Die erste internationale Konferenz für Learning Analytics and Knowledge beschreibt LA auf diese Weise (Long und Siemens, 2011):

> „Learning analytics is the measurement, collection, analysis and reporting of data about learners and their contexts, for purposes of understanding and optimising learning and the environments in which it occurs."

Siemens (2010) definiert LA wiederum folgendermaßen:

> „Learning analytics is the use of intelligent data, learner-produced data, and analysis models to discover information and social connections, and to predict and advise on learning."

Learning Analytics analysiert viele Informationen beziehungsweise Daten von einem Studenten oder einer Studentin, um damit den individuellen Lernprozess zu beobachten oder ihn vorherzusagen. Analytische Systeme erfassen diese gesammelten Daten und interpretieren daraus den angepassten Lernfortschritt des Studierenden oder der Studierenden (Elias, 2011; Johnson et al., 2011).

Khalil und Ebner (2015) stellen einen Learning Analytics Life Cycle vor, welcher aus vier Komponenten besteht (siehe Abbildung 2.10):

- **Learning Environment**
  Die Komponente *Learning Environment* umfasst die verfügbaren Bildungs- und Lernplattformen, sowie die Personen, die in einer solchen Umgebung involviert sind. Diese Stakeholder lassen sich in vier Gruppen zusammenfassen. In Abbildung 2.10 werden die Gruppen aufgelistet.

- **Big Data**
  Ein Studierender oder eine Studierende produzieren durch die Anwendung einer Lernplattform viele Daten. Solche Daten setzen sich unter anderem aus den persönlichen Daten oder den akademischen Informationen, die zum Beispiel die Noten oder die eingeschriebenen Kurse des Studierenden oder der Studierenden sind, zusammen.

- **Analytics**
  Das Ziel der Analyse ist das Herauslesen von aufschlussreichen Mustern aus den gesammelten Daten.

- **Act**
  Mit dem Ergebnis der Analyse und den erfassten Mustern wird versucht, die Zielvorgaben von LA zu erfüllen. Die Ziele sind beispielsweise die Vorhersage der Leistung eines Lernenden oder einer Lernenden in einem Kurs oder die Empfehlung von neuen Kursen und Büchern.

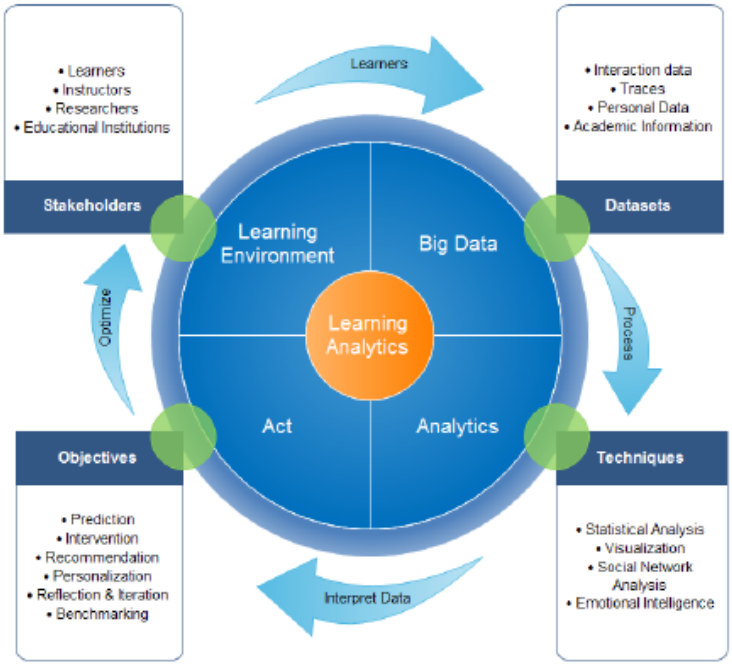


Abbildung 2.10.: Die Abbildung zeigt den Learning Analytics Life Cycle. Quelle: Khalil und Ebner, 2015

3. Applikationen

Im vorigen Kapitel wurde ein Einblick gegeben, wie die Erkennung einer Handschrift funktioniert. In diesem Abschnitt werden nun Applikationen vorgestellt, die über eine solche Technik verfügen. Das Hauptaugenmerk für diese Applikationen liegt auf dem mobilen Lernen, da für diese Arbeit eine iOS-Mathematik-Lernapplikation implementiert wurde. Außerdem werden nur Applikationen vorgestellt, die auf einem iOS-Betriebssystem ausgeführt werden können.

3.1. Todo Math

Das Spiel *Todo Math* von Enuma ist die umfangreichste Lernapplikation für Kinder, die in diesem Kapitel vorgestellt wird. Die Applikation kann auf einem iPhone oder iPad gespielt werden. Das Spiel kann nach der Eingabe eines Nicknamen begonnen werden. Das Kind hat die Wahl zwischen einem täglichen Training, einer freien Spielwahl oder einem Missionsmodus. Im Modus *Todo Math*„Freie Wahl“ kann aus dem gesamten Minispiele-Pool ein Minispiel ausgewählt werden. Die Minispiele teilen sich in folgenden Kategorien auf: *Zählen und Kardinalität*, *Zahlen und Operationen*, *Mathematisches Denken*, *Uhrzeit lesen* und *Geometrie*. Vor allem die Kategorie *Zahlen und Operationen* ist für diese Arbeit relevant und wird näher beschrieben. Die Spiele in dieser Kategorie sind *Schnelles Addieren*, *Schnelles Subtrahieren* und *Schnelles Multiplizieren*. Alle diese genannten Minispiele nutzen eine Hand-

schrifterkennung. Die Abbildung 3.1 zeigt das Minispiel *Schnelles Addieren*. Das tägliche Training und der Missionsmodus sind in Schwierigkeitsstufen aufgeteilt. Je höher das Level ist, desto mehr Aufgaben müssen erfüllt werden, um eine Belohnung zu erhalten. Als Belohnung winkt eine Monsterkarte, die den aktuellen Lernfortschritt widerspiegelt. Das Spiel verfügt zusätzlich zu einem normalen Modus für Kinder auch über einen Modus für Eltern. Die Eltern haben damit jederzeit einen Überblick über die gespielten Minispiele und können den Lernfortschritt des Kindes mitverfolgen.

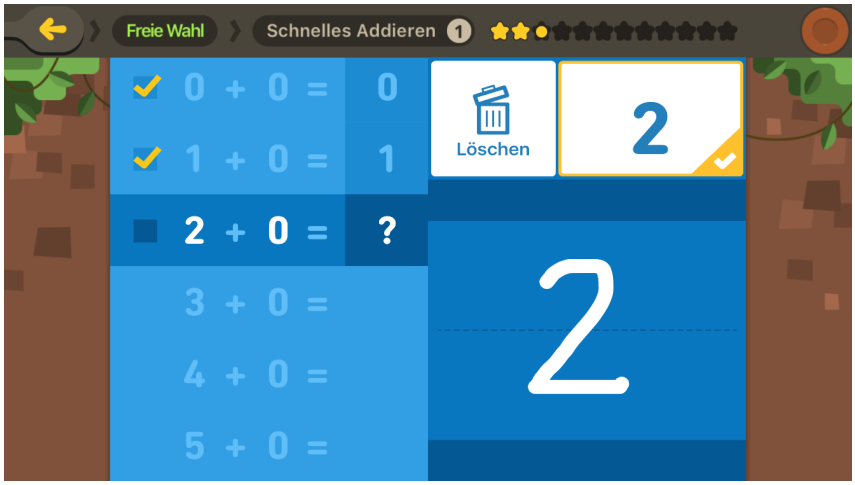


Abbildung 3.1.: Die Appildung zeigt das Spiel *Todo Math* im Minispiel *Schnelles Addieren*.

3.2. Handwriting Math Training for Kids

Das nächste Lernspiel mit einer Handschrifterkennung, das im *App Store* erhältlich ist, ist *Handwriting Math Training for Kids* von *ringsbell.net*. Dieses Spiel verfügt, wie das Spiel *Todo Math*, über einen Einstellungsbereich für Eltern. In diesem Bereich können sie Kinder hinzufügen und ihnen ihre bevorzugte Hand, mit der sie ihre Zahlen schreiben, einstellen. Das Hauptspiel findet auf einem Dach statt. Auf diesem Dach befinden sich

jeweils links und rechts ein Kamin und in der Mitte ein Fenster. Die Aufgabe des Spiels besteht darin, die vorgegebenen Zahlen in einer über dem Fenster vordefinierten Box nachzuschreiben. Bei jeder richtigen Zahl fallen Münzen, geprägt mit der nachgezeichneten Zahl, vom Himmel. Um in ein nächstes Level zu gelangen, müssen die heruntergefallenen Münzen vom Dach fallen. Fällt eine Münze in einen der beiden Kamine, öffnet sich das Fenster und die Spielfigur *Tsugesan* wirft verschiedene Gegenstände aus dem Fenster, welche zusätzliche Punkte liefern, sobald sie vom Dach runtergefallen sind. (siehe Abbildung 3.2)

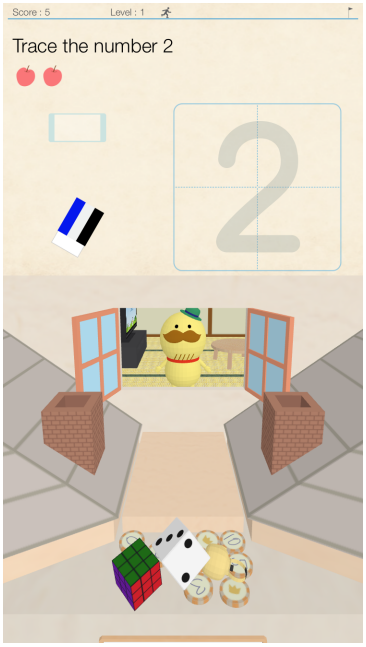


Abbildung 3.2.: Die Abbildung zeigt die Spielfigur *Tsugesan* vom Spiel *Handwriting Math Training for Kids*, wie sie einen Würfel aus dem Fenster wirft.

3.3. Handwriting Math Training

Die Entwickler und Entwicklerinnen *ringsbell.net* haben noch eine weitere Applikation mit einer Handschrifterkennung im *App Store*. Sie trägt den Namen *Handwriting Math Training*. Die App bietet im Hauptmenü den Addition-, Subtraktion- und den Multiplikationsmodus als Spielmöglichkeit an. Alle Möglichkeiten sind bis auf die mathematische Operation identisch. In den Spielmodi werden nacheinander mathematische Aufgaben gestellt, die so schnell wie möglich gelöst werden sollen. Die Anzahl der gelösten Aufgaben und die dafür benötigte Zeit werden im Hauptmenü unter dem Log-Tab zusammengefasst. Abbildung 3.3 zeigt das Spiel im Subtraktionsmodus.

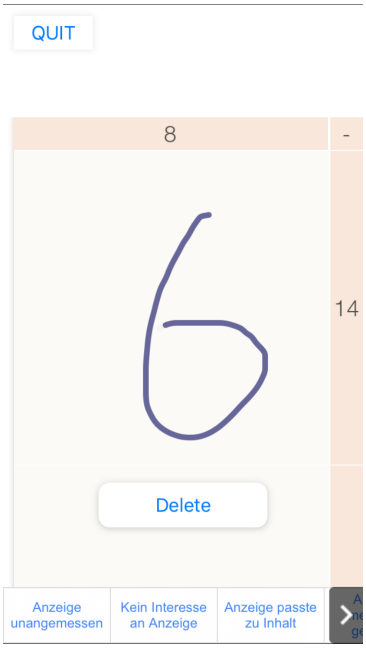


Abbildung 3.3.: Der *Subtraktionsmodus* im Spiel *Handwriting Math Training*

3.4. Write Math

Die Lernapplikation *Write Math* ist von den Entwicklern und Entwicklerinnen *Mutomedia*. Der Anwender oder die Anwenderin kann zwischen zwei Spielarten wählen. Die erste Spielart ist der Übungsmodus. In diesem Modus können die Aufgaben ohne Zeitdruck gelöst werden. Vor jedem Spiel kann der Zahlenraum, die mathematische Operation (Addition, Subtraktion, Multiplikation oder Division), die Anzahl der Aufgaben und die Aktivierung eines Sprachassistenten oder einer Sprachassistentin gewählt werden. Die zweite Spielart ist der Speedtest, in der die Applikation zusätzlich die Geschwindigkeit und die Genauigkeit des Anwenders oder der Anwenderin prüft. In der Abbildung 3.4 wird ein Screnshot vom beschriebenen Speedtest gezeigt. Am Ende eines Spieldurchlaufs werden in beiden Spielarten die Lösungen zu den gestellten Aufgaben und den gegebenen Antworten zusammengefasst.

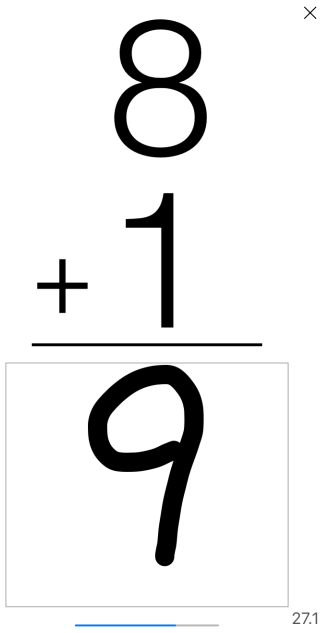


Abbildung 3.4.: Die Abbildung veranschaulicht den *Speedtest* in der *Write Math* Applikation.

3.5. ABCFunkid Calculus Lite

Die letze vorgestellte Applikation hat den Namen *ABCFunkid Calculus Lite* und stammt von den Entwicklern *Pyxsys SARL*. Sie kann, im Gegensatz zu den anderen aufgezählten Applikationen, nur auf einem iPad gespielt werden. Der Hauptbildschirm teilt sich in zwei Bereiche. Der obere Bereich ist für die Spieleinstellungen verantwortlich. Hier kann der Spieler oder die Spielerin die Schwierigkeit des Spiels verändern oder den Überblick über die bisher gelösten Aufgaben einsehen. Der untere Bereich ist wiederum in zwei Spielmodi, den Additions- und Multiplikationsmodus, aufgeteilt. In beiden Spielarten bekommt der Spieler oder die Spielerin zehn Rechenaufgaben gestellt, die in einer vorgegebenen Zeit gelöst werden sollen. Abbildung 3.5 zeigt das Spielgeschehen im Multiplikationsmodus. Die Applikation hat außerdem, so wie die *Todo Math* App, eine Sprachassistentin oder einen Sprachassistenten, der die aktuelle Rechenaufgabe vorlesen kann.

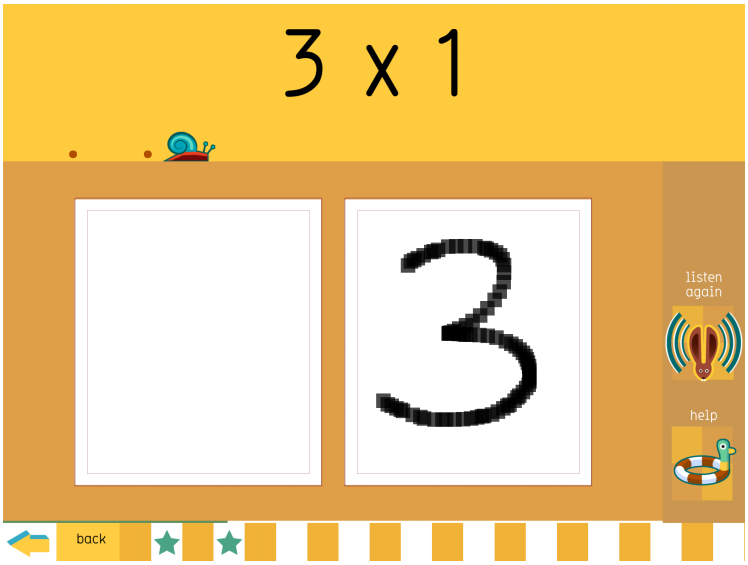


Abbildung 3.5.: Die Abbildung zeigt das Spiel *ABCFunkid Calculus Lite*.

