

Hochschule Aalen

MASTERS-STUDIENGANG MACHINE LEARNING AND DATA ANALYTICS

Projektarbeit

Sudoku-Löser auf einem iPhone

(Erkennung des Sudoku aus einem Kamerabild, Trennung der Zahlen und Lösen durch einen Lösungsalgorithmus)

betreut von
Prof. Dr. Ulrich Klauck

26. April 2021

Eidesstattliche Erklärung:

Hiermit erkläre ich, Markus Schmidgall, dass ich die vorliegenden Angaben in dieser Projektarbeit wahrheitsgetreu und selbständig verfasst habe.

Weiterhin versichere ich, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben, dass alle Ausführungen, die anderen Schriften wörtlich oder sinngemäß entnommen wurden, kenntlich gemacht sind und dass die Arbeit in gleicher oder ähnlicher Fassung noch nicht Bestandteil einer Studien- oder Prüfungsleistung war

Ort, Datum

Unterschrift(Student)

Bestätigung der inhaltlichen Richtigkeit:

Der vorliegende Bericht wurde durch den zuständigen Betreuer Gregor Grambow Korrektur gelesen und auf inhaltliche Korrektheit geprüft.

Ort, Datum

Unterschrift(Betreuer)

Inhaltsverzeichnis

Eidesstattliche Erklärung	2
Abbildungsverzeichnis	4
Tabellenverzeichnis	5
Listings	6
1 Einleitung	8
1.1 Aufgaben-/Fragestellung	8
2 Grundlagen	9
2.1 Sudoku	9
2.1.1 Allgemein	9
2.1.2 Lösungsvorgang	10
2.2 Convolutional Neural Network	12
2.2.1 Convolutional Layer	13
2.2.2 Pooling Layer	13
2.3 Node.js, P5.js und ML5.js	14
3 Praktische Umsetzung der Aufgabenstellung	16
3.1 Projektstruktur	16
3.1.1 Ordner	16
3.1.2 Dateien	17
3.2 Trainingsprozess	23
3.2.1 Datensatz	23
3.2.2 MNIST Handwritten Digits	23
3.2.3 Selbst erstellter Datensatz	24
3.2.4 Trainungsvorgang	25
3.3 Zahlenerkennung	28
3.4 Der Lösungsalgorithmus	31
4 Applikations-Walkthrough	38
4.1 StartScreen	38
4.2 UploadScreen	39
4.3 UploadImageScreen	40
4.4 VideoScreen	41
4.5 ImageScreen	42
4.6 SudokuScreen	43
4.7 CalculatedScreen	44
5 Diskussion und Ausblick	45

Abbildungsverzeichnis

1	Gefülltes Sudokufeld	9
2	Cross-Hatching	11
3	Counting Methode	11
4	Aufbau eines CNNs	12
5	Convolution im CNN	13
6	Maxpooling2D	13
7	Http-Server installieren	14
8	Node Server starten	14
9	Setup und Draw Funktion	15
10	Import ML5	15
11	Projektstruktur	16
12	Convert Handwritten Digits	23
13	Java created Digits	24
14	Trainingsverlauf	27
15	Modellparameter	27
16	Ausgangssudoku	30
17	81 Unterbilder	30
18	Erkannte Ziffern	30
19	Startscreen	38
20	UploadScreen	39
21	UploadImagesScreen	40
22	VideoScreen	41
23	ImageScreen - Android	42
24	ImageScreen - iPhone	42
25	SudokuScreen	43
26	CalculatedScreen - Upload Image	44
27	CalculatedScreen - Video Image	44

Tabellenverzeichnis

Code Beispiele

1	index.html	17
2	sketch.js - Initialisierung	18
3	sketch.js - Setup	18
4	sketch.js - Uploadscreen	19
5	sketch.js - Videoscreen	19
6	sketch.js - Uploadimagescreen	20
7	sketch.js - Imagescreen	20
8	sketch.js - SudokuScreen	20
9	sketch.js - CalculatedScreen	21
10	sketch.js - Draw Funktion 1	21
11	sketch.js - Draw Funktion 2	22
12	convertImages.py	23
13	createDataset.pde	24
14	training_Sketch.js - Initialisierung	25
15	training_Sketch.js - Preload	25
16	training_Sketch.js - Setup	26
17	sketch.js - Preload	28
18	getDigits.js	29
19	solve_Sudoku.js - square-coordinates	31
20	solve_Sudoku.js - getGrid	31
21	solve_Sudoku.js - solve	32
22	solve_Sudoku.js - get_row/column/square	32
23	solve_Sudoku.js - complete_cell	33
24	solve_Sudoku.js - square-coordinates	34
25	solve_Sudoku.js - square-coordinates	34
26	solve_Sudoku.js - is_solved	35
27	solve_Sudoku.js - square-coordinates	36
28	solve_Sudoku.js - square-coordinates	37

Literatur

- [1] Frankfurter Allgemeine Zeitung, *Der Heilige Gral der Sudokus*, <https://www.faz.net/aktuell/wissen/physik-mehr/mathematik-der-heilige-gral-der-sudokus-11682905.html> (2012).
- [2] Markus Schmidgall, *Visualisierung neuronaler Netze* (2019).

1 Einleitung

1.1 Aufgaben-/Fragestellung

Im Laufe der Projektarbeit soll eine auf dem iPhone ausführbare Applikation entwickelt werden, die es dem Benutzer erlaubt, mit dem mobilen Gerät ein Foto von einem noch zu lösenden Sudokufeld aufzunehmen, und dieses von der Applikation lösen zu lassen.

Zunächst erfolgt die Beschäftigung mit dem Thema des Lösen eines Sudokus, um Grundlagen für die weitere Bearbeitung der Aufgabenstellung zu erlangen.

Weiter ist das Ziel, sich mit der Bilderkennung und der Erstellung eines zu der Aufgabenstellung passendes Modells auseinander zu setzen.

Auch die Erarbeitung eines effizienten Lösungsalgorithmus für die Sudokufelder ist ein essentieller Bestandteil der Projektarbeit.

Anschließend erfolgt die Auswahl eines zur Aufgabenstellung passenden Frameworks um die Umsetzung des erwarteten Ergebnisses zu erreichen. Hierbei ist die Wahl des Frameworks nicht vorgegeben.

2 Grundlagen

Im Abschnitt der Grundlagen sollen dem Leser Themen näher erklärt werden, um die Zusammenhänge mit der Fragestellung der Projektarbeit besser zu verstehen. Dies beinhaltet allgemeine Grundlagen, wie zum Beispiel was Sudoku eigentlich ist. Außerdem werden technische Grundlagen angesprochen. Dabei geht es vor allem um verwendete Frameworks, oder wie Machine Learning im Bereich der Bilderkennung funktioniert.

2.1 Sudoku

2.1.1 Allgemein

Sudoku ist ein aus Japan kommendes Zahlenrätsel. Vorgegeben ist dabei in der klassischen Variante immer ein 9x9 Felder großes Quadrat. Dieses ist in 9 kleinere 3x3 große Blöcke aufgeteilt. Die noch leeren Felder sollen wie in Abbildung 1 so mit Zahlen von 1 bis 9 gefüllt werden, dass jede Zeile(Orange), jede Spalte(Grün) und jeder 3x3 Block(Rot) jede Zahl genau einmal enthält.

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

Abbildung 1: Gefülltes Sudokufeld

Quelle: Selbst erstellt

Jedes Sudoku das eine eindeutige Lösung hat, muss mindestens 17 schon vorher eingetragene Ziffern enthalten. Es gibt rund 6,7 Trilliarden Möglichkeiten, die Ziffern 1 bis 9 regelkonform in einem Sudoku zu plazieren. „Allerdings gibt es unter diesen Lösungen Ähnlichkeiten. Sudokus weisen nämlich etliche Symmetrien auf. Man kann ein Sudoku drehen oder spiegeln, Zeilen oder Spalten einer Blockreihe vertauschen oder Zahlen gegeneinander auswechseln und erhält dennoch jedes Mal wieder gültige Sudokus. Zählt man alle symmetrischen Lösungen nur einmal, bleiben von den 6,7 Trilliarden Möglichkeiten nur noch 5,5 Milliarden übrig.“[1]

Vom klassischen Sudoku gibt es mittlerweile einige abgewandelte Versionen, die durchaus interessant und abwechslungsreich sind, allerdings sind diese im Lösungsprozess noch deutlich aufwändiger, weshalb diese Applikation sich ausschließlich auf das Standard 9x9 Sudoku konzentriert.

2.1.2 Lösungsvorgang

Es gibt fast so viele verschiedene Möglichkeiten ein Sudoku durch logische Folgerungen zu lösen wie verschieden Sudokus. Hier sollen exemplarisch einige davon vorgestellt werden, die gerade zum lösen einfacherer Sudokus beitragen können.

Der einfachste Algorithmus der ein Computer ausführen kann um die Lösung eines Sudokus zu finden ist ein Brute-Force Algorithmus mit Backtracking. Dies ist allerdings nur eine sinnvolle Möglichkeit der Lösung für den Computer, für einen Menschen ist diese Vorgehensweise sehr zeitintensiv und umständlich. Bei Backtracking Algorithmus wird jedes freie Feld von vorne her mit der ersten passenden Ziffer gefüllt. Dies passiert so lange, bis im aktuellen Feld keine Ziffer mehr gefunden werden kann, die passt. In diesem Fall geht der Algorithmus ein Feld zurück, und probiert dort die nächsthöhere passende Ziffer aus. Dieses Vorgehen passiert so lange, bis das Sudoku gelöst ist. Dieser Algorithmus funktioniert bei jedem Sudoku das eine eindeutige Lösung hat.

Gerade um Rechenleistung und Zeit zu sparen ist es von Vorteil, den Lösungsalgorithmus für Sudoku deutlich intelligenter zu programmieren. Einer der ersten Methoden um ein Sudoku intelligent zu lösen ist das **Cross-Hatching**. Hierbei wird eine Ziffer im Feld platziert, indem man basierend auf Ziffern in der selben Zeile, Spalte und dem selben Quadrat, die Möglichkeiten eliminiert. Im vorliegenden Beispiel soll in der oberen mittleren Box die Zahl 8 gefunden werden. Die Quadrate links und rechts davon enthalten aber schon jeweils eine 8 in Reihe 1 bzw Reihe 2. Somit bleibt für die mittlere Box nur noch Reihe 3 übrig um die 8 zu platzieren. Da hier nur noch ein freies Feld ist, kann die 8 mit Sicherheit platziert werden.

