Dokumentation Projekt „Project\_Image\_Recommender\_2024“

Studierende: Fouad Mokhtari, Lucas Müller

Professor: Florian Huber

Kurs: D4.1.2 Big Data Engineering

**Inhaltsverzeichnis**

**Part 1: Bildempfehlungssoftware**

1.1 Motivation / Ziele des Projekts S. 2

1.2 Programm Design S. 3

1.2.1 Berechnung der Features S. 3

1.3 Ähnlichkeit der Bilder S. 5

1. 4 Performance Analyse + Laufzeitoptimierung S. x

1.5 Diskussion des Programms S. x

**Part 2: Big Data Analyse**

2.1 Diskussion S. x

**Part 1: Bildempfehlungssoftware**

1.1 Motivation / Ziele des Projektes

Ziel des Projekts ist es, eine Python Software zu schreiben, um ähnliche Bilder möglichst schnell zu identifizieren.

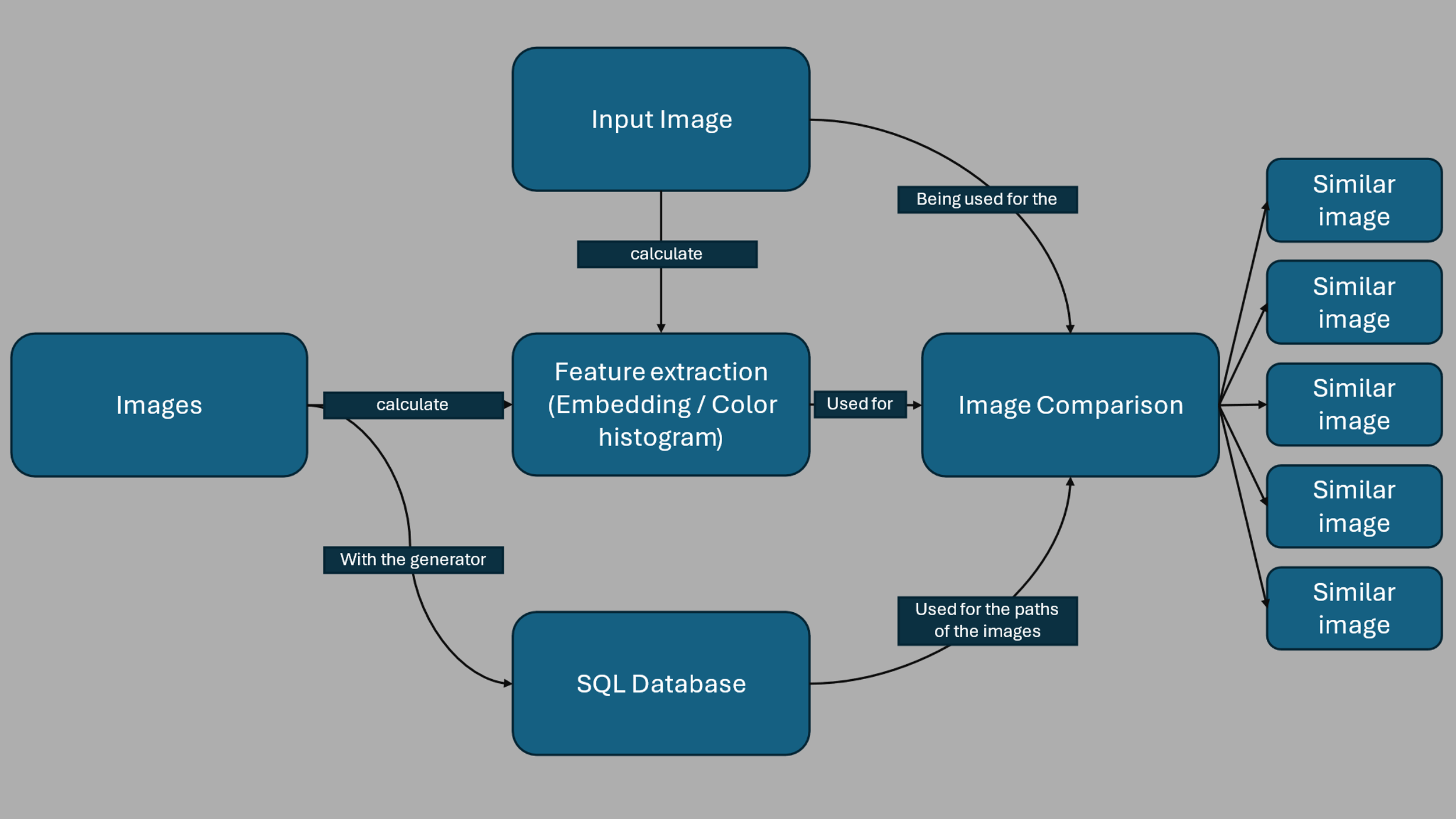
Dafür wurde uns ein Datensatz mit knapp 450.000 Bildern zur Verfügung gestellt. Die Bilder sind quer durchmischt und beinhalten viele verschiedene Motive, wir selbst haben uns nicht alle Bilder angeschaut.

