

# Number Guesser

Projektdokumentation

des Studienganges Informatik  
an der Dualen Hochschule Baden-Württemberg Stuttgart

von

Markus Limbacher

28. Juni 2020

Matrikelnummer, Kurs

9123121, STG-TINF18C

Fach

Microservices mit Node.js

Dozent

Dr. Ingolf Buttig

# Inhaltsverzeichnis

<b>INHALTSVERZEICHNIS .....</b>	<b>II</b>
<b>ABKÜRZUNGEN.....</b>	<b>III</b>
<b>ABBILDUNGSVERZEICHNIS .....</b>	<b>III</b>
<b>1. EINLEITUNG.....</b>	<b>1</b>
1.1. PROBLEMSTELLUNG .....	1
1.2. VERKNÜPFUNG ZU VORLESUNGSINHALTEN .....	1
1.2.1. <i>Express.js</i> .....	1
1.2.2. <i>Tensorflow</i> .....	1
<b>2. UMSETZUNG .....</b>	<b>2</b>
2.1. LÖSUNGSANSATZ / VERWENDETE TECHNOLOGIEN .....	2
2.1.1. <i>GitHub:</i> .....	2
2.1.2. <i>Swagger:</i> .....	2
2.1.3. <i>Express:</i> .....	2
2.1.4. <i>Tensorflow:</i> .....	3
2.1.5. <i>HTML, CSS, JS:</i> .....	3
2.2. ARCHITEKTUR .....	4
2.2.1. <i>MNIST</i> .....	5
2.2.2. <i>Tensorflow ML-Modell</i> .....	5
2.2.3. <i>API</i> .....	8
2.2.4. <i>Canvas</i> .....	9
2.3. PROJEKTSTRUKTUR.....	10
<b>3. ZUSAMMENFASSUNG.....</b>	<b>11</b>
3.1. FAZIT .....	11
3.2. WEITERE DOKUMENTATION.....	12

## Abkürzungen

<b>API</b>	Application Programming Interface
<b>CNN</b>	Convolutional Neural Network
<b>CSS</b>	Cascading Style Sheets
<b>GPU</b>	Graphics Processing Unit
<b>HTML</b>	Hypertext Markup Language
<b>JS</b>	JavaScript
<b>ML</b>	Machine Learning
<b>MNIST</b>	Modified National Institute of Standards and Technology
<b>NPM</b>	Node Package Manager

## Abbildungsverzeichnis

ARCHITEKTUR-DIAGRAMM .....	4
BEISPIEL CNN, DAS VOM TENSORFLOW-TEAM ENTWICKELT WURDE .....	6
BILD WELCHES EIN BENUTZER GEZEICHNET HAT .....	7
ERGEBNIS DER ANALYSE VON ABBILDUNG 3 .....	7
CNN DAS IN DIESEM PROJEKT VERWENDET WURDE .....	7
DER ALGORITHMUS ZUM KONVERTIEREN EINES 140x140-PIXEL-BILD IN EIN 28x28-PIXEL-BILD .....	9
BEISPIEL FÜR DIE BILDER DIE IN DER MNIST-DATENBANK ENTHALTEN SIND.....	11

# 1. Einleitung

## 1.1. *Problemstellung*

Ziel dieses Projekts ist die Entwicklung einer Webapplikation, die ihren Benutzern die Möglichkeit bietet, mithilfe eines Zeichenfensters Bilder von handgeschriebenen Zahlen zu erstellen. Diese Bilder werden dann von der App analysiert mit dem Ziel, die auf dem Bild enthaltene Zahl zu erkennen. Dem Benutzer soll dann das Ergebnis der Analyse präsentiert werden, wobei für jede Zahl ein Prozentsatz angegeben wird, um zu zeigen, wie sicher die Anwendung ist, dass diese Zahl die richtige ist. Die Analyse der Zahlen soll mithilfe von Machine Learning (kurz: ML) umgesetzt werden.

## 1.2. *Verknüpfung zu Vorlesungsinhalten*

Bei der Entwicklung dieser Applikation kamen Technologien zum Einsatz, die in der Vorlesung „Microservices mit Node.js“ vorgestellt und diskutiert wurden.

