Hochschule für Technik und Wirtschaft Berlin
University of Applied Sciences

Staatsbibliothek zu Berlin
Preußischer Kulturbesitz

ABSCHLUSSBERICHT

INTERNATIONALER STUDIENGANG MEDIENINFORMATIK

Masterprojekt M6: Chasing Unicorns and Vampires in a Library

Eingereicht am: [redacted]

Eingereicht von: [redacted]

Projektbetreuer: Dr.-Ing. David Zellhöfer

INHALTSVERZEICHNIS

ZIELSETZUNG

Ziel des Projekts **Chasing Unicorns and Vampires in a Library** war die Entwicklung einer webbasierten Applikation, die es erlaubt, den digitalen Bestand der Staatsbibliothek aus einer neuen Perspektive zu erkunden. Im Fokus stand dabei nicht länger die Komponente Text, sondern das Bild. Der Nutzer sollte primär Bildkategorien entdecken und miteinander vergleichen können. Ferner war ihm die Möglichkeit zu bieten, inhaltliche Zusammenhänge zwischen Büchern zu erkennen und zu verstehen. Zu diesem Zweck galt es, die Bilder selbst sowie die Metadaten der Bücher für die spätere Visualisierung entsprechend aufzubereiten. Dafür war in erster Linie eine Klassifizierung der Bilder von großer Bedeutung. Um dem Benutzer in der Anwendung eine zusätzliche Explorationsmöglichkeit bereitzustellen, sollten außerdem für alle Illustrationen ähnliche Elemente ermittelt werden. Darüber hinaus waren diverse Visualisierungsarten für eine sinnvolle und zielgerichtete Darstellung der Metadaten zu evaluieren.

DATENFLÜSSE

Die Staatsbibliothek zu Berlin stellt für sämtliche Digitalisate ihrer Sammlung entsprechende METS-MODS-Dateien zum Download bereit. Diese bilden die Grundlage für zwei parallele Arbeitsschritte zur der Datengewinnung. Zum einen dienen sie dem Auslesen der Metadaten aller Bücher, welche später in der Datenbank gespeichert werden. Zum anderen werden auf diese Weise die TIF-Dateien darin ausgewiesener Illustrationen heruntergeladen. Alle TIF-Dateien werden im gleichen Schritt in JPEGs umgewandelt, um diese für die spätere Verwendung zur Verfügung zu stellen und die Datenmenge für weitere Verarbeitungsschritte zu reduzieren. Da die Ermittlung der Illustrationen durch eine OCR stattfand, enthielt die Datenmenge nicht nur verwendbare Bilder, sodass es zuerst einer Bereinigung bedurfte. Dabei wurden beispielsweise leere Seiten, Stempel oder Farbkarten aussortiert. Im weiteren Verlauf erfolgte die Klassifizierung der Bilder, um in der Anwendung zukünftig die Filterung nach Kategorien zu ermöglichen. Neben der Berechnung der dominanten Farben wurde außerdem eine Feature-Extraktion durchgeführt. Unter Verwendung der resultierenden Featurevektoren wurden zusätzlich nach dem “Bag of Visual Words”-Prinzip die jeweils fünf ähnlichsten Bilder ermittelt. Eine Datenbank speichert die Ergebnisse aller Verarbeitungsschritte und stellt sie der Anwendung zur Verfügung. Möchte der Nutzer nun z.B. die Kategorie “Pferd” betrachten, sendet die Applikation eine entsprechende REST-Anfrage an einen Server, der wiederum mit der Datenbank kommuniziert.

Zur späteren Darstellung der Daten auf einer geographischen Karte wurden die Koordinaten der zuvor aus den METS-MODS Dateien entnommenen Orten von Open Street Map abgerufen und ebenfalls in der Datenbank gespeichert.

ARCHITEKTUR

Die Applikation, welche vom Endbenutzer verwendet wird, lässt sich in die zwei Bereiche Backend (Server) und FrontEnd (Applikation) unterteilen. Der Server wird in diesem Projekt für zwei Aufgaben verwendet: Zum einen liefert er dem Frontend alle benötigten Daten, zum anderen stellt er dieses dem Benutzer zur Verfügung.

Das Backend

Das Backend besteht aus einem Python Flask-Server, welcher, wie oben kurz beschrieben. sowohl das Frontend bereitstellt als auch auf HTTP-REST-Calls antwortet und damit Daten an das Frontend übergibt. Sein Aufbau besteht aus dem Serverbereich selbst und zusätzlichen Python Skripten, die sich separat um die Datenbeschaffung sowie -aufbereitung kümmern. Diese sind jeweils in den Ordnern *Helpers* und *Server/app/* untergebracht. Im Folgenden werden kurz die Hauptaufgaben der beiden Ordner zusammen mit einigen Bedienungshinweisen beschrieben. Zur Ausführung sind Python (Version 3.7) und pip (Version 10.01) und MongoDB vorausgesetzt.

Der Helpers Ordner

Im nachstehenden Abschnitt werden die einzelnen Unterordner des Ordners *Helpers* im Detail betrachtet, ihre jeweiligen Ziele erläutert und zum Teil mögliche Herangehensweisen für weiterführende Projekte aufgelistet. Es ist zu beachten, dass jeder Unterordner eine spezifische Aufgabe erfüllt und dadurch eine andere Python Umgebungen besitzt. Es wird deshalb empfohlen für jeden Unterordner eine eigene virtuelle Python Umgebung zu erstellen und darin die in der *requirements.txt* befindlichen Abhängigkeiten zu installieren.

- Erläuterung zur Erstellung von virtuellen Python Umgebungen ist hier zu finden: https://realpython.com/python-virtual-environments-a-primer/
- Alternativ mit Anaconda: https://conda.io/docs/user-guide/tasks/manage-environments.html

Im großen Ganzen dienen die Skripte in diesem Bereich der Datenbeschaffung und ihrer Aufbereitung für das Backend. Abhängigkeiten mit *tensorflow-gpu*[1] wurden auf einem Windows 10 Rechner mit Nvidia GPU entwickelt, da es deutlich performanter ist. Je nach Betriebssystem gibt es unterschiedliche Unterstützung dieser Abhängigkeiten und zur Not kann es sich lohnen, sich die Liste der Anbieter dieser Abhängigkeiten im Anaconda Cloud[2] anzuschauen, um spezifische Versionen dieser auszuwählen.

[1] Setting up tensorflow-gpu: https://medium.com/@lmoroney_40129/installing-tensorflow-with-gpu-on-windows-10-3309fec55a00
[2] Anaconda Cloud: https://anaconda.org/

download_and_filter_sbb_images

Ziel ist es zunächst, erforderliche Daten, in diesem Fall alle Bilder der Digitalisate der Staatsbibliothek, zu gewinnen. Dieser Unterordner enthält das Skript *sbbget.py*, welches eben diese Aufgabe erledigt. Dabei gilt zu beachten, dass es sich um mehrere 100k Bilder im TIFF-Format handelt, was zu einer langen Downloadzeit (mehrere Tage) führt. Für den Fall, dass das Script abbricht, ist das Skript erneut zu starten. Es wird dann an der Stelle des Abbruchs fortfahren. Das Skript iteriert durch die gegebene Liste mit PPN-Einträgen (*OCR-PPN-Liste.txt*) und lädt alle ausgewiesenen Bilder eines Buches, welches durch eine PPN referenziert ist, herunter.

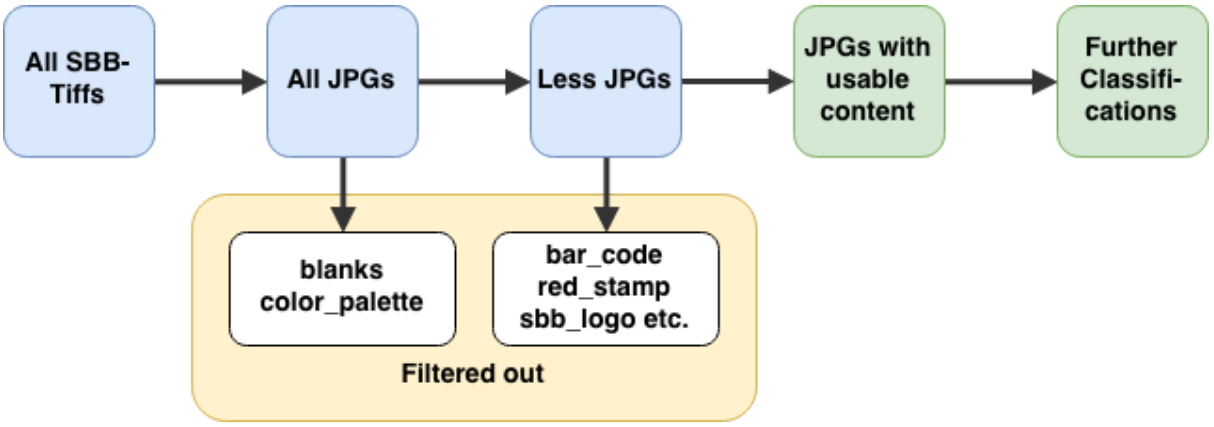
Zum Starten des Skripts ist folgender Befehl auszuführen: `python sbbget.py`

Nachdem das Skript mit der PPN-Liste beendet ist, sollte ein Ordner erstellt worden sein (*sbb*), worin sich die heruntergeladenen Bilder befinden. Diese sind ferner nach der PPN-Bezeichnung geordnet. Da jedoch nicht alle Bilder verwendbar sind, ist es sinnvoll, diese zunächst herausfiltern, um die Anzahl der Bilder zu reduzieren und die Qualität der Ausgangsdaten zu verbessern. Zur weiteren Optimierung der Laufzeit wurde beschlossen, alle brauchbaren TIFF-Dateien in JPGs umzuwandeln und mit diesen weiterzuarbeiten. Das Aussortieren und die Umwandlung erledigt das Skript *classify-all.py*. Hiermit wurden beispielsweise diese Bilder aussortiert:

- Leere Seiten
- Farbpaletten
- Barcodes
- Rote Stempelmarken
- SBB-Logo

Das genannte Skript verwendet zur Filterung einen ML-Algorithmus und benötigt als Input Trainingsdaten in Form von Bildern. Diese Bilder stammen aus dem zuvor erzeugten *sbb*-Ordner. Die Anzahl der Bilder kann stark aufgrund ihrer Qualität bzw. der Ähnlichkeiten innerhalb einer Kategorie variieren. So wurden zum Beispiel für die Kategorie “blanks” (Leere Seiten) 1500 Exemplare gesammelt. Wenn sich die Merkmale der Bilder wenig voneinander unterscheiden, sind durchaus bereits 500 Elemente hinreichend. Die Anzahl unterliegt daher der Qualität der Bilder einer jeweiligen Kategorie sowie der Verschiedenheit von Kategorien. Eine detaillierte Beschreibung dieser Skripte ist in der dazugehörigen *readme.md*-Datei enthalten[3].

