# 1. Ziel des Projektes

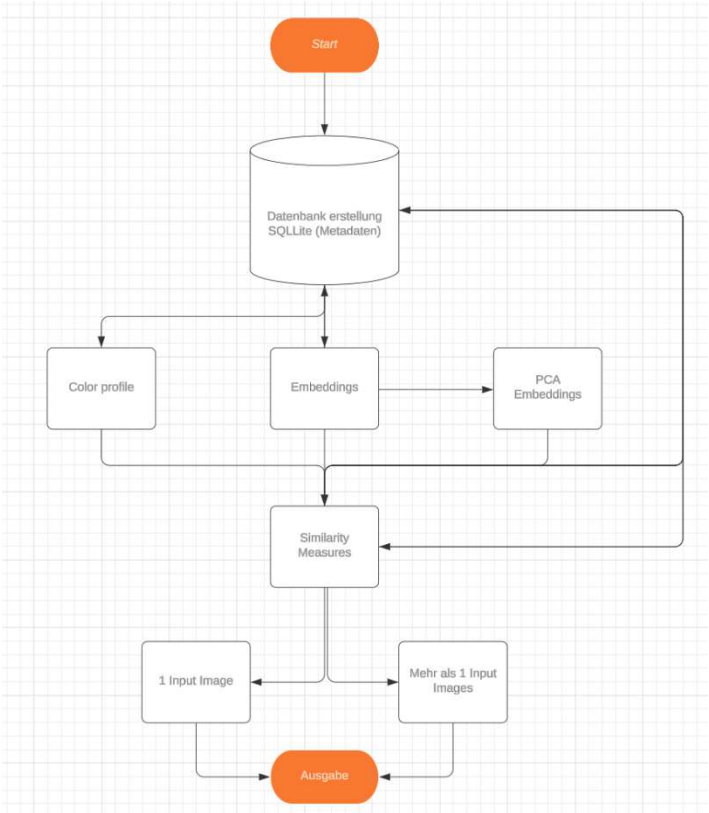
Im Rahmen des Vertiefungsmoduls „Big Data Analysis“ haben wir in Zweiergruppen an unserem ersten Big-Data-Projekt gearbeitet.

Das erste Ziel dieses Projekts war es, eine Python-Software zu entwickeln, die eine möglichst genaue und effiziente Suche in einem Bilddatensatz von etwa 500.000 Bildern ermöglicht. Die Software sollte in der Lage sein, zu einem oder mehreren vorgegebenen Input Bilder die 5 ähnlichsten Bilder aus dem Datensatz zu identifizieren und auszugeben.

Das zweite Ziel bestand darin, die Positionen der Bilder in einem 2D- oder 3D-Raum zu visualisieren, um zu überprüfen, wie gut ähnliche Bilder gruppiert werden können. Diese Aufgabe sollte uns helfen, den großen Bilddatensatz besser zu verstehen und erste Erfahrungen im Umgang mit solch umfangreichen Daten zu sammeln. Durch die Anwendung von Algorithmen zur Dimensionsreduktion wie UMAP und t-SNE haben wir gelernt, wie man hochdimensionale Daten auf eine anschauliche Weise darstellt und welche Herausforderungen dabei auftreten.

Im Rahmen des Projektes haben wir im Pair-Programming  an einem leistungsstarken Rechner zusammengearbeitet.

# 2. Program design (sketch + text)



**Hauptelemente der Software:**

1. Datenbankerstellung (SQLite)

    - Funktion: Die Software beginnt mit der Erstellung einer SQLite-Datenbank, die alle Bildmetadaten wie Dateipfade und Bildgrößen speichert.

    - Interaktion: Diese Datenbank dient als zentraler Speicher für die effiziente Verwaltung und Abfrage der Bildinformationen.

2. Color Histogramm (Farbprofile):

    - **Funktion:** Die Software berechnet Farbhistogramme, um die Farbverteilung der Bilder zu analysieren und zu vergleichen.
    - **Interaktion:** Diese Farbprofile werden berechnet und in einer Pickle-Datei gespeichert, um sie für spätere Abfragen schnell verfügbar zu machen.