### 1.2.1. Express.js

Express.js (oder kurz „Express“) ist ein Open-Source-Framework für Webanwendungen, das seinen Benutzern die Möglichkeit bietet, Webanwendungen und APIs zu entwickeln. Mit über 10 Millionen wöchentlichen Downloads ist es eines der beliebtesten und am weitesten verbreiteten NPM-Pakete.

### 1.2.2. Tensorflow

Tensorflow ist eine Open-Source-Bibliothek, die vom Google Brain Team entwickelt wurde. Tensorflow bietet viele mathematische Funktionen, die zur Lösung zahlreicher Probleme und Aufgaben verwendet werden können. Die Hauptanwendung ist die Entwicklung von Modellen, die maschinelles Lernen nutzen, wie zum Beispiel neuronale Netze.

## 2. Umsetzung

### 2.1. *Lösungsansatz / verwendete Technologien*

#### 2.1.1. GitHub:

GitHub wurde für die Versionsverwaltung verwendet. GitHub ist ein Tochterunternehmen von Microsoft das, seinen Benutzern ermöglicht, ihre Projekte online mithilfe des Versionsverwaltungssystems Git zu verwalten. Dadurch ist es beispielsweise möglich, dass mehrere Nutzer gleichzeitig an verschiedenen Versionen desselben Projekts arbeiten. Obwohl es bei dieser praktischen Arbeit keine Zusammenarbeit zwischen mehreren Entwicklern gab, erwies sich Git trotzdem als ein sehr wichtiges und nützliches Werkzeug zur besseren Verfolgung von Änderungen.

Das GitHub Repository, das für dieses Projekt angelegt wurde, ist unter folgendem Link zu erreichen: <https://github.com/wodyy666/Microservices>.

#### 2.1.2. Swagger:

Swagger wurde eingesetzt, um die für dieses Projekt erforderliche API zu entwerfen und zu dokumentieren. Swagger ist ein Open-Source Werkzeug, das zur Beschreibung von APIs gemäß dem OpenAPI-Standard verwendet wird.

Die Swagger-Dokumentation enthält Informationen zu allen Endpunkten, die von dem Backend bereitgestellt werden. Hier kann man also nachlesen, wie auf die verschiedenen Methoden des Backends zugegriffen werden kann und welche Daten übergeben werden müssen, damit das Backend die Anfrage bearbeiten kann. Zuletzt werden alle Antwortoptionen einschließlich ihres Inhalts festgehalten. Dies würde es einem Entwickler ermöglichen, ein Frontend an ein Backend anzubinden, ohne die Implementierungsdetails des Backends zu kennen. Stattdessen kann dieser Entwickler alle Informationen, die er benötigt, aus der Swagger-Dokumentation für die API entnehmen.

#### 2.1.3. Express:

Wie in der Einleitung erwähnt wurde in diesem Projekt das NPM-Paket Express verwendet. Mithilfe von Express wurde ein Webserver entwickelt. Dieser Webserver enthält sowohl das Frontend als auch das Backend dieser Anwendung.

#### 2.1.4. Tensorflow:

Zur Analyse der Bilder wurde Tensorflow verwendet. Es wäre auch möglich gewesen, einen der Azure Cognitive Services zu nutzen, die diese Funktionalität bereits bieten, aber das Ziel des Projekts war es ein eigenes Modell zu entwickeln, zu trainieren und dann zu verwenden.

Eine wichtige Entscheidung für dieses Projekt war die Wahl des richtigen Tensorflow-Moduls, denn obwohl es im NPM-Repository ein Modul namens Tensorflow gibt, ist dieses Modul nicht die offizielle Tensorflow-Bibliothek für Node.js und nur mit Linux kompatibel. Stattdessen wurde in diesem Projekt das „@tensorflow/tfjs-node“-Paket verwendet, das mit Linux, Windows und Mac kompatibel ist.