[3] download_and_filter_sbb_images::
https://github.com/CouchCat/imi-unicorns/tree/master/Helpers/download_and_filter_sbb_images



Aufgrund der Anzahl von Trainingsdaten und ihrer Verschiedenheit untereinander wurde die Filterung zweimal angewendet, um möglichst gute Ergebnisse zu erzielen. Diese kann jedoch noch vereinfacht werden.

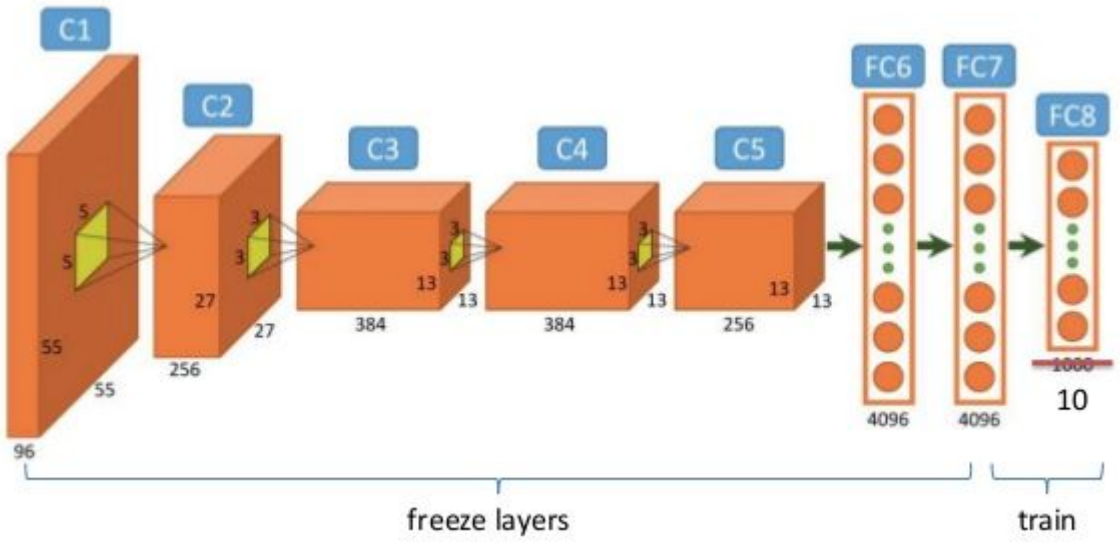
classify_and_create_json

In diesem Unterordner geht es um die Klassifizierung der relevanten Bilder. Da die Klassifizierung solch einer großen Datenmenge mit einiger Rechenzeit verbunden ist, wurde diesbezüglich beschlossen, den Prozess aus der Endanwendung auszulagern, um dort den Rechenaufwand zu minimieren. Dementsprechend galt es, stattdessen die Klassifizierung im voraus durchzuführen und die Ergebnisse in einem Format zu serialisieren, das eine einfache Speicherung in der Datenbank erlaubt. Hierfür wurde das JSON Format verwendet. Details zur Struktur der JSON-Datei können in der *readme.md*[4] nachgelesen werden.

Um eine sinnvolle Klassifizierung durchführen zu können, ist ein passendes Model zu finden. Dazu existieren verschiedene Ansätze. In vorliegenden Projekt wurden für diese Aufgabe sowohl ein eigenes Model mit **Fine-Tuning**[5] als auch das vortrainierte Model selbst eingesetzt. Fine-Tuning bedeutet, dass ein vortrainiertes Model lediglich als Basis für eine benutzerdefinierte Klassifizierung dient. In diesem Fall fand das **ImageNet-Model** VGG16 Verwendung, da dieses vergleichsweise leicht zu verstehen und zu implementieren ist. Überdies sind dazu reichlich Ressourcen und Quellen für Hilfestellungen vorhanden. Es ist üblich, vortrainierte Netze wie das VGG16 einzusetzen, um auf diese Weise die Dauer des Trainings bedeutend zu verkürzen. Denn das Trainieren der Models selbst kann von einigen Minuten bis zu mehreren Wochen andauern. Neben dem VGG16-Netz sind unter anderem auch ResNet50 und InceptionV3 weitere geeignete Models (vorgeschlagene Architekturen von neuronalen Netzen). Sie zählen zu den ImageNet-Models und stellen gute Kandidaten dar, da sie allesamt das Ziel verfolgen, mit möglichst geringer Fehlerquote die 1000 verschiedenen Kategorien (**Klassen**) des ImageNet Wettbewerbs (ILSVRC) zu klassifizieren. Anstatt die resultierenden 1000 Klassen zu nutzen, kann das Model zusätzlich modifiziert werden (**Fine-Tuning**), sodass dieses am Ende Ergebnisse zu einer festgelegten Anzahl frei

[4] classify_and_create_json:
https://github.com/CouchCat/imi-unicorns/tree/master/Helpers/classify_and_create_json
[5] Transfer Learning bzw. Fine-Tuning
https://www.analyticsvidhya.com/blog/2017/06/transfer-learning-the-art-of-fine-tuning-a-pre-trained-model/

gewählter Kategorien liefert. Demzufolge müssen nicht zwangsläufig ausschließlich die im ImageNet vorgeschlagenen Kategorien verwendet werden, sondern es können auch eigene ergänzt werden.



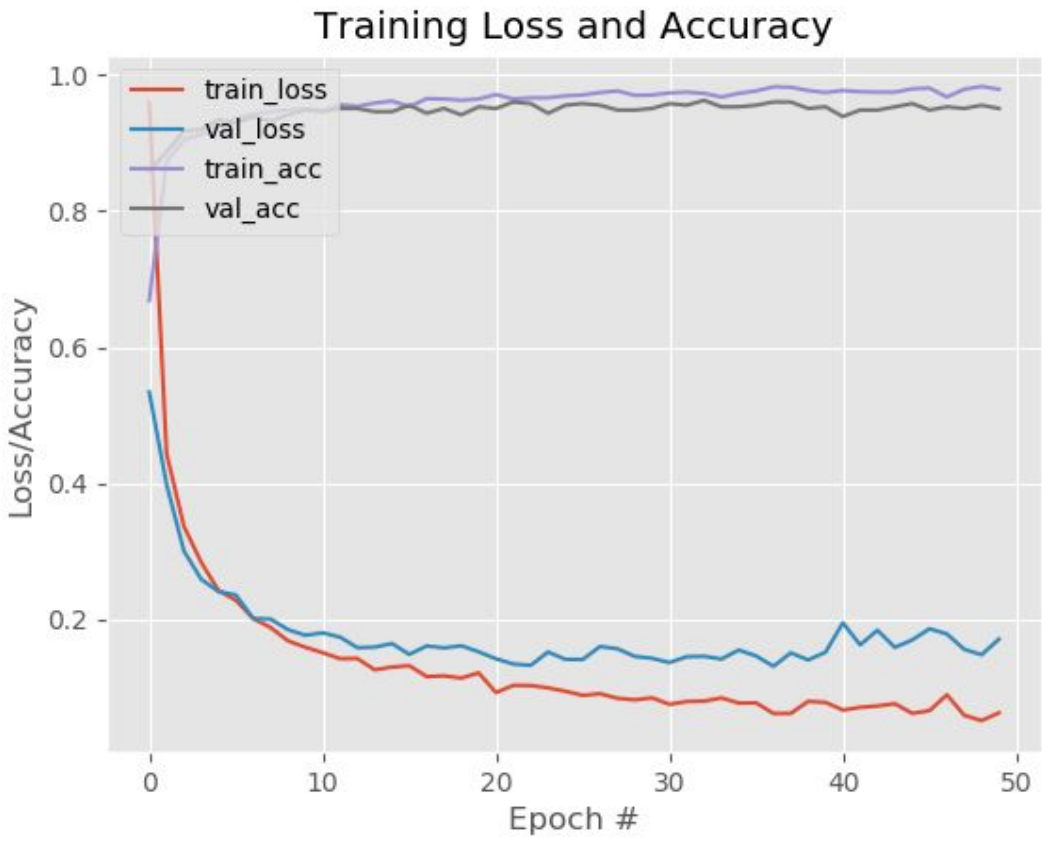
In dieser Illustration wird ein neuronales Netz (Convolutional Neural Network) so modifiziert, dass alle Schichten außer der letzten Schicht beim Trainieren nicht modifiziert werden (freeze layers). So bleibt die Qualität dieses vortrainierten Netzes erhalten und man spart sich Rechenaufwand beim Trainieren. Die letzte Schicht wurde so modifiziert, dass sie, statt nach 1000 verschiedenen Kategorien, nur noch nach 10 festgelegten Kategorien klassifizieren soll.