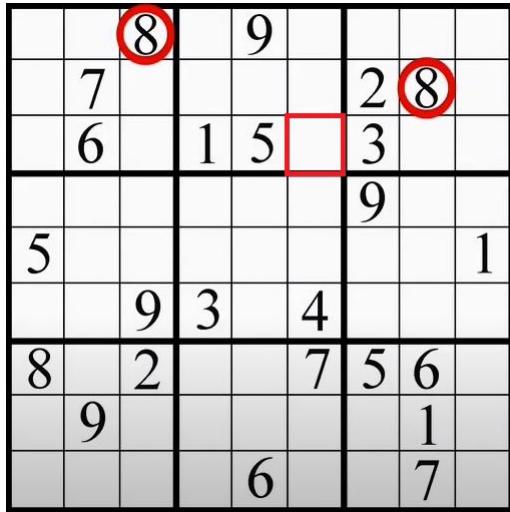


Abbildung 2: Cross-Hatching
Quelle: Selbst erstellt

Zum Lösen des gesamten Sudokus kann nun jedes 3x3 Feld der Reihe nach nach dieser Methode kontrolliert werden. Oft ist es sinnvoll diesen Vorgang zu wiederholen, da Ziffern in späteren Feldern Ziffern in frühen Feldern ausschließen können. Danach kann man die **Counting** Methode anwenden. Hierbei sucht man das Sudoku nach leeren Feldern ab, in die durch Betrachtung der Zeile, Spalte und 3x3 Quadrat nur noch eine einzige Möglichkeit übrig bleibt.

	1	2	3	4	5	6	7	8	9
1	279	467	2679	679	35679	579	2347	8	1
2	178	14678	3	167	2	178	47	5	9
3	12789	5	12678	1679	13678	9	4	237	26
4	15789	178	15789	3	1789	6	289	12	4
5	13789	2	16789	1479	14789	1789	389	16	5
6	1389	1368	4	5	189	2	389	7	36
7	2358	38	258	246	456	5	1	9	7
8	6	17	127	1279	179	3	5	4	8
9	4	9	157	8	157	157	6	3	2

Abbildung 3: Counting Methode
Quelle: <https://sudoku.ironmonger.com/howto/counting/docs.tpl?setreferrerself=true>

Diese Vorgehensweise durch Cross-Hatching und Counting führt bei einfacheren Sudokus meist schon ans Ziel. Beide Lösungsalgorithmen sind auch in der Applikation implementiert. Bei schwereren Sudokus sind oft deutlich kompliziertere Methoden notwendig um eine Lösung zu finden. Einen Überblick über diese Methoden ist unter folgendem Link zu finden -> <https://sudoku.ironmonger.com/howto/basicRules/docs.tpl?setreferrerself=true>

2.2 Convolutional Neural Network

Da es bei der vorliegenden Aufgabenstellung um die Klassifikation von Bildmaterial handelt, ist zur Umsetzung dieser am besten ein CNN oder auch Convolutional Neural Network geeignet.

Der folgende Abschnitt ist aus meiner eigenen Projektarbeit kopiert[2]

„Ein Convolutional Neural Network (CNN), auf deutsch auch faltendes neuronales Netzwerk ist ein Konzept des maschinellen Lernens, genauer des Bereichs Deep Learning. CNNs werden vor allem zur Bildererkennung und Klassifizierung verwendet. Ansatz eines Convolutional Neural Networks ist es, die Vorgänge die auch im Gehirn des Menschen beim Erkennen von Bildern nachzubilden. Anders als bei anderen neuronalen Netzen sind die Neuronen im Convolutional Neural Network nicht komplett miteinander verbunden. Jedes Neuron ist nur mit wenigen Neuronen aus dem vorhergehenden und nachfolgenden Layer verbunden. Dies unterscheidet sich nur in der letzten Ausgabeschicht, wo alle Neuronen miteinander verbunden sind.“

Convolutional Neural Networks bestehen aus zwei Blöcken verschiedenen Arten von Schichten. Der erste Block ist der Convolution Block, welcher meist aus mehreren Convolutional Sichten und einer Pooling Schicht besteht. Hiervon kann es im Netz mehrere Blöcke hintereinander geben. Diese Blöcke sind dafür zuständig, bestimmte Eigenschaften im Bild zu entdecken. Danach kommt ein Klassifizierungsblock, welcher aus Ausgabeschichten besteht. Diese ist zuständig für die Klassifizierung der Eingabedaten.

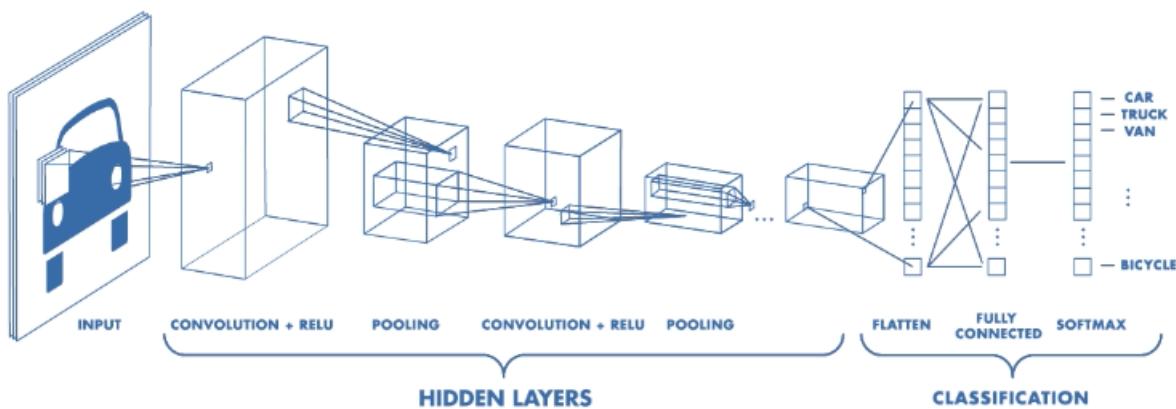


Abbildung 4: Aufbau eines CNNs

Quelle: <https://link.springer.com/article/10.1365/s40702-020-00641-8>

2.2.1 Convolutional Layer

In den Convolutional Schichten werden die Eingabedaten mit Hilfe eines Filters komprimiert und als Ausgabedaten zur nächsten Schicht weitergeleitet.

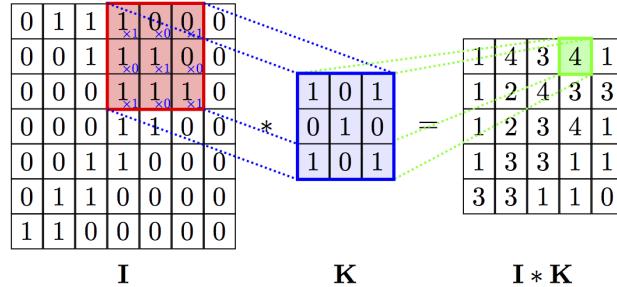


Abbildung 5: Convolution im CNN

Quelle: <https://www.quora.com/Why-do-we-use-convolutional-layers>

2.2.2 Pooling Layer

Nach den Convolutional Schichten wird meist eine Pooling Schicht verwendet, um die Anzahl an Parametern zu verringern. Dies wird durch eine Funktion gemacht, welche im angegebenen Pooling Bereich die Werte komprimiert. Beim maxpooling in Abbildung 6 werden jeweils vier Werte genommen und über die Funktion der höchste der vier Werte als Ausgabewert des Bereichs definiert. Diese Reduktion der Parameter führt zu einer Steigerung der Performance du weniger Rechenzeit, und bessere Kontrolle gegen Überanpassung (overfitting).

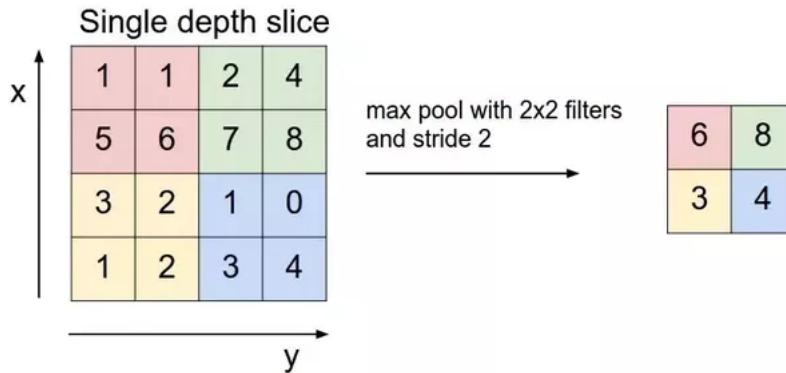


Abbildung 6: Maxpooling2D

Quelle: <https://medium.com/@duanenielsen/deep-learning-cage-match-max-pooling-vs-convolutions-e42581387cb9>

Output Layer

Der letzte Block im Netz ist wie schon erwähnt zur Klassifizierung zuständig. Die einzelnen Schichten im Block sind voll miteinander verbunden. Sinnvollerweise wählt man die Anzahl der Neuronen in der letzten Schicht äquivalent zur Anzahl an zu kategorisierenden Klassen.

Als erste Schicht im Klassifizierungsblock wird eine Flatten Schicht zur Überführung der dreidimensionalen Daten in eindimensionale Daten, die von den Klassifizierungsschichten verarbeitet werden können.“

2.3 Node.js, P5.js und ML5.js

Quelle: <https://p5js.org/get-started/>

Quelle: <https://nodejs.org/en/download/>

Quelle: <https://learn.ml5js.org/>

Zur Umsetzung der Aufgabe wurde eine Javascript Bibliothek mit dem Namen P5.js verwendet. Diese Bibliothek ist vor allem sinnvoll um grafische Elemente zu kreieren. P5.js bringt von Haus aus viele Funktionen mit, um visuelle Elemente auf den Bildschirm zu bringen. Programme in P5.js lassen sich entweder über den Webeditor, oder wie im aktuellen Fall, auf einem lokal eingerichteten Server erstellen.

Um einen lokalen Server auf dem Rechner aufzusetzen gibt es mehrere Möglichkeiten. Hier wurde node.js verwendet. Node.js lässt sich in 3 Schritten relativ einfach aufsetzen. Zuerst muss das Node Package runtergeladen und installiert werden. Danach muss das verwendete Modul **http-server** über npm installiert werden.

```
npm install -g http-server
```

Abbildung 7: Http-Server installieren

Quelle: <https://github.com/processing/p5.js/wiki/Local-server>

Zuletzt nur noch in den Ordner des aktuellen Projektes navigieren und dort über den Befehl **http-server** einen lokalen Server starten. Unter der IP 127.0.0.1:8080 lässt sich nun das Projekt im Browser anzeigen.

```
[bvlabs-MacBook-Pro:sudoku-solver ms$ http-server
Starting up http-server, serving .
Available on:
  http://127.0.0.1:8080
  http://192.168.0.206:8080
Hit CTRL-C to stop the server
```

Abbildung 8: Node Server starten

Quelle: Selbst erstellt

Damit P5.js richtig funktioniert, muss die Main Datei 2 vordefinierte Funktionen enthalten (Abbildung 9). Die erste davon ist die Setup Funktion. Diese wird nur einmal am Anfang der Ausführung des Programms aufgerufen und die vor allem dazu, Umgebungseigenschaften zu definieren. Hierzu gehören zum Beispiel die Bildschirmgröße oder die Hintergrundfarbe. Auch verwendete Bilder können hier geladen werden.