3.6. Zusammenfassung

Die vorgestellten Applikationen zeigen, dass die Erkennung von handgeschriebenen Zahlen gut und sinnvoll in Mathematik-Lernapplikationen eingesetzt werden kann. Die Erkennungsmethoden, die in den verschiedenen Applikationen zum Einsatz kommen, können leider ohne Quellcode-Einsicht nicht identifiziert werden.

Die Applikationen verdeutlichen außerdem die bedeutungsvolle Rolle der Speicherung von erzielten Ergebnissen. Das zusätzliche Mitspeichern erhöht den Lerneffekt des Anwenders und der Anwenderin und führt dadurch zu einem schnelleren Lernfortschritt.

Als Basis für die Entwicklung der Lernapplikation dieser Arbeit werden nun die Ansätze der aufgezählten Applikationen und die daraus gewonnenen Ideen beziehungsweise Erkenntnisse herangezogen.

4. Prototyp

Die ersten beiden Kapitel haben einen theoretischen Einblick und die praktische Verwendung einer Handschrifterkennung gezeigt. Im nachfolgenden Kapitel wird nun die Umsetzung der mathematischen Lernanwendung, dem *1x1 Trainer*, behandelt. Zuallererst werden die verwendeten Technologien beschrieben, danach folgt eine nähere Beschreibung, wie die Applikation implementiert und wie die Erkennung der handgeschriebenen Zahlen umgesetzt wurde.

4.1. Verwendete Technologien

In diesem Unterkapitel werden alle Technologien vorgestellt, die in der entwickelten Applikation zum Einsatz kommen. Die Beschreibung beginnt mit dem Betriebssystem, auf dem die Anwendung läuft, und der Programmiersprache, mit der sie implementiert wurde. Anschließend folgt das Netzwerkprotokoll, welches für den Datenaustausch mit einem Server benötigt wird, und das benutzte neuronale Netz, das die Erkennung der Handschrift ermöglicht.

4.1.1. iOS - Swift

Das Lernspiel dieser Arbeit läuft auf dem Apple Betriebssystem iOS. Alle iPhones und iPads verwenden das iOS-Betriebssystem. Um das Spiel auf einem der Geräte zu installieren, wird mindestens die iOS-Version 10 benötigt. Mit der Veröffentlichung der iOS-Version 10 am 13.09.2017[1] wurde zeitgleich auch Swift mit der Version 3.0 und Xcode 8.0[2] veröffentlicht. Swift ist eine von Apple entwickelte Programmiersprache. Sie ermöglicht es nun, zusätzlich zur Programmiersprache Objective-C, eine mobile Applikation für alle Apple Geräte zu erstellen. Für die Entwicklung einer iOS-Applikation mit der Programmiersprache Swift muss die Entwicklungsumgebung Xcode benutzt werden. Xcode ist ebenfalls von Apple und ist ausschließlich auf einem Mac-Computer ausführbar. Die Entwicklungsumgebung enthält einen Compiler, Designer, Text-Editor, Debugger, einfach alles um eine iOS-Applikation entwickeln zu können. Das Testen oder Debuggen auf einem *echten* iPhone oder iPad ohne einer *Apple Developer Program*[3] Lizenz ist erst ab Xcode 8.0 möglich. Davor konnte eine in der Entwicklung stehende Applikation nur mit einem Simulator getestet werden.

Das Betriebssystem, oder besser gesagt das iOS Software Development Kit (SDK), stellt einem Entwickler und einer Entwicklerin viele nützliche Frameworks für die Applikationsprogrammierung zur Verfügung, wie etwa das *UIKit* Framework, mit dem die Benutzerschnittstelle definiert werden kann. Ein weiteres Framework, das das iOS SDK mit sich bringt, ist das *Core Data* Framework. *Core Data* verwaltet Datenmodelle und ermöglicht eine persistente Speicherung dieser definierten Modelle. Das Datenmodell beschreibt die Struktur der Daten in Bezug auf ihre Instanzen, Properties und deren Beziehungen untereinander.

[1] https://goo.gl/XLVVdx, letzter Aufruf: 28.8.2017
[2] https://goo.gl/M8QxsQ, letzter Aufruf: 28.8.2017
[3] https://goo.gl/dvp3Yy, letzter Aufruf: 28.8.2017

4.1.1.1. CocoaPods

In der Implementierung einer Software wird in den meisten Fällen auf existierende Softwarelösungen, den sogenannten *Third-Party*-Softwarekomponenten, zurückgegriffen. Eine einfache Möglichkeit solche Komponenten einem iOS-Projekt hinzuzufügen, ist die Verwendung von *CocoaPods*[4]. *CocoaPods* ist ein benutzerfreundlicher Bibliotheksmanager, der das Verwalten von externen Bibliotheken mit einer einzigen Textdatei übernimmt. Um damit Bibliotheken in Xcode-Projekte einzubinden, muss *CocoaPods* zunächst auf einem Mac-Rechner installiert werden. Nach der Installation von *CocoaPods* wird mit dem in Auflistung 4.1 gezeigten Terminal-Befehl im Projektverzeichnis eine Textdatei, das Podfile, erstellt.

```
$ pod init
```

Auflistung 4.1.: Terminal-Befehl, um *CocoaPods* zu einem Xcode-Projekt hinzuzufügen

In diesem Podfile werden die Bibliotheken eingetragen. Danach werden sie mit dem Befehl in Auflistung 4.2 heruntergeladen und dem Projekt automatisch hinzugefügt.

```
$ pod install
```

Auflistung 4.2.: Die Auflistung zeigt den Terminal-Befehl mit dem externe Bibliotheken installiert und dem Xcode-Projekt hinzugefügt werden

[4]https://goo.gl/Tjgh29, letzter Aufruf: 28.8.2017

4.1.2. SOAP

Das *Simple Object Access Protocol (SOAP)* ist ein Protokoll, welches für den Austausch von Daten in einer verteilten Umgebung verwendet wird. Es basiert auf der Erweiterbaren Auszeichnungssprache (XML) und funktioniert mit verschiedenen Transportprotokollen (Curbera et al., 2002).

Das Protokoll besteht laut Box et al. (2000) aus drei Teilen:

- **Envelope**
  Eine SOAP-Nachricht muss aus einem SOAP-Envelope und einem SOAP-Body bestehen. Zusätzlich zu diesen beiden Elementen kann der Nachricht ein SOAP-Header hinzugefügt werden, welcher aber nicht zwingend notwendig ist. Der Envelope-Teil repräsentiert das XML-Dokument als eine SOAP-Nachricht. Der Body enthält wichtige Informationen für den Datenaustausch und im Header können der Nachricht anwendungsspezifische Informationen beigefügt werden.

- **Zeichenkodierung**
  Damit spezifische Datentypen in einem XML-Dokument erkannt werden können, hat das Protokoll eine eingebaute Zeichenkodierung. Es wird zwischen einfachen Typen, wie beispielsweise int oder string, und einem aus mehreren einfachen Typen zusammengefassten Container unterschieden. Ein Beispiel für einen solchen Container ist die aus der Programmierung bekannte Struktur.

- **Remote Procedure Call (RPC)**
  Das SOAP kann mittels dem *RPC* eine existierende Methode auf dem Empfängersystem aufrufen. Für einen solchen Aufruf müssen der Methodenname und die dafür benötigten Argumente in der SOAP-Nachricht angegeben werden. Der Rückgabewert der aufgerufenen Methode wird genauso wie die Aufrufnachricht mittels einer SOAP-Nachricht zurückgesendet.

4.1.3. Convolutional Neural Network

Das *Convolutional Neural Network (CNN)* erzielt in der Erkennung von handgeschriebenen Zahlen sehr gute Ergebnisse (Simard et al., 2003; Ciregan et al., 2012). Ein CNN ist ein spezielles feedforward neuronales Netz und unterscheidet sich durch ihre Struktur von anderen neuronalen Netzen. So ist ein Neuron in den verdeckten Schichten nicht mit allen Neuronen der nachfolgenden Schicht verbunden, sondern nur mit einem kleinen Bereich der vorigen Schicht (Bishop, 2006). Die Abbildung 4.1 zeigt die angesprochene Grundstruktur eines CNN. Sie setzt sich aus einer Convolutional- und einer Subsampling-Schicht zusammen.

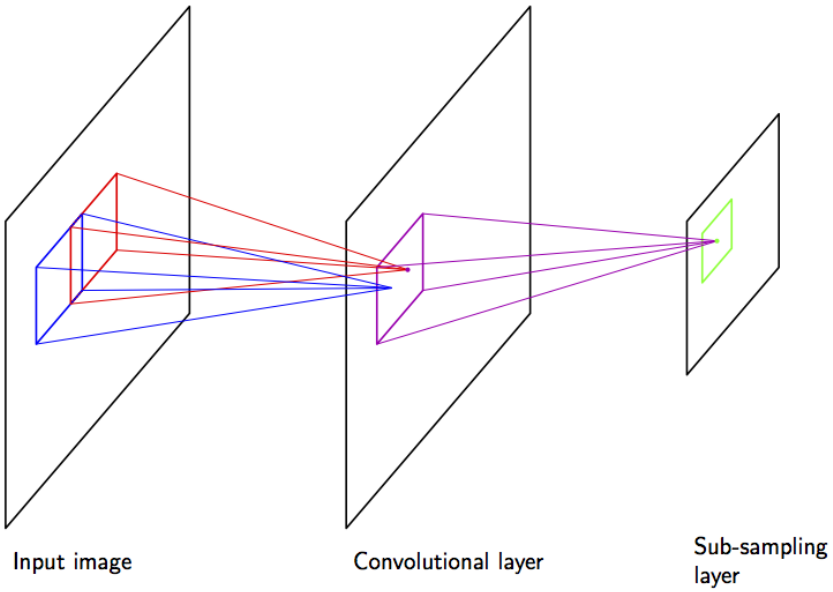


Abbildung 4.1.: Die Abbildung veranschaulicht die Verbindung zwischen den einzelnen Schichten eines CNN. Quelle: Bishop, 2006

Die Convolutional-Schicht unterteilt sich in mehrere Ebenen. Jede einzelne Ebene dieser Schicht extrahiert mit einem Merkmalsfilter ein anderes

Merkmal. Aus diesem Grund werden diese Ebenen auch als *feature maps* bezeichnet (Bishop, 2006; Zeiler und Fergus, 2014).

Nach einer Convolutional-Schicht kommt eine Subsampling-Schicht zum Einsatz. Anstatt eines Subsampling kann auch eine Pooling-Schicht eingesetzt werden. In dieser Schicht wird die Auflösung von den Ebenen der vorangegangen Convolutional-Schicht reduziert. Beim Max-pooling wird beispielsweise ein kleiner quadratischer Bereich über die Ebene gelegt und daraus immer der größte Wert gewählt, bis die gesamte Ebene abgetastet wurde (Ciresan et al., 2011).

Das CNN setzt sich aus ein oder mehreren Schichten, bestehend aus einer Convolutional- und einer Subsampling-Schicht, zusammen und schließt mit einer voll verbundenen Schicht ab (Ciresan et al., 2011). Die Abbildung 4.2 zeigt den Aufbau eines mehrschichtigen CNN für eine Erkennung von handgeschriebenen Zahlen.

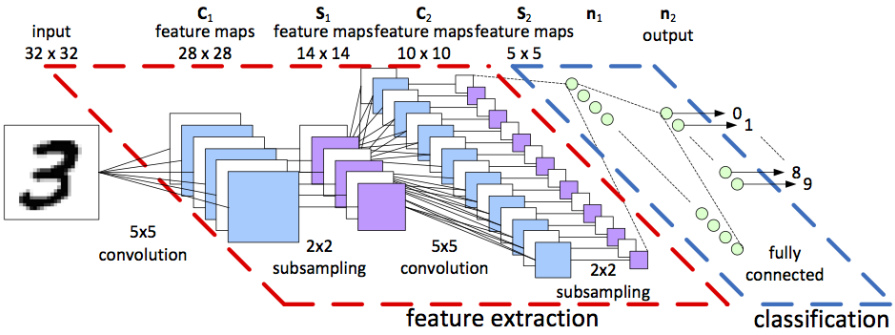


Abbildung 4.2.: EIn mehrschichtiges *Convolutional Neural Network* für eine Erkennung von handgeschriebenen Zahlen. Quelle: Peemen et al., 2011

4.2. Implementierung

Dieses Unterkapitel beinhaltet die detaillierte Beschreibung zur Umsetzung der Lernapplikation *1x1 Trainer*. Es setzt sich aus der Erklärung der entwickelten Lernapplikation, den verwendeten externen Bibliotheken sowie den benötigten Algorithmus und einer genauen Erläuterung der Funktionsweise der Handschrifterkennung zusammen.

4.2.1. Idee

Der *1x1 Trainer* ist eine Mathematik-Lernapplikation, die das Üben oder Lernen des Einmaleins ermöglicht. Die Applikation ist im Apple's *App Store* erhältlich und kann daher nur auf einem iPhone oder iPad ausgeführt werden. Sie wurde hauptsächlich für Kinder im Alter von sechs bis zwölf Jahren entwickelt. Aus diesem Grund ist eine klare und strukturierte Benutzeroberfläche ein sehr wichtiger Punkt. Ein Kind sollte sich nicht durch zu viel Text ablenken oder entmutigen lassen. Deshalb verzichtet die Applikation auf viel Text und setzt eher auf den Einsatz von visuellen Elementen. Die Besonderheit dieser Lernapplikation ist allerdings die Fähigkeit, handgeschriebene Zahlen zu erkennen. Die Antworten der gestellten Rechenaufgaben können einfach mit dem Finger auf den Bildschirm des mobilen Gerätes geschrieben werden. Die Umsetzung und der genaue Ablauf dieser Erkennung werden in Kapitel 4.2.7 beschrieben.

4.2.2. Layout

Das Design ist für eine Lernapplikation für Kinder von großer Bedeutung. Kinder, die diese Applikation verwenden, sollen durch das gewählte Design der Benutzeroberfläche angesprochen und zugleich auch motiviert werden.

Nachfolgend werden die einzelnen Screens des *1x1 Trainer* kurz beschrieben, damit ein besseres Verständnis für die Applikation entwickelt werden kann.

4.2.2.1. Hauptmenü

Nach dem Start der Applikation ist das Hauptmenü der erste Screen, den der Anwender oder die Anwenderin zu Gesicht bekommt. Er ist die zentrale Schaltstelle der Applikation. Der Screen enthält zwei Buttons, die den Spieler oder die Spielerin jeweils zu einer anderen Spielvariante führen. (siehe Abbildung 4.3) Eine genaue Erklärung zu diesen zwei Spielmodi ist in Kapitel 4.2.3 zu finden.

