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.

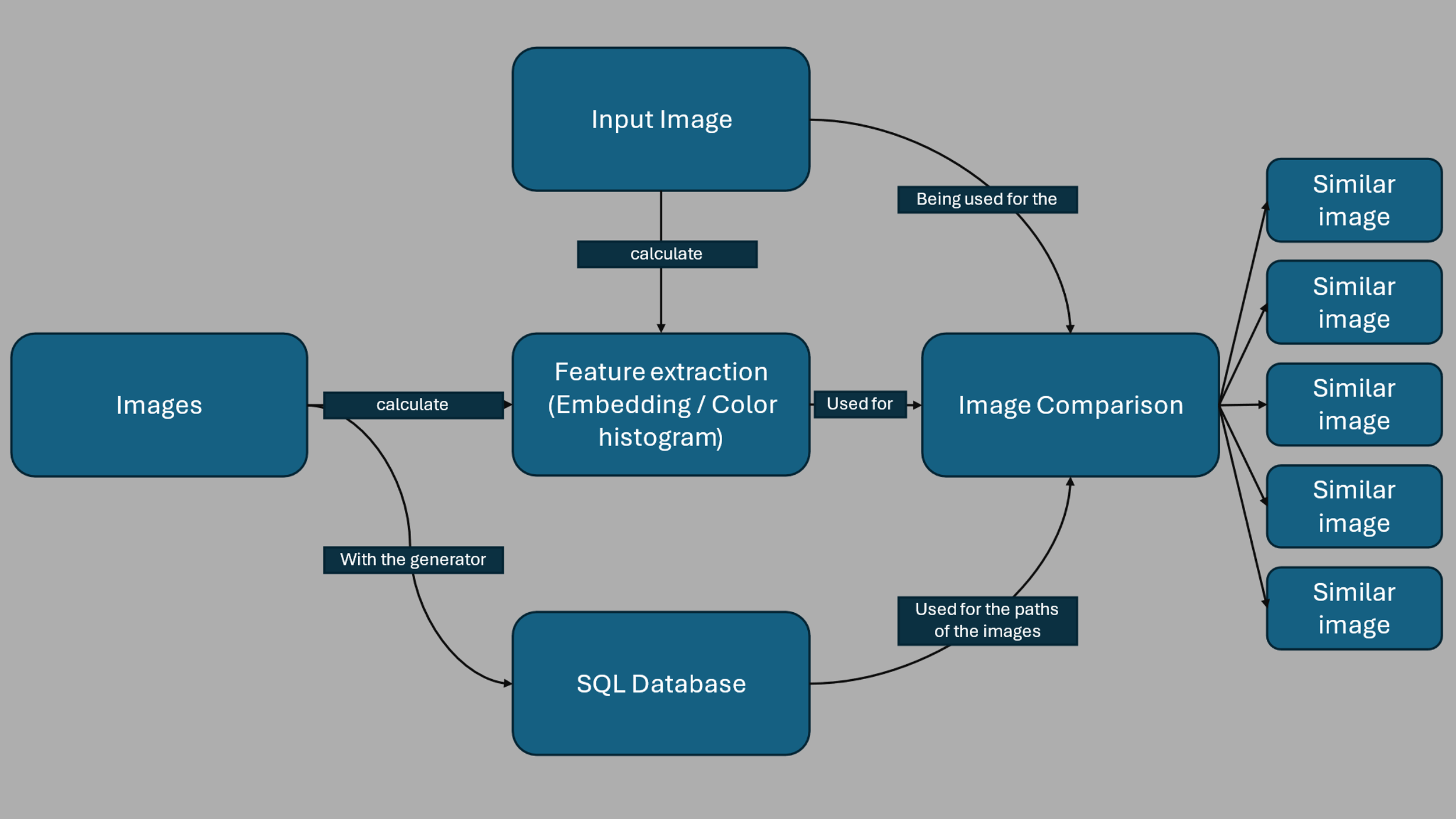
Dafür wurde uns ein Datensatz mit knapp 450.000 Bildern zur Verfügung gestellt. Die Bilder sind quer durchmischt und beinhalten viele verschiedene Motive, wir selbst haben uns ebenfalls nicht alle Bilder angeschaut.

