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

Abkürzungen

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

Abbildungsverzeichnis

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

2.1.1. verwendete Technologien

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 bessern Verfolgung von Änderungen.

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

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.

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.

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.

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.s

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.

2.1.2. 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.

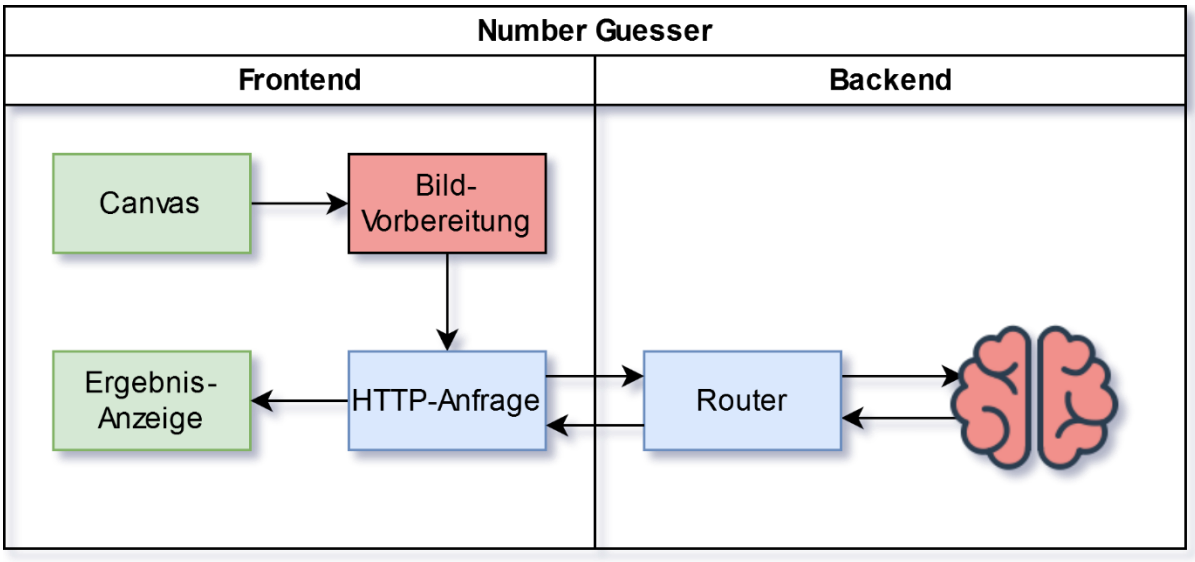

Abbildung 1: Architektur-Diagramm

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.

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 überflüssie 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 vorhergenden 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.

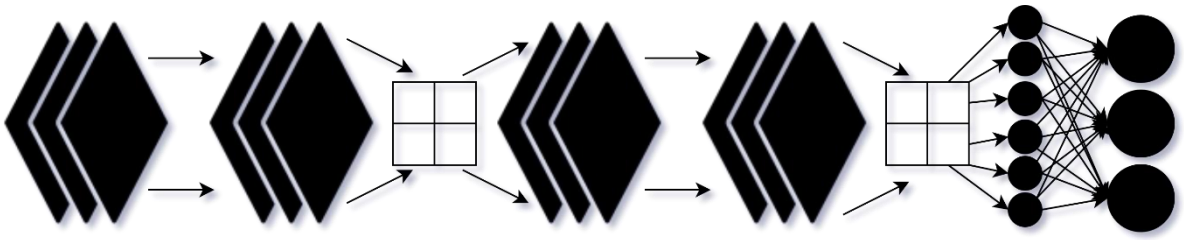

Abbildung 2: Beispiel CNN, das vom Tensorflow-Team entwickelt wurde

Das Beispielmodell, besteht aus 8 Schichten. Abbildung 2 zeigt eine vereinfachte Dartstellung der einzelnen Schichten. Das Modell ist sehr groß und beinhaltet 594.922 Parameter, die trainiert werden müssen. Die 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

Durch die Größe von dem Beispielmodell dauerte es sehr lange, bis das Modell trainiert wurde und die Analyse eines Bildes erwies sich als sehr Ressourcenintensiv. Beim Testen des Modells stellte sich außerdem heraus, das es sich teilweise auf sehr kleine Merkmale konzentierte. Dies lag daran, dass das Modell sich aufgrund seiner vielen Parameter sehr viele Merkmale in den Trainingsdaten merken konnte. Dadurch war das Modell an die Trainingsdaten ein wenig überangepasst und versuchte bereits aus kleinen Merkmalen Schlüsse zu ziehen. Die Abbildung 3 und 4 zeigen ein Beispiel für ein solches Verhalten. Obwohl die Zeichnung in Abbildung 3 kaum Informationen enthält und eigentlich keine Zahl zu erkennen ist, geht das Modell davon aus, das es sich bei dem Bild um die Zahl 4 handeln muss.