Von diesem Paket gäbe es noch eine GPU-Version, die die Grafikkarte des jeweiligen Rechners verwendet. Diese Version wurde nicht verwendet, da der für die Entwicklung verwendete Laptop nur über eine integrierte Grafikkarte verfügte. In Zukunft wäre es jedoch möglich, das derzeit verwendete Paket durch seine GPU-Version zu ersetzen, um eine bessere Leistung zu erzielen. Die zusätzliche Leistung einer Grafikkarte sorgt jedoch nur dafür, dass das Modell schneller trainiert werden kann und die Analyse eines Bildes schneller abgeschlossen ist. Eine Grafikkarte hätte keinen Einfluss auf die Qualität der Ergebnisse.

#### 2.1.5. HTML, CSS, JS:

Das Frontend wurde mit HTML, CSS und JS entwickelt. Es wurden keine Frontend-Frameworks verwendet, da der Umfang der zu entwickelnden Website nicht groß genug war, um den Einsatz eines solchen Frameworks lohnenswert zu machen.

Drei Funktionalitäten wurden in dem Frontend implementiert. Zum einen wird hier das Bild von dem Benutzer erstellt, damit dieses später analysiert werden kann. Zu diesem Zweck wurde ein Canvas-Element in die Website integriert, in dem der Benutzer die Zahlen mithilfe seiner Maus zeichnen kann.

Zum anderen ist das Frontend in der Lage, das vom Benutzer gezeichnete Bild in ein Format zu konvertieren, das vom Backend verarbeitet werden kann. Ein Teil der Bildverarbeitung und -analyse wird also bereits auf der Client-Seite erledigt. Nach dieser Verarbeitung übernimmt das Frontend auch die Kommunikation mit dem Backend über die zuvor erwähnte API, die über http-Aufrufe angesprochen wird.

Zuletzt dient das Frontend auch zur Anzeige der Analyseergebnisse mithilfe mehrerer Balken, die die erkannte Zahl darstellen.

Das Frontend wurde mit dem Firefox-Browser entwickelt und getestet. Aus diesem Grund kann die Benutzererfahrung bei der Verwendung von Firefox besser sein. Die Anwendung sollte jedoch auf allen Browsern funktionieren.

## 2.2. Architektur

Die Architektur dieser Anwendung lässt sich grob in Frontend und Backend unterteilen (siehe Abbildung 1). Das Frontend enthält alles, was auf der Client-Seite, d.h. auf dem Rechner eines Benutzers stattfindet, während das Backend alles enthält, was auf dem Server geschieht.

Das Frontend enthält die Funktionalität zum Zeichnen einer Zahl und zur Ausgabe des Ergebnisses. Dies sind die Berührungspunkte der Anwendung und dem Benutzer. Darüber hinaus findet im Frontend bereits ein Teil Bildverarbeitung statt, da das gezeichnete Bild in ein brauchbares Format konvertiert werden muss. Als Format wurde ein eindimensionales Float-Array mit 784 Einträgen gewählt. Ein Eintrag repräsentiert ein Pixel des 28 Pixel breiten und 28 Pixel hohen Bildes. Da das Bild keine unterschiedlichen Farben hat, ist es nicht notwendig, Farbinformationen zu speichern. Das Frontend kommuniziert mit dem Backend über http-Anfragen.

Alle eingehenden Anfragen an das Backend werden vom Express-Server empfangen und entsprechend an das ML-Modell weitergeleitet. Dieses Modell verarbeitet dann das Array, das das Bild enthält, und gibt als Ergebnis ein Array mit 10 Einträgen zurück. Jeder Eintrag steht für eine Ziffer.