Die Software soll dazu in der Lage sein, anhand eines Input Images die fünf ähnlichsten Bilder zu identifizieren. So ähnlich wie es bei Google Lens funktioniert. Das Input Bild wird verarbeitet und mit den vorhandenen Bildern verglichen. Der Output sind dann die fünf ähnlichsten Bilder.

Des Weiteren soll das Programm uns ermöglichen, anhand von fünf Input Bilder ebenfalls wieder die fünf ähnlichsten Bilder zu finden. Die Schwierigkeit besteht darin, Bilder zu identifizieren, die thematisch eine möglichst hohe Ähnlichkeit zu den Input Bildern haben. Geben wir beispielsweise zwei Fotos von einem Wald und drei Fotos von einem See in das Programm, sollten Fotos mit Wald und See als Ausgabe herauskommen.

Die Ausgabe der ähnlichsten Fotos soll innerhalb weniger Sekunden geschehen. Dafür ist es notwendig, dass die vorhandenen Bilder aus dem Datensatz bereits vorverarbeitet sind.

1.2 Programm Design



[1] Figur 1: Programm Sketch

Zuerst einmal bekommen wir die 450.000 Bilder und müssen diese laden. Dies basiert mithilfe unseres Generators. Damit erstellen wir uns zuerst einmal die SQL-Datenbank, um die nötigsten Informationen über unsere Bilder festzuhalten. In der SQL-Datenbank hat jedes Bild eine eigene ID. Außerdem speichern wir den Pfad des Bildes ab und die Größe. Die Pfade in der SQL-Datei benutzen wir dann, um durch diese zu iterieren und die Features der Bilder zu berechnen. Dies passiert einmal mit den Embeddings (näheres dazu später…) und einmal mit Farbhistogrammen.

Die berechneten Features werden gespeichert. Nun erfolgt dieselbe Berechnung mit dem Input-Foto. Die berechneten Features und die des Input-Fotos vergleichen wir nun miteinander. Wie dieser Schritt erfolgt, wird in Punkt 1.3 genauer erläutert. Der Vergleich zwischen den Input-Features und denen der Bilddatenbank liefert uns nun einen Wert. Dieser variiert je nach Methode. Wir sortieren die Bilder so, dass wir die fünf ähnlichsten Bilder ausgegeben bekommen.

1.2.1 Feature Berechnung

Zuerst einmal erläutern wir den Weg mithilfe von Embeddings.

Ein Embedding ist ein hochdimensionaler Vektor. Dieser Vektor repräsentiert die Features des Bildes. Um die Embeddings zu berechnen, haben wir uns für das ResNet50 Modell entschieden. Das ist ein neuronales Netz, welches auf dem ImageNet Datensatz vortrainiert wurde. Der ImageNet Datensatz beinhaltet Millionen von Bildern über tausende von Kategorien hinweg. Zuerst wird das Bild für das Modell vorverarbeitet. Dies beinhaltet beispielsweise das Anpassen der Größe auf 224 x 224 Pixel. Es prozessiert das Bild durch multiple Schichten, um das Embedding zu berechnen. Die Schichten im Netzwerk wenden convolutional Filter auf das Bild an, um Muster, Kanten und Ecken im Bild zu erkennen. Die Pooling Layer im Netzwerk reduzieren die Features, um deren Dimensionalität zu reduzieren und machen das Modell damit effizienter.