Unter den 1000 ImageNet Kategorien befanden sich einige, die für das Projekt geeignet waren, weshalb das vortrainierte Model auch unmodifiziert zum Einsatz kam. Die Ergebnisse wurden anschließend nach verwertbaren Klassen gefiltert. Da sich die Models von ImageNet jedoch eher an modernen Alltagsgegenständen und Objekten orientieren, ist ein Großteil der Kategorien nicht optimal im Hinblick auf die historischen Bilder der Staatsbibliothek. Infolgedessen ist es notwendig, genügend passende Trainingsdaten zu sammeln. Neben der Bedienungsanleitung für die Klassifizierungsskripte ist die *readme.md*-Datei auch mit Links zu nützlichen Ressourcen versehen. Einer davon führt zu einem Skript, durch welches Trainingsdaten mit Hilfe von Google Images schnell und einfach generiert werden können.

Letztlich soll dieser Abschnitt als kleine Hilfestellung zur Optimierung neuronaler Netze (Models) dienen. Dafür ist ein Grundverständnis über den Aufbau sowie Workflow von neuronalen Netzen[6] (nützliche Links dazu sind in der *readme.md* hinterlegt) vorausgesetzt. Nachdem man sich damit etwas auseinandergesetzt hat, fällt schnell auf, dass die Qualität[7] des Netzes nicht nur von der Auswahl und Anzahl der Trainingsdaten abhängt, sondern auch von einer Vielzahl von Parametern, die vor dem Training zu definieren sind, sowie von den

[6] Grundlagen von neuronalen Netzen:
https://www.raywenderlich.com/188-beginning-machine-learning-with-keras-core-ml
[7] Ansätze um die Qualität und Performanz von neuronalen Netzen zu verbessern:
https://machinelearningmastery.com/improve-deep-learning-performance/

Algorithmen, die angewendet werden können. Dennoch ist ein empirisches Vorgehen nicht zu vermeiden. Daher sind Ergebnisse stets festzuhalten, zu evaluieren und so anzupassen, dass die nächste Iteration bessere Resultate liefert. Die Techniken zur Evaluation[8] sind vielfältig, so wie auch die Visualisierungsmöglichkeiten. Es lohnt sich daher zu wissen, welche Art von Evaluation für das zu untersuchende Model zweckdienlich ist und darauf aufbauend entweder eine eigene Visualisierung zu erstellen (zum Beispiel mit der Python Library matplotlib) oder vorhandene Frameworks wie Tensorboard[9] hinzuzuziehen.



In diesem Projekt wurde neben Tensorboard auch matplotlib verwendet, um den Verlauf von Training Loss and Accuracy zu visualisieren. Um zu überprüfen, ob das Netz Fortschritte macht, ist es üblich den Trainingsdatensatz in Training- und Validierungsdatensatz zu unterteilen. Beim Trainieren wird iterativ versucht, die Parameter des Netzes so zu modifizieren, dass die Fehlerquote (Loss) minimiert und die Genauigkeit (Accuracy) maximiert werden. Bei steigender Iteration tendiert das Netz dazu, sich zu sehr an den Trainingsdatensatz anzupassen (Overfitting). Der Validierungsdatensatz wird deshalb zur Überprüfung, ob das Netz auch außerhalb des Trainingsdatensatz gut klassifizieren kann, verwendet. Hierzu wäre es ebenfalls sinnvoll, Cross-Validation durchzuführen, um Under- und Overfitting zu vermeiden. Aus zeitlichen Gründen wurde dieser Schritt jedoch ausgelassen (siehe Ausblick).

---

[8] Techniken der Evaluation:
https://machinelearningmastery.com/metrics-evaluate-machine-learning-algorithms-python/
[9] Visualisierung der Performanz von ML-Models:
https://fizzylogic.nl/2017/05/08/monitor-progress-of-your-keras-based-neural-network-using-tensorboard/

calculate_color_information

Dieser Unterordner enthält das Skript *dominant_colors.py*, das für die Berechnung der dominanten Farben sowie der Durchschnittsfarbe jedes Bildes zuständig ist. Als Datenbasis dient dabei der *content*-Ordner, welcher im Schritt zuvor durch das Skript *classify-all.py* erstellt wurde. Um die Durchschnittsfarben ohne großen Arbeitsaufwand im Frontend darstellen zu können und zukünftig eine Filterung nach Farben zu ermöglichen, werden die ermittelten RGB-Werte den nächsten Webcolors zugeordnet. Zur anschließenden Berechnung der vorerst zehn häufigsten Farben dient der von Scikit-Learn bereitgestellte Clustering Algorithmus Mini-Batch K-Means[10]. Da ein Großteil der Bilder tendenziell monochrom ist, besteht die Gefahr, dass die dominanten Farben nach dem Mapping auf die websicheren Farben zusammenfallen und demzufolge mehrfach auftreten. Um diese Redundanz weitgehend zu vermeiden, wurde zu Beginn eine höhere Clusteranzahl gewählt. Dies erzielt aufgrund der Miteinbeziehung seltenerer Farben eine bessere Repräsentation des Bildes. Nach dem Mapping erfolgt die Ermittlung der fünf dominanten (wenn möglich einmaligen) Farben. Alle Ergebnisse werden schließlich in der Pickle-Datei *color_infos.pkl* gespeichert und an die Datenbank übergeben.

Das Skript kann mit dem Befehl `python dominant_colors.py` ausgeführt werden. Weitere Informationen zu benötigten Packages sind in der entsprechenden *readme.md*[11] im GitHub-Repository zu finden.

bovw_and_simiar_images

Skripte und sonstige Ressourcen, die mit der Berechnung ähnlicher Bilder zusammenhängen, sind hier abzulegen. Dazu zählen die bereits enthaltenen Skripte *features_resnet.py* und *ball_tree.py*. Ersteres umfasst die Erstellung von Histogrammen basierend auf dem **Bag of Visual Words**-Ansatz[12]. Dazu werden im ersten Schritt die Featurevektoren vom letzten Convolutional Layer des auf ImageNet trainierten Netzwerks RestNet50[13] ausgelesen. Die gesamte Menge der extrahierten Features repräsentiert dabei den Bag. Zur Beschleunigung der Rechenzeiten folgte eine Dimensionsreduktion des Bags mittels einer PCA[14]. Der verkleinerte Datensatz bildet die Grundlage für die Erstellung eines Codebooks mit einem Clustering-Verfahren. In diesem Projekt wurde dafür, wie bereits bei der Berechnung der

---

[10] K-Means Algorithmus:
https://towardsdatascience.com/clustering-using-k-means-algorithm-81da00f156f6
http://scikit-learn.org/stable/modules/generated/sklearn.cluster.MiniBatchKMeans.html#sklearn.cluster.MiniBatchKMeans

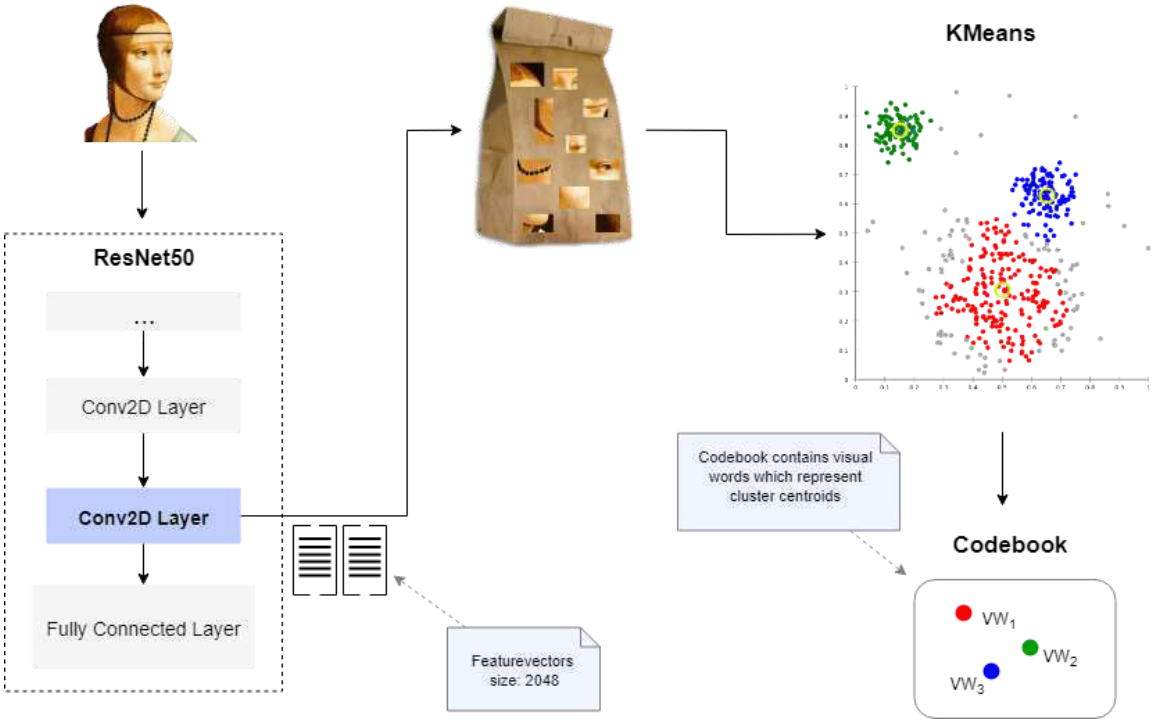
[11] Readme zum calculate_color_information-Ordner:
https://github.com/CouchCat/imi-unicorns/tree/master/Helpers/caclulate_color_information

[12] Bag of Visual Words: https://gurus.pyimagesearch.com/the-bag-of-visual-words-model/

[13] ResNet50: https://keras.io/applications/#resnet50

[14] Principal Component Analysis:
https://towardsdatascience.com/pca-using-python-scikit-learn-e653f8989e60

dominanten Farben, der Mini-Batch K-Means Algorithmus eingesetzt. Die daraus hervorgehenden Zentroide stellen die visuellen Wörter des Codebooks dar.