Die Software soll dazu in der Lage sein, anhand eines Input Images die 5 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 x 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

Wie eben beschrieben, haben wir zwei unterschiedliche Ansätze, um ähnliche Bilder zu finden. Daher benötigen wir auch zwei unterschiedliche Methoden, um Bilder miteinander zu vergleichen.

Um die Ähnlichkeit zwischen zwei Bildern anhand der Embeddings festzustellen, benutzen wir die Cosinus Ähnlichkeit. Die Cosinus Ähnlichkeit berechnet den Cosinus der Winkel zwischen zwei Vektoren. Dabei ergibt sich als Ergebnis eine Zahl zwischen 1 (gleicher Vektor) und -1 (entgegengesetzter Vektor). Die Cosinus-Ähnlichkeit ist nützlich, um hochdimensionale Vektoren miteinander zu vergleichen, welche uns im Falle der Embedding Berechnung vorliegt.

Der Vorteil der Cosinus-Ähnlichkeit liegt darin, dass die Methode die Orientierung der Vektoren anstatt deren Größe (Magnitude). Vektoren können unterschiedliche Größen haben und dennoch ähnlichen Inhalt widerspiegeln. Des Weiteren ermöglichen die normierten Ergebnisse eine klare Interpretation. Außerdem ist die Cosinus-Ähnlichkeit einfach und effizient. Sie erfordert nur die Berechnung des Skalarprodukts und der Normen des Vektors, was insbesondere bei großen Datensätzen wichtig ist.

Dadurch, dass die Cosinus-Ähnlichkeit nur die Richtung der Vektoren betrachtet, geht die Information über die Länge (Magnitude) der Vektoren verloren. In sehr hochdimensionalen Räumen kann die Cosinus-Ähnlichkeit an Aussagekraft verlieren, da die meisten Vektoren tendenziell orthogonal zueinander sind.

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.

Bei der Methode mithilfe des Farbistogramms verwenden wir die Bhattacharyya-Distanz, eine statistische Methode zur Messung der Ähnlichkeit zwischen zwei Verteilungen. Für jedes Histogramm in unserer Datenbank berechnen wir die Ähnlichkeit zwischen dem Histogramm des Eingabebildes und dem gespeicherten Histogramm des Vergleichsbildes. Dies erfolgt über folgenden Weg.

Die Bhattacharyya-Distanz wird definiert als:

(4)

wobei BC (P, Q) der Bhattacharyya-Koeffizient ist, der die Überlappung zwischen den beiden Verteilungen misst:

(5)

Hierbei gilt:

* Q(i) und P(i) sind die Wahrscheinlichkeiten für die i-te Kategorie (Bin) in den Verteilungen P und Q,
* n ist die Anzahl der Kategorien.

Der Bhattacharyya-Koeffizient liegt im Bereich [0, 1]. Ein Wert von 1 bedeutet, dass die Verteilungen identisch sind. Ein Wert von 0 bedeutet, dass die Verteilungen keine Überlappungen haben. Da die Distanz mithilfe von (4) berechnet wird, führt ein Koeffizient von 1 zu einer Distanz von 0, was maximale Ähnlichkeit bedeutet. Je kleiner der Bhattacharyya-Koeffizient, desto größer die Distanz und desto unähnlicher sind die Verteilungen.

Die Wahl der Bhattacharyya-Distanz (im Folgenden BD genannt) hat verschiedene Vorteile. Zum einen ist sie robust gegenüber kleinen Schwankungen in den Histogrammen, da sie die Überlappung der Verteilungen direkt misst. Außerdem ist die BD relativ einfach und erfordert keine komplexen Operationen.

Allerdings kann die BD anfällig für Ausreißer sein, insbesondere wenn die Histogramme nicht normiert sind. Hinzu kommt, dass die Distanz nicht die Reihenfolge der Bins berücksichtigt, was für einige Anwendungen von Bedeutung sein könnte.

Trotz dieser Nachteile haben für uns wir diese Methode entschieden, da wir die Histogramme normiert haben und die Reihenfolge der Bins für unseren Use-Case nicht die größte Bedeutung hat.

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

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

1.4 Performance Analyse + Laufzeitoptimierung

1.5 Diskussion des Programms

**Part 2: Big Data Analyse**

2.1 Diskussion

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.

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.