3. Embeddings:
    - **Funktion:** Die Embeddings werden mit Hilfe des vortrainierten ResNet50-Modells erstellt, um hochdimensionale Vektoren mit komplexen Informationen der Bilder zu generieren.

    - **Interaktion:** Diese Embeddings werden in einer Pickle-Datei gespeichert, um die spätere Ähnlichkeitssuche zu beschleunigen.

4. PCA Embeddings:
    - **Funktion:** Die Software verwendet PCA, um die Dimensionen der Embeddings zu reduzieren und die Berechnungen effizienter zu gestalten.

    - **Interaktion:** Die reduzierten Embeddings werden in Pickle-Dateien gespeichert und für die Ähnlichkeitsbewertung verwendet.

5. Similarity Measurers:
    - **Funktion:** Zunächst werden für die Input Bilder jeweils Farbhistogramme und Embeddings berechnet. Diese werden dann mit den gespeicherten Farbprofilen und Embeddings der Pickle Dateien verglichen.

    - **Vergleichsmethoden:** Für die Farbhistogramme werden verschiedene Methoden verwendet, wie 'correlation', 'bhattacharyya'. Die Embeddings werden mithilfe der Cosine Similarity verglichen.

    - **Interaktion:** Basierend auf den berechneten Similarity Measures wird eine Anfrage an die Datenbank gestellt, um über die UUIDs die Dateipfade der ähnlichen Bilder abzurufen. Diese Bilder werden anschließend verwendet, um die Ergebnisse visuell darzustellen, indem sie dem Benutzer zur Ansicht zusammen mit dem Input Bilder geplottet werden.

6. Input Images + Ausgabe:
    - **Funktion:** Der Benutzer kann ein bis fünf Input Bilder festlegen, um ähnliche Bilder im Datensatz zu finden.

- **Für ein einzelnes Bild:** Die Software vergleicht das Bild mit den Vektoren in den Pickle-Dateien und findet die 5 besten Übereinstimmungen, basierend auf den Similarity Measures.

- **Für bis zu fünf Bilder:** Die Software berechnet den Mittelwert der Embeddings und Farbhistogramme der Input Bilder und sucht dann nach den 5 besten Übereinstimmungen, die idealerweise mit allen Input Bildern eine gewisse Ähnlichkeit besitzen.

- **Interaktion:** Die Software plottet die 5 ähnlichsten Bilder und stellt sie visuell den Eingabebildern gegenüber, um die Übereinstimmungen anschaulich darzustellen.

## Verwendete Libraries und Key Features:

**Speicherung und Serialisierung:**

- pickle: Serialisierung von Python-Objekten, wie z.B. Farbprofile und Embeddings, um diese in Pickle-Dateien zu speichern.

**Numerische und wissenschaftliche Berechnungen:**

- numpy: Durchführung numerischer Berechnungen und Arbeiten mit Arrays.

**Bildverarbeitung:**

- cv2 (OpenCV): Umfassende Bildverarbeitung, einschließlich:

  - Farbhistogramme: Berechnung und Vergleich von Farbhistogrammen.

  - Ähnlichkeitsmessung: Vergleich von Histogrammen mit verschiedenen Methoden wie Korrelation, Chi-Quadrat, Intersection und Bhattacharyya-Distanz.

- PIL (Pillow): Bildverarbeitung und -manipulation, insbesondere das Laden und Vorverarbeiten von Bildern.

**Maschinelles Lernen und neuronale Netze:**

- torch, torchvision: Arbeiten mit neuronalen Netzen, einschließlich der Erstellung von Embeddings mit vortrainierten Modellen wie ResNet50.

- CUDA: Nutzung von GPU-Beschleunigung, um die Berechnung von neuronalen Netzen und anderen rechenintensiven Operationen zu beschleunigen.

**scikit-learn:**

- IncrementalPCA, KernelPCA, PCA: Reduktion der Dimensionen von großen Datensätzen und Erfassung von Zusammenhängen zwischen Bildmerkmalen.

- cosine_similarity: Berechnung der Cosinus-Ähnlichkeit zwischen Bildvektoren.