Das Modell spuckt einen 2048-dimensionalen Vektor aus, der die Features der Bilder beinhaltet. Diese Vektoren speichern wir in einer Pickle Datei, da diese später schnell zu laden ist.

Für diesen Ansatz benutzen wir eine Reihe von Bibliotheken. Zum einen verwenden wir die Bibliothek Dask. Dask ist eine parallele Computerbibliothek, die für die Verarbeitung großer Datenmengen entwickelt wurde. Es ermöglicht uns eine verteilte Berechnung und die effiziente Nutzung der Ressourcen.

Des Weiteren verwenden wir ProgressBar, welches ein Diagnosewerkzeug von Dask ist, dass Fortschrittsbalken für Dask-Berechnungen anzeigt.

Unser zweiter Ansatz beruht auf Farbhistogrammen.

Das Histogramm eines Bildes ist eine grafische Darstellung der Verteilung der Farbinformationen in einem Bild. Es zeigt, wie viele Pixel eines Bildes zu bestimmten Farbwerten gehören. Für ein Farbbild kann man das Histogramm für jede Farbkomponente (z.B. Rot, Grün, Blau in RGB) oder das HSV-Histogramm berechnen. HSV steht für Hue, Saturation und Value. Hue bezeichnet den Farbton, gemessen in Grad (0 bis 360). Aufgrund von Symmetrie reicht allerdings die Halbierung auf 0-180. Die Saturation ist die Intensität der Farbe, die von 0 (grau) bis 255 (volle Sättigung) reicht. Value ist die Helligkeit der Farbe, ebenfalls von 0 (schwarz) bis 255 (volle Helligkeit).

Um das HSV-Histogramm eines Bildes zu berechnen, konvertieren wir das Bild zunächst von BGR (Blau-Grün-Rot) nach HSV. Dafür werden die Farbwerte jedes Pixels in die entsprechenden Werte (H, S, V) umgewandelt. Zunächst werden die Farbwerte normalisiert, um in den Bereich [0, 1] zu fallen. Dann werden die Maximal- und Minimalwerte unter den normalisierten Farbkomponenten berechnet. Der Farbton wird basierend auf dem maximalen Farbwert berechnet. Wenn der berechnete Farbton negativ ist, wird er um 360° erhöht, um ihn in den Bereich [0, 360] zu bringen. Die Sättigung der Farbe gibt an, wie rein oder intensiv die Farbe ist und wird folgendermaßen berechnet.

(2) mit (Cmax = Maximalwert und Cmin = Minimalwert)

Die Helligkeit ist einfach der Maximalwert der normalisierten RGB-Komponenten.

(3)

Das extrahierte Histogramm ist ein mehrdimensionales Array. Die Dimensionen und die Form dieses Arrays hängen davon ab, wie viele Kanäle und Bins (Kategorien) wir beim Berechnen des Histogramms spezifizieren. In unserer Methode verwenden wir 32 Bins.

Um die Arrays später effizienter miteinander vergleichen zu können, wird das mehrdimensionale Array in ein eindimensionales Array umgewandelt.

Der dritte Ansatz ist eine Kombination der ersten beiden Ansätze und beinhaltet keine neuen Techniken.

1.3 Ähnlichkeit der Bilder

Um ähnliche Bilder verglichen mit dem Input-Bild zu finden, verwenden wir die Nearest-Neighbor Search. Als Eingabe erhält diese Funktion die Embeddings bzw. die Histogramme der Bilder. Es gibt mehrere Distanzmittel, mit der sich die Nearest-Neighbor Search durchführen lässt, wir verwenden die euklidische Distanz.