Abbildung 4.3.: Hauptbildschirm des 1x1 Trainer

4.2.2.2. Login

Der Login-Screen erscheint, wenn der Benutzer oder die Benutzerin den „Trainer"-Button im Hauptmenü antippt und noch kein User angemeldet ist. Dieser Screen ist für das Anmelden mit einem TU-Graz-Learning-Analytics-Konto[5] zuständig. Falls eine Person über kein Benutzerkonto verfügt, kann sie sich über den „BIST DU NEU?"-Button schnell und einfach ein Benutzerkonto anlegen. Nach korrekter Eingabe der Anmeldedaten und positiver Rückmeldung vom Server wird der *Onboard*-Screen angezeigt. Die Abbildung 4.4 zeigt den *Login*-Screen.

Abbildung 4.4.: Die Abbildung zeigt den Login-Bildschirm der entwickelten Lernapplikation.

[5]https://schule.learninglab.tugraz.at/, letzter Aufruf: 28.8.2017

4.2.2.3. Onboarding

Der *Onboard*-Screen wird vor jedem Spielstart angezeigt. Es gibt zwei mögliche Szenarien, um ihn aufzurufen. Der erste Weg ist durch den „Kurzspiel"-Button im Hauptmenü. Der andere Weg ist durch eine erfolgreiche Anmeldung im *Login*-Screen. Dieser Screen erklärt dem Anwender oder der Anwenderin, dass die Zahlen in vordefinierte Boxen geschrieben werden sollen. Zu diesem Zweck bekommt der Anwender oder die Anwenderin eine Zahl zum Nachschreiben. Erst wenn die Zahl korrekt nachgeschrieben wurde, kann das Spiel mit dem „SPIELEN"-Button gestartet werden. (siehe Abbildung 4.5)

Abbildung 4.5.: Die Abbildung zeigt die Onboard-Phase und die korrekte Nachzeichnung der vorgegebenen Zahl.

4.2.2.4. Hauptspiel

Das Spiel startet und der *Spiel*-Screen erscheint. Der *Spiel*-Screen beinhaltet viele Spielelemente. Diese sind unter anderem die zwei schwarz umrandeten Boxen, die in der Mitte des Screens ihren Platz finden. In diesen beiden Boxen sollen die Antworten zu den gestellten Rechenaufgaben geschrieben werden. Weiters befinden sich unterhalb der Boxen zwei Buttons. Der linke Button ist der „OK"-Button und bestätigt die Eingabe der geschriebenen Zahl. Der rechte Button ist der „LÖSCHEN"-Button. Er löscht die Eingabe in beiden Boxen, damit eine neue Eingabe getätigt werden kann. Am unteren Rand des *Spiel*-Screens ist außerdem eine Animation, die die Anzahl der gelösten Beispiele repräsentiert, sowie ein „X"-Button, der sich im linken unteren Eck befindet. Mit diesem Button kann das Spiel sofort beendet werden. Die grüne Tafel am oberen Rand enthält die Rechenaufgabe, die gelöst werden soll. Links und rechts neben der Aufgabe befinden sich die Spieluhr und der aktuelle Punktestand. Der genaue Spielablauf in beiden Spielvarianten wird in Kapitel 4.2.3 beschrieben. Die Abbildung 4.6 zeigt den *Spiel*-Screen.

Abbildung 4.6.: Die Abbildung zeigt das Spielgeschehen im *Kurzspiel*-Modus.

4.2.2.5. Spielzusammenfassung

Die Applikation fasst nach jedem Spieldurchlauf das Spiel im *Spielzusammenfassung*-Screen zusammen. Sie zeigt die erreichten Punkte und bietet die Möglichkeit, sich alle gestellen Rechenbeispiele mit den zugehörigen Lösungen und den gegebenen Antworten anzusehen. Außerdem fordert sie die Eingabe eines Namens. (siehe Abbildung 4.7) Der eingegebene Name wird danach mit dem Punktestand in den Highscore-Daten gespeichert.

Abbildung 4.7.: Die Spielzusammenfassung wird nach jedem Spielmodi angezeigt und fasst alle Ergebnisse zusammen.

4.2.2.6. Bestenliste

Der *Bestenliste*-Screen zeigt je nach Spielvariante eine Rangliste mit den bislang am meist erreichten Punkten und den dazugehörigen Namen. Er kann entweder im *Onboard*-Screen oder nach der Spielzusammenfassung betrachtet werden. In Abbildung 4.8 wird eine Beispiel-Bestenliste gezeigt.

Abbildung 4.8.: Die Bestenliste-Tabelle im *1x1 Trainer*

4.2.3. Spielvarianten

Der *1x1 Trainer* verfügt über zwei Spielmodi, die sich das gleiche Spielprinzip teilen. Sie unterscheiden sich nur durch die Auswahl der Aufgaben und durch das Mitspeichern des Lernfortschrittes. Im ersten Modus steht ein Server im Hintergrund, der mit Hilfe von einem Algorithmus die passenden Rechenaufgaben auswählt. Im anderen Modus werden hingegen die Rechenaufgaben mit reinem Zufallsprinzip ausgewählt. Dem Spieler oder der Spielerin werden in beiden Spielvarianten 20 Rechenaufgaben gestellt, für die es pro Aufgabe ein Zeitlimit von 60 Sekunden gibt. Je schneller der Spieler oder die Spielerin die Rechenaufgabe richtig löst, desto mehr Punkte bringt die gelöste Aufgabe. Die maximale Punkteanzahl für eine Frage sind 250 Punkte. Die Punktezahl ergibt sich demnach aus folgender Formel:

$$\frac{\text{verbleibende Zeit}}{\text{erlaubte Zeit pro Frage}} \cdot \text{maximale Punkte Zahl pro Frage}$$

Am Ende jedes Spiels werden die erreichten Punkte und die gestellten Beispiele zusammengefasst, sodass ein zusätzlicher Lerneffekt entstehen soll.

4.2.3.1. Trainer

Der erste Modus ist der *Trainer*. Dieser Modus benötigt eine bestehende Internetverbindung und ein Benutzerkonto[6]. Der Internetzugang ist unbedingt erforderlich, da die Applikation sonst keine Verbindung zum TU-Graz-Server herstellen kann. Die Kommunikation ist mit den aufgelisteten SOAP-Nachrichten in Kapitel 4.1.2 aufgebaut. Um den *Trainer*-Modus spielen zu können, muss sich ein Anwender oder eine Anwenderin zuvor mit einem Benutzerkonto anmelden. Der *Trainer*-Modus holt sich die Spieldaten von einem Server. Der Server speichert vom angemeldeten Benutzer oder der angemeldeten Benutzerin mit dem Prinzip von *Learning Analytics* den gesamten Lernfortschritt. Ein auf dem Server liegender Algorithmus, der in Kapitel 4.2.5 genauer erläutert wird, berechnet mit diesem Lernfortschritt die Rechenaufgaben, die dem Schwierigkeitsgrad des Benutzers oder der Benutzerin entsprechen.

4.2.3.2. Kurzspiel

Der Modus *Kurzspiel* ist die zweite Möglichkeit den *1x1 Trainer* zu spielen. Er kann im Gegensatz zum *Trainer* auch ohne einen Internetzugang und einem Benutzerkonto gespielt werden. Das *Kurzspiel* speichert den Lernfortschritt nicht mit, einzig der Punktestand von einem Spieldurchlauf wird für die Highscore-Tabelle gespeichert. Da diesem Modus kein Server im Hintergrund zur Verfügung steht, der neue Rechenaufgaben liefert, bedarf es einer anderen Möglichkeit, um den Benutzer oder die Benutzerin mit

[6] https://schule.learninglab.tugraz.at/, letzter Aufruf: 28.8.2017

neuen Rechenaufgaben zu fordern. Dies wird mit dem, in Kapitel 4.1.1 kurz angesprochenen, *Core Data*-Framework umgesetzt. Alle möglichen Rechenaufgaben werden beim ersten Start der Applikation initialisiert und im persistenten Speicher der Applikation gespeichert. Sobald das Spiel eine Aufgabe anfordert, wählt die Applikation durch Zufallsprinzip eine neue Aufgabe aus dem persistenten App-Speicher aus.

4.2.4. Pods

Der *1x1 Trainer* nutzt für bestimmte Problemstellungen die Hilfe und Unterstützung von externen Bibliotheken. Für die Installation dieser Bibliotheken wird CocoaPods verwendet. Folgende Bibliotheken, die in CocoaPods auch als *Pods* oder *CocoaPod* bezeichnet werden, finden in der Applikation ihre Verwendung:

- **Alamofire**
  Alamofire ist eine Schnittstelle für die von Apple bereitgestellten Klassen und Funktionen zur Kommunikation mit dem *Hypertext Transfer Protocol (HTTP)*. Sie erleichtert die Handhabung und die Implementierung solcher Netzwerkverbindungen. Die Applikation setzt Alamofire für das Versenden und Empfangen von den in Kapitel 4.1.2 defnierten SOAP-Nachrichten ein.

- **SWXMLHash**
  Der CocoaPod *SWXMLHash* ist eine Bibliothek, die das Parsen von XML-Dokumenten erleichtet. Der *1x1 Trainer* verwendet sie für das Parsen von empfangenen SOAP-Nachrichten, um die darin enthaltenen Informationen herauszulesen.

- **AEXML**
  AEXML ermöglicht, zusätzlich zum Lesen, auch das Schreiben von XML-Dokumenten. Ihre Verwendung findet die Bibliothek in der Erstellung und Initialisierung der SOAP-Nachrichten.

- **CryptoSwift**
  Mit dem *CryptoSwift* Pod können kryptographische Verfahren mit der Programmiersprache Swift eingesetzt werden. Sie verschlüsselt die Anmeldeinformationen eines Anwenders oder einer Anwenderin in der SOAP-Methode *isUserAllowed* .

- **SAConfettiView**
  Die Lernapplikation soll für Kinder möglichst ansprechend sein. Deswegen findet die Bibliothek *SAConfettiView* auch ihren berechtigten Einsatz in dieser implementierten Applikation. Sie hat die besondere Fähigkeit, einen Konfettiregen über den gesamten Bildschirm regnen zu lassen und wird im *Spielzusammenfassung*-Screen eingesetzt.

4.2.5. Algorithmus

Die Lernapplikation nutzt einen von Schön et al. (2012) entwickelten Algorithmus, der im *Trainer*-Modus für die Auswahl der Rechenaufgaben verantwortlich ist. Der Algorithmus ermöglicht ein individuelles Lernen, indem er Rechenbeispiele in Abhängigkeit vom vorhandenen Wissen des Anwenders oder der Anwenderin bereitstellt. Er besteht aus den im folgenden näher beschriebenen Komponenten. Ein Zusammenspiel dieser Komponenten zeigt das Ablaufdiagramm in Abbildung 4.11.

4.2.5.1. Fragen (Rechenbeispiele)

Alle Rechenbeispiele sind im Vorfeld anhand ihres Schwierigkeitsgrades mit einer Zahl zwischen null und eins eingeteilt worden. Aufgaben, die leichter zu lösen sind, bekommen eine hohe Wahrscheinlichkeit (1,0), dagegen bekommen schwierig zu lösende Aufgaben eine niedrige Wahrscheinlichkeit (0,0). Die beantworteten Rechenaufgaben werden in der Datenbank durch folgende Werte klassifiziert:

- **0**: Diese Klasse enthält Rechenaufgaben, die falsch beantwortet wurden.
- **1**: Zu dieser Klasse werden Aufgaben zugeordnet, die richtig beantwortet wurden.
- **2**: Eine Frage, die zweimal hintereinander richtig beantwortet wurde, ordnet sich dieser Klasse zu.

4.2.5.2. Lernfähigkeit

Die Lernfähigkeit ist ein Parameter, der dem Algorithmus zeigt, wie hoch das Können des Anwenders oder der Anwenderin ist. Er soll außerdem eine

Hilfestellung für den Algorithmus sein, welche Fragen für den Benutzer oder der Benutzerin schwieriger oder leichter zu beantworten sind. Die Lernfähigkeit wird mit einem Wert zwischen null und eins repräsentiert. Damit der Anwender oder die Anwenderin nicht überfordert oder unterfordert ist, ist die Lernfähigkeit in zwei Bereiche, dem erweitertem Lernbereich und dem aktuellen Lernbereich, unterteilt. Der aktuelle Lernbereich beginnt bei null und endet mit dem Wert der aktuellen Lernfähigkeit. Der erweiterte Lernbereich liegt bis zu 25% über der aktuellen Lernfähigkeit. Die Abbildung 4.9 zeigt eine visuelle Darstellung der Lernfähigkeit.

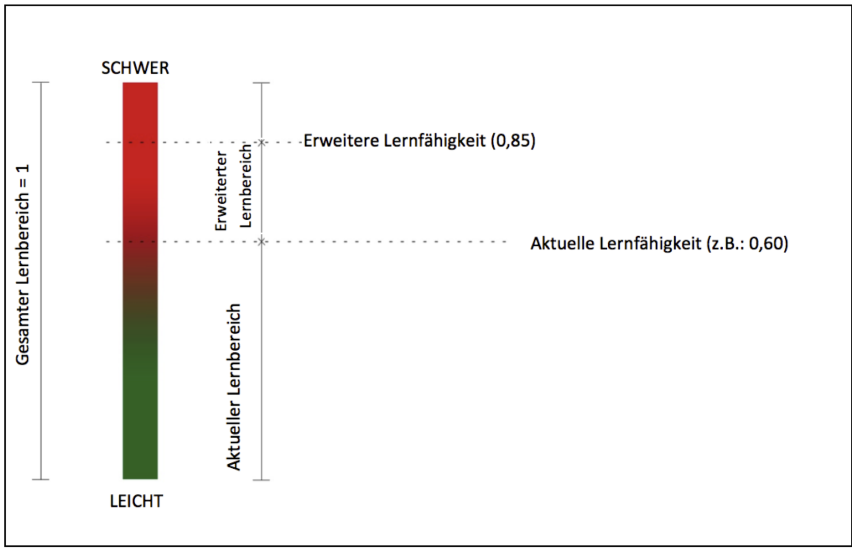


Abbildung 4.9.: Die Abbildung zeigt die visuelle Darstellung der Lernfähigkeit. Quelle: (Schön et al., 2012)

4.2.5.3. Vortest

Die Lernfähigkeit ist dem Algorithmus beim allerersten Spielen nicht bekannt. Aus diesem Grund werden dem Anwender oder der Anwenderin zu Beginn zwei Rechenaufgaben gestellt. Mit den Antworten aus diesen

beiden Rechenaufgaben berechnet sich der Algorithmus die Lernfähigkeit. Die Abbildung 4.10 stellt die Vorgehensweise des Vortests dar.