**Visualisierung:**

- matplotlib: Erstellen von Diagrammen und Plots zur Visualisierung der Ergebnisse.

**Parallele Verarbeitung:**

- ThreadPoolExecutor: Ermöglicht die parallele Ausführung von Aufgaben, um die Verarbeitungsgeschwindigkeit zu erhöhen.

**Fortschrittsanzeige:**

- tqdm: Zeigt Fortschrittsbalken an, um den Fortschritt bei längeren Aufgaben zu vermitteln. Ist bei Längeren Prozessen gut, um Fortschritt zu sehen darf man sich aber nicht drauf verlassen.

**Datenbankmanagement:**

- sqlite3: Verwaltung und Abfrage von Bildmetadaten in einer SQLite-Datenbank.

**metadata.py:**

Der Ablauf beginnt mit der Erstellung einer SQLite-Datenbank, die eine Tabelle für die extrahierten Informationen enthält. Das Programm durchläuft das Verzeichnis in 500er-Schritten und weist jedem Bild im JPG-, JPEG- oder PNG-Format eine UUID zu. Dabei werden Dateiname, Pfad, Verzeichnis und Pixelmaße erfasst und gesammelt. Nach jedem Schritt werden die Daten in die Tabelle eingetragen. Nicht lesbare Dateien werden im Logfile vermerkt. Checkpoints und eine Pickle-Datei sichern den Fortschritt, um bei Abstürzen oder Problemen Datenverlust zu verhindern.

**Image_similarity.py:**

Zunächst wird ein vortrainiertes ResNet50-Modell aus PyTorch geladen. Die letzte Schicht des Modells wird entfernt, da diese ausschließlich für die Klassifikation verwendet wird, die bei der Ähnlichkeitserkennung nicht benötigt wird.

Das Modell wird anschließend auf die GPU geladen, falls eine vorhanden ist, um die Performance zu steigern. Die Bilder werden vor der Verarbeitung auf 224x224 Pixel skaliert und normalisiert.

Eine Verbindung zur SQLite-Datenbank (image_metadata.db) wird hergestellt, um die UUIDs und Dateipfade abzurufen. Auch hier wird in Batches gearbeitet, um Speicherprobleme zu vermeiden und die Effizienz zu steigern.

Die vorverarbeiteten Bilder werden dann in das ResNet50-Modell geladen, um die Features zu extrahieren und Embedding-Vektoren zu erzeugen. Diese Vektoren werden zusammen mit den zugehörigen UUIDs in einer Pickle-Datei (combined_embeddings.pkl) gespeichert.

Nach jedem Batch wird der Speicher bereinigt, um mögliche Speicherprobleme zu verhindern.

**Similarity_Measures**

Unsere Hauptmethoden zur Erkennung von Bildähnlichkeiten sind in drei Dateien gegliedert:

**Ähnlichkeitserkennung mit ResNet50 Image Embeddings:**

In diesem Ansatz wird ein vortrainiertes ResNet50-Modell verwendet, wobei der letzte Layer entfernt wurde, um Bild-Embeddings zu extrahieren. Diese Embeddings werden in Batches aus einer zuvor erstellten Pickle-Datei geladen. Ein Eingabebild wird auf 224x224 Pixel skaliert, in einen Tensor umgewandelt und normalisiert. Anschließend wird das Bild durch das Modell verarbeitet, um ein Vektor-Embedding zu erzeugen. Dieses Embedding wird dann mit den Embeddings aus der Datenbank verglichen, wobei die Ähnlichkeit über die Cosine-Similarity berechnet wird. Die Ergebnisse werden nach absteigender Ähnlichkeit sortiert, und die UUIDs der fünf ähnlichsten Bilder werden zurückgegeben. Die Pfade dieser Top-Bilder werden aus der Datenbank abgerufen und visualisiert.