15

Dementsprechend gilt es eine Codebook-Größe zu bestimmen, bei welcher die visuellen Wörter den vollständigen Datensatz möglichst gut abbilden. In diesem Fall kam ein Codebook mit 8500 visuellen Wörtern zum Einsatz. Zum Schluss wird für jedes Bild ein Histogramm berechnet. Dabei wird automatisch im selben Ordner für alle angegebenen Codebook-Größen eine entsprechende Pickle-Datei mit dem Namen *histograms_XX.pkl*, die jeweils alle Histogramme enthalten, erstellt.

Das Skript wird in gleicher Weise gestartet: `python features_resnet.py`

Die Histogramme sind für die Ausführung des zweiten Skriptes, *ball_tree.py*, erforderlich. Eine Länge von 8500 Wörtern bedeutet, dass die Datenpunkte in einem 8500-dimensionalen Raum liegen. Um sinnvolle Ergebnisse bei dieser hohen Anzahl von Dimensionen zu erzielen, ist vor allem die Unterteilung des Raumes entscheidend. Ein Großteil der Verfahren bzw. Datenstrukturen ist für hochdimensionale Räume nicht geeignet. Im Projekt wurde die von Scikit-Learn zur Verfügung gestellte Datenstruktur **Ball Tree**[16], ein binärer Baum mit hypersphärischen Knoten, der besonders bei Nearest-Neighbor-Problemen mit vielen Dimensionen effizient ist, eingesetzt.

[15] Prozess zur Berechnung der Histogramme nach dem Bag of Visual Word-Prinzip
[16] Ball Tree:
https://en.wikipedia.org/wiki/Ball_tree
http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.BallTree.html

Sollten die beiden genannten Skripte an verschiedenen Orten abgelegt worden sein, so ist die gewünschte Histogramm-Datei in den Ordner, welcher die *ball_tree.py* enthält, zu verschieben.

Es ist ferner anzumerken, dass vorab ein weiterer Arbeitsschritt notwendig ist. Da einige Bilder aus dem *content*-Ordner aufgrund mangelnder Qualität nicht klassifizierbar waren, befindet sich nur ein Teildatensatz in der Datenbank. Die Berechnung der Histogramme musste jedoch wegen der begrenzt verfügbaren Bearbeitungszeit parallel zur Klassifizierung stattfinden, sodass aus diesem Grund für alle Bilder Histogramme erstellt wurden. Folglich war es unerlässlich, die Histogramme von den Bildern, die sich in der Datenbank befinden, aus der Gesamtmenge auszulesen. Die Datei *images_new_names.pkl* enthält die Namen aller Bilder. Mithilfe dieser Datei werden die Einträge mit den Namen in der Datenbank verglichen, um so die entsprechenden Indices der Pickle-Datei herauszuschreiben und in der *image_ids.pkl* zu speichern. Die Selektion der dazugehörenden Histogramme erfolgte anhand dieser Pickle-Datei. Aus den verbleibenden Histogrammen wird anschließend ein Ball Tree erstellt, um daraus die fünf nächsten Nachbarn für jedes Element zu bestimmen. Um das Skript ausführen zu können, müssen sich daher beide Dateien ebenfalls im selben Ordner befinden. Letztere ist bereits im GitHub-Repository enthalten. Auch zu diesem Ordner können in der *readme.md*[17] entsprechende Ausführungsanweisungen nachgelesen werden.

[17] Readme zu bovw_and_similar_images-Ordner:
https://github.com/CouchCat/imi-unicorns/tree/master/Helpers/bovw_and_simiar_images

Der Server

Installieren der Abhängigkeiten und Starten des Servers

Es wird geraten, eine virtuelle Umgebung von Python zu verwenden, um eine strukturierte Umgebung auf dem lokalen Computer beizubehalten. Alle nachstehenden Befehle sind im Ordner *Server/* auszuführen:

- Erstellen der Python Virtual Environment (venv):
  ```
  python -m venv .
  ```
- Aktivieren der Venv:
  ```
  source ./bin/activate
  ```
- Installieren der Python-Packages:
  ```
  pip install -r requirements.txt
  ```
- Erstellen der Systemvariable:
  ```
  export FLASK_APP=./app/__init__.py
  ```
- Starten des Flask Servers:
  ```
  flask run
  ```

Klassen und Data Controller für den Server

__init__.py	Ist für die Instanziierung des Servers mit allen notwendigen Datenbankverbindungen und Data-Controller-Abhängigkeiten zuständig.
import.py	Hilfsklasse zum Importieren und Verarbeiten von Daten in der Datenbank.
api.py	Ist dafür zuständig, dem Server die im *WebApp/dist/*-Ordner enthaltenen Daten bekannt zu machen und bietet damit die Funktionalität eines File-Servers. Des Weiteren beinhaltet die Klasse eine Sammlung aller REST-Anfragen, die der Server kennen soll/muss, um dem Frontend die notwendigen Daten zu liefern. Diese REST-Calls sind serverseitige Funktionen, wobei dieser Methoden von den Controllern benutzt, um Datenbankabfragen auszuführen.
book_controller.py	Ist für alle Abfragen, die an die “books”-Collection der Datenbank gerichtet und damit buchspezifisch sind, zuständig. Zum Beispiel ruft dieser Controller die Zeitspanne gegebener Kategorien oder Staatsbibliothek-Kategorien ab.
image_controller.py	Ist für alle Abfragen, die an die “images”-Collection der Datenbank gerichtet und bildspezifisch sind, zuständig.

features_controller.py	Ist für alle Abfragen, die an die “features”-Collection der Datenbank gerichtet sind, zuständig und liefert alle in der Datenbank gespeicherten Kategorien (ermittelt durch einen Bilderkennungsalgorithmus) zurück.
genre_controller.py	Ist für alle Abfragen, die an die “genres”-Collection der Datenbank gerichtet sind, zuständig und liefert alle in der Datenbank gespeicherte, Stabi- Kategorien zurück.
color_controller.py	Ist für alle Abfragen, die an die “colors”-Collection der Datenbank gerichtet sind, zuständig und liefert alle in der Datenbank gespeicherten Farb- Kategorien zurück.
maps_controller.py	Ist für alle Abfragen, die an die “maps”-Collection der Datenbank gerichtet sind, zuständig und liefert alle von der Maps-Component benötigten Daten zurück.

Das Frontend

Das Frontend ist eine Applikation, die mit dem Framework Angular 2 (Version 5) entwickelt wurde und auf dem Konzept Model-View-ViewModel basiert. Zur Entwicklung der Anwendung diente TypeScript, welches nach dem Kompilieren / Interpretieren in diverse JavaScript-Dateien von npm übersetzt wird. Die Installation der Packages, die in dem Frontend verwendet werden, erfolgte über den Package Manager npm (Version 6.4.0). Dieser ist in der Installation von NodeJs (Version 10.5.0) enthalten.

Installieren der Abhängigkeiten und bauen der Applikation

Alle folgenden Befehle werden im *WebApp/* Ordner ausgeführt.

- Installieren der Abhängigkeiten aus der package.json
  ```
  npm install
  ```
- Interpretieren der TypeScript-Dateien in JavaScript (einmalig)
  ```
  npm run build
  ```
- Dauerhaftes Interpretieren der TypeScript-Dateien in JavaScript bei andauernder Aktivität des Befehls (watch deamon)
  ```
  npm start
  ```

Aufbau der Angular Anwendung

Die Applikation ist eine “Single-Page”-Anwendung und besteht aus verschiedenen Angular Komponenten sowie Services. Eine Angular-Komponente bildet das Grundgerüst und beinhaltet ein HTML-Template, welches den visuellen Teil der Applikation ausmacht. Ein Service ist ein Controller, der wiederum Daten für eine spezifische Komponente abruft, diese verarbeitet und zurückgibt. Des Weiteren dienen Interfaces der Typisierung gegebener Daten

aus der Datenbank und dem DOM. Die Gestaltung der Applikation wird mittels SCSS / CSS realisiert. SCSS ist eine Erweiterung von CSS und vereinfacht die Erstellung von Stylesheets. Mit SASS oder einem anderen entsprechenden Programm wird es in CSS umgewandelt. Alle Bilder sind im Ordner *./dist/ChasingUnicornsAndVampires/assets/images/* abzulegen, damit diese vom Server erkannt, gelesen und zurückgeliefert werden können.

Services

dom.service.ts	Ist eine Helper Klasse für den modal.service und durchsucht das DOM nach vordefinierten HTML-element-ids, um in diesen Angular 2 Komponenten zu erstellen.
modal.service.ts	Benutzt den bereits erwähnten dom.service und ist für die Instanziierung von Angular-Komponenten, die modale Dialoge ergeben, zuständig.