Die Nearest Neighbor Search mit der euklidischen Distanz funktioniert, indem sie für einen gegebenen Datenpunkt die nächsten Nachbarn in einem mehrdimensionalen Raum findet. Die euklidische Distanz misst dabei die "Luftlinie" zwischen zwei Punkten. Um dies zu erreichen, wird zunächst der Datensatz mit Punkten in einem n-dimensionalen Raum vorbereitet, wobei jeder Punkt durch einen Vektor dargestellt wird. Für einen neuen Punkt, der keinen bekannten Nachbarn hat, berechnet der Algorithmus die euklidische Distanz zu jedem Punkt im Datensatz. Diese Distanz wird durch die Wurzel der Summe der quadrierten Differenzen der jeweiligen Koordinaten berechnet. Nachdem die Distanzen berechnet wurden, wählt der Algorithmus die k Punkte aus dem Datensatz aus, die die kleinsten euklidischen Distanzen zu dem neuen Punkt aufweisen. Diese k Punkte gelten als die nächsten Nachbarn des neuen Punktes. Wenn k gleich 1 ist, gibt der Algorithmus den Punkt im Datensatz zurück, der die kleinste Distanz zu dem neuen Punkt hat. Wenn k größer als 1 ist, gibt der Algorithmus die k Punkte mit den kleinsten Distanzen zurück.

Zusatz für die Methode, bei der fünf Bilder als Input in die Software gefüttert werden:

Um die ähnlichsten Bilder zu finden, die im Idealfall thematisch zu allen fünf Input-Bildern passen, berechnen wir das durchschnittliche Embedding. Dafür wird der Mittelwert der 2048 Dimensionen für alle Input-Fotos genommen. Dieses neue Embedding wird nun dafür verwendet, ähnliche Bilder in unserem Datensatz zu finden.

Zusatz für die Methode, bei der fü nf Bilder als Input in die Software gefüttert werden:

Der Prozess umfasst drei Hauptschritte: Berechnung der durchschnittlichen Histogramme, Auffinden der ähnlichsten Bilder basierend auf diesen Histogrammen, und Anzeigen der Bilder zur Visualisierung.

Der erste Schritt des Prozesses besteht darin, das durchschnittliche Farbhistogramm für eine Reihe von Eingabebildern zu berechnen. Für jeden angegebenen Bildpfad wird Bild geladen und sein Farbhistogramm im angegebenen Farbraum berechnet. Die Histogramme aller Eingabebilder werden dann gemittelt, um ein einzelnes, zusammengefasstes Histogramm zu erstellen. Dieses durchschnittliche Histogramm dient als repräsentatives Merkmal der Eingabebilder und wird in den folgenden Schritten verwendet, um ähnliche Bilder im Datensatz zu finden.

1.4 Performance Analyse + Laufzeitoptimierung

Beim Plotten der Bilder hatten wir ebenfalls am Anfang zu ineffiziente Funktionen. Zu Beginn des Projekts haben wir die Bilder anhand der Pfade vom Computer geöffnet, was natürlich nicht effizient genug war. Der zweite Versuch basierte auf der Image Bibliothek, allerdings hat auch dieser Ansatz zu lange gedauert. Die Funktion, die wir nun auch zum Visualisieren der Bilder verwenden nutzt Matplotlib, um die Bilder schnell und effizient zu laden.

Um fünf ähnliche Bilder aus dem kompletten Datensatz zu finden, benötigt die Embedding-Version circa sechs bis sieben Sekunden. Davon gehen circa vier bis fünf Sekunden dafür drauf, die Pfade der ähnlichsten Bilder herauszufinden und der Rest der Zeit wird benötigt, um die Bilder zu plotten. Um anhand von fünf Input Fotos die ähnlichsten zu finden, benötigt das Programm circa fünf Sekunden. Davon wird innerhalb einer Sekunde das durchschnittliche Histogramm der Eingabebilder berechnet, drei Sekunden braucht der Code dann, um die ähnlichsten Pfade zu finden und der Rest der Zeit wird für das Plotten der Bilder verwendet. Allerdings variieren diese Zahlen auch abhängig vom Computer und von der Größe der Eingabebilder. Daher kann es auch sein, dass bestimmte Teile der Funktion mal länger brauchen und mal kürzer.   
Um die fünf ähnlichsten Fotos anhand der Histogramme zu finden, benötigt das Programm circa fünf Sekunden und mit fünf Input Fotos ebenfalls. Die Aufteilung der Zeit ist ähnlich wie bei dem Code mit den Histogrammen. Die meiste Zeit (ca. 2/3) werden die Features des Input-Bildes mit den Features der anderen Bilder verglichen. Die restliche Zeit wird verwendet, um die Features des Input-Bildes zu berechnen und zum Schluss, um die Bilder zu visualisieren.