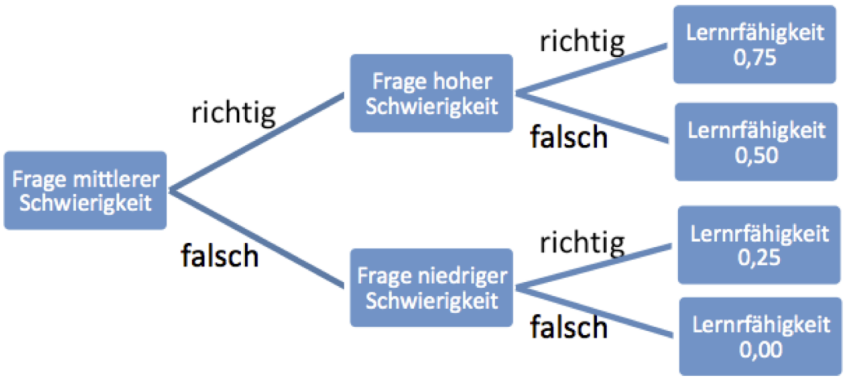


Abbildung 4.10.: Ablauf des Vortests. Quelle: (Schön et al., 2012)

4.2.5.4. Auswahl einer Frage

Die Auswahl der Rechenaufgabe bestimmt eine Zufallszahl zwischen null und eins. Dazu werden drei mögliche Fälle behandelt:

- **1.Fall**: Ist die Zufallszahl kleiner oder gleich 0,05, wählt der Algorithmus eine Rechenaufgabe aus dem aktuellen Lernbereich mit der Klassifizierung 2.
- **2.Fall**: Die Zufallszahl ist größer 0,05 und kleiner gleich 0,15, so wählt er ebenfalls eine Aufgabe aus dem aktuellen Lernbereich mit einer 1-Klassifizierung.
- **3.Fall**: Der dritte Fall trifft ein, wenn die Zahl größer als 0,15 ist. Dann wählt der Algorithmus ein unbekanntes Rechenbeispiel aus dem aktuellen oder erweiterten Lernbereich.

4.2.5.5. Verarbeitung der Antwort

Mit jeder Beantwortung einer Rechenaufgabe verändert sich die Lernfähigkeit und die Klassifizierung der beantworteten Frage. Eine falsch beantwortete Frage wird der Klasse *0* zugeordnet. Bei einer richtigen Antwort muss zwischen einer neuen Frage oder einer bereits korrekt beantworteten Frage unterschieden werden. Die neue Frage kommt deshalb in Klasse *1* und die bereits richtig beantwortete Aufgabe in Klasse *2*.

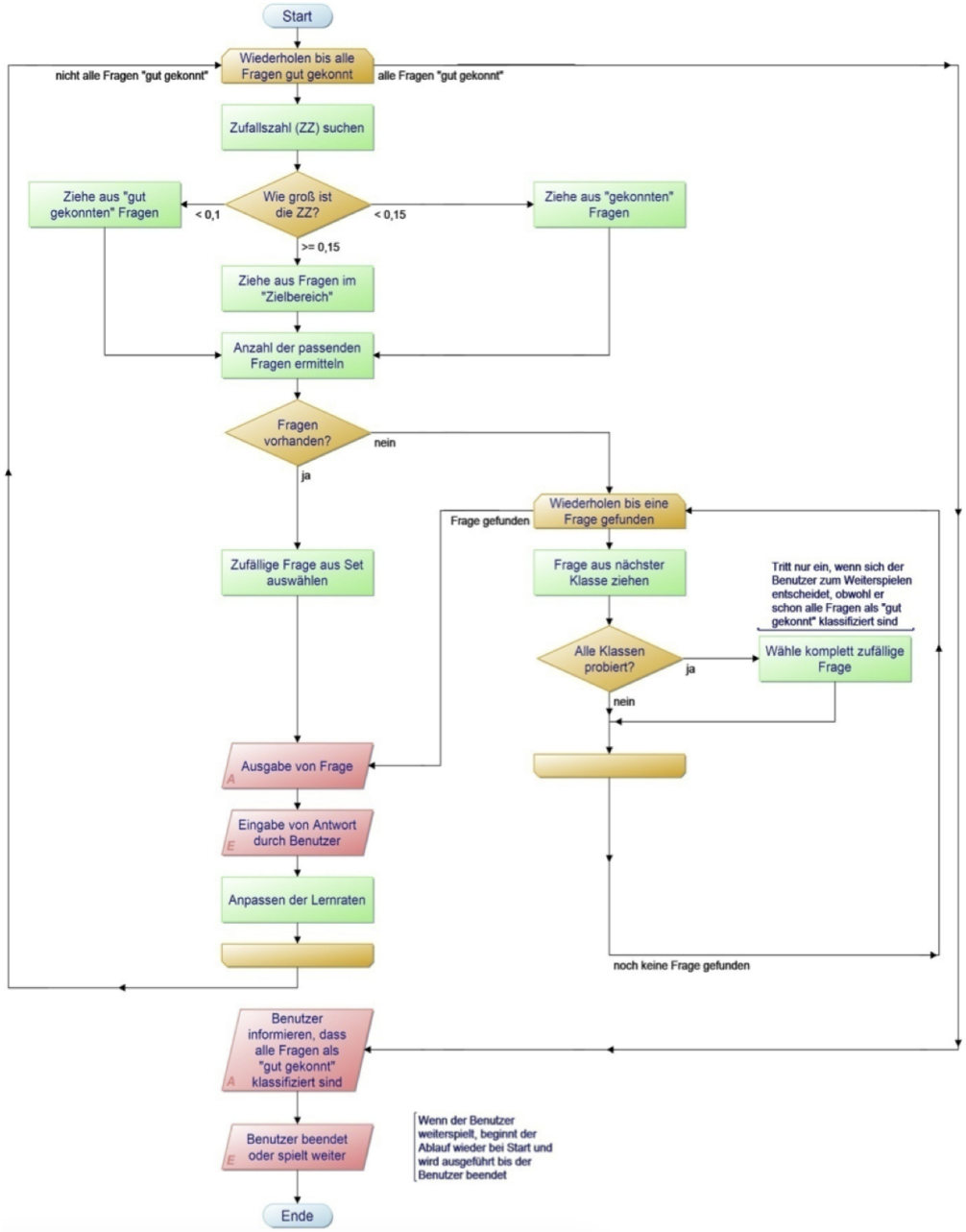


Abbildung 4.11.: Die Abbildung zeigt das Ablaufdiagramm des vorgestellten Algorithmus. Quelle: (Schön et al., 2012)

4.2.6. SOAP Requests

Der *1x1 Trainer* nutzt das Netzwerkprotokoll *SOAP* für die Kommunikation mit dem Server und dem Algorithmus. Die mobile Applikation verwendet folgende Methoden:

4.2.6.1. isUserAllowed

Diese Anfrage dient zur Anmeldung. Sie überprüft, ob sich ein Anwender oder eine Anwenderin mit den korrekten Benutzerdaten anmeldet.

- Request
  - *username*: Der Benutzername des Users
  - *password*: Das Passwort des Users
  - *idApp*: Die Identifikationsnummer der Applikation
  - *hmacClient*: Der HMAC-Wert dient zur Überprüfung der Vertraulichkeit
- Response
  - *accepted*: Dieser Wert gibt an, ob die Anmeldung erfolgreich war.
  - *idUser*: Die Identifikationsnummer des Users
  - *message*: Eine Nachricht für zusätzliche Informationen
  - *hmac*: Zur Überprüfung der Vertraulichkeit

4.2.6.2. sessionService

Die *sessionService*-Methode teilt dem Server vor jedem Spielstart mit, dass ein neues Spiel beginnt.

- Request
  - *idPlatform*: Die Plattform Identifikationsnummer
  - *idUser*: Die Identifikationsnummer des angemeldeten Users
- Response
  - *return*: Die Identifikationsnummer der neuen Session

4.2.6.3. questionService

Die Methode *questionService* fordert vom Server eine neue Rechenaufgabe an.

- Request
  - *idUser*: Die Identifikationsnummer des angemeldeten Users
- Response
  - *id*: Die Identifikationsnummer des angemeldeten Users
  - *label*: Die Aufgabe als Text
  - *answer*: Die Lösung der Rechenaufgabe

4.2.6.4. answerService

Die *answerService*-Methode teilt dem Server die Antwort der gestellten Rechenaufgabe mit.

- Request
  - *answer*: Eingegebene Lösung des Users
  - *questionId*: Die Identifikationsnummer der Rechenaufgabe
  - *reactionTime*: Die benötigte Zeit für die Beantwortung
  - *sessionId*: Die Identifikationsnummer der Session
- Response
  - *return*: Wert, ob die Anfrage erfolgreich verarbeitet wurde

4.2.7. Schrifterkennung

Das entwickelte Lernspiel nutzt für die Erkennung der handgeschriebenen Zahlen ein mehrschichtiges Convolutional Neural Network. Das CNN besteht aus zwei Convolution-Schichten, an der jeweils eine Max-Pooling-Schicht anschließt und einer voll verbundenen Schicht. Die Abbildung 4.2 zeigt die Struktur des verwendeten CNN in der Applikation.

4.2.7.1. Training

Das definierte *Convolutional Neural Network* gehört zuerst mit geeigneten Daten antrainiert, damit es eine handgeschriebene Nummer erkennt. Für das Training des neuronalen Netzes werden *TensorFlow* und die *MNIST*-Datenbank verwendet.

TensorFlow ist eine von Google entwickelte Bibliothek für maschinelles Lernen (Abadi et al., 2016). Diese Bibliothek baut das neuronale Netz auf und lernt es mit der *MNIST*-Datenbank an. Die *MNIST*-Datenbank ist eine Bilderdatenbank, die Bilder von handgeschriebenen Zahlen in der Größe von 28x28 Pixel enthält. Sie beinhaltet 60.000 Trainingsbilder und weitere 10.000 Bilder für das Testen des angelernten neuronalen Netzes (LeCun et al., 1998).

4.2.7.2. Anwendung

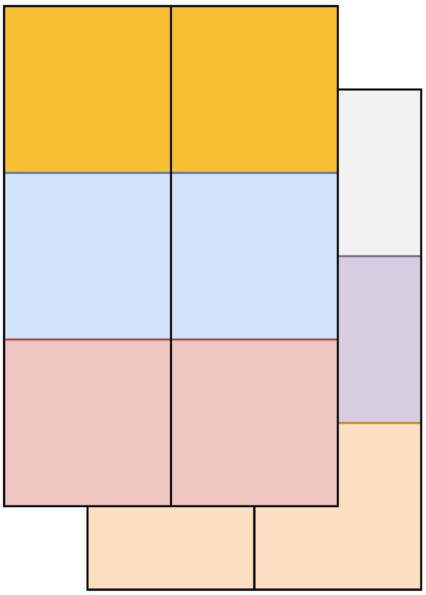
Das Trainingsskript speichert vom antrainierten *Convolutional Neural Network* die Gewichtungs- und Bias-Werte von allen Schichten in separate Dateien. Damit der *1x1 Trainer* das CNN mit den gespeicherten Werten einsetzen kann, wird die in iOS-10 eingeführte *Basic neural network subroutines* verwendet.

BNNS ist Teil des *Accelerate*-Frameworks[7] und bietet dem Entwickler und der Entwicklerin Funktionen für das Ausführen von neuronalen Netzen. Das *BNNS* unterstützt in einer iOS-Applikation nur neuronale Netze, die bereits mit Testdaten angelernt wurden.

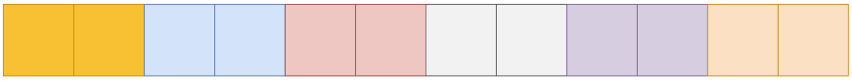
Das Einlesen der *TensorFlow*-Daten mit der *BNNS* birgt Gefahren, was dazu führen kann, dass das neuronale Netz nicht korrekt funktioniert. Es existieren Unterschiede bei der Anordnung der Pixelwerte in einem Array zwischen der Bibliothek *TensorFlow* und der *BNNS*. *BNNS* erwartet, dass die Pixelwerte in Form von Zeile für Zeile und Kanal für Kanal in der Datei vorliegen. *TensorFlow* speichert die Daten dagegen in einer komplett anderen Form. Die Bilddaten liegen in Form von Pixel pro Kanal für Pixel pro Kanal vor. In Abbildung 4.12 werden die Unterschiede der Repräsentation der Bilddaten graphisch dargestellt. Eingesetzte Matrixoperationen im Trai-

[7] https://developer.apple.com/documentation/accelerate , letzter Aufruf: 28.08.2017

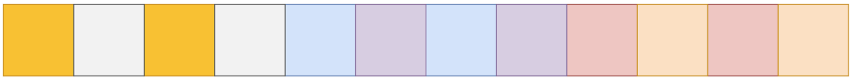
ningsskript beheben diese Probleme.



(a) Anordnung der Pixel in einem Bild



(b) BNNS Anordnung



(c) Tensorflow Anordnung

Abbildung 4.12.: Abbildung (a) zeigt die vorgegebene Anordnung der Pixel in einem Bild. Die Abbildungen (b) und (c) zeigen die Anordnungen der Pixel, die von BNNS und Tensorflow unterstützt wird. Quelle: https://goo.gl/Mfgx8M, letzter Aufruf 29.08.2017

Die *NeuralNetwork*-Klasse und die *BNNSBuilder*-Klasse sind für das Einlesen der Daten und die Erstellung des *Convolutional Neural Network* verantwortlich. Der *BNNSBuilder* ist ein Wrapper um das *BNNS* und dient zu einem übersichtlicheren und strukturierteren Aufbau des neuronalen Netzes. Auf-

listung 4.3 zeigt den Quellcode für die Erstellung des CNN.

```
1  private class func setupNetwork() -> BnnsNetwork {
2      return BnnsBuilder()
3          .shape(width: 28, height: 28, channels: 1)
4          .kernel(width: 5, height: 5)
5          .convolve(weights: weights[0], bias: weights[1])
6          .shape(width: 28, height: 28, channels: 32)
7          .maxpool(width: 2, height: 2)
8          .shape(width: 14, height: 14, channels: 32)
9          .convolve(weights: weights[2], bias: weights[3])
10         .shape(width: 14, height: 14, channels: 64)
11         .maxpool(width: 2, height: 2)
12         .shape(width: 7, height: 7, channels: 64)
13         .connect(weights: weights[4], bias: weights[5])
14         .shape(size: 1024)
15         .connect(weights: weights[6], bias: weights[7])
16         .shape(size: 10)
17         .build()!
18 }
```

Auflistung 4.3.: Quellcode für die Erstellung des neuronalen Netzes

Das Schreiben der Zahlen ist in zwei vordefinierten Quadraten erlaubt. Je ein Quadrat steht für eine Zahl. Dies bringt zum einen den Vorteil, dass die Applikation die einzelnen Zahlen aus einer Eingabe nicht herausfiltern muss. Zum anderen bleibt die Applikation übersichtlich und einfach zu bedienen. Die Abbildung 4.6 zeigt diese zwei Boxen für die Eingabe der handgeschriebenen Zahlen.

Die Klasse *DrawingView* repräsentiert ein solches Quadrat. Sie hält außerdem eine Instanz von der *CanvasView*-Klasse, die für das Schreiben der Zahlen eingesetzt wird. Die *CanvasView*-Klasse ist von der Klasse *UIImageView*[8] abgeleitet und verwaltet alle Events und Operationen, die für das Zeichnen in einem Quadrat benötigt werden. Sie nutzt dafür folgende drei Touch-Events:

[8]https://developer.apple.com/documentation/uikit/uiimageview, letzter Aufruf: 28.08.2017

- **touchesBegan**
  Die Methode *touchBegan* wird ausgeführt, wenn der Berührungspunkt im Quadrat liegt. Sie speichert nur den Berührungspunkt, um ihn gegebenenfalls in der nächsten Event-Methode zeichnen zu können.

- **touchesMoved**
  Der Finger zieht in dem definierten Rechteck weiter. Daraufhin ruft das Betriebssystem die *touchesMoved*-Methode auf. Diese Methode zeichnet mit der *drawLine*-Funktion eine Linie in das Quadrat, die vom letzten gespeicherten Berührungspunkt startet und mit dem aktuellen Berührungspunkt endet.

- **touchesEnded**
  Diese Methode wird aufgerufen, sobald der Finger den Kontakt mit dem Quadrat verliert. Sie zeichnet den letzten Berührungspunkt und startet die Erkennung mit der Delegate-Funktion *canvasViewDidFinish-Drawing*.