Die zweite dringend benötigte Funktion ist die Draw Funktion. Diese steuert den gesamten Prozess des Zeichnens von Elementen auf dem Bildschirm. Sie wird direkt nach der Setup Funktion aufgerufen. Sie wird abhängig von der Framerate in gleichen Zeitabständen neu aufgerufen und durchlaufen. In Abschnitt 3.1 wird darauf noch genauer eingegangen. Optional kann noch eine dritte Funktion hinzugefügt werden. Diese Funktion ist Preload, und enthält allen Code, der vor dem eigentlichen Programm ausgeführt werden soll, um den Ablauf des benutzergesteuerten Programms nicht zu stören. Im Trainingsprozess des Modells werden hier zum Beispiel die verwendeten Bilder geladen, was eine ganze Weile dauert.

The screenshot shows a code editor window with a dark theme. On the left, there's a sidebar titled "FOLDERS" showing a single folder named "empty-example" containing "index.html" and "sketch.js". The main area has two tabs: "index.html" and "sketch.js". The "sketch.js" tab is active and displays the following code:

```
function setup() {
  // put setup code here
}

function draw() {
  // put drawing code here
}
```

At the bottom of the editor, it says "Line 7, Column 2" and "Spaces: 2 JavaScript".

Abbildung 9: Setup und Draw Funktion
Quelle: <http://staging.p5js.org/get-started/>

Als letztes externes Javascript Paket wurde die Bibliothek ML5 eingebunden (Abbildung 10). ML5 wurde designed um vor allem in der Entwicklungsumgebung mit P5.js eingesetzt zu werden. Dies sieht man schon alleine am Namen. Allerdings lässt sich ML5 auch in andere javascriptbasierte Umgebungen einbinden.

```
<script src="https://unpkg.com/ml5@latest/dist/ml5.min.js"></script>
```

Abbildung 10: Import ML5
Quelle: <https://learn.ml5js.org/>

ML5 ist eine Abstraktionsebene von Tensorflow.js und soll die Funktionalität von maschinellem Lernen einem breiteren Publikum zugänglich machen, in dem es viele dem Benutzer viele Aufgaben abnimmt. In ML5 gibt es mehrere vordefinierte Netzarchitekturen, die für verschiedene Anwendungsbereiche geeignet sind. Die Verwendungsweise dieses Pakets wird in Abschnitt 3.1 noch genauer erklärt.

3 Praktische Umsetzung der Aufgabenstellung

Nachdem nun die verwendeten Konzepte und Frameworks bekannt sind, soll es im folgenden Kapitel darum gehen, wie diese Komponenten dazu eingesetzt wurden, die vorliegende Aufgabenstellung zu lösen.

3.1 Projektstruktur

Um einen groben Überblick zu erhalten, wie das Projekt aufgebaut ist, soll zunächst einmal die Projektstruktur sowohl textuell als auch visuell dargestellt werden. Wie in Abbildung 11 zu sehen, besteht das Projekt aus verschiedenen Ordnern und Dateien.

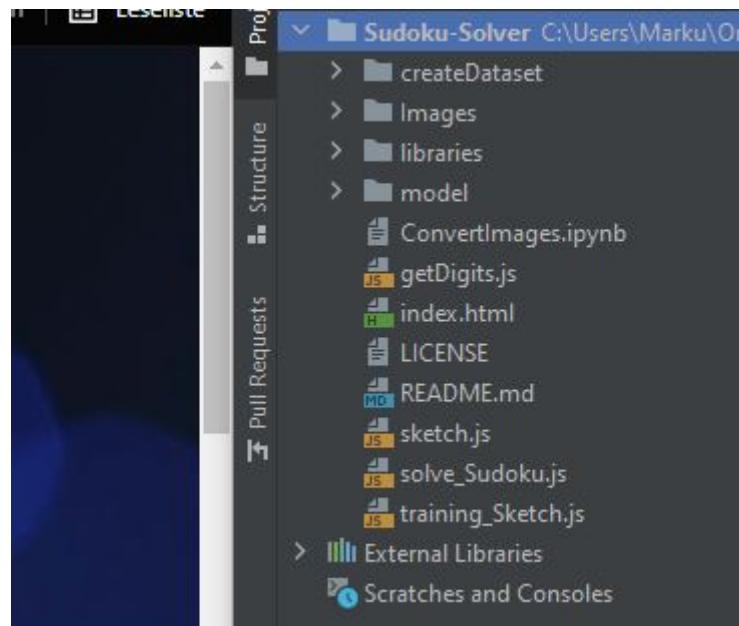


Abbildung 11: Projektstruktur
Quelle: selbst erstellt

3.1.1 Ordner

Der Ordner **createDataset** enthält den Code für die Erstellung des Datensatzes in Kapitel 3.2.3. Dieser ist separiert, da er zum einen nicht den Applikationsprozess betrifft, und zum anderen in Java und nicht in Javascript ist. Im Ordner **Images** sind alle erstellten und verwendeten Bilder gespeichert. **Libraries** ist eine lokale Speicherung der eingebundenen Bibliotheken die in der Index.html eingefügt werden. Im Ordner **model** sind schließlich die gespeicherten Dateien aus Kapitel 3.2.4 enthalten, die das Modell später lädt, um eine Klassifikation der Kamerabilder vorzunehmen.

3.1.2 Dateien

Der Einstieg in ein Projekt auf Basis von P5.js bildet wie bei jeder anderen Webanwendung auch die index.html Datei. Darin sind wie in Listing 1 zu sehen, verschiedene HTML tpyische Abgaben enthalten, wie zum Beispiel die Sprache oder der Titel. Interessant für das Projekt sind dann aber die Zeilen 10 bis 21. Hier werden die verschiedenen verwendeten Bibliotheken eingebunden, und dem Projekt gesagt, in welcher Reihenfolge die selbst erstellten Javascript Dateien geladen werden sollen. Dabei muss die Main Datei immer zuletzt geladen werden. Bei der Ausführung ist zu beachten, dass Zeile 21 auskommentiert ist. Dies ist wichtig, da Zeile 21 alleine für den Trainingsprozess des Modells zuständig ist, und dieser im eigentlich Applikationsverlauf nicht ausgelöst werden soll. Zeile 19 und 21 sind quasi mutually exclusive, dh. es darf immer nur eine davon wirklich aufgerufen werden, die andere muss auskommentiert sein.

```
1 <!DOCTYPE html>
2 <html lang="de">
3   <head>
4     <meta charset="UTF-8">
5     <meta http-equiv="X-UA-Compatible" content="IE=edge">
6     <meta name="viewport" content="width=device-width, initial-scale=1">
7
8     <title>Sudoku-Solver </title>
9
10    <!-- Einbinden der P5.js Bibliothek -->
11    <script src="https://cdnjs.cloudflare.com/ajax/libs/p5.js/1.1.9/p5.min.js"></script>
12    <!-- Einbinden der ML5 Bibliothek -->
13    <script src="https://unpkg.com/ml5@latest/dist/ml5.min.js"></script>
14    <!-- Javascript Datei zum Lösen des Sudokus -->
15    <script src="solve_Sudoku.js"></script>
16    <!-- Javascript Datei zum Erkennen der Ziffern -->
17    <script src="getDigits.js"></script>
18    <!-- Main Javascript Datei -->
19    <script src="sketch.js"></script>
20    <!-- Javascript Datei zum Training des Netzes -->
21    <!--<script src="training_Sketch.js"></script>-->
22
23    <style>
24      body {
25        margin:0;
26        padding:0;
27        overflow: hidden;
28      }
29      canvas {
30        margin: auto;
31      }
32    </style>
33  </head>
34  <body>
35  </body>
36 </html>
```

Listing 1: index.html

Sowohl die Dateien **training_Sketch.js**, **getDigits.js**, und **solve_Sudoku.js** werden in Kapitel 3.2.4, 3.3 und 3.4 noch genauer beschrieben, weshalb hier nur der Aufbau der Datei **sketch.js** erläutert werden soll.

Zuerst werden alle globalen Variablen initialisiert, die im Verlauf der Codeausführung verwendet werden. Hierzu gehören die Variablen für Bilder und das Videosignal, verschiedene Arrays zur Speicherung der Sudokufelder, wie auch die Steuervariablen für die verschiedenen Screens und Buttons. (Listing 2).

```

1 let _startScreen = true; // Screen um die Auswahl ob Upload oder über
2   Kamera aufnehmen anzuzeigen
3 let _uploadScreen = false; // Screen zum Upload eines Sudoku Bildes
4 let _videoScreen = false; // Screen der das Videosignal darstellt
5 let _uploadImageScreen = false; // Screen der das erkannte hochgeladene
6   Bild darstellt
7 let _imageScreen = false; // Screen der das Bild aus dem Videosignal
8   darstellt
9 let _sudokuScreen = false; // Screen der das zugeschnittene Bild darstellt
10 let _calculatedScreen = false; // Screen der das gelöste Sudoku darstellt
11
12 let uploadButton;
13 let videoButton;
14 let uploadImageButton;
15 let getImageButton;
16 let cropImgButton;
17 let resetButton;
18 let calculateButton;
```

Listing 2: sketch.js - Initialisierung

In der vorher schon erwähnten Setup Funktion (Listing 3) werden dann das Canvas erstellt, die Buttons platziert und das Bild für das Logo im StartScreen geladen. Die Buttons definieren auch, welche Funktion beim Klicken aufgerufen werden soll (Zeile 11 und 18).

```

1 function setup() {
2   createCanvas(windowWidth, windowHeight - 100);
3   logo = loadImage("Images/Logo.jpg");
4   background(255);
5   let buttonwidth = 140;
6   let buttonheight = 50;
7   /* Erstellen der Buttons zur Steuerung durch die App*/
8   uploadButton = createButton('Sudoku hochladen');
9   uploadButton.position(windowWidth / 2 - 40 - buttonwidth, windowHeight
10  - 120);
11  uploadButton.size(buttonwidth, buttonheight);
12  uploadButton.mousePressed(uploadScreen);
13 ...
14 ...
15  calculateButton = createButton('Sudoku berechnen!');
16  calculateButton.position(windowWidth / 2 + 40, windowHeight - 120)
17  calculateButton.size(buttonwidth, buttonheight)
18  calculateButton.mousePressed(calculatedScreen)
19  calculateButton.hide();
20 }
```