Die folgenden Service-Klassen beinhalten HTTP-GET-Methoden, um entsprechende Daten bei Bedarf von der Datenbank abzurufen:

book.service.ts	Buchspezifische Informationen
category.service.ts	Kategoriespezifische Informationen
color.service.ts	Farbspezifische Informationen
genre.service.ts	Genre-spezifische Informationen
image.service.ts	Bildspezifische Informationen
maps.service.ts	Daten, die von der map.component benötigt werden

Komponenten

app.module.ts	Besitzt die Aufgabe, alle notwendigen Klassen, Komponenten, Services, Directories, etc. zu instanziieren und diese mit ihren Abhängigkeiten, Typen sowie applikationsspezifischen Einbindungen zu verwalten. Ferner definiert sie sämtliche Routes / Pfade.
app.component.ts	Bildet den Einstiegspunkt der Anwendung, an dem alle Komponenten zusammengefügt werden. Zusätzlich ruft sie einen Router auf, der die jeweilige, zum Pfad passende Komponente zurückgibt.
cluster-view.component.ts	Repräsentiert die URL “localhost:500/cluster-view” und enthält die TimeSliderComponent sowie mindestens eine

	ClusterRowComponent. Sie visualisiert außerdem gewählte Kategorien zusammen mit ihren Bildern und zusätzlichen Features, die durch weitere Komponenten umgesetzt sind.
map-view.component.ts	Repräsentiert die URL “localhost:5000/map-view”. Darüber hinaus umfasst sie neben einer TimeSliderComponent die Logik, um die Weltkarte anzuzeigen und alle notwendigen Events, Daten sowie Popups zu verarbeiten.
cluster-row.component.ts	Dient der Anzeige einer gewählten Kategorie als separate Reihe in der Benutzeroberfläche. Beim Starten der Applikation wird eine ClusterRowComponent instanziiert. Allerdings bleibt diese Funktionalität verborgen. Zur Laufzeit ist es dann möglich, bis zu (derzeit) drei Reihen hinzuzufügen. Sie beinhaltet ferner die beiden Komponenten ModalMenuComponent und CanvasComponent.
canvas.component.ts	Visualisiert eine übergebene Kategorie.
image-modal.component.ts	Bei Auswahl eines Bildes erfolgt die Instanziierung der Komponente, um das Bild in einem modalen Dialogfenster vergrößert anzuzeigen. Das Dialogfenster stellt zudem Detailinformationen wie dem Titel des Buches, das dieses Bild enthält, einer URL zum Digitalisat, dem Autor, dem Erscheinungsort sowie -jahr bereit.
modal-menu.component.ts	Realisiert die Funktionalität zur Auswahl einer Kategorie in der Anwendung. Sie wird durch den ModalService erzeugt und übergibt die gewählte Kategorie zur Darstellung ihrer Bilder an die ClusterRowComponent.
time-slider.component.ts	Visualisiert einen TimeSlider, der als Filter in Bezug auf das Erscheinungsjahr dient. Dieser erhält initial eine Zeitspanne und gibt diese anhand von Events aus.

Models

Die nachstehenden Interfaces typisieren die Daten der Applikation und machen diese definiert verwendbar:

- *category.model.ts*
- *color-category.model.ts*
- *stabi-category.model.ts*
- *image.model.ts*

Datenbank

Ein Backup der Datenbank liegt im Repository womit man den aktuellsten Stand der Datenbank ohne Probleme bei sich aufsetzen kann.
Voraussetzung ist Mongo installiert und eingestellt zu haben.

Zum wiederherstellen der Datenbank muss man nur den folgenden Befehl ausführen:

```
mongorestore -d unicorns ./UnicornsDB/unicorns
```

Hiernach hat man den aktuellsten Stand der Datenbank lokal hinterlegt.

HERAUSFORDERUNGEN

Generell ist bei Weiterführung des Projektes zu beachten, dass Rechenzeiten aufgrund der großen Datenmenge unter Umständen mehrere Tage betragen können. Dies sollte bei der Zeitplanung berücksichtigt werden. Auch das Kopieren der Daten von sowie auf externe Festplatten dauerte bis zu 4 Stunden. Die Zeit für Datenübertragung ist dementsprechend einzuplanen.

Visualisierung

Trotz Bereinigung und Zusammenfassung aller Daten lag die Datenmenge nach wie vor im Big-Data-Bereich. Besonders die sehr hohe Anzahl der Bilder erforderte Lösungen zur besseren Darstellung, um geringe Ladezeiten zu gewährleisten. Selbst mit Pagination und Thumbnails betragen die Wartezeiten bis zu fünf Sekunden.

Im Bereich der Webanwendungen existieren unzählige Frameworks, wobei einige leichter zu verstehen sind als andere. Da die Entwickler in verschiedenen Bereichen Erfahrung besaßen, konnten sie bezüglich des Frameworks keinen gemeinsamen Nenner finden. Folglich war ein Teil des Entwicklungsteams gezwungen, sich neues Wissen in kurzer Zeit anzueignen und sich mit einem neuen Framework vertraut zu machen. In diesem Fall fiel die Wahl auf Angular, da es den Eindruck vermittelte, einen leichten Einstieg zu ermöglichen. Im weiteren Verlauf des Projektes zeigte sich jedoch, dass dieses Framework vergleichsweise viel Erfahrung bedingt. So waren beispielsweise spezielle Wünsche nicht einfach umzusetzen und die Integration einiger Plugins wie d3.js nicht problemlos möglich.

Machine Learning

In den meisten Fällen wurden Ergebnisse im Pickle-Format serialisiert. Die Verwendung ist sehr unkompliziert und es lassen sich diverse Python-Objekte damit speichern. Jedoch

zeigte sich durch die sehr große zu verarbeitende Datenmenge ein entscheidender Nachteil: Im Hinblick auf den Arbeitsspeicherverbrauch arbeitet Pickle sehr ineffizient. Das Laden von Dateien nahm zum Teil bis zu 80GB RAM in Anspruch, sodass einige Skripte nicht ohne Erweiterung des Arbeitsspeichers ausführbar waren. Nach einigen damit verbundenen Schwierigkeiten wurde daher nach einem alternativen Format gesucht. Das Format HDF5 eignet sich besonders gut für große Datensätze, da dieses speziell darauf ausgelegt ist.

Die Berechnung der ähnlichen Bilder stellte in verschiedenen Punkten, wovon einige durch die begrenzt verfügbare Zeit verursacht wurden, eine Herausforderung dar.

Das Skript zur Erstellung des Ball Trees musste beim ersten Versuch nach einer Woche Laufzeit abgebrochen werden. Der Parameter “leaf_size” des Ball Trees gibt an, ab welcher Elementanzahl die nächsten Nachbarn per Brute Force zu ermitteln sind. Ein großer Wert induziert eine erhöhte Rechenzeit zum Finden der nächsten Nachbarn, wohingegen bei einem niedrigeren Wert der Speicherbedarf ansteigt und sich damit einhergehend die Erstellung des Baumes verlangsamt.

Der Parameterwert sowie die Histogramm-Länge konnten aus bereits genannten zeitlichen Gründen nicht optimiert werden.

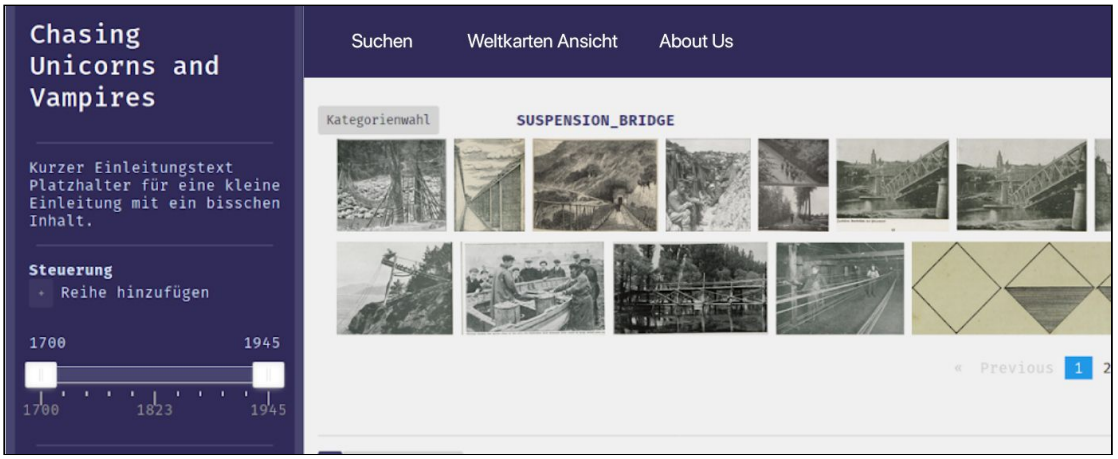
Ähnliche Herausforderungen zeigten sich auch bei der Klassifizierung der Bilder mittels Convolutional Neural Networks (CNNs). Wie auch bei anderen ML-Algorithmen, entscheiden bei dieser Methode nicht nur die Art und Menge von Trainingsdaten darüber, wie performant das Model das Klassifikationsproblem lösen kann, sondern auch viele Parameter und Aufbereitungsmöglichkeiten. Eine generelle Auseinandersetzung mit den Vor- und Nachteilen der verschiedenen Vorgehensweisen ist deshalb wichtig, um Zeit beim Trainieren des Netzes einzusparen. Die beiden Hauptprobleme, die bei vielen ML-Algorithmen auftauchen, sind Over- und Underfitting. Je nachdem, welches Problem zu lösen ist (Klassifikation, Regression, etc.), sind die verschiedenen Lösungsansätze, die speziell auf diese Probleme abzielen, näher zu betrachten. Bei diesem Projekt wurde viel Zeit damit verbracht, Netze zu verbessern, da ein Großteil der Gruppe nicht mit dem Thema vertraut war und daher empirisch bzw. experimentell vorgegangen wurde. Das Team musste sich auch die Frage stellen: Sollte ein vortrainiertes Netz wie VGG16 verwendet werden, obwohl es für das Problem (historische Bilder anstatt moderne Gegenstände) etwas ungeeignet ist oder sollte lieber auf ein völlig neues Model zurückgegriffen werden, bei welchem der zeitliche Trainingsaufwand höher ist, um ein zufriedenes Ergebnis zu erzielen? Letztendlich kam eine Vielzahl von Methoden für das Endprodukt zum Einsatz, da einige vortrainierte Netze nur teilweise gut abschnitten und fehlende Klassen mit eigenen Netzen ergänzt werden mussten. Das Prinzip Trial and Error war hier demnach nicht zu vermeiden. Eine motivierte Experimentierlust ist bei einem Projekt dieser Art daher unerlässlich.

