



UNIVERSITÄT
LEIPZIG

UNIVERSITÄT LEIPZIG

ABTEILUNG DATENBANKEN

BIG DATA - PRAKTIKUM

Konzeptioneller Entwurf zum
Thema
*"Traffic Analysis with Deep
Learning"*

Ali Al-Ali und Jeremy Puchta

26. Mai 2019

Inhaltsverzeichnis

Inhaltsverzeichnis	I
Abbildungsverzeichnis	II
1 Einleitung	1
2 Architektur	2
3 Ablauf	3
Literaturverzeichnis	III

Abbildungsverzeichnis

1	Architektur der visionierten Webanwendung	2
---	---	---

1 Einleitung

Die vorliegende Ausarbeitung dokumentiert die Konzeption des Projekts *Traffic Analysis with Deep Learning*. Innerhalb der Ausarbeitung wird insbesondere der Ablauf und die Architektur der visionierten Software konzeptionell dargestellt. Dabei wird weiterhin auf die verfolgte Zielstellung und die Aufgaben eingegangen, die zur Erfüllung des Ziels gelöst werden müssen.

Das Ziel des Projekts ist die Erstellung einer Webanwendung, welche Daten über das Verkehrsaufkommen am Leipziger Ring sammelt, statistisch analysiert und visualisiert. Als Datengrundlage werden die unter <https://www.l.de/webcam.html> öffentlich bereitgestellten Webcambilder des Leipziger Rings verwendet. Diese werden regelmäßig mithilfe eines Webscraping-Algorithmus abgerufen und dem Datensatz hinzugefügt (**T1**).

Innerhalb der gesammelten Bilddateien werden die für das aktuelle Verkehrsaufkommen relevanten Objekte erfasst. Dazu wird ein für den Anwendungsfall der Objekterkennung vortrainiertes neuronales Netz implementiert. Der Leistungsstand des neuronalen Netzes wird im ersten Schritt untersucht, bevor im zweiten Schritt eine Verbesserung der Leistungsfähigkeit vorgenommen wird (**T2**). Anschließend erfolgt die Transformation der gesammelten Bilddaten in Zeitreihendaten (**T3**).

Abschließend wird der Zeitreihendatenbestand für die nachfolgende Analyse und Visualisierung innerhalb der Webanwendung aufbereitet. Die Webanwendung stellt ein Dashboard zur Verfügung, welches die aufbereiteten Verkehrsdaten grafisch visualisiert (**T4**).

Die aufbereiteten Verkehrsdaten geben Informationen über die gegenwärtige Verkehrslage am Leipziger Ring preis und ermöglichen es Rückschlüsse auf mögliche Über- oder Unterlastungen des betrachteten Verkehrsbereich zu ziehen. Aus diesen Analysen ist es möglich Handlungen für eine Verbesserung der Verkehrssituation abzuleiten. Daher sind die Ergebnisse der Ausarbeitung von Interesse für politische Amtsträger, Studenten verschiedener Fachrichtungen sowie interessierte Bürger der Stadt Leipzig.

Im Folgenden werden zunächst die zur Realisierung der visionierten Webanwendung ausgewählten Technologien kurz vorgestellt. Im Anschluss werden die Architektur der Anwendung und der geplante Ablauf des Projekts beschrieben.

2 Architektur

In diesem Kapitel erfolgt die Präsentation der Architektur der visionierten Webanwendung. Das UML-Komponentendiagramm in Abbildung 1 visualisiert die Bestandteile der Anwendung.

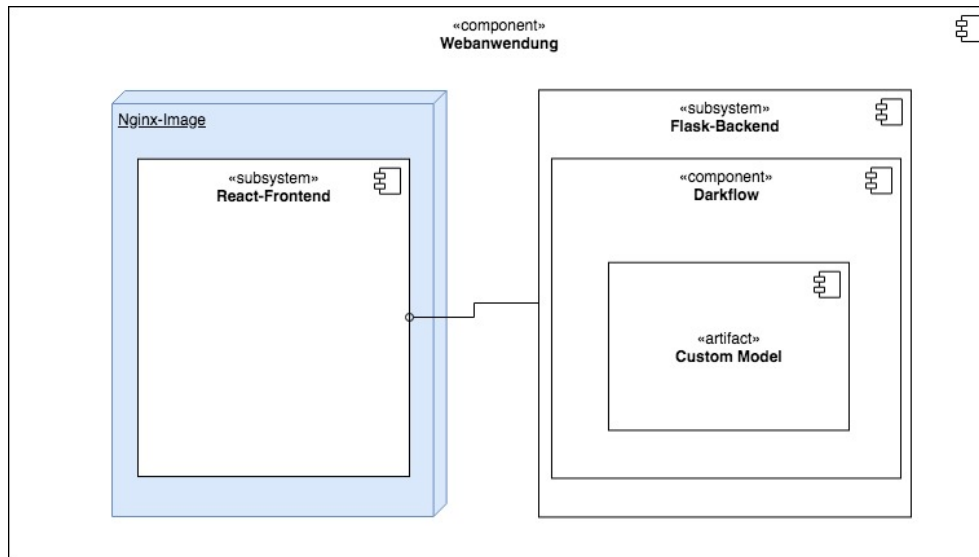


Abbildung 1: Architektur der visionierten Webanwendung

Zur Identifizierung von relevanten Objekten wird die *YOLO*-Architektur genutzt. Bei *YOLO* handelt es sich um ein neuronales Netz, dass zum detektieren sowie zum lokalisieren von Objekten entworfen wurde. Die in diesem Projekt verwendete *YOLO*-Implementierung heißt *Darkflow*.