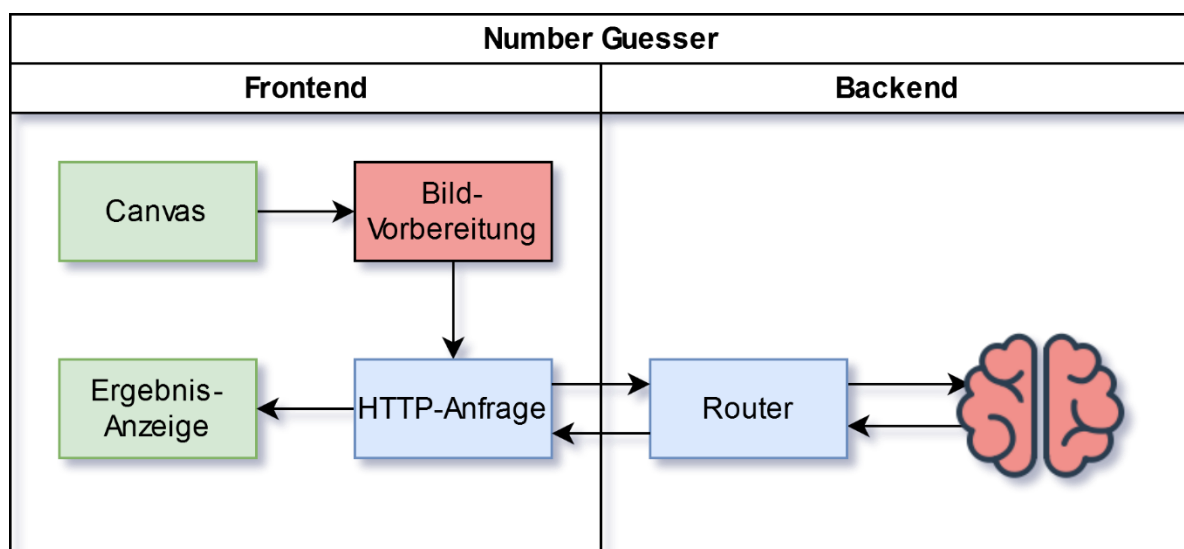


Abbildung 1: Architektur-Diagramm

### 2.2.1. MNIST

Für die Entwicklung eines ML-Modells werden große Datenmengen benötigt. Diese Daten werden von dem Modell für das Training verwendet, mit dem Ziel, dass das Modell in der Lage ist, Muster in den Daten zu erkennen, die dann in der Zukunft auf neue Daten angewendet werden können. In diesem Projekt wurde die MNIST-Datenbank für das Training des Modells verwendet. Diese Datenbank enthält 70.000 Schwarzweißbilder von handschriftlichen Zahlen.

### 2.2.2. Tensorflow ML-Modell

Dieses online verfügbare Beispiel wurde als Vorlage für das ML-Modell verwendet:

<https://github.com/tensorflow/tfjs-examples/tree/master/mnist-node>

Dieses Modell ist ein convolutional neural network (kurz: CNN). Diese Art von Modell wird typischerweise zur Bildverarbeitung verwendet. Ein CNN kann aus drei möglichen Arten von Schichten aufgebaut werden.

Es gibt die Convolutional Layers, die dem Modell auch seinen Namen geben. Bei diesem Schichttyp wird das Bild mithilfe von Filtern analysiert, die auf das Bild angewendet werden. Je mehr Filter pro Schicht angewendet werden, desto mehr Merkmale können im Bild erkannt werden.

Dann gibt es auch noch die Pooling Layers. Diese Schichten sorgen dafür, dass überflüssige Informationen verworfen werden, indem immer nur die stärksten Neuronen in einem Bereich an die nächste Schicht weitergegeben werden. Die Größe dieses Bereichs ist typischerweise 2 breit und 2 hoch. Das bedeutet, dass diese Schicht sich immer vier benachbarte Neuronen aufnimmt und das stärkste, d.h. dasjenige mit dem höchsten Wert, an die nächste Schicht weitergibt.

Schließlich gibt es die Fully Connected Layers oder auch Dense Layers. Diese Schichten bestehen aus mehreren Neuronen, die mit jedem Neuron in der vorhergehenden Schicht verbunden sind. Diese Art von Schichten findet man typischerweise als letzte Schicht in einem Modell. Die Anzahl der Neuronen in der letzten Schicht entspricht auch der Anzahl der möglichen Ergebnisse der Analyse.