AUSBLICK

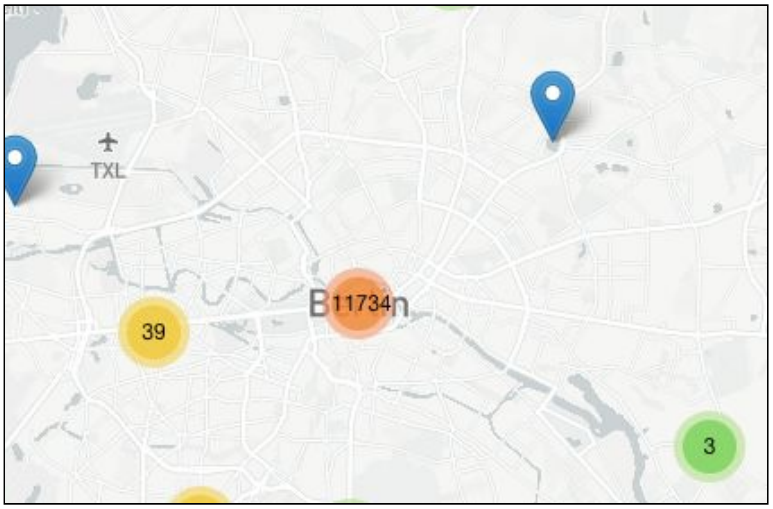
Visualisierung

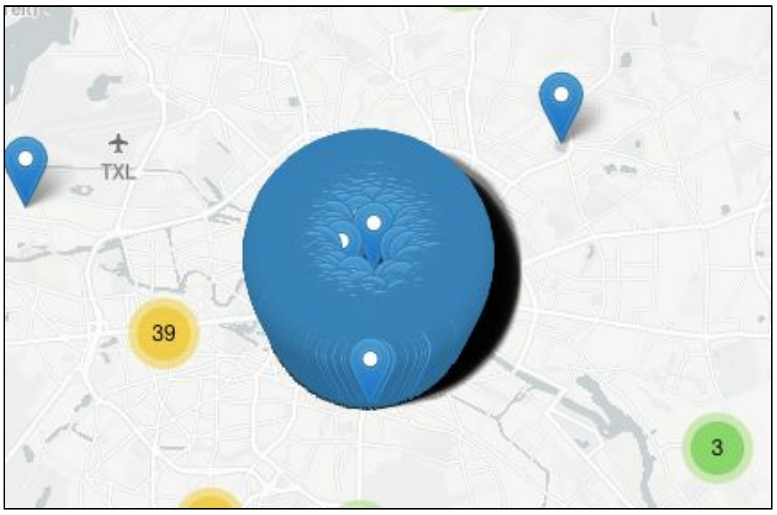
In Bezug auf den Visualisierungsteil sind nachfolgend einige Ansätze zur Verbesserung sowie Aspekte, die es zu untersuchen gilt, gelistet:

- Welche Alternativen gibt es, um die Rechenzeit zur Visualisierung der Clusterkarte zu verringern?
- Des Weiteren ist die Implementierung einer Standard-Navigation zu empfehlen. Diese erleichtert das Finden von inhalts- sowie anwendungsbezogenen Informationen.

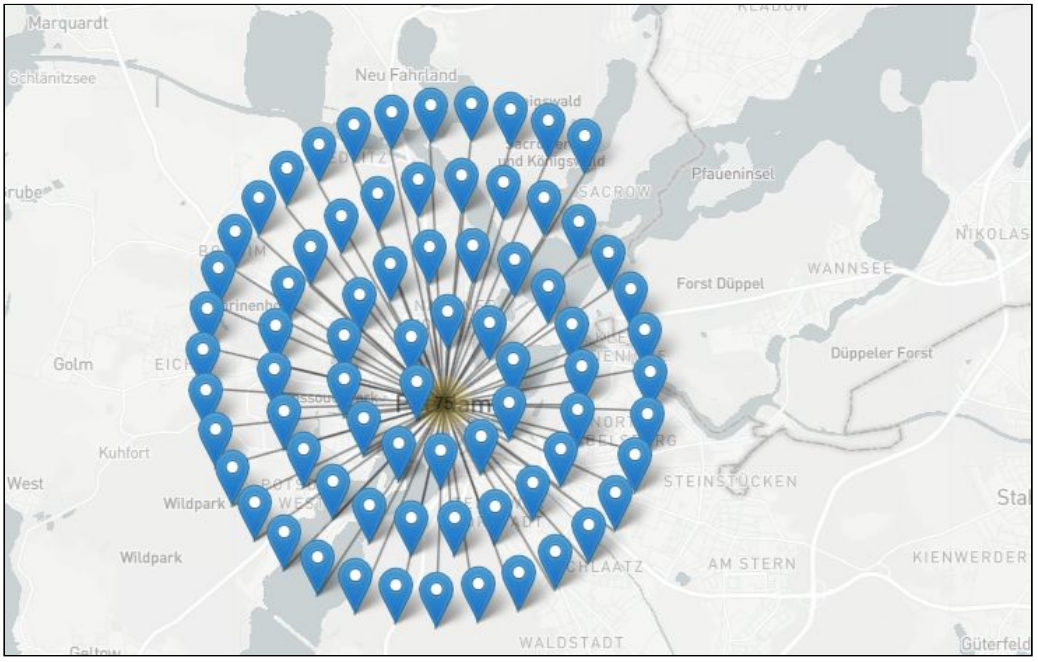


- Allgemein sind mehr Interaktionsmöglichkeiten zu schaffen und das Design der Benutzerschnittstelle zu verbessern, damit der Nutzer schneller an gewünschte Inhalte gelangt.
- Problematisch ist derzeit die Anzeige von besonders großen Clustern auf der Karte, da viele Bücher aufgrund desselben Erscheinungsortes einen gemeinsamen Marker besitzen. Zurzeit haben alle Bücher (11734), die sich in Berlin befinden, die gleiche Koordinaten - longitude 13.3888599 und latitude 52.5170365.