Die Webanwendung basiert auf einer Client-Server-Architektur. Auf Serverseite wird das leichtgewichtige Python-Framework *Flask* genutzt. Flask wurde mit dem Ziel entwickelt seinen Nutzern einen einfachen und schnellen Einstieg zu ermöglichen, jedoch mit der Möglichkeit bis hin zu komplexen Anwendungen zu skalieren (vgl. siehe [1]).

Im Frontend kommt die von *Facebook* entwickelte JavaScript-Library *React* zum Einsatz. Wie andere komponentenbasierte Frontend-Frameworks stellt auch *React* einen komfortablen und schnellen Weg zur Erstellung von UI-Komponenten bereit (vgl. siehe [2]). Die im Frontend erstellten Grafiken werden mithilfe von *p5.js* erstellt. Dabei handelt es sich wiederum um eine JavaScript-Library für die Erstellung von grafischen und interaktiven Inhalten (vgl. siehe [3]).

Sämtliche Komponenten des Systems werden mithilfe der Container-virtualisierungs-Technologie Docker deployed. Docker ermöglicht die Bereitstellung und den Betrieb von Linux-Containern. Innerhalb dieser Container können Images deployed werden, die unter anderem auf *Dockerhub* (<https://hub.docker.com/>) bereitgestellt werden (vgl. siehe [4]). Im

Rahmen des Projekts wird beispielsweise das offizielle Image des *Nginx*-Webservers verwendet, um den Production-Build des Frontends bereitzustellen.

3 Ablauf

Dieses Kapitel richtet den Blick auf den Ablauf des Systems. Dabei ist der Ablauf unterteilt in zwei unabhängige Phasen:

1. Trainings- und Testingphase des neuronalen Netzes
2. Betriebsphase der Webanwendung

Zunächst wird der Ablauf der ersten Phase vorgestellt. Diese befasst sich mit dem Training und Testing des neuronalen Netzes. Das verwendete neuronale Netz basiert auf der YOLO-Architektur, welche in [5] dokumentiert ist. Weiterhin wird ein vortrainiertes Model zur Objekterkennung verwendet. Dieses bietet bereits die Möglichkeit 80 Klassen zu erkennen und zu klassifizieren. Aufbauend auf diesem Fundament, wird das Model entsprechend der im Rahmen des Projekts aversierten Domäne optimiert. Dabei spricht man von *Training*.

Im Rahmen der Trainingsphase wird der zusammengestellte und mit Labels versehene Datensatz verwendet, um das neuronale Netz zu trainieren. Hierbei werden die Bilder iterativ durch das neuronale Netz via *Forward Pass*-Prozess geschickt. Dabei detektiert und klassifiziert das neuronale Netz die im Bild vorliegenden Objekte und vergleicht anschließend die Resultate mit den im Input-Datensatz manuell gelabelten Objekten. Im Fehlerfall passt das neuronale Netz die Kantengewichte mit Hilfe der *Back Propagation*-Technik an. Zur Validierung des neuronalen Netzes werden dem Netz unbekannte Bilder als Input bereitgestellt. Damit wird überprüft, ob das Netz in der Lage ist auf unbekannte Input-Daten zu reagieren und ob beispielsweise eine Überanpassung existiert.

Das Ergebnis der Trainingsphase ist ein auf den Projektanwendungsfall spezialisiertes Model. Dieses Model wird im Folgenden in das Backend der Webanwendung integriert. Zur Laufzeit der Webanwendung wird stündlich das aktuell veröffentlichte Webcambild (siehe <https://www.1.de/webcam.html>) gecrawled. Dieses wird vom Flask-Backend durch das neuronale Netz geschickt. Das Resultat stellt die enthaltenen Objekte in einer JSON-Datei dar. Die enthaltenen relevanten Werte werden aus der JSON-Datei extrahiert und in eine Datenbank gespeichert. Der gesammelte Datenbestand wird im Anschluss aufbereitet, harmonisiert und analysiert, bevor sie an das Frontend geschickt werden. Im Frontend werden die aus den Daten resultierenden Statistiken nutzerfreundlich visualisiert.

Literaturverzeichnis

- [1] Flask. <https://palletsprojects.com/p/flask/>. (abgerufen am: 20.05.2019).
- [2] React Homepage. <https://reactjs.org/>. (abgerufen am: 20.05.2019).
- [3] p5.js Homepage. <https://p5js.org/>. (abgerufen am: 20.05.2019).
- [4] Was ist Docker? <https://www.redhat.com/de/topics/containers/what-is-docker>. (abgerufen am: 20.05.2019).
- [5] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015.