Listing 3: sketch.js - Setup

Für jeden in der Applikation verwendeten Screen gibt es eine eigene Funktion, die die Steuerung der darin enthaltenen Elemente und Funktionen beinhaltet. Exemplarisch hierzu der in Listing ?? dargestellte Code. Um Platz zu sparen, werden im folgenden nur noch Abweichungen oder zusätzlicher Code in den einzelnen Funktionen dargestellt. Die Boolean Screenvariablen von Zeile 3 bis 9 werden in der Draw Funktion verwendet um zu definieren, welcher Abschnitt dieser Funktion gerade ausgeführt werden soll. So entsteht ein Walkthrough durch die Applikation. Gleichermaßen passiert in Zeile 11 bis 17, hier wird definiert welche Buttons in welchem Screen eingeblendet werden sollen. Im UploadScreen wird dann noch ein Bild von einem Sudoku geladen, welches danach zur Erkennung von Ziffern verwendet wird. Die ersetzt die Funktionalität des Hochladens.

```

1 function uploadScreen() {
2   console.log("UploadScreen");
3   _startScreen = false;
4   _uploadScreen = true;
5   _videoScreen = false;
6   _uploadImageScreen = false;
7   _imageScreen = false;
8   _sudokuScreen = false;
9   _calculatedScreen = false;
10  /* **** */
11  uploadButton.hide();
12  videoButton.hide();
13  uploadImageButton.show();
14  getImageButton.hide();
15  cropImgButton.hide();
16  calculateButton.hide();
17  resetButton.show();
18  /* **** */
19  img = loadImage("Images/Sudoku.jpg");
20  img2 = img;
21 }
```

Listing 4: sketch.js - Uploadscreen

Im **Videoscreen** wird zusätzlich noch das Videosignal abhängig vom Gerätetyp eingelesen (Zeile 1 bis 13) und auf dem Bildschirm dargestellt (Zeile 14 bis 19).

```

1 if (is_desktop) {
2   video = createCapture(VIDEO);
3 } else {
4   video =
5     createCapture({
6       audio: false,
7       video: {
8         facingMode: {
9           exact: "environment"
10        }
11      }
12    });
13 }
14 if (is_desktop) {
15   video.position(windowWidth / 2 - video.width, 0);
16 } else {
17   video.position(0, 0);
18   video.size(windowWidth, windowHeight + 100);
```

```
19 }
```

Listing 5: sketch.js - Videoscreen

Im **UploadImageScreen** wird die Funktion `getDigits` aufgerufen, welche wie in Abschnitt 3.3 beschrieben die Ziffernerkennung durchführt.

```
1 numbers2d = await getDigits(img2);
```

Listing 6: sketch.js - Uploadimagescreen

Im der Funktion **ImageScreen** wird das aktuelle Videosignal als Bild gespeichert. Danach wird das Videosignal gestoppt und versteckt. Anschließend werden die 2 Arrays deklariert, welche die Bildschirmberührungen zum Zuschneiden des Bildes speichern. Außerdem wird der Draw Funktion über die Zeile 7 mitgeteilt, dass sie das rote Quadrat an den Koordinaten in den zwei Arrays zeichnen soll.

```
1 img = video.get(0, 0, video.width, video.height);
2 video.stop();
3 video.hide();
4 mouselicksx = [];
5 mouselicksy = [];
6 showSquare = true;
```

Listing 7: sketch.js - Imagescreen

Die Funktion **SudokuScreen** wird das Zuschneiden des Bildes gesteuert. In Zeile 2 bzw. 8 wird überprüft, ob die Arrays die Koordinaten für das Zuschneiden beinhalten. Wenn ja, wird vom Bild in Variable `img` eine Kopie erstellt die an den Koordinaten zugeschnitten ist und in der Variable `img2` gespeichert. Sollte dies nicht der Fall sein, dann wird das Bild in Zeile 11 quadratisch zugeschnitten. Dies setzt voraus, dass der Benutzer der Applikation das Videosignal des Sudokus vorher relativ passgenau in den Bildschirm eingepasst hat. Außerdem wird auch hier die Bildvariable an die Funktion zur Erkennung der Ziffern weiter gegeben.

```
1 if (is_desktop){
2     if (mouselicksx.length >= 3) {
3         img2 = img.get(mouselicksx[1], mouselicksy[1], mouselicksx[2] -
4             mouselicksx[1], mouselicksy[2] - mouselicksy[1]);
5     } else {
6         img2 = img;
7     }
8 }
9 if (mouselicksx.length >= 3) {
10    img2 = img.get(mouselicksx[1], mouselicksy[1], mouselicksx[2] -
11        mouselicksx[1], mouselicksy[2] - mouselicksy[1]);
12 } else {
13    img2 = img.get(5, 5, windowHeight-10, windowHeight-10);
14 }
15 numbers2d = await getDigits(img2);
```

Listing 8: sketch.js - SudokuScreen

Die letzte Steuerfunktion der Screens ist die Funktion ***CalculatedScreen***. Von Zeile 1 bis 11 wird hier ein Array in Sudokuform erstellt, welches zur Ausführung der Lösungsfunktion in Zeile 12 verwendet wird. In Zeile 13 wird das eindimensionale Feld in ein zweidimensionales Feld umgewandelt.

```

1 const grid = [ // Fallback Feld, da die Erkennung nicht gut funktioniert
2   [5, 3, 0, 0, 7, 0, 0, 0, 0],
3   [6, 0, 0, 1, 9, 5, 0, 0, 0],
4   [0, 9, 8, 0, 0, 0, 0, 6, 0],
5   [8, 0, 0, 0, 6, 0, 0, 0, 3],
6   [4, 0, 0, 8, 0, 3, 0, 0, 1],
7   [7, 0, 0, 0, 2, 0, 0, 0, 6],
8   [0, 6, 0, 0, 0, 0, 2, 8, 0],
9   [0, 0, 0, 4, 1, 9, 0, 0, 5],
10  [0, 0, 0, 0, 8, 0, 0, 7, 9]
11 ];
12 final1d = getGrid(grid);
13 while (final1d.length) final2d.push(final1d.splice(0, 9));

```

Listing 9: sketch.js - CalculatedScreen

In der ***Draw*** Funktion werden alle für das Frontend wichtige Information verarbeitet und die verschiedenen Bilder, Buttons und Sudokufelder auf dem Bildschirm ausgegeben. Diese Funktion auch nur annähernd in ihrer Gänze zu beschreiben und darzustellen würde den Rahmen dieses Berichtes sprengen, weshalb hier nur exemplarisch in Listing 10 das Zeichnen eines einzigen 9x9 Sudokufeldes inklusive des Füllens der Felder dargestellt werden soll.

In Zeile 1 und 2 werden die Farbe und die Linienbreite der Zeichnung definiert. In den zwei For-Schleife von Zeile 3 bis 10 werden dann alle waagrechten und senkrechten Linien anhand der vorher schon definierten Zellgröße gezeichnet. Die äußeren Linien und die welche die 3x3 Felder im Sudoku einschließen werden dunkler gezeichnet.

```

1 stroke(245);
2 strokeWeight(1);
3 for (let i = 1; i < 9; i++) {
4   line(cell_size * (1 / 2 + i), cell_size / 2, cell_size * (1 / 2 + i),
5     cell_size * (10 - 1 / 2));
6   line(cell_size / 2, cell_size * (1 / 2 + i), cell_size * (10 - 1 / 2),
7     cell_size * (1 / 2 + i));
8 }
9 stroke(45);
10 for (let i = 0; i <= 3; i++) {
11   line(cell_size * (1 / 2 + i * 3), cell_size / 2, cell_size * (1 / 2 + i
12     * 3), cell_size * (10 - 1 / 2));
13   line(cell_size / 2, cell_size * (1 / 2 + i * 3), cell_size * (10 - 1 /
14     2), cell_size * (1 / 2 + i * 3));
15 }

```

Listing 10: sketch.js - Draw Funktion 1

Danach werden dann anhand von Koordinaten auf dem Bildschirm die 81 noch leeren Felder des Sudokus mit dem Inhalt des im Array final2d gefüllt. Ziffern die nicht 0 sind werden dabei schwarz gezeichnet. Ziffern die 0 sind, was vor dem Berechnen der Sudokulösung vorkommt, werden grün gezeichnet. Sollte im Array irgendwas anderes gespeichert sein, was auf einen Fehler hindeutet, wird was auch immer darin enthalten ist orange gezeichnet.

```
1 noStroke();
2 textSize(20);
3 for (let i = 0; i < 9; i++) {
4   for (let j = 0; j < 9; j++) {
5     if (final2d[i][j] !== 0) {
6       stroke("black");
7       fill("black");
8       text(final2d[i][j], cell_size * (j + 1), cell_size * (i + 1) +
6);
9     } else if (final2d[i][j] > 0) {
10       stroke("green");
11       fill("green");
12       text(final2d[i][j], cell_size * (j + 1), cell_size * (i + 1) +
6);
13     } else if (final2d[i][j] === 0) {
14       stroke("green");
15       fill("green");
16       text("", cell_size * (j + 1), cell_size * (i + 1) + 6);
17     } else {
18       stroke("orange");
19       fill("orange");
20       text(final2d[i][j], cell_size * (j + 1) - 12, cell_size * (i +
1) + (j % 3 - 1) * 12 + 6);
21     }
22 }
```

Listing 11: sketch.js - Draw Funktion 2

3.2 Trainingsprozess

Um eine spätere Ziffernerkennung anhand der Kamerabilder zu ermöglichen, ist es erforderlich, dafür ein Modell zu trainieren. Hierbei sind mehrere Schritte notwendig.

3.2.1 Datensatz

Der erste und wichtigste Schritt um den Trainingsprozess zu bewältigen, ist die Auswahl bzw. Erstellung eines zu der Aufgabe passenden Datensatzes. Da es bei der vorliegenden Applikation darum geht, Ziffern in einem gegebenen Sudokufeld zu erkennen und dann das Sudoku zu lösen, sollte auch der Datensatz diese Aufgabe repräsentieren. Außerdem müssen die Daten vor dem Training so bearbeitet werden, dass sie zu der Architektur des verwendeten Netzes passen. Dies betrifft vor allem die Dimensionalität des Inputssignales, wie die Höhe und Breite, sowie die Farben des Bildes, dh. ob farbig (4) oder Graustufen (1).