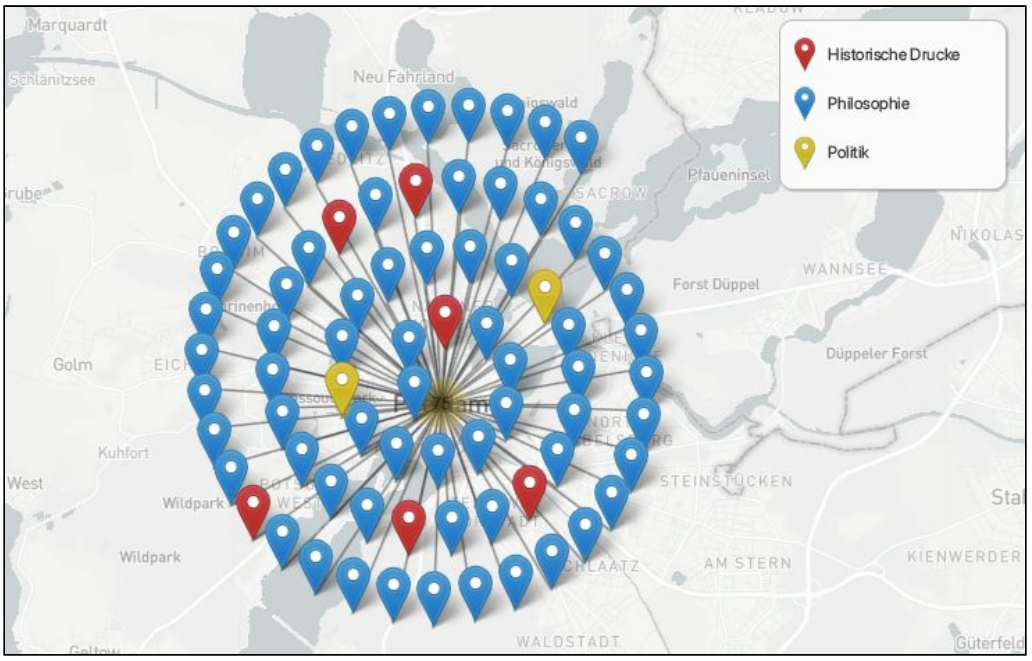




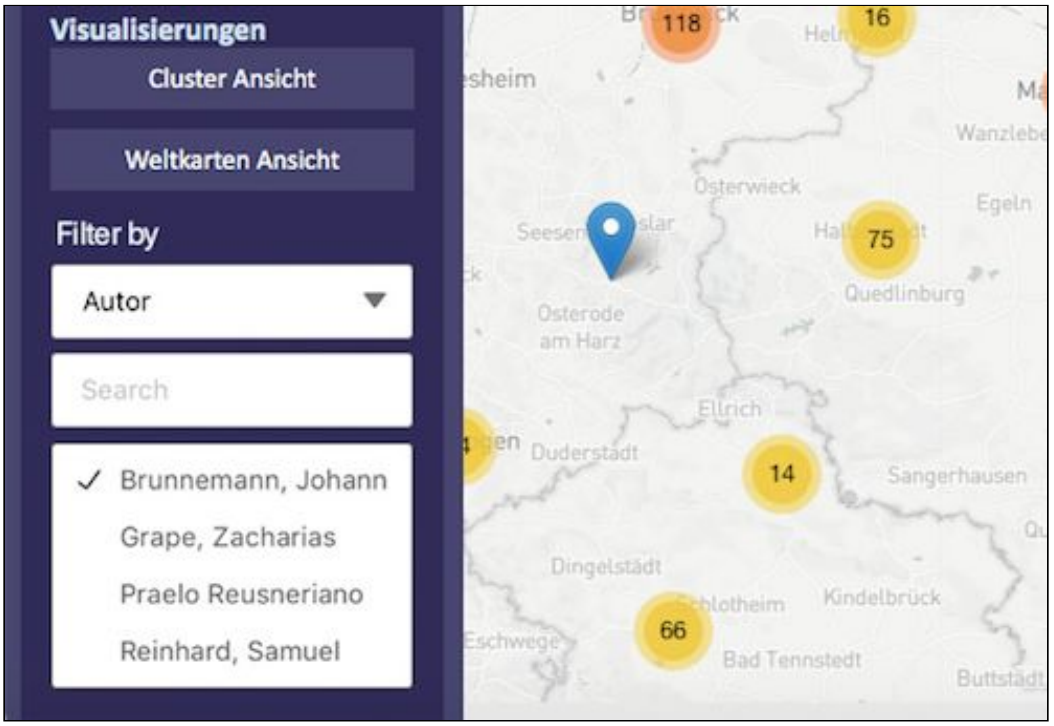
Ein mögliche Lösungsansatz ist die Speicherung verschiedener Breiten- und Längengrade in der Datenbank. Die Koordinaten können (automatisch) leicht voneinander abweichen z.B. longitude 13.3888599 wird 13.3881000 und latitude 52.5170365 wird 52.5165000, sodass die Cluster kleiner werden (z.B. bis zum 200 Bücher pro Cluster), was zu einer besseren Abgrenzung führt. Unten ist ein Screenshot des gewünschten Ergebnisses.



- Eine zusätzliche farbliche Kennzeichnung verschiedener Marker auf der Clusterkarte würde eine höhere Übersichtlichkeit erzielen.



- Außerdem ist die Ergänzung von Funktionen, die es dem Nutzer ermöglichen, Autoren nach bestimmten Kriterien zu suchen, sinnvoll. Damit könnte der Benutzer beispielsweise alle Autoren aus derselben Stadt oder desselben Genres finden.

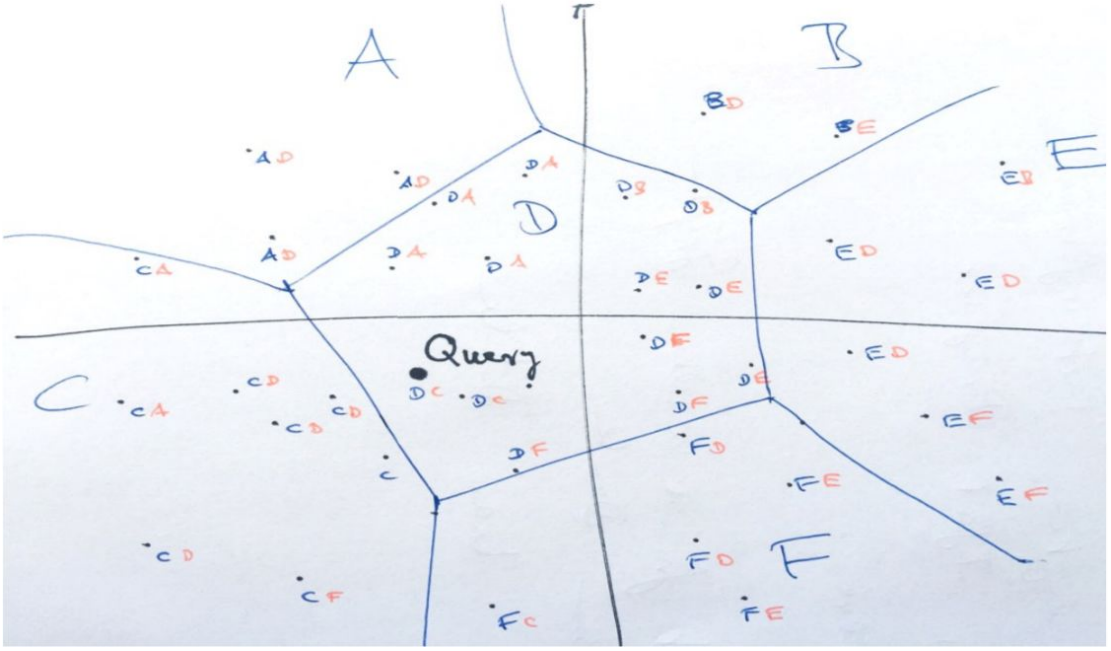


Machine Learning

Obwohl für Pickle-Dateien kein Größenlimit ermittelt werden konnte, sollte bei Weiterführung des Projektes auf die Daten-Serialisierung mittels dieses Formats, wenn möglich, verzichtet werden.

Zur Berechnung der ähnlichen Bilder sind zukünftig folgende Punkte zu evaluieren:

- Gibt es bessere Alternativen zum Ball Tree, die bessere Resultate erzielen? Zum Beispiel könnte “Überlappendes Clustering”, bei dem jedes Bild zu mehreren Clustern gehören kann, eingesetzt werden. Dieses Verfahren erzeugt für ein Query-Bild einen größeren Suchraum, ohne dabei die besonders kostenintensive Brute-Force-Methode anwenden zu müssen.



18

- Wie können bei Verwendung dieser Datenstruktur sowohl Rechenzeit als auch Speicherbedarf so gering wie möglich gehalten werden?
- Welche Histogramm-Länge ist für die Nutzung des „Bag of Visual Words“-Ansatzes geeignet?

Ein weiterer Ansatz ist außerdem die Verbesserung der Feature-Extraktion. Das ResNet50 liefert nur bedingt gute Resultate. Bessere Ergebnisse sind zu erwarten, wenn das neuronale Netz zur Klassifizierung optimiert und als Grundlage zur Extraktion verwendet wird.

Aus Zeitmangel wurden auch folgende Punkte in diesem Projekt nur wenig oder gar nicht behandelt. Nachstehend sind mögliche Verbesserungsvorschläge aufgelistet, die sich speziell auf das Klassifikationsproblem im classify_and_create_json Unterordner beziehen:

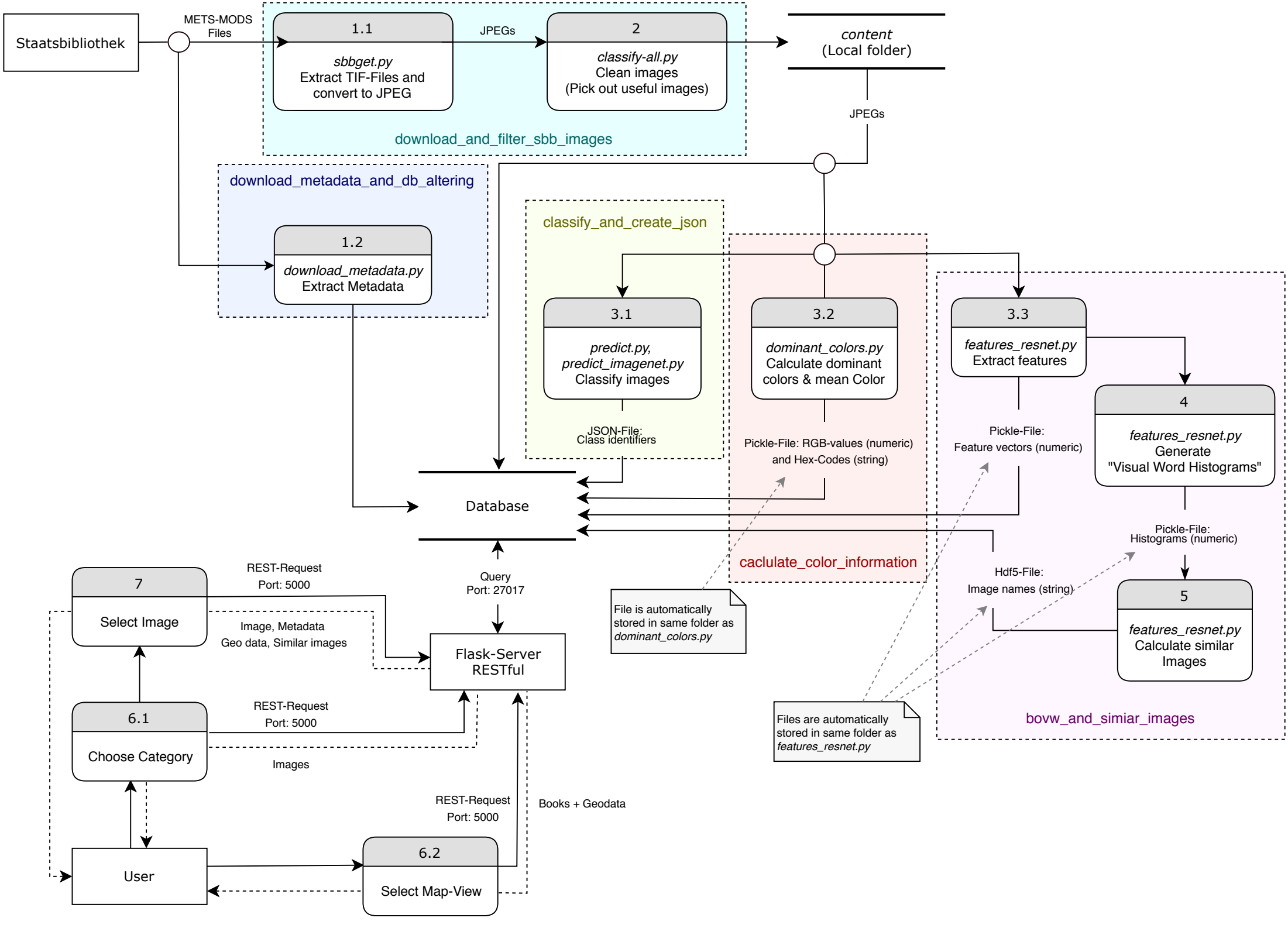
[18] Illustration entnommen aus den Lehrvortrags-Folien von Prof. Dr.-Ing. Kai Uwe Barthel