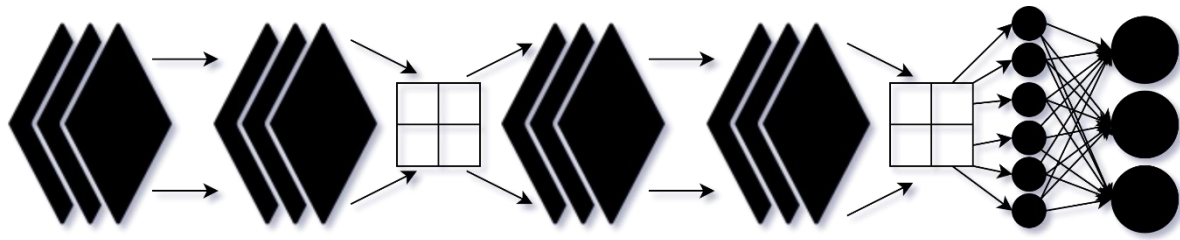


Abbildung 2: Beispiel CNN, das vom Tensorflow-Team entwickelt wurde

Das Beispielmodell, besteht aus 8 Schichten. Abbildung 2 zeigt eine vereinfachte Darstellung der einzelnen Schichten. Das Modell ist sehr groß und enthält 594.922 Parameter, die trainiert werden müssen. Tabelle 1 zeigt den genauen Aufbau der einzelnen verwendeten Schichten.

Typ	Eigenschaften	Parameter
Convolutional Layer	32 Filter	320
Convolutional Layer	32 Filter	9248
Pooling Layer	2x2 Cluster	n/a
Convolutional Layer	64 Filter	18496
Convolutional Layer	64 Filter	36928
Pooling Layer	2x2 Cluster	n/a
Dense Layer	512 Neuronen	524800
Dense Layer	10 Neuronen	5130

Tabelle 1: Schichten des CNN, das vom Tensorflow-Team entwickelt wurde

Aufgrund der Größe des Beispielmodells dauerte es lange, das Modell zu trainieren, und die Analyse eines Bildes erwies sich als sehr ressourcenintensiv. Beim Testen des Modells stellte sich auch heraus, dass es sich manchmal auf sehr kleine Merkmale konzentrierte. Dies war darauf zurückzuführen, dass das Modell aufgrund seiner vielen Parameter in der Lage war, sich viele Merkmale in den Trainingsdaten zu merken. Infolgedessen war das Modell etwas zu stark an die Trainingsdaten angepasst und versuchte, selbst aus kleinen Merkmalen Schlussfolgerungen zu ziehen. Die Abbildung 3 und 4 zeigen ein Beispiel für ein solches Verhalten. Obwohl die Zeichnung in Abbildung 3 kaum Informationen enthält und keine Zahl erkennbar ist, geht das Modell davon aus, dass es sich bei dem Bild um die Zahl „4“ handeln muss.