3.2.2 MNIST Handwritten Digits

Die erste spontane Idee an verwendbare Daten zu kommen war, einen schon vorhandenen Datensatz zu verwenden. Hierbei bot sich der MNIST-Datensatz der handgeschriebenen Ziffern an, welcher auch meist der verwendete Datensatz ist, wenn man mit Machine Learning das erste Mal in Berührung kommt. Dieser Datensatz beinhaltet 60.000 Bilder von handgeschriebenen Ziffern im Format 28x28 Pixeln. Um diesen Datensatz nutzen zu können, mussten die Bilder allerdings noch farblich invertiert werden, da beim online verfügbaren Datensatz die Ziffern weiß und der Hintergrund schwarz ist. Deshalb wurde mit Python (Listing 12) ein Programm geschrieben, das automatisch alle Bilder des Datensatz invertiert. In Abbildung 12 sind die vorher-nachher Ergebnisse zu sehen.

```
1 from PIL import Image , ImageOps
2
3 for i in range(10):
4     for j in range(1,3501):
5         im = Image.open('trainingSet/' + str(i) + '/img' + str(j) + '.jpg')
6         img_invert = ImageOps.invert(im)
7         im_invert.save('newtrainingSet/' + str(i) + '/img' + str(j) + '.jpg')
```

Listing 12: convertImages.py

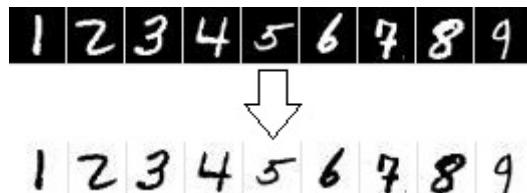


Abbildung 12: Convert Handwritten Digits

Quelle: Selbst erstellt

3.2.3 Selbst erstellter Datensatz

Nachdem der erste Ansatz mit den handgeschrieben Ziffern ausführlich und länger mit der fertigen Applikation getestet wurde, war klar, dass dieser Datensatz und das daraus generierte Modell nicht ausreichend genau funktioniert. Deshalb wurde kurzerhand ein eigener Datensatz erstellt, der die realen Input Daten besser repräsentieren sollte. Hierbei kam Processing auf Basis von Java zum Einsatz. Hierbei wurden wie im Code (Listing 13) zu sehen ist, von jeder Ziffer von 0 bis 9 1500 Bilder erstellt. Dabei wurde sowohl die Farbe des Hintergrunds, die Größe der einzelnen Ziffer, als auch die Position im Bild selbst leicht verändert. Somit ist jedes Bild im Datensatz etwas unterschiedlich, und das Modell lernt mit verschiedenen Lichtverhältnissen und verschiedenen Größen der gegebenen Ziffern umzugehen. In Abbildung 13 ist jeweils ein Beispiel dieser Bilder zu sehen. Ein rein einfarbiges Bild wurde in diesem Datensatz als Referenz für ein leeres Sudokufeld (Ziffer 0) erstellt, da im vorherigen Datensatz oft leere Felder trotzdem als eine der Ziffern von 1 bis 9 erkannt wurden. Dies soll mit dieser Vorgehensweise verhindert werden.

```
1 PFont f;
2 void setup() {
3     size(28,28);
4     f = createFont("Arial",16,true);
5 }
6 void draw() {
7     for (int i = 1; i < 1501; i++) {
8         for (int j = 0; j < 10; j++) {
9             float backg = random(250,255);
10            float fontsize = random(20,25);
11            float textx = random(width/2-8,width/2-3);
12            float texty = random(height/2+8,height/2+11);
13            background(backg);
14            textAlign(f, fontsize);
15            fill(0);
16            if(j == 0){
17                text("", textx , texty);
18            } else {
19                text(j, textx , texty);
20            }
21            saveFrame("data/" + j + "/Img" + i + ".jpg");
22        }
23    }
24 }
```

Listing 13: createDataset.pde

	1	2	3	4
5	6	7	8	9

Abbildung 13: Java created Digits
Quelle: Selbst erstellt

3.2.4 Trainingsvorgang

Um die Bilder aus dem Datensatz dem Modell zugänglich zu machen, müssen diese geladen werden. Hierzu werden für jedes vorhanden Label der Daten ein eigenes Array erstellt, um die Bilder im weiteren Verlauf besser separieren zu können. Diese Arrays werden in Listing 14 initialisiert. Außerdem wird die Variable initialisiert, in der im nächsten Abschnitt das Modell gespeichert wird.

```
1 /* Initialisierung der verwendeten Arrays */
2 let zero = [];
3 let one = [];
4 let two = [];
5 let three = [];
6 let four = [];
7 let five = [];
8 let six = [];
9 let seven = [];
10 let eight = [];
11 let nine = [];
12 let numberClassifier;
```

Listing 14: training_Sketch.js - Initialisierung

Die **Preload** Funktion wird in P5.js vor dem Laden des eigentlichen Inhalts der Seite ausgeführt, was in diesem Fall besonders nützlich ist, um die große Datenmenge an Bildern in die einzelnen Arrays zu laden. In der Konsole im Entwicklertool wird außerdem noch der Fortschritt der geladenen Bilder in 100er Schritten angezeigt.

```
1 /* Laden der Bilder aus dem Datensatz */
2 function preload() {
3     for (let i = 0; i < 1500; i++){
4         let index = nf(i+1);
5         zero[i] = loadImage('Images/ownTrainingSet/0/Img'+index+'.jpg');
6         one[i] = loadImage('Images/ownTrainingSet/1/Img'+index+'.jpg');
7         two[i] = loadImage('Images/ownTrainingSet/2/Img'+index+'.jpg');
8         three[i] = loadImage('Images/ownTrainingSet/3/Img'+index+'.jpg');
9         four[i] = loadImage('Images/ownTrainingSet/4/Img'+index+'.jpg');
10        five[i] = loadImage('Images/ownTrainingSet/5/Img'+index+'.jpg');
11        six[i] = loadImage('Images/ownTrainingSet/6/Img'+index+'.jpg');
12        seven[i] = loadImage('Images/ownTrainingSet/7/Img'+index+'.jpg');
13        eight[i] = loadImage('Images/ownTrainingSet/8/Img'+index+'.jpg');
14        nine[i] = loadImage('Images/ownTrainingSet/9/Img'+index+'.jpg');
15        if (i%100==0) {
16            console.log((i/100+1)*100+' images loaded!');
17        }
18    }
19 }
```

Listing 15: training_Sketch.js - Preload

In der **Setup** Funktion passiert der eigentliche Trainingsprozess. In Listing 16 wird in Zeile 8 das verwendete neuronale Netz in die Variable `numberClassifier` geladen. Dabei erwartet die Funktion `ml5.neuralNetwork` ein Dictionary aus verschiedene Key-Value Paaren damit bekannt ist, wie das Training auszusehen hat. Hier ist einmal die Inputform entscheidend, in diesem Fall die Größe der verwendeten Bilder ([28, 28, 4]). Des Weiteren muss dem Netz mitgeteilt werden, was für eine Aufgabe bewältigt werden soll, hier die Klassifikation von Bildern (imageClassification).

In Zeile 9 bis Zeile 20 werden die Bilder in das Netz geladen. In Zeile 21 werden diese Daten noch normalisiert. In Zeile 22 schließlich wird das neuronale Netz dann auf dem geladenen Datensatz trainiert. Hierbei erhält man auch im Browser eine visuelle Rückmeldung des Trainingsvorgangs wie in Abbildung 14 zu sehen ist.

Die Funktion `finishedTraining` ist eine Callback Funktion, die am Ende des Trainings aufgerufen wird um das trainierte Modell mit seiner Architektur und seinen Gewichten zu speichern. Diese werden im weiteren Verlauf geladen um den eigentlichen Klassifizierungsvorgang durchführen zu können.

```

1  /* Training des Modells mit den vorher geladenen Daten */
2  function setup() {
3    let options = {
4      inputs: [28,28,4],
5      task: 'imageClassification',
6      debug: true
7    }
8    numberClassifier = ml5.neuralNetwork(options);
9    for (let j = 0; j < one.length; j++) {
10      numberClassifier.addData({image:zero[j]},{label:'0'});
11      numberClassifier.addData({image:one[j]},{label:'1'});
12      numberClassifier.addData({image:two[j]},{label:'2'});
13      numberClassifier.addData({image:three[j]},{label:'3'});
14      numberClassifier.addData({image:four[j]},{label:'4'});
15      numberClassifier.addData({image:five[j]},{label:'5'});
16      numberClassifier.addData({image:six[j]},{label:'6'});
17      numberClassifier.addData({image:seven[j]},{label:'7'});
18      numberClassifier.addData({image:eight[j]},{label:'8'});
19      numberClassifier.addData({image:nine[j]},{label:'9'});
20    }
21    numberClassifier.normalizeData();
22    numberClassifier.train({epochs: 20}, finishedTraining);
23  }
24  function finishedTraining() {
25    numberClassifier.save();
26    console.log("Training finished!");
27 }
```

Listing 16: training_Sketch.js - Setup

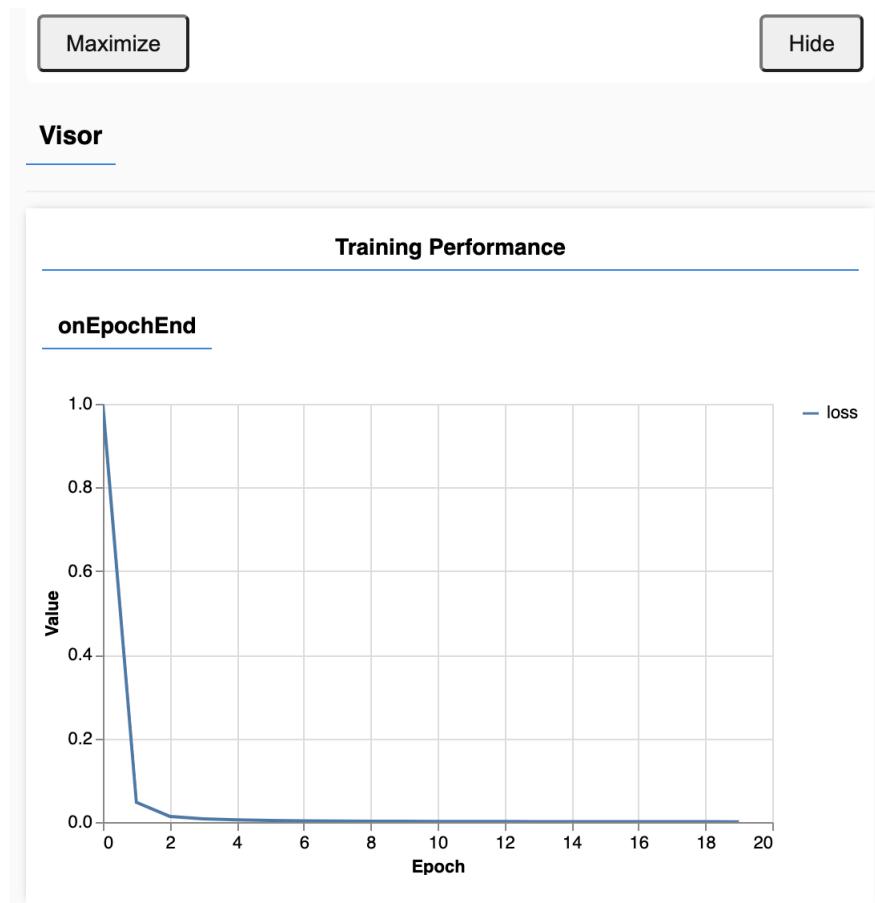


Abbildung 14: Trainingsverlauf
Quelle: Selbst erstellt

Model Summary

Layer Name	Output Shape	# Of Params	Trainable
conv2d_Conv2D1	[batch,24,24,8]	808	true
max_pooling2d_MaxPooling2D1	[batch,12,12,8]	0	true
conv2d_Conv2D2	[batch,8,8,16]	3,216	true
max_pooling2d_MaxPooling2D2	[batch,4,4,16]	0	true
flatten_Flatten1	[batch,256]	0	true
dense_Dense1	[batch,10]	2,570	true