**Ähnlichkeitserkennung mit Farbhistogrammen:**

Bei dieser Methode wird die Farbverteilung der Bilder anhand von vier verschiedenen Methoden verglichen: Korrelation, Chi-Square, Intersection und Bhattacharyya. Zunächst wird eine Pickle-Datei mit den Farbhistogrammen der Bilder geladen. Der Benutzer wählt die Vergleichsmethode, und ein Histogramm des Eingabebildes wird erstellt, normalisiert und flach gemacht. Falls mehrere Eingabebilder vorhanden sind, wird ein Durchschnittshistogramm berechnet. Die Histogramme der Datenbank werden in Batches mit dem des Eingabebildes verglichen. Je nach gewählter Methode werden die Ergebnisse sortiert: Bei Korrelation und Intersection werden die höchsten Werte, bei Chi-Square und Bhattacharyya die niedrigsten Werte als ähnlichste ausgewählt. Auch hier werden die Pfade der fünf ähnlichsten Bilder aus der Datenbank abgerufen und visualisiert.

**PCA-Dimensionenreduktion der Embeddings:**

In diesem Abschnitt werden die Bild-Embeddings mithilfe von PCA auf eine niedrigere Dimension reduziert. Diese reduzierten Embeddings werden dann zur Ähnlichkeitserkennung verwendet. Der Vergleich der Embeddings erfolgt wie bei der ResNet50-Methode, jedoch unter Verwendung der durch PCA transformierten Daten. Nach der Berechnung der Cosine-Similarity werden die fünf ähnlichsten Bilder ermittelt und ausgegeben.

# 3. **Image similarity**

Im Rahmen des Projektes haben wir verschiedene Methoden zur Messung der Similarity von Bildern kennengelernt und implementiert. Im Wesentlichen habe wir drei Ansätze verwendet: Color Histogramms, Embedding-basierte Vergleiche und PCA-reduzierte Embeddings. Im Folgenden beschreibe und vergleichen wir diese Methoden sowie ihre Vor- und Nachteile.

## 1. Color Histogramms Similarity:

**Beschreibung und Erklärung:**

Diese Methode basiert auf der Analyse der Farbverteilung in Bildern mithilfe von Farbhistogrammen. Ein Color Histogramm zeigt, wie häufig bestimmte Farben in einem Bild vorkommen. Um die Ähnlichkeit zwischen zwei Bildern zu bestimmen, haben wir die Farbhistogramme verglichen. Dabei kamen verschiedene Metriken zum Einsatz:

- **Korrelation:** Misst die lineare Beziehung zwischen den Farbverteilungen zweier Bilder. Wenn die Histogramme stark korreliert sind, bedeutet dies, dass die Farbverteilungen ähnlich sind.

- **Chi-Quadrat:** Misst die Unterschiede zwischen den erwarteten und den tatsächlichen Häufigkeiten in den Farbverteilungen. Es bewertet, wie stark die Histogramme voneinander abweichen.

- **Intersection:** Misst den Überschneidungsbereich zwischen den Farbverteilungen. Diese Metrik berechnet, wie viel von einem Histogramm mit dem anderen übereinstimmt.

- **Bhattacharyya-Distanz:** Misst die Ähnlichkeit zwischen zwei Wahrscheinlichkeitsverteilungen. Es handelt sich um eine probabilistische Metrik, die die Wahrscheinlichkeit beschreibt, dass zwei Verteilungen ähnlich sind.

**Vorteile Color Histogramm:**

- **Robustheit gegenüber Skalierung und Rotation:** Da Color Histogramme keine räumlichen Informationen berücksichtigen, sind sie unempfindlich gegenüber Änderungen in der Bildgröße oder Rotation. Dies macht sie robust bei der Erkennung von Bildern mit unterschiedlichen Ausrichtungen oder zuschnitten.

- **Effiziente Berechnung**: Die Berechnung von Color Histogrammen ist einfach und schnell, was sie zu einer guten Wahl für die schnelle Analyse und den Vergleich großer Bilddatenbanken macht.

**Nachteile Color Histogramm:**

- **Verlust räumlicher Informationen:** Diese Methode berücksichtigt nicht die räumliche Anordnung der Farben im Bild. Dadurch können Bilder mit ähnlichen Farbverteilungen, aber unterschiedlicher Struktur, als ähnlich eingestuft werden, obwohl sie es nicht sind.