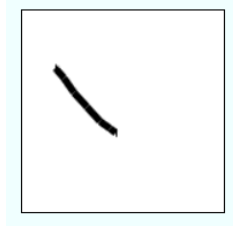


Abbildung 3: Bild welches ein Benutzer gezeichnet hat

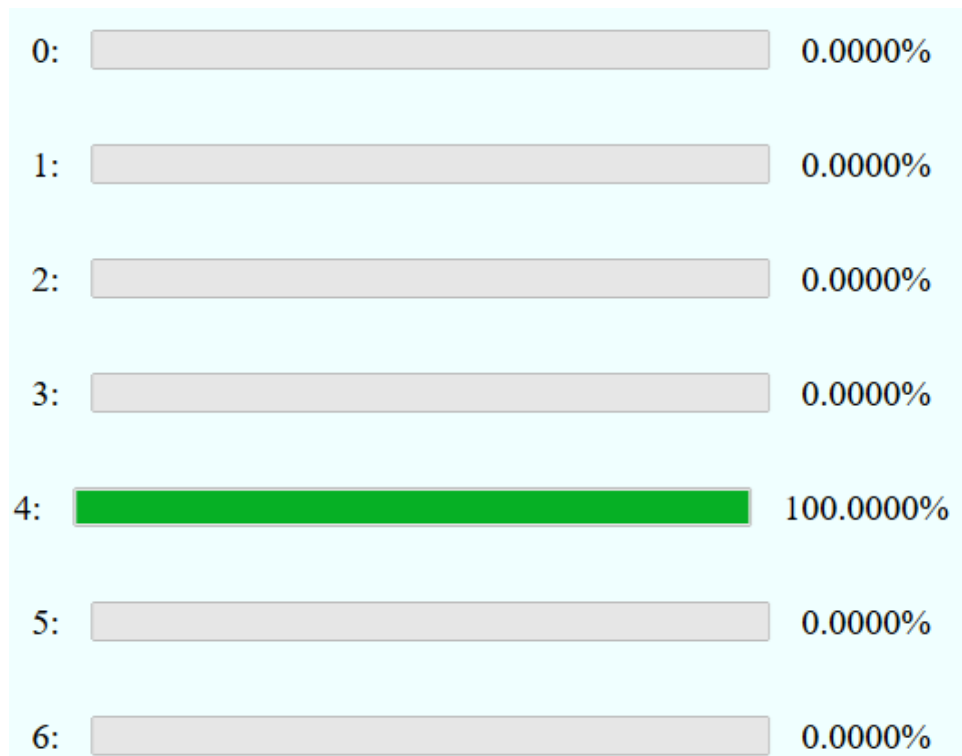


Abbildung 4: Ergebnis der Analyse von Abbildung 3

Aus diesem Grund wurde das Beispielmmodell für dieses Projekt leicht modifiziert. Das Modell besteht jetzt nur noch aus 7 Schichten anstelle der ursprünglichen 8 Schichten, die vorher verwendet wurden (siehe Abbildung 5).

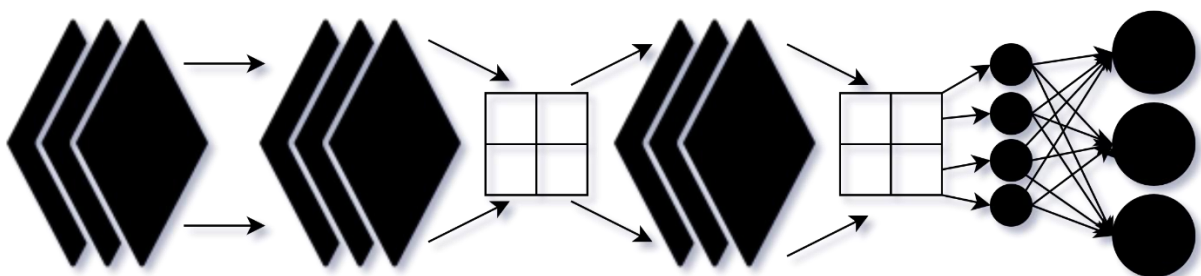


Abbildung 5: CNN das in diesem Projekt verwendet wurde

Die Größe dieser Schichten wurde ebenfalls geändert, sodass das Modell jetzt nur noch aus 33.082 Parametern besteht. Dies entspricht nur noch etwa 5,6% der ursprünglichen Größe. Details zu den neuen Schichten sind in Tabelle 2 zu sehen.

<i>Typ</i>	<i>Eigenschaften</i>	<i>Parameter</i>
<i>Convolutional Layer</i>	16 Filter	160
<i>Convolutional Layer</i>	16 Filter	2320
<i>Pooling Layer</i>	2x2 Cluster	n/a
<i>Convolutional Layer</i>	32 Filter	4640
<i>Pooling Layer</i>	2x2 Cluster	n/a
<i>Dense Layer</i>	32 Neuronen	25632
<i>Dense Layer</i>	10 Neuronen	330

Tabelle 2: Schichten des CNN, das in diesem Projekt verwendet wurde

Aufgrund der reduzierten Größe nahm die Genauigkeit des Modells leider etwas ab. Beim Testen des neuen Modells war dies jedoch nicht so offensichtlich, und das Modell war immer noch in der Lage, zwischen den verschiedenen Zahlen zu unterscheiden.

Der Vorteil dieser Änderung war, dass das Modell nun nicht mehr so viele Ressourcen für die Analyse eines Bildes benötigt. Darüber hinaus wurde die Zeit, die für das Training des Modells benötigt wurde, stark reduziert. Dies lag zum einen daran, dass die Trainingsbilder aufgrund der geringeren Größe des Modells schneller verarbeitet werden konnten. Zum anderen lag es aber auch daran, dass jetzt nur noch 10 Trainingsläufe (vorher waren es 50) notwendig sind, um das Modell ausreichend zu trainieren. Dies ist darauf zurückzuführen, dass weniger Parameter trainiert werden mussten.