Alle Screens, die eine Erkennung vom gezeichneten Inhalt der Quadrate benötigen, implementieren das *CanvasViewDelegate* Protokoll. Das Protokoll stellt die Funktion *canvasViewDidFinishDrawing* zum Implementieren bereit, da die Ergebnisse der Erkennung in den einzelnen Screens unterschiedlich verarbeitet werden.

Nachdem der Anwender oder die Anwenderin in eines der vordefinierten Quadrate gezeichnet hat, ruft die oben genannte Event-Methode *touchesEnded* die *canvasViewDidFinishDrawing*-Methode des aktuellen Screens auf. In dieser Methode werden alle in diesem Screen vorhandenen *DrawingView*-Instanzen mit der Erkennung ihres Inhaltes beauftragt. Dafür ruft die Applikation die Funktion *classifyDrawingInput* mit jedem *DrawingView*-Quadrat als Parameter auf. Diese Funktion erstellt zunächst mit Hilfe des

UIKit-Frameworks[9] und dem Inhalt der übergebenen Box ein UIImage[10]. Danach wird das erstellte UIImage für die Erkennung vorverarbeitet.

Die Vorverarbeitung beinhaltet die Zentrierung der geschriebenen Zahl im Bild und die darauffolgende Skalierung des UIImages. Die Skalierung ist notwendig, da das *Convolutional Neural Network* ausnahmslos eine Bildgröße von 28x28 Pixel benötigt.

Nach der Vorverarbeitung startet die Erkennung mit dem vorverarbeiteten UIImage. Das Bild durchläuft daraufhin die unten angeführten Schritte, die ergänzend in Abbildung 4.13 graphisch dargestellt werden.

1. Die erste Schicht ist eine Convolutional-Schicht und berechnet 32 Merkmale für jeden der 5x5 Kernel.
2. Die zweite Schicht ist eine Max-Pooling-Schicht mit einem 2x2 Filter. Diese Schicht verkleinert die Bildgröße auf 14x14 Pixel.
3. Anschließend berechnet eine weitere Convolutional-Schicht 64 Merkmale für jeden der 5x5 Kernel.
4. Danach folgt eine weitere Max-Pooling-Schicht mit einem 2x2 Filter, die das Bild auf 7x7 Pixel verkleinert.
5. Nach diesen vier Schichten folgt eine voll verbundene Schicht. Die voll verbundene Schicht dient zur Verarbeitung des gesamten Bildes.
6. Abschließend kommt eine *Softmax*-Schicht[11] zum Einsatz, die aus dem Ausgangsvektor der vorigen Schicht einen Vektor mit zehn Einträgen erzeugt. Die zehn Einträge stellen die Klassen der Zahlen von null bis neun dar. Der höchste Wert in diesem Vektor gibt die Klassenzugehörigkeit an.

[9] https://goo.gl/UV4EJ2, letzter Aufruf: 28.08.2017

[10] https://developer.apple.com/documentation/uikit/uiimage, letzter Aufruf: 28.08.2017

[11] https://goo.gl/i2KcnA , letzter Aufruf: 28.08.2017

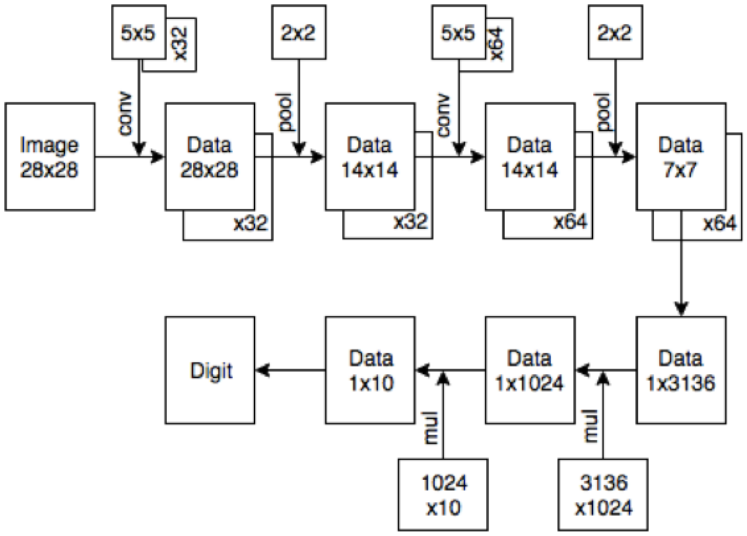


Abbildung 4.13.: Die Abbildung veranschaulicht die beschriebenen Schritte der Bild- und Datentransformationen des CNN. Quelle: https://goo.gl/Mfgx8M, letzter Aufruf 29.08.2017

5. Evaluation

Das nachfolgende Kapitel beschreibt die Evaluation der implementierten Lernapplikation und deren Ergebnisse. Das erste Unterkapitel behandelt den Aufbau und die Durchführung der Evaluation. Danach werden die Ergebnisse der Evaluation erläutert und im letzten Abschnitt dieses Kaptitels werden die gewonnenen Erkenntnisse dieser Evaluation zusammengefasst und diskutiert.

5.1. Aufbau

Die Evaluation wurde mit der iPad-Klasse in der Volksschule Graz-Hirten durchgeführt. Die Klassenlehrerin Dipl.-Päd. Silvana Aureli konnte bei der Evaluation bedauerlicherweise nicht teilnehmen. Sie wurde von der Klassenlehrerin der 1.b-Klasse Frau Dipl.-Päd. Sylvia Ruhs vertreten. Die iPad-Klasse der VS Graz-Hirten besteht aus 25 Kindern und besitzt 11 iPads.

Für die Evaluation konnten leider nur sieben iPads genutzt werden, da die restlichen vier iPads zu diesem Zeitpunkt noch keine iOS-Version 10 installiert hatten. Am Tag der Evaluation waren 20 der 25 Kinder anwesend. Aus diesem Grund wurden die Schulkinder am Anfang der Evaluation in zwei 7er-Gruppen und einer 6er-Gruppe aufgeteilt. Nach einer kurzen Vorstellung meiner Person startete die erste der drei Gruppen mit der Evaluation.

Zu Beginn herrschte kurzzeitig eine hektische Stimmung, da vergessen wurde, den WLAN-Router für die benötigte Internetverbindung einzuschalten. Danach beruhigte sich die Stimmung wieder und jede Gruppe spielte nacheinander für 15 Minuten den *1x1 Trainer*. Das Spiel wurde den Kindern zuvor allerdings nicht erklärt. Sie sollten dadurch nicht voreingenommen werden und sich selber einen Eindruck vom Spiel machen, da ihre eigene Meinung für die spätere Befragung von Bedeutung ist. Die Abbildung 5.1 zeigt die zweite Gruppe beim Spielen des *1x1 Trainers*.

Abbildung 5.1.: Die Abbildung zeigt die Kinder der VS Graz-Hirten beim Spielen mit dem 1x1 Trainer

Kinder sind oft sehr zurückhaltend und schüchtern, wenn sie alleine befragt werden. Um dieses Problem zu lösen, wurden die Gruppen deshalb, nachdem sie den *1x1 Trainer* gespielt hatten, in noch kleinere Gruppen zu je vier oder drei Schulkinder aufgesplittet. So ergaben sich schlussendlich sechs Gruppen. Diese Gruppen wurden alle im Lehrerzimmer befragt, da

in der VS Graz-Hirten kein freier Raum zur Verfügung stand. Diese Befragung ist auch der Hauptteil der Evaluation der Lernapplikation. In dieser Befragung wurden den Schulkindern nacheinander vier unterschiedliche Fragen gestellt:

1. Das Spiel hat mir gefallen.
2. Ich habe mich gleich im Spiel zurechtgefunden.
3. Mit dem Finger die Zahlen zu schreiben, macht mir Spaß.
4. Ich würde den „1x1 Trainer“ auch zu Hause spielen.

Als Hilfestellung für die Beantwortung der genannten Fragen wurden den Schulkindern fünf Smileys vorgelegt:

Tabelle 5.1.: Evaluation-Bewertungen-Smileys

	Sehr gut / Ja auf alle Fälle
	Gut / Ja denke schon
	Normal / Naja ich bin mir nicht ganz sicher
	Schlecht / Nein eher nicht
	Sehr schlecht / Nein auf keinen Fall

5.2. Ergebnisse

In diesem Unterkapitel werden die einzelnen Gruppenergebnisse zusammengefasst.

5.2.1. Gruppe 1

Die allererste Gruppe, die befragt wurde, bestand aus vier Kindern. Die Abbildung 5.2 zeigt die Antworten der Befragung.

1. *Das Spiel hat mir gefallen.*
   Bei dieser Frage war sich die Gruppe sehr schnell einig und bewertete sie mit dem besten Smiley. Das Spiel hat den Kindern sehr gut gefallen, weil es ihrer Meinung nach interessant ist und es dem Anwender oder der Anwenderin zusätzlich lernt, schöner zu schreiben.
2. *Ich habe mich gleich im Spiel zurechtgefunden.*
   Die Schulkinder fanden das Spiel aufgrund der Handschrifterkennung zwischenzeitlich zu schwer. Das wiederholte Schreiben der Zahl hat sie genervt. Deswegen wurde diese Frage nur mit einem hellgrünen Smiley bewertet.
3. *Mit dem Finger die Zahlen zu schreiben, macht mir Spaß.*
   Bis auf die Erkennung der Zahlen hat den Kindern das Schreiben mit dem Finger in dieser Gruppe Spaß gemacht, weil die Zahlen leicht und schnell geschrieben werden können. Die Frage bekam von der Gruppe die beste Bewertung.
4. *Ich würde den „1x1 Trainer“ auch zu Hause spielen.*
   Alle Kinder in dieser Gruppe würden den *1x1 Trainer* auch zu Hause spielen. Sie bewerteten die Frage mit dem dunkelgrünen Smiley.

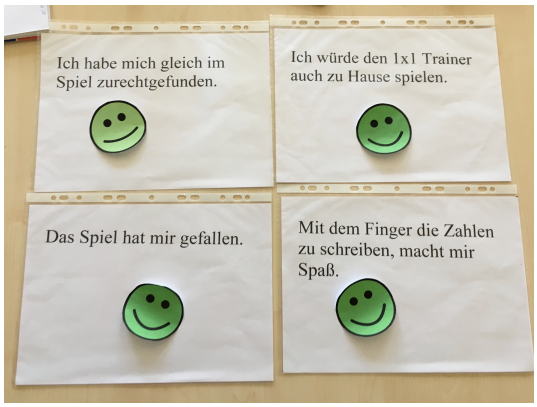


Abbildung 5.2.: Ergebnisse der Gruppe 1

5.2.2. Gruppe 2

Die zweite Gruppe setzte sich aus drei Schulkindern zusammen. In der Abbildung 5.3 wird das Ergebnis der gestellten Fragen gezeigt.

1. *Das Spiel hat mir gefallen.*
   Die Gruppe fand das Spiel super. Jedoch nervte es die Kinder, dass machmal eine falsche Zahl von der Lernapplikation erkannt wurde. Bewertet wurde die Frage mit einem hellgrünen Smiley.
2. *Ich habe mich gleich im Spiel zurechtgefunden.*
   Ein Kind in dieser Gruppe hatte Anfangsschwierigkeiten und wusste nicht, was wo gemacht werden sollte. Ein anderes Schulkind wiederum tat sich bei der Anmeldung mit den Benutzerdaten schwer. Es wollte sich mit seinem echten Namen anmelden und nicht mit den Benutzerkontodaten, die alle Kinder vor der Evaluation bekommen haben. Aus diesen Gründen wurde die Frage mit einem gelben Smiley beantwortet.

3. *Mit dem Finger die Zahlen zu schreiben, macht mir Spaß.*
   Mit dem Finger die Zahlen zu schreiben, machte den Kindern aus dieser Gruppe Spaß, wurde aber andererseits auch als anstrengend und schwer bewertet, weil die Applikation die handgeschriebenen Zahlen oft falsch erkannte. Sie geben daher die Zahlen lieber mit dem Ziffernblock ein. Die Kinder beantworteten diese Frage mit einem hellgrünen Smiley.
4. *Ich würde den „1x1 Trainer" auch zu Hause spielen.*
   Die Kinder halten das Spiel für eine gute Idee. Es macht ihnen Spaß und verbessert ihrer Meinung nach die Einmaleins-Kenntnisse. Wenn sie Zeit haben, dann würden sie den *1x1 Trainer* zu Hause spielen. Diese Frage wurde als einzige der vier Fragen mit einem dunkelgrünen Smiley bewertet.

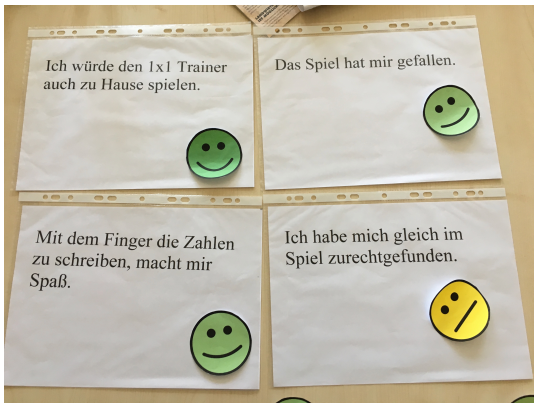


Abbildung 5.3.: Ergebnisse der Gruppe 2

5.2.3. Gruppe 3

Die dritte Gruppe bestand wieder aus vier Kindern. Die Abbildung 5.4 veranschaulicht die Bewertungen der einzelnen Fragen dieser Gruppe.

1. *Das Spiel hat mir gefallen.*
   Der *1x1 Trainer* hat den Kindern in dieser Gruppe gut gefallen, weil die Zahlen selbst zu schreiben sind. Das Einzige was die Kinder gestört hat, ist die hin und wieder falsche Erkennung der geschriebenen Zahlen. Die Frage wurde mit einem hellgrünen Smiley bewertet.
2. *Ich habe mich gleich im Spiel zurechtgefunden.*
   Die Schulkinder haben sich im Spiel zurechtgefunden und wussten sofort was zu tun war. Folglich wurde die Frage von der Gruppe mit einem dunkelgrünen Smiley beantwortet.
3. *Mit dem Finger die Zahlen zu schreiben, macht mir Spaß.*
   Ein Kind von dieser Gruppe hatte keinen Spaß mit dem Finger zu schreiben, da die Applikation seine Handschrift schwer erkannte. Es tippt die Zahlen lieber mit einem Ziffernblock ein. Die anderen Kinder empfanden das Schreiben mit dem Finger als lustig und leicht. Die Gruppe bewertete die Frage mit einem dunkelgrünen Smiley.
4. *Ich würde den „1x1 Trainer“ auch zu Hause spielen.*
   Wenn die Kinder Lust auf das Einmaleins hätten, würden sie den *1x1 Trainer* auch zu Hause spielen. Sie beantworteten die Frage mit einem gelben Smiley.