Abbildung 15: Modellparameter
Quelle: Selbst erstellt

3.3 Zahlenerkennung

Um das zuvor in Kapitel 3.2.4 trainierte Modell zum Erkennen von Ziffern zu verwenden, muss dieses nun der Main Javascript Klasse zur Verfügung gestellt werden. Sehr ähnlich wie zuvor, muss auch hier erstmal eine Variable erstellt werden, die dann das neuronale Netz mit seinen festgelegten Optionen speichert (Zeile 3 bis 7). Außerdem werden in Zeile 13 die vorher gespeicherten Modeldateien (Gewichte und Architektur) geladen. Dies geschieht auch in der Main Klasse in der Funktion **Preload** damit die Applikation direkt nach dem ersten Laden des Browsers darauf zugreifen kann.

```
1 /* Laden des vorher gespeicherten Modells */
2 async function preload() {
3     let options = {
4         inputs: [28, 28, 4],
5         task: 'imageClassification',
6     }
7     numberClassifier = ml5.neuralNetwork(options);
8     const modelDetails = {
9         model: 'model/model.json',
10        metadata: 'model/model_meta.json',
11        weights: 'model/model.weights.bin'
12    }
13    numberClassifier.load(modelDetails, modelLoaded);
14 }
```

Listing 17: sketch.js - Preload

Die Funktion **getDigits** (Listing 18) führt die eigentliche Erkennung der Ziffern im Sudokufeld aus. Der Funktion wird beim Aufruf ein Bild mitgegeben, welches vom Benutzer aufgenommen und zugeschnitten wurde. In Zeile 3 wird das Array initialisiert, in dem im Laufe der Funktion die erkannten Ziffern gespeichert werden. In Zeile 4 bis Zeile 9 wird das Bild in 81 gleich große Unterbilder aufgeteilt, welche dann einzeln vom Erkennungsalgorithmus untersucht werden. Diese 81 Bilder sind in Abbildung 17 zu sehen. Diese Bilder werden dann auf 28x28 Pixel skaliert, damit sie als Input vom neuronalen Netz verwendet werden können. Ab Zeile 11 passiert dann die eigentliche Erkennung. Die Funktion **numberClassifier.classify** erwartet als zweites Argument eine Callback Funktion, welche in diesem Fall inline definiert wird. Diese Funktion die ohne Bezeichnung auskommt, da sie nie aufgerufen wird, speichert die Ergebnisse der Erkennungsfunktion in das Array numbers (Zeile 23 bis 28). Sollte sicher der Algorithmus mehr als 99 Prozent sicher sein mit seiner Vorhersage, wird diese Vorhersage gespeichert, ansonsten wird diese Stelle im Array mit einer 0 gefüllt. In Abbildung 18 sieht man welche Ziffern vom neuronalen Netz erkannt wurden, im Gegensatz zu Abbildung 16, wo die tatsächlichen Ziffern abgebildet sind. Am Ende der Funktion wird das Array mit den Ziffern an den Funktionsaufruf zurückgegeben.

```

1 /* Funktion zur Erkennung der Ziffern */
2 async function getDigits(img) {
3     let numbers = [];
4     let imgwidth = img.width / 9;
5     let imgheight = img.height / 9;
6     const promises = [];
7     for (let i = 0; i < 9; i++) {
8         for (let j = 0; j < 9; j++) {
9             let subimg = img.get(j * imgwidth+imgwidth*0.1, i * imgheight+
imgwidth*0.1, imgwidth*0.8, imgheight*0.8);
10            subimg.resize(28, 28);
11            promises.push(new Promise(function (resolve, reject){
12                numberClassifier.classify({image: subimg}, function (err,
13                results) {
14                    if (err) {
15                        console.error(err);
16                        reject(err);
17                        return;
18                    }
19                    let label = int(results[0].label);
20                    let confidence = nf(100 * results[0].confidence , 2, 0)
21 ;
22                    if (!numbers[i]) {
23                        numbers[i] = [];
24                    }
25                    if (confidence >= 99) {
26                        numbers[i][j] = label;
27                        resolve(label);
28                    } else {
29                        numbers[i][j] = 0;
30                        resolve(0);
31                    }
32                });
33            }));
34        }
35    }
36    await Promise.all(promises);
37    return numbers;
38 }

```

Listing 18: getDigits.js

5	3		7				
6			1	9	5		
	9	8				6	
8			6				3
4		8	3				1
7			2			6	
	6			2	8		
		4	1	9			5
			8		7	9	

Abbildung 16: Ausgangssudoku

5	3		7				
6		1	9	5			
	9	8			6		
8			6			3	
4		8	3			1	
7			2			6	
	6			2	8		
		4	1	9		5	
			8		7	9	

Abbildung 17: 81 Unterbilder

5	3		7				
6		3	9	5			
	9	8			6		
8			6			3	
4		8	3			2	
7			2			6	
	6			2	8		
		4	3	9		3	
			8		7	9	

Abbildung 18: Erkannte Ziffern

Quelle: Selbst erstellt

3.4 Der Lösungsalgorithmus

Als letzter Schritt im Applikationsprozess steht nach dem Erkennen der Ziffern im Sudokufeld das Lösen des Sudokus an. Hierbei soll ein etwas intelligenterer Lösungsalgorithmus als nur Bruteforce zum Einsatz kommen. Im Folgenden soll der Ablauf des Lösungsalgorithmuses anhand von Codebeispielen aufgezeigt werden.

Als Hilfsarray für künftige Methoden wird ein Array erstellt, welches als Wert die Integerzahl gespeichert hat, welche angibt, in welchem der 9 3x3 Blöcke die jeweilige Stelle im Array liegt.

```
1 var square_coordinates = [
2     [1, 1, 1, 2, 2, 2, 3, 3, 3],
3     [1, 1, 1, 2, 2, 2, 3, 3, 3],
4     [1, 1, 1, 2, 2, 2, 3, 3, 3],
5     [4, 4, 4, 5, 5, 5, 6, 6, 6],
6     [4, 4, 4, 5, 5, 5, 6, 6, 6],
7     [4, 4, 4, 5, 5, 5, 6, 6, 6],
8     [7, 7, 7, 8, 8, 8, 9, 9, 9],
9     [7, 7, 7, 8, 8, 8, 9, 9, 9],
10    [7, 7, 7, 8, 8, 8, 9, 9, 9]
11 ]
```

Listing 19: solve_Sudoku.js - square-coordinates

GetGrid ist die Funktion, die als Schnittstelle aus der Main Javascript Datei aufgerufen wird. Als Input bekommt die Funktion ein zu lösendes Sudokufeld als zweidimensionales Array, und liefert ein eindimensionales Array mit dem gelösten Sudoku zurück.

```
1 function getGrid(grid) {
2     let newArr = [];
3
4
5     for(let i = 0; i < solve(grid).length; i++)
6     {
7         newArr = newArr.concat(solve(grid)[i]);
8     }
9     return newArr;
10 }
```

Listing 20: solve_Sudoku.js - getGrid

Die Funktion **solve** ist die Hauptfunktion in der Klasse solve_Sudoku.js. Sie steuert den weiteren Lösungsalgorithmus und überprüft, ob das gesamte Sudokufeld schon gelöst ist (Zeile 3, 6 und 11), und ob im letzten Lösungsschritt eine weitere Ziffer im Feld hinzugefügt wurde (Zeile 2, 5 und 10). Dies ist besonders bei schweren Sudokufeldern wichtig, da es hier vorkommen kann, dass mit normalen Lösungsalgorithmen keine Lösung gefunden werden kann. In diesem Fall greift die solve Funktion dann auf einen Brute-force Backtracking Algorithmus zurück, die den Rest des Sudokufeldes löst.

```

1 function solve(board) {
2     let updated = true
3     let solved = false
4     while (updated && !solved) {
5         updated = one_value_cell_constraint(board)
6         solved = is_solved(board)
7     }
8
9     if (!solved) {
10        board = backtrack_based(board)
11        solved = is_solved(board)
12    }
13
14    return board
15 }
```

Listing 21: solve_Sudoku.js - solve

get_row, get_column und get_square sind Hilfsfunktionen um die jeweilige Zeile, Spalte oder 3x3 Quadrat mit den darin enthaltenen Ziffern und Positionen zu erhalten.

```

1 function get_row(board , row) {
2     return board[row]
3 }
4 function get_column(board , column) {
5     var col = []
6     for (let row = 0; row < 9; row++) {
7         col.push(board[row][column]);
8     }
9     return col
10 }
11 function get_square(board , square) {
12     let cells = []
13     for (let r = 0; r < 9; r++) {
14         for (let c = 0; c < 9; c++) {
15             if (square === square_coordinates[r][c]) {
16                 cells.push(board[r][c])
17             }
18         }
19     }
20     return cells
21 }
```

Listing 22: solve_Sudoku.js - get_row/column/square

In der Funktion **complete_cell** wird kontrolliert, welche Ziffern in einer bestimmten Zellen imm Array ([r][c]) noch möglich sind. In die Variable werden alle Ziffern gespeichert, die über die Hilfsfunktionen aus den Zeilen, Spalten und 3x3 Quadranten geholt werden. Diese Ziffern sind als Lösung nicht mehr möglich. Danach wird ein leeres Array erstellt, und in der for-Schleife mit den restlichen möglichen Ziffern gefüllt wird. In Zeile 9 wird