### 2.2.3. API

Die Spezifikation der API kann entweder der Datei swagger.json, die sich in diesem Ordner befindet, entnommen werden oder sie kann online unter folgendem Link eingesehen werden.

<https://app.swaggerhub.com/apis/woddy666/Microservices-NumberGuesser/1.0.0>

## 2.2.4. Canvas

Da die in der MNIST-Datenbank enthaltenen Bilder nur 28 Pixel breit und 28 Pixel hoch sind, wäre das Fenster zum Zeichnen sehr klein gewesen. Um dieses Problem zu lösen, wurde dieses Element mittels CSS vergrößert. Dieser Ansatz, ist jedoch begrenzt nützlich, da bei einem zu großen Vergrößerungsfaktor das vergrößerte Element nun unscharf ist. Daher wurde das Element nur mit einem Vergrößerungsfaktor von 1.5 vergrößert.

Die andere Maßnahme ist die Vergrößerung des Bildes von 28x28 Pixel auf das 5-fache, also 140x140 Pixel. Da das ML-Modell jedoch eine Eingabe von 28x28 Pixel erwartet, muss das Bild vor dem Senden wieder verkleinert werden. Abbildung 6 zeigt den Algorithmus dafür.

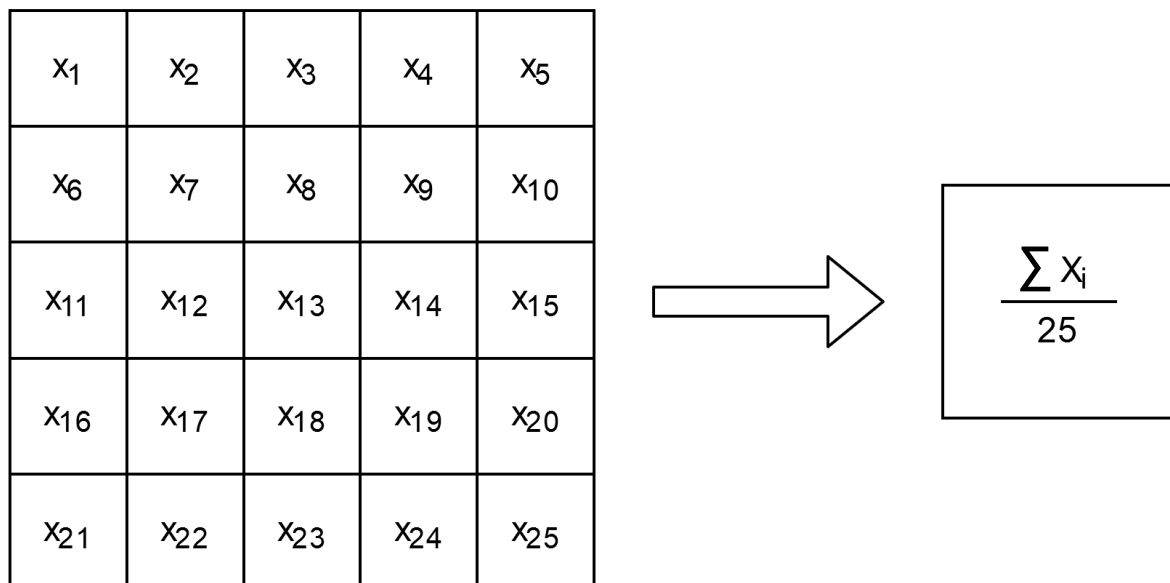


Abbildung 6: Der Algorithmus zum Konvertieren eines 140x140-Pixel-Bild in ein 28x28-Pixel-Bild

Vor dem Senden eines Bildes an das Backend wird das 140x140 Pixel große Bild in 784 Flächen der Größe 5x5 aufgeteilt. Für jede Fläche wird der Durchschnitt aller Pixel berechnet. Das Ergebnis ist der Wert des Pixels des 28x28-Pixel-Bildes an dieser Position.