- **Eingeschränkte Genauigkeit:** Besonders bei komplexen Bildern, die viele Details oder unterschiedliche Objekte enthalten, reicht die Color Histogramm-Methode oft nicht aus, um präzise Ähnlichkeiten zu erkennen.

**Korrelation:**

- **Vorteil:** Robust gegenüber linearen Verschiebungen in der Farbverteilung; funktioniert gut bei Bildern, die ähnliche Farbbereiche aufweisen.

- **Nachteil:** Weniger effektiv bei komplexen Bildern, die nicht streng linear verwandte Farbverteilungen aufweisen.

**Chi-Quadrat:**

- Vorteil: Empfindlich gegenüber Unterschieden in der Farbverteilung, gut geeignet für Bilder mit deutlich unterschiedlichen Farbmustern.

- Nachteil: Kann empfindlich auf kleine Änderungen und Rauschen reagieren, was zu ungenauen Ergebnissen führen kann.

**Intersection:**

- Vorteil: Einfach und effizient, liefert gute Ergebnisse, wenn zwei Bilder stark überlappende Farbverteilungen haben.

- Nachteil: Weniger empfindlich bei feinen Unterschieden in der Farbverteilung; berücksichtigt nicht die Gesamtgröße der Histogramme.

**Bhattacharyya-Distanz:**

- Vorteil: Erfasst sowohl Form als auch Lage der Farbverteilungen; geeignet für komplexe Verteilungen.

- Nachteil: Rechenintensiver als andere Metriken; möglicherweise weniger effizient bei sehr großen Datensätzen.

## 2. Embedding-basierte Methode

Beschreibung und Erklärung:

Die Embedding-basierte Methode nutzt ResNet50, um aus Bildern Merkmalsvektoren (Embeddings) zu extrahieren, hierbei haben den letzten Layer entfernt da wir diesen nicht für unsere Zwecke benötigen. Diese Embeddings repräsentieren die wichtigen Merkmale eines Bildes in einem hochdimensionalen Raum. Um die Ähnlichkeit zwischen zwei Bildern zu bestimmen, haben wir die Cosine Similarity zwischen den Embeddings berechnet.

**Vorteile:**

- **Hohe Genauigkeit:** Tiefe neuronale Netze sind in der Lage, komplexe Muster in Bildern zu erfassen. Dadurch ist diese Methode oft sehr genau, insbesondere bei Bildern mit vielen Details.

- **Vielseitigkeit:** Diese Methode funktioniert gut bei einer Vielzahl von Bildern, unabhängig von deren Inhalt oder Komplexität.

**Nachteile:**

- **Rechenintensiv:** Die Berechnung von Embeddings erfordert erhebliche Rechenressourcen, insbesondere bei großen Datasets. Hier war die Nutzung von GPUs (z.B. über CUDA) notwendig, um die Berechnungen effizient durchzuführen.

- **Schwer nachvollziehbar:** Da Embeddings in einem hochdimensionalen Raum liegen, ist es schwierig, die spezifischen Merkmale zu identifizieren, die zur Ähnlichkeit zwischen zwei Bildern beitragen.

## 3. PCA-reduzierte Embeddings

**Beschreibung und Erklärung:**

Um die Effizienz zu steigern, habe ich PCA (Principal Component Analysis) eingesetzt, um die Dimensionen der Embeddings zu reduzieren. Diese Methode zielt darauf ab, die Berechnungen zu beschleunigen und weniger Speicher zu verbrauchen, indem die wichtigsten Merkmale beibehalten und die unwichtigen verworfen werden. Hierbei haben wir auch eine kleine Parameterstudie gemacht und verschiedene PCA's ausprobiert [50, 75, 100, 200]

**Vorteile:**

- **Effizienzsteigerung:** Durch die Reduktion der Dimensionen sind die Berechnungen schneller und weniger speicherintensiv, was besonders bei großen Datensätzen von Vorteil ist.

- **Erhaltung wichtiger Merkmale:** PCA hilft dabei, die wichtigsten Informationen im Datensatz zu bewahren, sodass die Qualität der Ähnlichkeitsbewertung weitgehend erhalten bleibt.