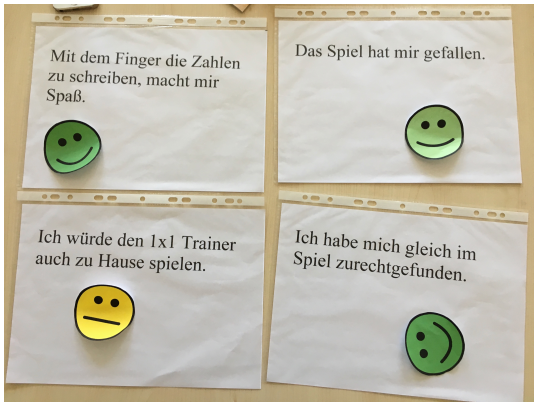


Abbildung 5.4.: Ergebnisse der Gruppe 3

5.2.4. Gruppe 4

Diese Gruppe setzte sich aus drei Kindern zusammen. Die Abbildung 5.5 zeigt die Bewertungen, die in der Gruppe getroffen wurde.

1. *Das Spiel hat mir gefallen.*
   Die Lernapplikation hat der gesamten Gruppe sehr gut gefallen. Ihnen hat außerdem gefallen, dass mit dem Spiel die Handschrift verbessert werden kann. Die Frage wurde mit dem besten Smiley bewertet.
2. *Ich habe mich gleich im Spiel zurechtgefunden.*
   Zwei Kindern war zuerst unklar, was sie in der Applikation machen müssen. Die Frage wurde nichtsdestotrotz mit einem guten Smiley bewertet.
3. *Mit dem Finger die Zahlen zu schreiben, macht mir Spaß.*
   Einem Schulkind wurde beim Schreiben mit dem Finger langweilig. Es hat lieber vorgegebene Zahlen zum Antippen. Ein anderes Kind bemängelte die schlechte Erkennung von schief geschriebenen Zahlen. Der gelbe Smiley ist die zusammen ausgewählte Bewertung dieser Frage.
4. *Ich würde den „1x1 Trainer“ auch zu Hause spielen.*
   Die Kinder würden den *1x1 Trainer* eher weniger zu Hause spielen, da das Lernspiel langweilig wird. Die Aussagen der Kinder spiegelt sich demnach auch in der Bewertung wider.

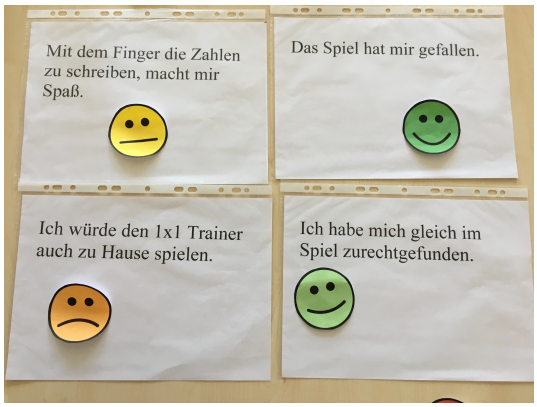


Abbildung 5.5.: Ergebnisse der Gruppe 4

5.2.5. Gruppe 5

In der fünften Gruppe wurden drei Kinder befragt. Die Antworten der Fragen sind in Abbildung 5.6 zu sehen.

1. *Das Spiel hat mir gefallen.*
   Die Schulkinder fanden das Spiel lustig. Sie ärgerten sich über die falsche Erkennung der Zahlen, bewerteten die Frage aber trotzdem mit einem dunkelgrünen Smiley.
2. *Ich habe mich gleich im Spiel zurechtgefunden.*
   Die befragten Kinder hatte keine Probleme sich im Spiel zurechtzufinden. Diese Frage wurde, wie die erste Frage, mit einem dunkelgrünen Smiley bewertet.
3. *Mit dem Finger die Zahlen zu schreiben, macht mir Spaß.*
   Den Schulkindern machte das Schreiben mit dem Finger aufgrund des großen Bildschirmes des iPads Spaß. Auf einem iPhone hätten die Kinder lieber einen Ziffernblock. Die Gruppe bewertete die Frage mit einem hellgrünen Smiley.

4. *Ich würde den „1x1 Trainer“ auch zu Hause spielen.*
   Alle Schulkinder dieser Gruppe würden den *1x1 Trainer* zu Hause spielen, aber nur wenn sie Zeit hätten und ihnen nicht langweilig ist. Die Bewertung dieser Frage war ein dunkelgrüner Smiley.

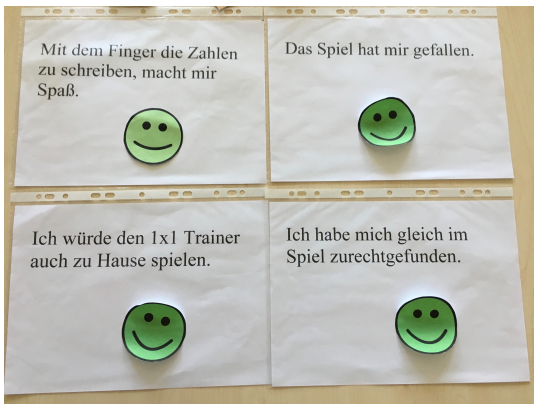


Abbildung 5.6.: Ergebnisse der Gruppe 5

5.2.6. Gruppe 6

Die letzte Gruppe in dieser Evaluation bestand wiederum aus drei Kindern. Die Antworten zu den gestellten Fragen werden in Abbildung 5.7 dargestellt.

1. *Das Spiel hat mir gefallen.*
   Das Spiel hat der ganzen Gruppe sehr gut gefallen. Sie gaben dieser Frage einen dunkelgrünen Smiley.
2. *Ich habe mich gleich im Spiel zurechtgefunden.*
   Die Kinder kannten sich im Spiel sofort aus. Nur bei der Erkennung bekamen sie falsche Ergebnisse. Deshalb wurde die Frage mit einem hellgrünen Smiley beantwortet.

3. *Mit dem Finger die Zahlen zu schreiben, macht mir Spaß.*
   Alle Kinder in dieser Gruppe hatten Spaß, mit dem Finger zu schreiben. Sie finden das Schreiben besser als die Zahlen einzutippen. Die Frage wurde mit einem dunkelgrünen Smiley bewertet.
4. *Ich würde den „1x1 Trainer" auch zu Hause spielen.*
   Mit dem *1x1 Trainer* würden die Schulkinder auch zu Hause lernen. Aus diesem Grund wurde der Frage wieder der beste Smiley gegeben.

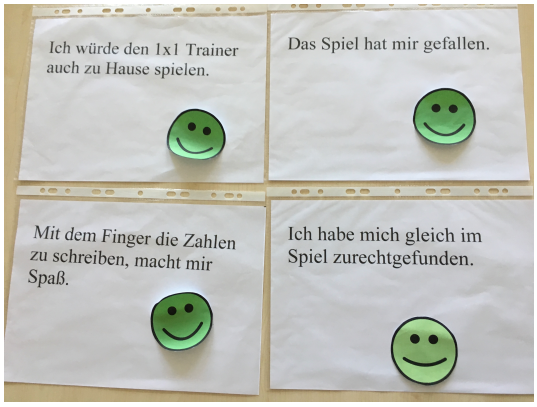


Abbildung 5.7.: Ergebnisse der Gruppe 6

5.3. Auswertung

Zurückblickend verlief die Evaluation ohne großen Zwischenfall. Der *1x1 Trainer* und die benötigte Verbindung mit dem Server funktionierten einwandfrei. Die Schulkinder hatten sehr viel Freude, die Zahlen mit ihren Fingern zu zeichnen. Aus der Durchführung und den Ergebnissen konnten trotzdem wichtige Erkenntnisse gewonnen werden.

Die Lernapplikation bekam grundsätzlich ein positives Feedback, dennoch zeigte die Evaluation, dass die Applikation einige Probleme mit ihrer Usability hat. Folgende Schwierigkeiten traten während der Evaluation auf:

- **Erkennung der Handschrift**
  Das Spiel erkannte ab und zu eine falsche Zahl, was bei den Kindern ein leichtes Frustrationsverhalten auslöste. Die Zahlen eins und sieben wurden vom neuronalen Netz oft falsch erkannt.

- **Anmeldung im Trainer-Modus**
  Die Anmeldung mit einem Benutzerkonto war für einige Kinder ein Problem. Sie versuchten ihren echten Namen in das Benutzernamenfeld einzugeben, obwohl sie die Benutzerdaten kurz vor der Evaluation bekommen haben.

- **Spielstart**
  Viele Kinder waren beim Spielstart überfordert und wussten nicht, was das Spiel von ihnen verlangt und was sie nun machen sollten.

- **Einstelliges Ergebnis**
  Ein weiteres Problem, das die Kinder kurzzeitig verwirrte, war die Frage, in welcher der zwei Boxen ein einstelliges Ergebnis geschrieben werden soll. Der *1x1 Trainer* akzeptiert jedenfalls beide Eingabemöglichkeiten.

6. Zusammenfassung und Ausblick

Das letzte Kapitel teilt sich in zwei Abschnitte auf. Im ersten Kapitel werden die verfasste Arbeit und die gewonnenen Erkenntnisse zusammengefasst. Der zweite Abschnitt gibt einen Ausblick, wie die implementierte mobile Applikation dieser Arbeit weiterentwickelt werden kann.

6.1. Zusammenfassung

Die vorliegende Arbeit hat gezeigt, welche Arten der Handschrifterkennung existieren und mit welchen Methoden die Vorgangsweise einer Erkennung aufgebaut sind. Die Mathematik-Lernapplikation *1x1 Trainer* wurde aufbauend auf dieses Kapitel und mit den in Kapitel 4.1 vorgestellten Technologien entwickelt.

Das Ergebnis der Evaluation verdeutlichte das Potenzial von Applikationen mit einer Handschrifterkennung und vor allem von Lernapplikationen, die über eine solche Technik verfügen. Die Kinder hatten eine große Freude die Zahlen auf den Bildschirm zu schreiben. Sie hatten allerdings eine geteilte Meinung, ob sie die Zahlen lieber mit dem Finger oder mit einem Ziffernblock eintippen. Diese Aussage lässt sich aber höchstwahrscheinlich auf die Erkennungsfehler der Applikation zurückführen. Die falsche Erkennung einer Zahl wirkte sich sichtbar negativ auf die Motivation der Kinder aus.

Weiters offenbarte sich ein positiver Nebeneffekt durch die Evaluation. Die Applikation animierte Kinder die Zahlen schöner mit ihrem Finger zu zeichnen. Dieses Verhalten wurde durch die Befragung der Kinder festgestellt.

Zusammenfassend haben die Recherchen für die Arbeit ergeben, dass eine Handschrifterkennung durchaus in Lernapplikationen für Kinder ihren berechtigten Einsatz hat. Der Erfolg dieser Applikation hängt allerdings stark von der Erkennungsrate der eingesetzten Methodik ab.

Im Zuge des Technologiefortschrittes werden in naher Zukunft womöglich neue Ansätze für die Handschrifterkennung veröffentlicht, die bessere Resultate liefern können als wir Menschen. Ab diesem Zeitpunkt wird der Einsatz einer Handschrifterkennung in Lernapplikationen unumgänglich sein.

6.2. Ausblick

Der entwickelte *1x1 Trainer* kann aufgrund der aufgezeigten Schwierigkeiten in der Evaluation mit folgenden Ansätzen erweitert beziehungsweise verbessert werden:

- **Vortest**:
  Der *1x1 Trainer* nutzt im Trainer-Modus keinen Vortest, um das Können des Anwenders oder der Anwenderin zu bestimmen. Dadurch beginnen alle mit der geringsten Lernfähigkeit.

- **Handschrifterkennung**
  Die Handschrifterkennung funktioniert im Lernspiel gut, hat aber sicher noch ein Verbesserungspotenzial. Das verwendete Neuronale Netz kann beispielsweise besser antrainiert oder erweitert werden.

Eine weitere Möglichkeit die Erkennung zu verbessern, ist der Einsatz von zusätzlichen Vorverarbeitungstechniken.

- **Spielerklärungen**
  Der *1x1 Trainer* erklärt seine Spielmodi nur im Hauptbildschirm. Der genaue Spielablauf wird dagegen an keiner Stelle erklärt. Die Spielanleitungen könnten daher im Onboard-Screen eingefügt werden oder, wie in den vorgestellten Applikationen in Kapitel 3, mit einem Sprachassistenten vorgetragen werden.

- **Gamification**
  Um die Spielmotivation hoch zu halten, würde der Einsatz von Gamification-Elementen helfen. Damit würden der Anwender oder die Anwenderin wahrscheinlich über eine falsch erkannte Zahl leichter hinwegsehen und die Motivation würde daraufhin nicht so stark sinken.

Literatur

Abadi, Martin et al. (2016). »Tensorflow: Large-scale machine learning on heterogeneous distributed systems«. In: *arXiv preprint arXiv:1603.04467*.

Albus, John Edward et al. (2012). *Syntactic pattern recognition, applications*. Bd. 14. Springer Science & Business Media.

Arica, Nafiz und Fatos T Yarman-Vural (2001). »An overview of character recognition focused on off-line handwriting«. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 31.2, S. 216–233.

Bishop, Christopher M (2006). *Pattern recognition and machine learning*. springer.

Box, Don et al. (2000). *Simple object access protocol (SOAP) 1.1*.

Buades, Antoni, Bartomeu Coll und Jean-Michel Morel (2005). »A review of image denoising algorithms, with a new one«. In: *Multiscale Modeling & Simulation* 4.2, S. 490–530.

Caudill, Maureen (1987). »Neural Networks Primer, Part I«. In: *AI Expert* 2.12, S. 46–52. ISSN: 0888-3785. URL: http://dl.acm.org/citation.cfm?id=38292.38295.

Ciregan, Dan, Ueli Meier und Jürgen Schmidhuber (2012). »Multi-column deep neural networks for image classification«. In: *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. IEEE, S. 3642–3649.

Ciresan, Dan C et al. (2011). »Flexible, high performance convolutional neural networks for image classification«. In: *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*. Bd. 22. 1. Barcelona, Spain, S. 1237.

Curbera, Francisco et al. (2002). »Unraveling the Web services web: an introduction to SOAP, WSDL, and UDDI«. In: *IEEE Internet computing* 6.2, S. 86–93.

Eden, Murray (1962). »Handwriting and pattern recognition«. In: *IRE Transactions on Information Theory* 8.2, S. 160–166.

Elias, T. (2011). *Learning Analytics: Definitions, Processes and Potential.* URL: https://pdfs.semanticscholar.org/732e/452659685fe3950b0e515a28ce89d9c5592a.pdf (besucht am 07. 09. 2017).

Fischer, Andreas et al. (2013). »A fast matching algorithm for graph-based handwriting recognition«. In: *International Workshop on Graph-Based Representations in Pattern Recognition*. Springer, S. 194–203.

Genchi, Hiroshi et al. (1968). »Recognition of handwritten numerical characters for automatic letter sorting«. In: *Proceedings of the IEEE* 56.8, S. 1292–1301.

Hecht-Nielsen, Robert (1988). »Neurocomputing: picking the human brain«. In: *IEEE spectrum* 25.3, S. 36–41.

Impedovo, Donato und Giuseppe Pirlo (2008). »Automatic signature verification: The state of the art«. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 38.5, S. 609–635.

Jain, Anil K, Robert P. W. Duin und Jianchang Mao (2000). »Statistical pattern recognition: A review«. In: *IEEE Transactions on pattern analysis and machine intelligence* 22.1, S. 4–37.

Johnson, L. et al. (2011). »The 2011 Horizon Report«. In: URL: http://redarchive.nmc.org/publications/horizon-report-2011-higher-ed-edition (besucht am 07. 09. 2017).

Keysers, Daniel et al. (2017). »Multi-language online handwriting recognition«. In: *IEEE transactions on pattern analysis and machine intelligence* 39.6, S. 1180–1194.