überprüft, ob dieses Array nur eine Ziffer enthält, dann ist diese Ziffer auch die einzige Lösung für die Zelle im Sudoku. Ansonsten wird das Array mit den Möglichkeiten an der Stelle [r][c] gespeichert.

```
1 function complete_cell(board, r, c) {
2     let used = [...get_row(board, r), ...get_column(board, c), ...
3     get_square(board, square_coordinates[r][c])]
4     let possibilities = []
5     for (let p = 1; p <= 9; p++) {
6         if (!used.includes(p)) {
7             possibilities.push(p)
8         }
9     }
10    if (possibilities.length === 1) {
11        board[r][c] = possibilities[0]
12        return true
13    } else {
14        board[r][c] = possibilities
15    }
16 }
```

Listing 23: solve_Sudoku.js - complete_cell

Die Funktion *appears_once_only* bildet die Grundlage für die in Kapitel 2.1.2 angesprochene Counting Methode. In der Funktion wird für eine gegebene Zelle im Array kontrolliert, welche verschiedenen Möglichkeiten von Ziffern darin enthalten sind. Sie dient als Hilfsfunktion für die Funktion *one_value_cell_constraint*

```

1 function appears_once_only(board, possibilities, segment, r, c) {
2     let updated = false
3     for (i = 0; i < possibilities.length; i++) {
4         let possibility = possibilities[i]
5         let counter = 0
6         segment.forEach(cell => {
7             if (Array.isArray(cell)) {
8                 if (cell.includes(possibility)) {
9                     counter++
10                }
11            } else {
12                if (cell === possibility) {
13                    counter++
14                }
15            }
16        })
17        if (counter === 1) {
18            board[r][c] = possibility
19            updated = true
20            break
21        }
22    }
23    return updated
24 }
```

Listing 24: solve_Sudoku.js - square-coordinates

```

1 function compare(expected, actual) {
2     let array1 = expected.slice()
3     let array2 = actual.slice()
4     return array1.length === array2.length && array1.sort().every(function
5         (value, index) { return value === array2.sort()[index] });
}
```

Listing 25: solve_Sudoku.js - square-coordinates

Hilfsfunktion, die kontrolliert, ob ein gegebenen Feld korrekt gelöst ist. Input ist ein zweidimensionales Array, der Rückgabewert ist boolean.

```
1 function is_solved(board) {
2     let expected = [1, 2, 3, 4, 5, 6, 7, 8, 9]
3     let valid = true
4     for (r = 0; r < 9 && valid === true; r++) {
5         if (!compare(expected, get_row(board, r))) {
6             valid = false
7         }
8     }
9     for (c = 0; c < 9 && valid === true; c++) {
10        if (!compare(expected, get_column(board, c))) {
11            valid = false
12        }
13    }
14    for (q = 1; q < 9 && valid === true; q++) {
15        if (!compare(expected, get_square(board, q))) {
16            valid = false
17        }
18    }
19    return valid
20 }
```

Listing 26: solve_Sudoku.js - is_solved

```

1 function backtrack_based(orig_board) {
2     let board = JSON.parse(JSON.stringify(orig_board));
3     for (let r = 0; r < 9; r++) {
4         for (let c = 0; c < 9; c++) {
5             if (board[r][c] === 0) {
6                 complete_cell(board, r, c)
7                 if (is_solved(board)) return board;
8                 let cell = board[r][c]
9                 if (Array.isArray(cell)) {
10                     for (let i = 0; i < cell.length; i++) {
11                         let board_2 = JSON.parse(JSON.stringify(board));
12                         board_2[r][c] = cell[i]
13                         if (completed_board = backtrack_based(board_2)) {
14                             return completed_board;
15                         }
16                     }
17                     return false // dead end
18                 }
19             }
20         }
21     }
22     return false;
23 }
```

Listing 27: solve_Sudoku.js - square-coordinates

```

1 function one_value_cell_constraint(board) {
2     updated = false
3     for (let r = 0; r < 9; r++) {
4         for (let c = 0; c < 9; c++) {
5             if (board[r][c] == 0) {
6                 updated = complete_cell(board, r, c) || updated
7             }
8         }
9     }
10    for (let r = 0; r < 9; r++) {
11        for (let c = 0; c < 9; c++) {
12            if (Array.isArray(board[r][c])) {
13                let possibilities = board[r][c]
14                updated = appears_once_only(board, possibilities, get_row(
15                    board, r), r, c) ||
16                        appears_once_only(board, possibilities, get_column(
17                    board, c), r, c) ||
18                        appears_once_only(board, possibilities, get_square(
19                    board, square_coordinates[r][c]), r, c) || updated
20            }
21        }
22    }
23    return updated
24 }
25 }
26 }
27 }
28 }

```

Listing 28: solve_Sudoku.js - square-coordinates

4 Applikations-Walkthrough

Der Abschnitt Applikations-Walkthrough dient zum einen nochmal als Zusammenfassung der vorher beschriebenen Schritte in einer mehr visuellen Form. Zum anderen aber auch als eine Art Gebrauchsanweisung um die Funktionen der Applikation anhand von Bildern nachvollziehen zu können. Die Applikation ist unter der URL <https://mashnak.github.io/Sudoku-Solver/> erreichbar.

4.1 StartScreen

Der StartScreen ist die erste Ansicht die der Nutzer von der App sieht. Hier stehen dem Benutzer 2 Buttons zur Verfügung. Er kann entscheiden ob er entweder ein schon vorher aufgenommenes Bild von einem Sudoku lösen lassen will (zumindest sollte das so sein, im Moment funktioniert das noch nicht). Oder du Benutzer kann seine im Handy verbauten Kamera dazu nutzen, ein Bild von einem Sudoku aufzunehmen, dieses zuschneiden zu lassen, und dann dieses Sudoku von der Applikation lösen zu lassen.



Abbildung 19: Startscreen
Quelle: Selbst erstellt

4.2 UploadScreen

Der UploadScreen sollte eigentlich die Möglichkeit bieten, ein Foto aus dem lokalen Speicher des Handys auszuwählen und der Applikation zur Verfügung zu stellen. In der aktuellen Version ist hier noch das dargestellte Sudokufeld als Bilddatei hinterlegt um die nachfolgende Funktionalität der Applikation zeigen zu können. Der Reset-Button ist in jeder Ansicht der Applikation exklusive des Startscreens vorhanden, um die Applikation neu zu starten. Der Button Foto hochladen lädt das Bild in den Speicher der Applikation und stößt das Erkennen des bereitgestellten Bildes an. Außerdem leitet ein Klick auf diesen Button den Benutzer zum nächsten Screen weiter.



Hier sollte eigentlich ein BildUpload sein!

Zum Test wird ein gemocktes Bild bereitgestellt

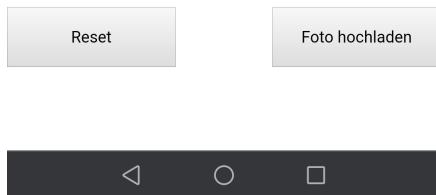


Abbildung 20: UploadScreen
Quelle: Selbst erstellt

4.3 UploadImageScreen

Der UploadImageScreen zeigt dem Benutzer der Applikation das Ergebnis des Erkennungsalgorithmus an. Der Button Sudoku berechnen leitet den Benutzer auf den nächsten Screen weiter, und löst das auf dem Bildschirm angezeigte Sudokufeld.

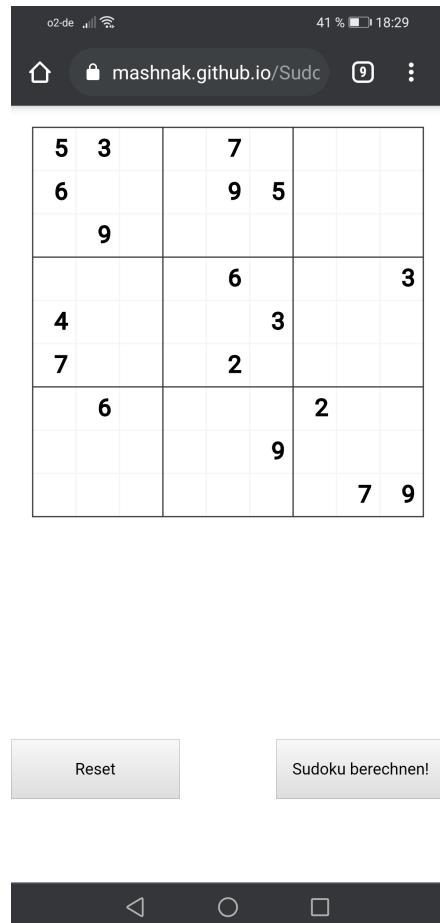


Abbildung 21: UploadImageScreen
Quelle: Selbst erstellt

4.4 VideoScreen

Im Videoscreen wird dem Benutzer das aktuelle Videomaterial der integrierten Kamera des Handys angezeigt. Der Benutzer sollte für eine gute Erkennung des Sudokus dieses so bündig wie möglich an den oberen, linken und rechten Rand des Kamerabildes einpassen. Sollte dies nicht gelingen gibt es auf dem nächsten Screen auch noch die Möglichkeit dies zu bearbeiten. Der Button Foto aufnehmen speichert das aktuelle Videosignal als Bild in der Applikation und gibt dieses intern an den nächsten Screen weiter.

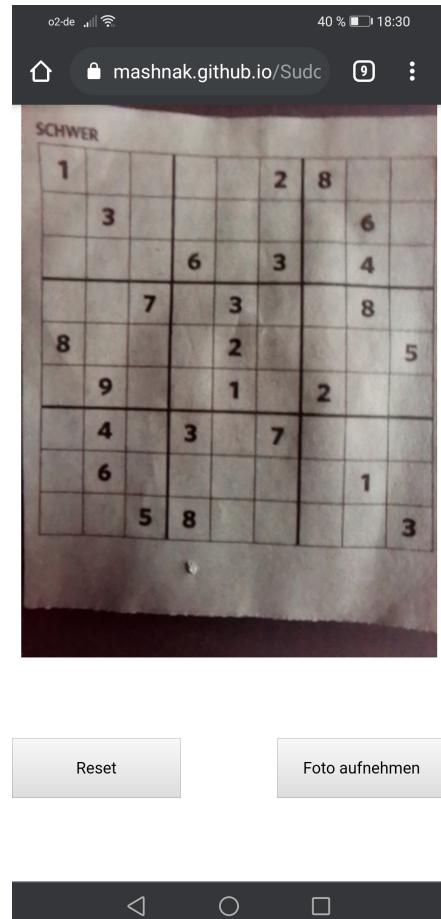


Abbildung 22: VideoScreen
Quelle: Selbst erstellt

4.5 ImageScreen

Der Imagescreen zeigt dem Benutzer das im vorherigen Screen aufgenommene Bild an. Der Nutzer kann dieses Bild entweder von der Applikation selbst zuschneiden lassen (Abbildung 23), oder dies selbst übernehmen (Abbildung 24). Bei der iPhone Version funktioniert das, indem der Benutzer die linke obere und die rechte untere Ecke des Sudokufeldes anklickt und somit dem Programm mitteilt, wie es das Bild zuschneiden soll. Hierbei ist zu beachten, das Sudoku vorher möglichst gerade fotografiert zu haben. Mit einem Klick auf den Button Sudoku zuschneiden wird intern das Bild zugeschnitten.