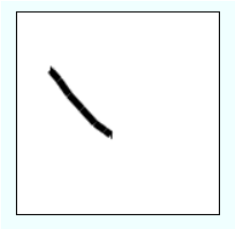

Abbildung 3: Bild welches ein Benutzer gezeichnet hat

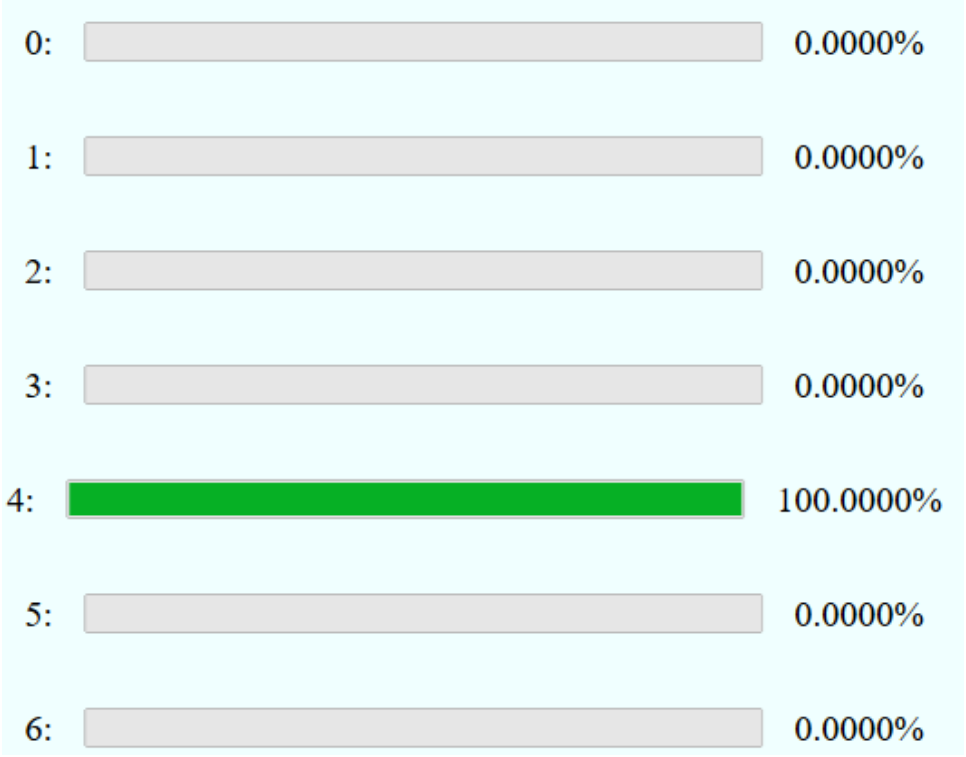

Abbildung 4: Ergebnis der Analyse von Abbildung 3

Aus diesem Grund wurde das Beispielmodell für dieses Projekt ein wenig verändert. Das Modell besteht nun nur noch aus sieben Schichten, anstatt der ursprünglichen 8 Schichten, die zuvor verwendet wurden (siehe Abbildung 5).

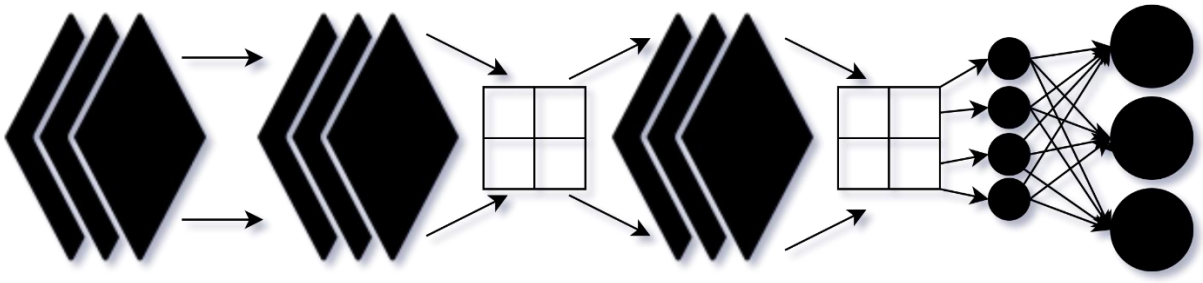

Abbildung 5: CNN das in diesem Projekt verwendet wurde

Die Größe dieser Schichten wurde auch verändert, sodass das Modell jetzt nur noch aus 33.082 Parametern besteht. Das sind nur noch etwa 5,6% der ursprünglichen Größe. Details zu den neuen Schichten können aus Tabelle 2 entnommen werden.

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

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

Durch die verringerte Größe lies die Genauigkeit des Modells leider ein wenig nach. Während dem Testen des neuen Modells viel das jedoch nicht so sehr auf und das Modell war immer noch in der Lage gut zwischen den unterschiedlichen Zahlen unterscheiden zu können.

Die Vorteile dieser Änderung waren, dass das Modell jetzt nicht mehr so viele Ressourcen benötigt, wenn es ein Bild analysiert. Außerdem nahm die Zeit, die notwendig war, um das Modell zu trainieren stark ab. Dies lag daran, dass zum einen aufgrund der verringerten Größe die Trainingsbilder schneller verarbeitet werden konnten. Zum anderen lag es jedoch auch daran, dass jetzt nur noch 10 Trainingsläufe (zuvor waren es 50) notwendig sind um das Modell ausreichend zu trainieren. Dies liegt daran, dass weniger Parameter trainiert werden mussten.

API specs

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

2.1.3. Projektstruktur

https://www.terlici.com/2014/08/25/best-practices-express-structure.html

2.2. Herausforderungen

Data ingest

Canvas resize and downsampling

→client side

Overfitting:

594922 → 33082 (Zeiteinsparung auch noch erwähnen?)

3. Zusammenfassung

3.1. Fazit

MNIST bias → Probeleme mit dataset: sehr klein; zentriert; schreibweise 1 (erkennt ne 7) (einfach beispiel MNIST bild einfügen)

Problem overfitting

Train endpoint besser nutzen

Draw update speed

3.2. Ausblick

3.3. Weitere Dokumentation

-swagger

.git

Readme

code