Khalil, Mohammad und Martin Ebner (2015). »Learning analytics: principles and constraints«. In: *Proceedings of world conference on educational multimedia, hypermedia and telecommunications*, S. 1326–1336.

Kruse, Rudolf et al. (2015). *Computational Intelligence Eine methodische Einführung in Künstliche Neuronale Netze, Evolutionäre Algorithmen, Fuzzy-Systeme und*

Bayes-Netze. 2. Aufl. Computational Intelligence. Wiesbaden: Springer Vieweg. ISBN: 978-3-658-10903-5. DOI: 10.1007/978-3-658-10904-2.

LeCun, Yann et al. (1998). »Gradient-based learning applied to document recognition«. In: *Proceedings of the IEEE* 86.11, S. 2278–2324.

Long, Phil und George Siemens (2011). »Penetrating the fog: Analytics in learning and education.« In: *EDUCAUSE review* 46.5, S. 30.

Mohamad, Muhammad'Arif et al. (2015). »A review on feature extraction and feature selection for handwritten character recognition«. In: *International Journal of Advanced Computer Science and Applications* 6.2, S. 204–212.

Otsu, Nobuyuki (1979). »A threshold selection method from gray-level histograms«. In: *IEEE transactions on systems, man, and cybernetics* 9.1, S. 62–66.

Peemen, Maurice, Bart Mesman und Henk Corporaal (2011). »Speed sign detection and recognition by convolutional neural networks«. In: *Proceedings of the 8th International Automotive Congress*, S. 162–170.

Plamondon, Réjean und Sargur N Srihari (2000). »Online and off-line handwriting recognition: a comprehensive survey«. In: *IEEE Transactions on pattern analysis and machine intelligence* 22.1, S. 63–84.

Priya, Anisha et al. (2016). »Online and offline character recognition: A survey«. In: *Communication and Signal Processing (ICCSP), 2016 International Conference on*. IEEE, S. 0967–0970.

Roy, A, S Dutta und M Das (2017). »Handwritten signature recognition and verification using artificial neural network«. In: *Computer, Communication and Electrical Technology: Proceedings of the International Conference on Advancement of Computer Communication and Electrical Technology (ACCET 2016), West Bengal, India, 21-22 October 2016*. CRC Press, S. 39.

Sahoo, Prasanna K, SAKC Soltani und Andrew KC Wong (1988). »A survey of thresholding techniques«. In: *Computer vision, graphics, and image processing* 41.2, S. 233–260.

Schön, Martin, Martin Ebner und Georg Kothmeier (2012). »It's Just About Learning the Multiplication Table«. In: *Proceedings of the 2Nd International Conference on Learning Analytics and Knowledge*. LAK '12. Vancouver,

British Columbia, Canada: ACM, S. 73–81. ISBN: 978-1-4503-1111-3. DOI: 10.1145/2330601.2330624. URL: http://doi.acm.org/10.1145/2330601.2330624.

Senior, Andrew W und Anthony J Robinson (1998). »An off-line cursive handwriting recognition system«. In: *IEEE transactions on pattern analysis and machine intelligence* 20.3, S. 309–321.

Siemens, George (2010). URL: http://www.elearnspace.org/blog/2010/08/25/what-are-learning-analytics/ (besucht am 07. 09. 2017).

Simard, Patrice Y, David Steinkraus, John C Platt et al. (2003). »Best practices for convolutional neural networks applied to visual document analysis.« In: *ICDAR*. Bd. 3, S. 958–962.

Simon, J-C (1992). »Off-line cursive word recognition«. In: *Proceedings of the IEEE* 80.7, S. 1150–1161.

Strecker, Stefan (1997). »Künstliche Neuronale Netze–Aufbau und Funktionsweise«. In: *Arbeitspapiere, Justus-Liebig-Universität, Gießen*.

Surinta, Olarik und Rapeeporn Chamchong (2008). »Image segmentation of historical handwriting from palm leaf manuscripts«. In: *Intelligent Information Processing IV*, S. 182–189.

Tapia, Ernesto und Raúl Rojas (2007). »A survey on recognition of on-line handwritten mathematical notation«. In:

Tappert, Charles C., Ching Y. Suen und Toru Wakahara (1990). »The state of the art in online handwriting recognition«. In: *IEEE Transactions on pattern analysis and machine intelligence* 12.8, S. 787–808.

Trier, Øivind Due, Anil K Jain und Torfinn Taxt (1996). »Feature extraction methods for character recognition-a survey«. In: *Pattern recognition* 29.4, S. 641–662.

Tubbs, Jack D (1989). »A note on binary template matching«. In: *Pattern Recognition* 22.4, S. 359–365.

Vinciarelli, Alessandro und Juergen Luettin (2001). »A new normalization technique for cursive handwritten words«. In: *Pattern recognition letters* 22.9, S. 1043–1050.

Zeiler, Matthew D und Rob Fergus (2014). »Visualizing and understanding convolutional networks«. In: *European conference on computer vision*. Springer, S. 818–833.

Appendix

Anhang A.

Quellcode

A.1. Handschrifterkennung

A.1.1. Convolutional Neural Network

```
import Foundation
import UIKit


class NeuralNetwork {

    init() {
        network = NeuralNetwork.setupNetwork()
    }

    static let weights: [[Float32]] = NeuralNetwork.readWeights()

    let network: BnnsNetwork

    private class func readWeights() -> [[Float32]] {

        func read(asset: String) -> [Float32]? {
            guard let data = NSDataAsset(name: asset)?.data else { return nil }
            return read(floats: data)
        }

        func read(floats: Data) -> [Float32]? {
            var res: [Float32] = Array(repeating: 0, count: floats.count / 4)
            guard floats.copyBytes(to:
                UnsafeMutableBufferPointer(start: &res, count: res.count)) == floats.count
```

```
26              else { return nil }
27          return res
28      }
29
30      let h1_h2_weights = read(asset: "model-h1w-5x5x1x32")!
31      let h1_h2_bias = read(asset: "model-h1b-32")!
32      let h2_h3_weights = read(asset: "model-h2w-5x5x32x64")!
33      let h2_h3_bias = read(asset: "model-h2b-64")!
34      let h3_h4_weights = read(asset: "model-h3w-3136x1024")!
35      let h3_h4_bias = read(asset: "model-h3b-1024")!
36      let h4_y_weights = read(asset: "model-h4w-1024x10")!
37      let h4_y_bias = read(asset: "model-h4b-10")!
38
39      return [h1_h2_weights, h1_h2_bias,
40              h2_h3_weights, h2_h3_bias,
41              h3_h4_weights, h3_h4_bias,
42              h4_y_weights, h4_y_bias]
43  }
44
45  private class func setupNetwork() -> BnnsNetwork {
46      return BnnsBuilder()
47          .shape(width: 28, height: 28, channels: 1)
48          .kernel(width: 5, height: 5)
49          .convolve(weights: weights[0], bias: weights[1])
50          .shape(width: 28, height: 28, channels: 32)
51          .maxpool(width: 2, height: 2)
52          .shape(width: 14, height: 14, channels: 32)
53          .convolve(weights: weights[2], bias: weights[3])
54          .shape(width: 14, height: 14, channels: 64)
55          .maxpool(width: 2, height: 2)
56          .shape(width: 7, height: 7, channels: 64)
57          .connect(weights: weights[4], bias: weights[5])
58          .shape(size: 1024)
59          .connect(weights: weights[6], bias: weights[7])
60          .shape(size: 10)
61          .build()!
62  }
63
64  func predict(image: Data) -> Int {
65      return predict(input: read(image: image))
66  }
67
68  func predict(input: [Float32]) -> Int {
69
70      let outputs = network.apply(input: input)
71
72      return outputs.index(of: outputs.max()!)!
73  }
74
75  func predictBatch(images: Data, count: Int) -> [Int] {
76
77      let outputs = network
78          .batch(input: read(image: images), count: count)
79          .map { $0.index(of: $0.max()!)! }
80
```

```
        return outputs
    }

    private func read(image: Data) -> [Float32] {
        return image.map { Float32($0) / 255.0 }
    }
}
```

Auflistung A.1.: Quellcode - NeuralNetwork-Klasse

```
import Accelerate

struct BnnsShape {
    let width: Int
    let height: Int
    let channels: Int

    var size: Int {
        get {
            return width * height * channels
        }
    }
}

class BnnsFilter {
    let filter: BNNSFilter
    let shape: BnnsShape

    init(filter: BNNSFilter, shape: BnnsShape) {
        self.filter = filter
        self.shape = shape
    }

    deinit {
        BNNSFilterDestroy(filter)
    }
}

struct BnnsNetwork {
    let network: [BnnsFilter]

    func apply(input: [Float32]) -> [Float32] {
        var outputs = input

        for layer in network {
            let inputs = outputs
            outputs = Array(repeating: 0, count: layer.shape.size)

            guard BNNSFilterApply(layer.filter, inputs, &outputs) == 0
                else { return [] }
        }

        return outputs
    }
```

```
45
46      func batch(input: [Float32], count: Int) -> [[Float32]] {
47          var outputs = input
48          var outputStride = input.count / count
49
50          for layer in network {
51              let inputs = outputs
52              let inputStride = outputStride
53
54              outputs = Array(repeating: 0, count: layer.shape.size * count)
55              outputStride = layer.shape.size
56
57              guard BNNSFilterApplyBatch(layer.filter, count, inputs,
58                                         inputStride, &outputs, outputStride) == 0
59                  else { return [] }
60          }
61
62          var result: [[Float32]] = []
63          outputStride = outputs.count / count
64
65          for row in 0..<count {
66              let res = Array(outputs[outputStride * row ..< outputStride * (row + 1)])
67              result.append(res)
68          }
69
70          return result
71      }
72  }
73
74  class BnnsBuilder {
75
76      var dataType: BNNSDataType {
77          get {
78              return BNNSDataTypeFloat32
79          }
80      }
81
82      private var descriptors: [LayerDescriptor] = []
83
84      private var inputShape: BnnsShape!
85      private var kernel: (width: Int, height: Int)!
86      private var stride = (x: 1, y: 1)
87      private var activation = BNNSActivationFunctionRectifiedLinear
88
89      func shape(width: Int, height: Int, channels: Int) -> Self {
90          let shape = BnnsShape(width: width, height: height, channels: channels)
91          inputShape = shape
92
93          if let lastFilter = descriptors.last {
94              lastFilter.output = shape
95          }
96
97          return self
98      }
99
```

```
100     func shape(size: Int) -> Self {
101         return shape(width: size, height: 1, channels: 1)
102     }
103
104     func kernel(width: Int, height: Int) -> Self {
105         kernel = (width: width, height: height)
106         return self
107     }
108
109     func stride(x: Int, y: Int) -> Self {
110         stride = (x: x, y: y)
111         return self
112     }
113
114     func activation(function: BNNSActivationFunction) -> Self {
115         activation = function
116         return self
117     }
118
119     func convolve(weights: [Float32], bias: [Float32]) -> Self {
120         let desc = ConvolutionLayerDescriptor()
121         desc.dataType = dataType
122         desc.input = inputShape
123         desc.kernel = kernel
124         desc.stride = stride
125         desc.weights = weights
126         desc.bias = bias
127         desc.activation = activation
128
129         descriptors.append(desc)
130         return self
131     }
132
133     func maxpool(width: Int, height: Int) -> Self {
134         let desc = MaxPoolingLayerDescriptor()
135         desc.dataType = dataType
136         desc.input = inputShape
137         desc.kernel = (width: width, height: height)
138
139         descriptors.append(desc)
140         return self
141     }
142
143     func connect(weights: [Float32], bias: [Float32]) -> Self {
144         let desc = FullyConnectedLayerDescriptor()
145         desc.dataType = dataType
146         desc.input = inputShape
147         desc.weights = weights
148         desc.bias = bias
149         desc.activation = activation
150
151         descriptors.append(desc)
152         return self
153     }
154
```

```
155     func build() -> BnnsNetwork? {
156         let building = descriptors.map { $0.build() }
157         let network = building.flatMap{$0}
158 
159         guard network.count == building.count else { return nil }
160 
161         return BnnsNetwork(network: network)
162     }
163 
164 
165     private class LayerDescriptor {
166         var dataType: BNNSDataType!
167         var input: BnnsShape!
168         var output: BnnsShape!
169 
170         func build() -> BnnsFilter? {
171             return nil
172         }
173     }
174 
175     private class ConvolutionLayerDescriptor : LayerDescriptor {
176         var kernel: (width: Int, height: Int)!
177         var stride: (x: Int, y: Int)!
178         var weights: [Float32]!
179         var bias: [Float32]!
180         var activation: BNNSActivationFunction!
181 
182         override func build() -> BnnsFilter? {
183 
184             let x_padding: Int = (stride.x * (output.width - 1) +
185                 kernel.width - input.width) / 2
186             let y_padding: Int = (stride.y * (output.height - 1) +
187                 kernel.height - input.height) / 2
188             let pad = (x: x_padding, y: y_padding)
189 
190             var imageStackIn = BNNSImageStackDescriptor(width: input.width,
191                                 height: input.height, channels: input.channels,
192                                 row_stride: input.width,
193                                 image_stride: input.width * input.height,
194                                 data_type: dataType, data_scale: 0, data_bias: 0)
195 
196             var imageStackOut = BNNSImageStackDescriptor(width: output.width,
197                                  height: output.height, channels: output.channels,
198                                  row_stride: output.width,
199                                  image_stride: output.width * output.height,
200                                  data_type: dataType, data_scale: 0, data_bias: 0)
201 
202             let weights_data = BNNSLayerData(data: weights, data_type: dataType,
203                                              data_scale: 0, data_bias: 0,
204                                              data_table: nil)
205             let bias_data = BNNSLayerData(data: bias, data_type: dataType,
206                                           data_scale: 0, data_bias: 0, data_table: nil)
207             let activ = BNNSActivation(function: activation, alpha: 0, beta: 0)
208 
209             var layerParams = BNNSConvolutionLayerParameters(x_stride: stride.x,
```

```
                            y_stride: stride.y, x_padding: pad.x,
                            y_padding: pad.y, k_width: kernel.width,
                            k_height: kernel.height,in_channels: input.channels,
                            out_channels: output.channels,
                            weights: weights_data, bias: bias_data,
                            activation: activ)

        var filterParams = defaultFilterParameters()

        guard let convolve = BNNSFilterCreateConvolutionLayer(&imageStackIn,
                              &imageStackOut, &layerParams, &filterParams)
            else { return nil }

        return BnnsFilter(filter: convolve, shape: output)
    }
}

private class MaxPoolingLayerDescriptor : LayerDescriptor {
    var kernel: (width: Int, height: Int)!

    override func build() -> BnnsFilter? {

        let stride = (x: kernel.width, y: kernel.height)

        let x_padding: Int = (stride.x * (output.width - 1) +
            kernel.width - input.width) / 2
        let y_padding: Int = (stride.y * (output.height - 1) +
            kernel.height - input.height) / 2
        let pad = (x: x_padding, y: y_padding)

        var imageStackIn = BNNSImageStackDescriptor(width: input.width,
                           height: input.height, channels: input.channels,
                           row_stride: input.width,
                           image_stride: input.width * input.height,
                           data_type: dataType, data_scale: 0, data_bias: 0)

        var imageStackOut = BNNSImageStackDescriptor(width: output.width,
                            height: output.height, channels: output.channels,
                            row_stride: output.width,
                            image_stride: output.width * output.height,
                            data_type: dataType, data_scale: 0, data_bias: 0)

        let bias_data = BNNSLayerData()
        let activ = BNNSActivation(function: BNNSActivationFunctionIdentity,
                                   alpha: 0, beta: 0)

        var layerParams = BNNSPoolingLayerParameters(x_stride: stride.x,
                        y_stride: stride.y, x_padding: pad.x, y_padding: pad.y,
                        k_width: kernel.width, k_height: kernel.height,
                        in_channels: input.channels, out_channels: output.channels,
                        pooling_function: BNNSPoolingFunctionMax,
                        bias: bias_data, activation: activ)

        var filterParams = defaultFilterParameters()