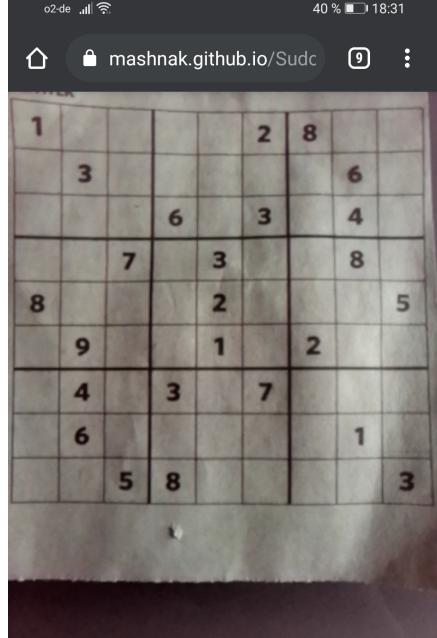


Abbildung 23: ImageScreen - Andro-
id
Quelle: Selbst erstellt

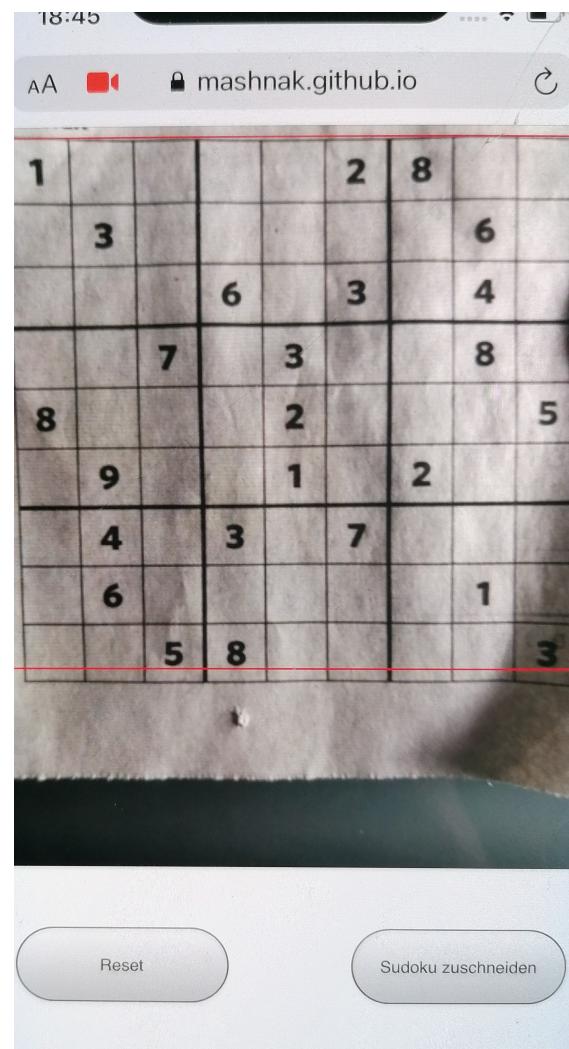


Abbildung 24: ImageScreen - iPhone
Quelle: Selbst erstellt

4.6 SudokuScreen

Im SudokuScreen wird das zugeschnittene Sudokubild dargestellt. Intern läuft in diesem Screen auch die Erkennung der Ziffern im Bild und theoretisch auch die Initialisierung des Lösungsalgorithmus mit diesen Daten. Allerdings ist der Erkennungsalgorithmus gerade bei selbst aufgenommenen Bildern noch nicht genau genug, weshalb hier intern ein vorher definiertes Array mit einem Sudoku geladen wird, was der Applikation erlaubt, einen Lösungsvorgang des Sudokus mit richtig auszuführen.

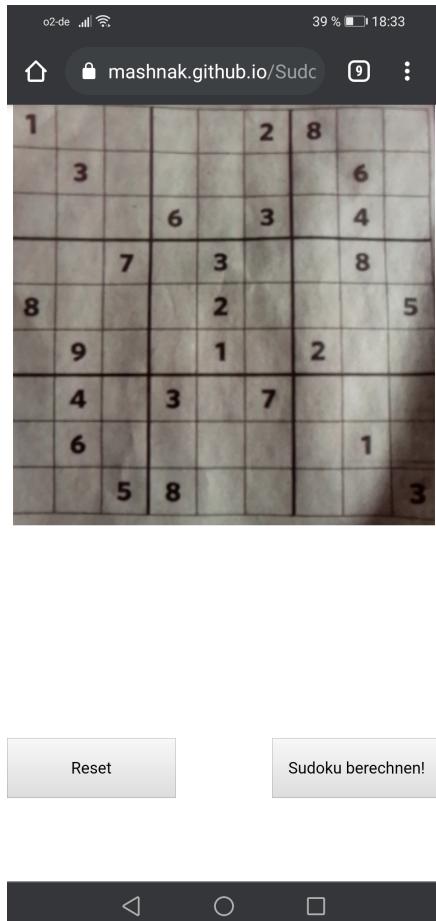
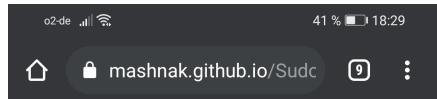


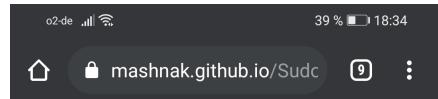
Abbildung 25: SudokuScreen
Quelle: Selbst erstellt

4.7 CalculatedScreen

Der letzte Screen der durch die Applikation führt ist der CalculatedScreen. Hier wird dem Benutzer das Ergebnis des vorher ausgeführten Lösungsvorgangs angezeigt. In Abbildung 26 ist das Ergebnis des hochgeladenen Sudokus angezeigt, welches mit real erkannten Ziffern gelöst wurde. In Abbildung 27 ist das intern geladene Sudoku gelöst dargestellt.



5	3	1	2	7	4	6	9	8
6	2	4	8	9	5	1	3	7
8	9	7	1	3	6	4	2	5
2	1	9	4	6	8	7	5	3
4	8	6	7	5	3	9	1	2
7	5	3	9	2	1	8	4	6
9	6	5	3	4	7	2	8	1
1	7	2	5	8	9	3	6	4
3	4	8	6	1	2	5	7	9



5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9



Abbildung 26: CalculatedScreen
Upload Image
Quelle: Selbst erstellt

Abbildung 27: CalculatedScreen
Video Image
Quelle: Selbst erstellt

5 Diskussion und Ausblick

Im Laufe der Projektarbeit musste ich mich den unterschiedlichsten Herausforderungen stellen und verschiedenste Probleme lösen um eine funktionierende Applikation zu erstellen. Zum einen sind da vor allem die theoretischen Grundlagen zu nennen, wie die Einarbeitung in die Funktionsweise und das Lösen von Sudokus und die Umsetzung dieser Lösungsmethodiken in einen funktionierenden Algorithmus in Javascript.

Des weiteren stelle die Auswahl des verwendeten Frameworks mich immer wieder vor leider teilweise (zumindest für mich) unlösbare Probleme. Hier ist zum Beispiel der Upload Prozess von Bildmaterial zu nennen, den ich unbedingt in die Applikation integrieren wollte, dies aber mit P5.js nicht zustande bekommen habe.

Auch die Entwicklungssprache Javascript war für mich mit wenigen Ausnahmen komplett neu, und gerade ihre Eigenschaft als synchrone, single-threaded Sprache hat mir in vielen Situation das Leben schwer gemacht, weil Code der in anderen Sprachen super funktioniert hätte, in Javascript ein komplett unvorhergesehenes Verhalten zeigte. Zum Ende hin hatte ich dann aber Erfahrung damit und konnte noch einige Probleme lösen, die mir über lange Zeit unlösbar schienen. Mit der jetzigen Erfahrung durch die Projektarbeit würde eine andere Auswahl von Framework, hier würde auf jeden Fall ReactJS oder ReactNative deutlich mehr Sinn machen, treffen. Hier wäre aber wahrscheinlich die Einarbeitung für den Umfang der Projektarbeit anteilmäßig deutlich zu groß gewesen.

Auch mit der Wahl des Frameworks zur Zifferkategorisierung bin ich rückblickend nicht sehr zufrieden. Ich glaube, dass sowohl der Datensatz von MNIST als auch mein eigen erstellter Datensatz für die Aufgabenstellung gut genug ist, allerdings ist die von ML5 bereitgestellte Architektur des Netzes nicht ausreichend um eine gute Erkennung der Ziffern zu gewährleisten. Hier könnte man in Zukunft auf das ML5 unterliegende Framework Tensorflow.js zurückgreifen, und selbst ein deutlich passenderes Netz kreieren.

Eine weitere Verbesserung der Funktionalität ist sicher auch die weitere Bearbeitung des aufgenommenen Bildmaterials. Hier könnte eine Erhöhung des Kontrast helfen, oder die perspektivische Korrektur des Bildmaterials. (Hier ein Beispiel dieser Vorgehensweise:

<http://www.recompile.in/2019/11/image-perspective-correction-using.html>).

Auch die zur Erkennung vorgenommene Aufteilung des Bildes in 81 Unterbilder könnte dynamisch durch einen Bilderkennungsalgorithmus übernommen werden, der zum Beispiel vertikale und horizontale Linien erkennt und dadurch erkennt, wo im Bild die Ziffern und leeren Stellen des Sudokus liegen.

Alles in allem hätte man aus der Idee sicher deutlich mehr machen können, allerdings war diese Projektarbeit für zukünftige Projekte ein wirklicher Lerneffekt, gerade was die Einarbeitungszeit für umfangreichere Projekte angeht. Die im Projekt aufgegriffene Idee der Applikation will ich in Zukunft mit anderen Frameworks nochmal neu aufgreifen und zu einer Lösung kommen, mit der ich selbst hundertprozentig zufrieden bin.