## 2.3. Projektstruktur

Tabelle 3 liefert einen Überblick über die Strukturierung dieses Projekts.

<i>Name</i>	<i>Beschreibung</i>
<i>/config</i>	In diesem Ordner befinden sich alle Konfigurationsdateien, die verwendet werden können, um Projekteinstellungen zu verändern
<i>/controllers</i>	In diesem Ordner befinden sich alle Router für die API, die sich darum kümmern die Anfragen an die unterschiedlichen Routen an die jeweilige Funktion weiterzuleiten
<i>/Doku</i>	In diesem Ordner befinden sich alle Dokumentationsdateien
<i>/helpers</i>	In diesem Ordner befinden sich die Dateien, die sich um die Analyse der Bilder kümmern
<i>/MNISTDataset</i>	In diesem Ordner befinden sich die Bilder, die zum Trainieren verwendet wurden
<i>/model</i>	In diesem Ordner befindet sich das fertig trainierte ML Modell, das die Bilder analysiert
<i>/public</i>	In diesem Ordner befinden sich alle statischen Dateien, die der Benutzer aufrufen kann
<i>.gitignore</i>	Diese Datei enthält alle Dateinamen, die nicht vom Versionsverwaltungssystem Git berücksichtigt werden sollen
<i>app.js</i>	Diese Datei ist diejenige, die die Anwendung startet
<i>LICENSE</i>	Die Lizenz dieses Projekts
<i>package.json</i>	Diese Datei enthält Informationen zu allen Abhängigkeiten von diesem Projekt
<i>README.md</i>	Diese Datei enthält Informationen zu Installation und Inbetriebnahme der Anwendung

## 3. Zusammenfassung

### 3.1. *Fazit*

Das Ziel des Projekts wurde erreicht, und es wurde eine funktionierende Anwendung entwickelt. Viele Probleme konnten behoben werden, um das Erlebnis für die Benutzer dieser Anwendung so gut wie möglich zu machen.

Einige Probleme bleiben jedoch bestehen. Eine ML-Modell ist nur so gut wie die Daten, mit denen es trainiert wurde. Die MNIST-Datenbank ist ein sehr nützliches Werkzeug, das einem Entwickler ermöglicht, schnell an Trainingsdaten zu kommen. Es gibt jedoch zwei Probleme an den MNIST-Bildern.

Das eine Problem ist, dass die Zahlen auf den Bildern immer zentriert sind und genau die Größe des Bildes haben. Das Ergebnis ist, dass, wenn ein Benutzer unserer Anwendung seine Zahl nicht groß und zentriert, sondern klein in eine Ecke zeichnet, das ML-Modell nicht mehr in der Lage ist, zu erkennen, um welche Zahl es sich handelt.

Das andere Problem ist, dass die Bilder nur Zahlen enthalten, welche von Schülern aus Amerika gezeichnet wurden. Abbildung 7 zeigt einige Beispielbilder aus der MNIST-Datenbank. Anhand dieser Abbildung ist erkennbar, dass zum Beispiel die Bilder einer 7 eher wie eine 1 hier in Deutschland aussehen.



Abbildung 7: Beispiel für die Bilder die in der MNIST-Datenbank enthalten sind

Quelle: [https://people.duke.edu/~ccc14/sta-663-2017/21\\_TensorFlow\\_Edward.html](https://people.duke.edu/~ccc14/sta-663-2017/21_TensorFlow_Edward.html)

### 3.2. *Weitere Dokumentation*

Wie oben erwähnt gibt es noch folgende zusätzliche Dokumentation zu diesem Projekt:

- Eine README-Datei mit Installationshinweisen
- Kommentare im Source-Code
- Eine Swagger-Datei mit der OpenAPI Spezifikation der API -> Dieses Dokument ist auch online unter diesem Link verfügbar:

<https://app.swaggerhub.com/apis/wodyy666/Microservices-NumberGuesser/1.0.0>

- Ein GitHub Repository mit allen Projektdateien:

<https://github.com/wodyy666/Microservices>