**Nachteile:**

- **Möglicher Informationsverlust:** Bei der Reduktion der Dimensionen kann es zu einem Verlust von wichtigen Details kommen, was die Genauigkeit der Ähnlichkeitsbewertung beeinträchtigen kann.

- **Anpassung erforderlich:** Es ist oft eine Herausforderung, die optimale Anzahl der zu behaltenden Dimensionen zu bestimmen, um den besten Kompromiss zwischen Effizienz und Genauigkeit zu erzielen.

**Vergleich der Methoden**

- **Genauigkeit:** Die embedding-basierte Methode liefert die genauesten Ergebnisse, da sie komplexe Bildmerkmale erfasst. Die PCA-reduzierte Methode bietet einen guten Kompromiss zwischen Genauigkeit und Effizienz. Die Farbhistogramm-Methode ist weniger genau, dafür aber schneller und einfacher zu berechnen.

- **Rechenaufwand:** Farbhistogramme sind am wenigsten rechenintensiv, während die embedding-basierte Methode den höchsten Rechenaufwand erfordert. Die PCA-reduzierte Methode ermöglicht eine schnellere Verarbeitung, während sie dennoch eine gute Genauigkeit bewahrt.

- **Anwendungsbereich:** Farbhistogramme eignen sich gut für Bilder, bei denen die Farbe das Hauptunterscheidungsmerkmal ist, während Embeddings besser für komplexe, inhaltlich reichhaltige Bilder geeignet sind.

## Fazit:

Durch die Arbeit an den Similarity Measures haben wir die Stärken und Schwächen verschiedener Methoden kennengelernt und verstanden, wie sie effektiv eingesetzt werden können. Jede Methode hat ihre spezifischen Vorteile und Einschränkungen, die sie je nach Projektanforderungen mehr oder weniger geeignet machen.

**Color histograms:**

Sind eine einfache und schnelle Methode, um die Farbverteilung von Bildern zu vergleichen. Sie eignen sich besonders für Aufgaben, bei denen Farbe das Hauptunterscheidungsmerkmal ist. Allerdings berücksichtigen sie nicht die räumliche Anordnung der Farben, was dazu führen kann, dass sie subtile Unterschiede im Bildinhalt übersehen

**Embedding-basierte Methoden:**

Nutzen neuronale Netze wie RestNET50 oder InceptionNetV3, um komplexe Merkmale wie Textur und Form zu erfassen, was sie für anspruchsvolle Aufgaben wie die Objekterkennung ideal macht. Diese Genauigkeit kommt jedoch mit hohem Rechenaufwand und einer gewissen Komplexität in der Interpretation der Ergebnisse.

**PCA-reduzierte Embeddings:**

Bieten einen Kompromiss zwischen Genauigkeit und Effizienz, indem sie die Dimension der Daten verringern und so die Verarbeitung beschleunigen, allerdings auf Kosten potenzieller Informationsverluste. Hierbei ist es wichtig einen guten Mittelwert zu finden zwischen Dimensions Reduzierung und intakt Haltung komplexer Informationen der Embeddings.

In der Praxis hängt die Wahl der Methode stark von den spezifischen Zielen des Projekts ab. Farbhistogramme sind nützlich für schnelle Kategorisierungen, während Embeddings bei komplexen Bildanalysen unübertroffen sind. PCA-reduzierte Embeddings bieten eine ausgewogene Lösung, wenn sowohl Genauigkeit als auch Effizienz wichtig sind.

# 4. Performance analysis ("profiling") + runtime optimization

## Benutzte Hardware:

- CPU-Kerne (physisch): 8
- CPU-Kerne (logisch): 16
- CPU-Frequenz: 4.80 GHz
- 32 GB Arbeitsspeicher
- NVIDIA GeForce RTX 3080
- CUDA Cores: 8704
- Speichergröße: 10 GB

In unserem Image Recommender gab es mehrere Engpässe, die die Performance beeinflussten, insbesondere in Bezug auf die Verarbeitungsdauer und die Effizienz der eingesetzten Methoden.