Im Laufe des Projekts mussten wir immer wieder an verschiedenen Stellen Funktionen optimieren und verändern, um den Code zu optimieren und an Laufzeit zu sparen.

1.5 Diskussion des Programms

Unser Programm funktioniert für die Anzahl der Bilder in der Datenbank ohne Probleme (in der Datenbank sind 450.000 Bilder). Zugegebenermaßen wären allerdings fünf bis sieben Sekunden für eine Suchmaschine etwas langsam. Nutzer sind dort schnellere Ladezeiten gewöhnt. Allerdings hängt die Laufzeit unseres Programms auch deutlich mehr von Faktoren ab wie Größe des Eingabebildes, dem verwendeten Computer usw...

Die größten Limitierungen ist das Vergleichen der Feature vom Input-Bild mit den Bildern in der Datenbank. Um dort auch für eine deutlich größere Zahl von Bildern umgehen zu können, müsste man die Suche nach den ähnlichsten Bildern umgestalten. Ein Ansatz wäre es, die Bilder bei der Berechnung der Features zu labeln, sodass man nicht alle Bilder mit dem Input-Bild vergleichen müsste. Wenn der Nutzer dann ein Bild von einem Wald eingeben würde, müsste man dieses nur mit anderen Waldbildern vergleichen und könnte sich den Rest sparen.

Des Weiteren gäbe es auch die Möglichkeit, von der exakten Suche nach den fünf ähnlichsten Bildern auf eine Suche auf eine nicht so feine Suche umzusteigen. Diese würde die Laufzeit weiter verringern, ggf. allerdings eine schwächere Ausgabe erzeugen.

**Part 2: Big Data Analyse**

2.1 Diskussion

Ein Bild, das Text, Screenshot, Karte enthält.

Automatisch generierte Beschreibung

[4] Figur 2: TSNE-Plot aller Bilder aus der Datenbank

Die Figur zeigt die Position aller Bilder in einem 2D-Plot. Ähnliche Bilder haben einen geringen Abstand zueinander und Bilder mit wenig / keinen Gemeinsamkeiten haben einen größeren Abstand zueinander. Wie man sieht, bilden sich dadurch gewisse Cluster mit Bildern ähnlicher Art. Es zeigt sich, dass ähnliche Bilder auch ähnliche Positionen aufweisen.

Um die 2048-dimensionalen Embeddings auf 2 (x und y Position) zu reduzieren, nutzen wir t-SNE. Dies ist ein Verfahren zur Dimensionsreduktion, das besonders gut darin ist, die Struktur in den Daten zu bewahren, insbesondere wenn es um Cluster oder Gruppen ähnlicher Punkte geht. Zuerst berechnet t-SNE die Wahrscheinlichkeiten Pij , dass zwei Punkte i und j in den ursprünglichen 2048-dimensionalen Daten ähnlich sind. Diese Wahrscheinlichkeiten basieren auf den euklidischen Distanzen zwischen den Datenpunkten in der hohen Dimension. Nun werden die Daten in den 2D-Raum projiziert. Es werden Wahrscheinlichkeiten Qij berechnet, die die Ähnlichkeiten der Punkte im neuen 2D-Raum darstellen. Ziel ist es, die Verteilung der Ähnlichkeiten Pij in der hohen Dimension möglichst gut in der niedrigen Dimension durch Qij zu approximieren.

Die Visualisierung ist eigentlich nicht für solch eine Vielzahl von Bildern geeignet, allerdings erkennt man dennoch gut, ob Bilder ähnlich sind oder nicht.