```

```
265             guard let pool = BNNSFilterCreatePoolingLayer(&imageStackIn,
266                                 &imageStackOut, &layerParams, &filterParams)
267                 else { return nil }
268 
269             return BnnsFilter(filter: pool, shape: output)
270         }
271     }
272 
273     private class FullyConnectedLayerDescriptor : LayerDescriptor {
274         var weights: [Float32]!
275         var bias: [Float32]!
276         var activation: BNNSActivationFunction!
277 
278         override func build() -> BnnsFilter? {
279 
280             var hiddenIn = BNNSVectorDescriptor(size: input.size,
281                                 data_type: dataType, data_scale: 0, data_bias: 0)
282             var hiddenOut = BNNSVectorDescriptor(size: output.size,
283                                 data_type: dataType, data_scale: 0, data_bias: 0)
284 
285             let weights_data = BNNSLayerData(data: weights, data_type: dataType,
286                                 data_scale: 0, data_bias: 0, data_table: nil)
287             let bias_data = BNNSLayerData(data: bias, data_type: dataType,
288                                 data_scale: 0, data_bias: 0, data_table: nil)
289             let activ = BNNSActivation(function: activation, alpha: 0, beta: 0)
290 
291             var layerParams = BNNSFullyConnectedLayerParameters(in_size: input.size,
292                                 out_size: output.size, weights: weights_data,
293                                 bias: bias_data, activation: activ)
294 
295             var filterParams = defaultFilterParameters()
296 
297             guard let layer = BNNSFilterCreateFullyConnectedLayer(&hiddenIn,
298                                     &hiddenOut, &layerParams, &filterParams)
299                 else { return nil }
300 
301             return BnnsFilter(filter: layer, shape: output)
302         }
303     }
304 }
```

Auflistung A.2.: Quellcode - BNNS Helferfunktionen

A.1.2. Bildverarbeitung

```
1 import Foundation
2 import UIKit
3 
4 class ImageProcessing {
5 
```

```
6       //--------------------------------------------------------------------------------
7       private class func getCenterOfMass(image: UIImage) -> UIImage? {
8           // calculate region of interest
9           let mass = centerOfMass(image: image)
10          // init target image size
11          let target = CGSize(width: 28, height: 28)
12          // crop image
13          let crop = image.cgImage?.cropping(to: mass)
14          // scale to defined size
15          return UIImage(cgImage: crop!).scaleToSize(newSize: target)
16      }
17
18      //--------------------------------------------------------------------------------
19      private class func centerCropResize(image: UIImage, target: CGSize) -> UIImage? {
20
21          let scale = max(target.width / image.size.width,
22                          target.height / image.size.height)
23          let scaledSize = CGSize(width: image.size.width * scale,
24                                  height: image.size.height * scale)
25
26          var rect = CGRect(origin: CGPoint(x: (target.width - scaledSize.width) / 2,
27                                            y: (target.height - scaledSize.height) / 2),
28                            size: scaledSize)
29
30          UIGraphicsBeginImageContextWithOptions(target, false, 1)
31          defer {
32              UIGraphicsEndImageContext()
33          }
34
35          image.draw(in: rect)
36
37          return UIGraphicsGetImageFromCurrentImageContext()
38      }
39
40      //--------------------------------------------------------------------------------
41      private class func centerOfMass(image: UIImage) -> CGRect {
42          let width = Int(image.size.width)
43          let height = Int(image.size.height)
44          let midpoint = CGRect(origin: CGPoint.zero, size: image.size)
45
46          guard let data = bytes(image: image) else { return midpoint }
47
48          var mass: CGFloat = 0
49          var rx: CGFloat = 0
50          var ry: CGFloat = 0
51          var minPoint = CGPoint(x: Int.max, y: Int.max)
52          var maxPoint = CGPoint(x: Int.min, y: Int.min)
53
54          for row in 0..<height {
55              for col in 0..<width {
56                  let px = 1 - CGFloat(data[row * width + col]) / 255
57                  guard px > 0 else { continue }
58
59                  let x = CGFloat(col)
60                  let y = CGFloat(row)
```

```
61
62                  mass += px
63                  rx += px * x
64                  ry += px * y
65
66                  if x < minPoint.x {
67                      minPoint.x = x
68                  }
69                  if x > maxPoint.x {
70                      maxPoint.x = x
71                  }
72                  if y < minPoint.y {
73                      minPoint.y = y
74                  }
75                  if y > maxPoint.y {
76                      maxPoint.y = y
77                  }
78              }
79          }
80
81          guard mass > 0 else { return midpoint }
82
83          let center = CGPoint(x: rx / mass, y: ry / mass)
84
85          let hx = max(center.x - minPoint.x, maxPoint.x - center.x)
86          let hy = max(center.y - minPoint.y, maxPoint.y - center.y)
87          let hh = max(hx, hy)
88
89          // check rect
90          var tempRoi = CGRect(origin: center,
91                               size: CGSize.zero).insetBy(dx: -hh, dy: -hh)
92
93          let roiX = (tempRoi.minX < 0) ? 0 : tempRoi.minX
94          let roiY = (tempRoi.minY < 0) ? 0 : tempRoi.minY
95          let roiWidth = (tempRoi.maxX < midpoint.width) ? tempRoi.maxX : midpoint.width
96          let roiHeight = (tempRoi.maxY < midpoint.height) ? tempRoi.maxY : midpoint.height
97
98          return CGRect(x: Int(roiX), y: Int(roiY), width: Int(roiWidth),
99                        height: Int(roiHeight))
100     }
101
102     //--------------------------------------------------------------------------
103     private class func bytes(image: UIImage) -> Data? {
104         let colorSpace = CGColorSpaceCreateDeviceGray()
105         let bitmapInfo = CGBitmapInfo()
106
107         guard let context = CGContext(data: nil, width: Int(image.size.width),
108                                       height: Int(image.size.height), bitsPerComponent: 8,
109                                       bytesPerRow: Int(image.size.width), space: colorSpace,
110                                       bitmapInfo: bitmapInfo.rawValue)
111             else { return nil }
112
113         context.draw(image.cgImage!, in: CGRect(origin: CGPoint.zero, size: image.size))
114
115         let data = Data(bytes: context.data!,
```

```
                        count: Int(image.size.width * image.size.height))

        return data
    }

    //-------------------------------------------------------------------------------
    public class func getImageData(image: UIImage) -> [Float32]? {
        guard let img = getCenterOfMass(image: image),
            let data = bytes(image: img)
            else { return nil }

        return data.map { 1 - Float32($0) / 255.0 }
    }
}
```

Auflistung A.3.: Quellcode - Bildverarbeitung Helferfunktionen

A.1.3. Zeichenboxen

```
import UIKit

class DrawingView : UIView {

    // grid properties
    let numberOfGridLines: Int = 2
    let borderWidth: CGFloat = (UI_USER_INTERFACE_IDIOM() == .pad) ? 2.5 : 2.0

    // canvas
    private(set) var canvas: CanvasView!
    private(set) var imgNumber: UIImageView!

    // auto-layout
    var isCanvasCreated: Bool = false

    required init?(coder aDecoder: NSCoder) {
        super.init(coder: aDecoder)
        setupView()
    }

    //-------------------------------------------------------------------------------
    fileprivate func setupView() {
        // enable events
        self.isUserInteractionEnabled = true
        // set background color
        self.backgroundColor = .clear
    }

    //-------------------------------------------------------------------------------
    func initDrawingCanvas() {
        // add canvas view
```

```
32          canvas = CanvasView(frame: CGRect(x: 0, y: 0, width: self.frame.width,
33                                            height: self.frame.height))
34          self.addSubview(canvas)
35      }
36
37      //--------------------------------------------------------------------------------
38      func setBackgroundNumber(number: Int) {
39          // load image
40          if let img = UIImage(named: "\(number)") {
41              if self.subviews.count > 1 {
42                  // remove number imageview
43                  self.subviews[0].removeFromSuperview()
44              }
45              // initialize the value of imageView with a CGRectZero, resize it later
46              let imageView = UIImageView(frame: self.bounds)
47
48              // set the appropriate contentMode and add the image to your imageView property
49              imageView.contentMode = .scaleToFill
50              imageView.image = img
51
52              // set new alpha value to show image not so strong
53              imageView.alpha = 0.25
54
55              // add the imageView to your view hierarchy
56              self.insertSubview(imageView, at: self.subviews.count - 1)
57          }
58      }
59
60      //--------------------------------------------------------------------------------
61      override func draw(_ rect: CGRect) {
62          // set border
63          self.layer.borderColor = UIColor.black.cgColor
64          self.layer.borderWidth = borderWidth
65
66          // draw grid lines
67          // calculate column width
68          let columnWidth: Int = Int(self.frame.size.width) / (self.numberOfGridLines + 1)
69
70          // ---------------------------
71          // Drawing column lines
72          // ---------------------------
73          for i in 1...self.numberOfGridLines {
74              // start point
75              let startPoint = CGPoint(x: columnWidth * i, y: 0)
76              // end point
77              let endPoint = CGPoint(x: startPoint.x, y: self.frame.size.height)
78
79              // draw line
80              drawDashedLine(from: startPoint, to: endPoint)
81          }
82
83          // calclulate row height
84          let rowHeight = Int(self.frame.size.height) / (self.numberOfGridLines + 1)
85
86          // ---------------------------
```

```
87          // Drawing row lines
88          // --------------------------
89
90          for j in 1...self.numberOfGridLines {
91              // start point
92              let startPoint = CGPoint(x: 0, y: rowHeight * j)
93              // end point
94              let endPoint = CGPoint(x: self.frame.size.width, y: startPoint.y)
95
96              // draw line
97              drawDashedLine(from: startPoint, to: endPoint)
98          }
99      }
100
101     //--------------------------------------------------------------------------
102     fileprivate func drawDashedLine(from: CGPoint, to: CGPoint) {
103         // init path
104         let  path = UIBezierPath()
105         //
106         path.move(to: from)
107         path.addLine(to: to)
108         // draw dashed line
109         let  dashes: [ CGFloat ] = [ 4.0, 4.0 ]
110         path.setLineDash(dashes, count: dashes.count, phase: 0.0)
111         path.lineWidth = self.borderWidth / 2.0
112         path.lineCapStyle = .butt
113         UIColor.lightGray.set()
114         path.stroke()
115     }
116 }
```

Auflistung A.4.: Quellcode - DrawingView-Klasse

```
1  import UIKit
2
3  class CanvasView: UIImageView {
4
5      // set in init method
6      var brushWidth: CGFloat {
7          return ((7 / 150) * frame.width).rounded()
8      }
9
10     fileprivate var lastDrawPoint = CGPoint.zero
11     fileprivate var swiped = false
12     var drawingAllowed = false
13
14     // delegate
15     weak var delegate: CanvasViewDelegate?
16
17     //--------------------------------------------------------------------------
18     required init?(coder aDecoder: NSCoder) {
19         super.init(coder: aDecoder)
20     }
21
```

```
22      //------------------------------------------------------------------------------
23      override init(frame: CGRect) {
24          super.init(frame: frame)
25          // set background color
26          self.backgroundColor = .clear
27          // enable events
28          self.isUserInteractionEnabled = true
29      }
30
31      //------------------------------------------------------------------------------
32      override func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?) {
33          guard let touch = touches.first, drawingAllowed else {
34              return
35          }
36
37          self.swiped = false
38
39          self.lastDrawPoint = touch.location(in: self)
40      }
41
42      //------------------------------------------------------------------------------
43      override func touchesMoved(_ touches: Set<UITouch>, with event: UIEvent?) {
44          guard let touch = touches.first, drawingAllowed else {
45              return
46          }
47
48          let currentPoint = touch.location(in: self)
49
50          if self.swiped {
51              self.drawLine(fromPoint: self.lastDrawPoint, toPoint: currentPoint)
52          } else {
53              self.drawLine(fromPoint: currentPoint, toPoint: currentPoint)
54              self.swiped = true
55          }
56
57          self.lastDrawPoint = currentPoint
58      }
59
60      //------------------------------------------------------------------------------
61      override func touchesEnded(_ touches: Set<UITouch>, with event: UIEvent?) {
62          guard let touch = touches.first, drawingAllowed else {
63              return
64          }
65
66          self.drawLine(fromPoint: self.lastDrawPoint, toPoint: self.lastDrawPoint)
67
68          // call delegate method
69          delegate?.canvasViewDidFinishDrawing()
70
71          super.touchesEnded(touches, with: event)
72      }
73
74      //------------------------------------------------------------------------------
75      private func drawLine(fromPoint: CGPoint, toPoint: CGPoint) {
76          autoreleasepool {
```

```
77              UIGraphicsBeginImageContextWithOptions(self.bounds.size, false, 0)
78              defer {
79                  UIGraphicsEndImageContext()
80              }
81
82              // create context
83              guard let context = UIGraphicsGetCurrentContext() else { return }
84
85              // draw in context
86              self.image?.draw(in: self.bounds)
87
88              // Append new line to image
89              context.move(to: fromPoint)
90              context.addLine(to: toPoint)
91              context.setLineCap(CGLineCap.round)
92              context.setLineWidth(brushWidth)
93              context.setStrokeColor(UIColor.black.cgColor)
94              context.strokePath()
95
96              self.image = UIGraphicsGetImageFromCurrentImageContext()
97          }
98      }
99
100     //------------------------------------------------------------------------------
101     func getImage() -> UIImage? {
102         var img: UIImage?
103
104         guard let notEmptyImage = self.image else {
105             return nil
106         }
107
108         autoreleasepool {
109             UIGraphicsBeginImageContext(self.bounds.size)
110             defer {
111                 UIGraphicsEndImageContext()
112             }
113
114             guard let context = UIGraphicsGetCurrentContext() else { return }
115
116             // add white background color
117             context.setFillColor(UIColor.white.cgColor)
118             context.fill(self.bounds)
119
120             // draw image into context
121             self.image?.draw(in: self.bounds)
122
123             img = UIGraphicsGetImageFromCurrentImageContext()
124         }
125
126         return img
127     }
128
129 }
```

Auflistung A.5.: Quellcode - CanvasView-Klasse

A.1.4. Erkennungsscreen

```
import Foundation

protocol CanvasViewDelegate : class {
    //--------------------------------------------------------------------------------
    func canvasViewDidFinishDrawing()
}
```

Auflistung A.6.: Quellcode - CanvasViewDelegate-Protokoll

```
import UIKit

class OCRViewController: MainViewController {

    // NEURAL NETWORK
    var network = NeuralNetwork()

    override func viewDidLoad() {
        super.viewDidLoad()
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }

    //--------------------------------------------------------------------------------
    func classifyDrawingInput(view: DrawingView) throws -> Int? {
        //        print("CLASSIFY INPUT")

        guard let image = view.canvas.getImage(),
            let data = ImageProcessing.getImageData(image: image)
            else {
                return nil
        }

        return network.predict(input: data)
    }
}
```

Auflistung A.7.: Quellcode - OCRViewController-Klasse