- **Cross-Validation**: Der gesamte Datensatz ist in 80 Prozent Trainings- und 20 Prozent Validierungsdaten eingeteilt. Jedoch erfolgte diese Aufteilung einmalig. Um zu untersuchen, ob der Trainingsdatensatz tatsächlich akkurate Ergebnisse liefert, ist sicherzustellen, dass jedes Bild mindestens einmal sowohl im Trainings- als auch im Validierungsdatensatz vorkommt. Hauptziel ist es, auf diese Weise einen Ausgleich zwischen Bias und Variance zu erreichen, wodurch die Wahrscheinlichkeit für Under- bzw. Overfitting minimiert wird.
- **YOLO:** Hauptsächlich kamen im Projekt ImageNet Models für die Klassifizierung zur Verwendung. Vortrainierte Models, die auf andere Datensätzen basieren, wurden dagegen vergleichsweise selten eingesetzt. Dazu zählt zum Beispiel das Object-Detection Model YOLOv3[19], welches auf dem COCO-Datensatz beruht. Es ist zu vermuten, dass dieses Model durch Fine-Tuning ebenfalls sinnvolle Ergebnisse liefert.
- **From Scratch**: Die Anwendung eines vortrainierten Netzes garantiert nicht zwingend die besten Resultate. Aus diesem Grund sind die Ergebnisse eines von Grund auf trainierten Netzes zu beurteilen.
- **Parameter ändern:** Bei einem weiterführenden Projekt gilt es, die Ergebnisse der bestehenden Models mit veränderten Parametern zu prüfen.
- **Evaluationen:** Im Rückblick hätte grundsätzlich eine zielstrebigere Identifizierung der Schwächen von eingesetzten Models durch eine systematische Aufzeichnung und Evaluation erzielt werden können. Zudem ist speziell bei Klassifikationsproblemen zu empfehlen, neben der Validation-/Test-Accuracy, auch weitere Evaluationsmethoden wie beispielsweise die Confusion Matrix miteinzubeziehen.
- **JSON:** Das Endergebnis des gesamten Klassifizierungsprozesses ist eine erzeugte JSON-Datei. Sie enthält unter anderem die Klassenzugehörigkeit eines jeden Bildes. Es kann allerdings von Vorteil sein, eine neue, effizientere JSON-Struktur, die darüber hinaus die Anzahl ermittelter Klassen sowie ihre Genauigkeiten umfasst, zu erstellen. In diesem Projekt wurden keine Angaben zu den Genauigkeiten gemacht.

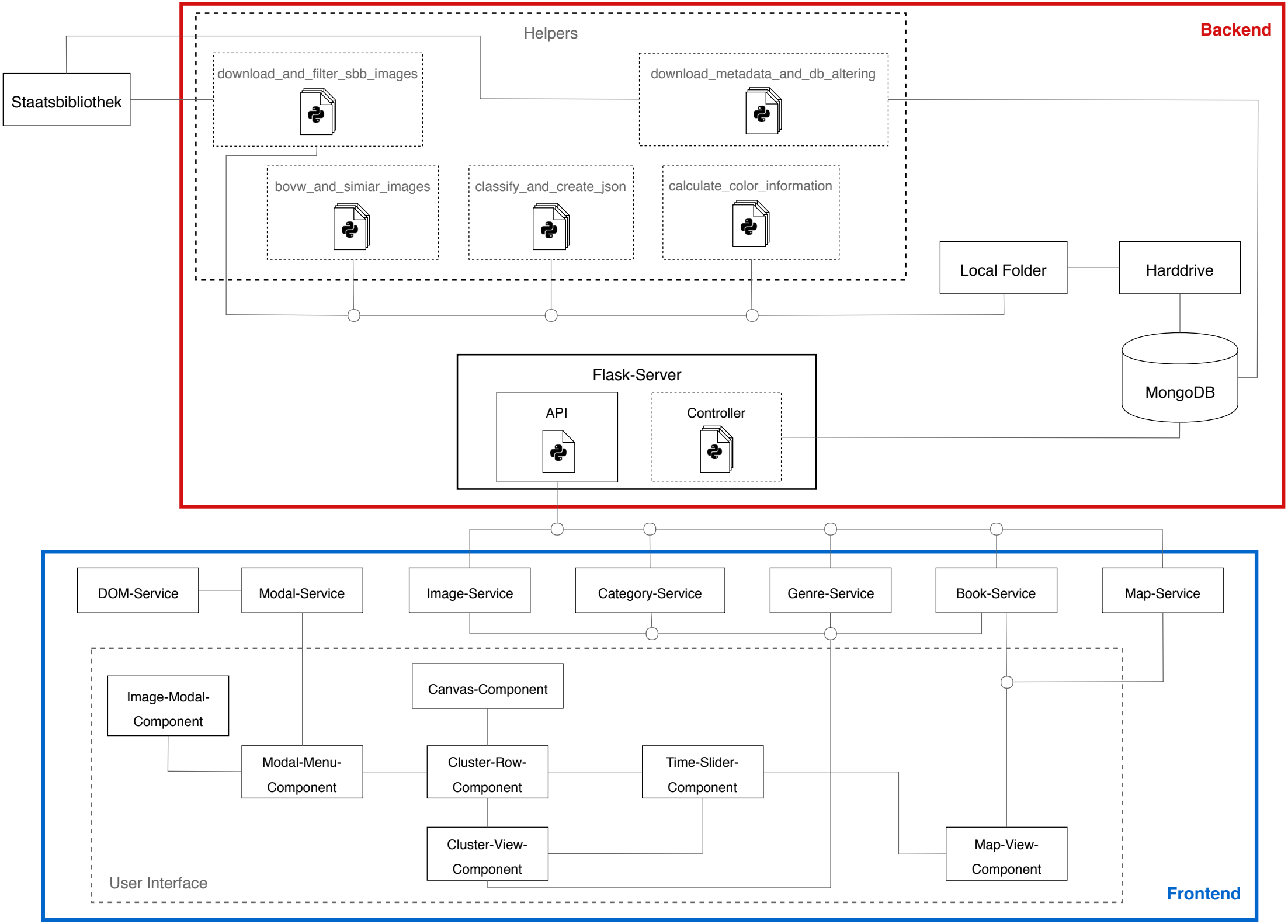
[19] YOLOv3: https://pjreddie.com/darknet/yolo/

ANHANG

A. Datenflussdiagramm


Staatsbibliothek
METS-MODS Files
1.1
sbbget.py
Extract TIF-Files and convert to JPEG
JPEGs
2
classify-all.py
Clean images
(Pick out useful images)
content
(Local folder)
JPEGs
download_and_filter_sbb_images
download_metadata_and_db_altering
1.2
download_metadata.py
Extract Metadata
classify_and_create_json
3.1
predict.py,
predict_imagenet.py
Classify images
JSON-File:
Class identifiers
3.2
dominant_colors.py
Calculate dominant colors & mean Color
Pickle-File: RGB-values (numeric) and Hex-Codes (string)
caclulate_color_information
3.3
features_resnet.py
Extract features
Pickle-File:
Feature vectors (numeric)
4
features_resnet.py
Generate
"Visual Word Histograms"
Pickle-File:
Histograms (numeric)
Hdf5-File:
Image names (string)
5
features_resnet.py
Calculate similar Images
bovw_and_simiar_images
Database
Query
Port: 27017
File is automatically stored in same folder as dominant_colors.py
Files are automatically stored in same folder as features_resnet.py
7
Select Image
REST-Request
Port: 5000
Image, Metadata
Geo data, Similar images
Flask-Server
RESTful
6.1
Choose Category
REST-Request
Port: 5000
Images
REST-Request
Port: 5000
Books + Geodata
User
6.2
Select Map-View

B. Architekturdiagramm


Backend
Helpers
Staatsbibliothek
download_and_filter_sbb_images
download_metadata_and_db_altering
bovw_and_simiar_images
classify_and_create_json
calculate_color_information
Local Folder
Harddrive
MongoDB
Flask-Server
API
Controller
DOM-Service
Modal-Service
Image-Service
Category-Service
Genre-Service
Book-Service
Map-Service
Image-Modal-Component
Canvas-Component
Modal-Menu-Component
Cluster-Row-Component
Time-Slider-Component
Cluster-View-Component
Map-View-Component
User Interface
Frontend