## Hauptengpässe und Laufzeiten:

**Erstellung der Embeddings:**

- **Laufzeit:** 15 Stunden und 48 Minuten für den gesamten Datensatz. (3,42 GB)

- **Problem:** Die Berechnung der Embeddings mit ResNet50 ist sehr rechenintensiv, besonders bei großen Datensätzen. Zu Beginn haben wir die Embeddings nur für einen kleinen Teil des Datensatzes berechnet, um die Funktionalität zu testen. Nachdem diese Tests gute Ergebnisse zeigten, haben wir die vollständigen Embeddings für den gesamten Datensatz erstellt. Dabei haben wir jedoch keine umfassenden Optimierungen durchgeführt.

**Berechnung der Farbhistogramme:**

- **Laufzeit:** 15 Stunden und 18 Minuten. (6,81 GB)

- **Problem:** Die Berechnung und Speicherung der Farbhistogramme für jedes Bild sind ebenfalls zeitaufwendig. Auch hier haben wir zunächst nur einen kleinen Teil des Datensatzes verarbeitet, um die Methode zu testen. Erst als die Ergebnisse zufriedenstellend waren, haben wir die Farbhistogramme für den gesamten Datensatz berechnen lassen. Zudem arbeiten wir mit einer Bin-Größe von 16x16x16, was mehr Informationen beibehält, aber nicht optimal für die Geschwindigkeit ist. Dabei haben wir jedoch keine umfassenden Optimierungen durchgeführt.

**Einrichtung der Datenbank:**

- **Laufzeit:** 8 Stunden. (91,7 MB)

- **Problem:** Die Einrichtung der Datenbank war zeitaufwendig, aber dieser Schritt muss nur einmal durchgeführt werden. Auch hier haben wir uns nicht die Mühe gemacht, den Prozess bis zum Optimum zu optimieren, da dies ein einmaliger Vorgang ist.

**Erstellung der Tensorboard-Loggings:**

- **Laufzeit:** ca 3 Stunden und 29 Minuten. (17,3 GB | 0-868 Log Dateien)

- **Optimierung:** Dieser Prozess wurde durch die Verwendung von Multiprocessing optimiert, was die Erstellung der Loggings für die Visualisierungen in Tensorboard beschleunigt hat.

**Abgleich der Eingabebilder mit normalen Embeddings (ohne PCA):**

- **Laufzeit:** Der Abgleich anhand der normalen Embeddings dauerte etwa 7.34 Sekunden.

- **Problem**: Obwohl der Vergleich recht schnell ist, könnte die Laufzeit noch verbessert werden, wenn der Vergleichsalgorithmus optimiert und die Daten effizienter verwaltet werden.

**Bilddarstellung (Plotting):**

- **Laufzeit:** 3.82 Sekunden pro Durchgang.

- **Problem:** Das Plotten der Bilder nimmt eine beträchtliche Zeit in Anspruch und könnte optimiert werden, um die Gesamtlaufzeit zu reduzieren.

## Optimierungen und Verbesserungen:

**Vergleich der Eingabebilder mit Embeddings:**

- **Laufzeit:** Die Berechnung der Embeddings für ein neues Bild dauert nur 0.1 Sekunden, während der Abgleich mit den gespeicherten Embeddings etwa 7.34 Sekunden dauert. Dies zeigt, dass die Nutzung von GPU-Beschleunigung und effizientem Speichermanagement eine schnelle Analyse ermöglicht.

**Vergleich der Eingabebilder mit Color Histogramms:**

- **Laufzeit:** Der Abgleich mit der Farbhistogramm-Methode dauert etwa 17 Sekunden. Obwohl dies langsamer ist, behalten wir durch die gewählte Bin-Größe von 16x16x16 mehr Informationen bei, was zu besseren Ergebnissen führen kann. Diese Methode könnte durch Optimierungen weiter beschleunigt werden.

**Batch-Verarbeitung:**

- **Vorteil:** Durch die Verarbeitung in Batches konnte die Belastung der GPU während der Berechnung der Embeddings und Farbhistogramme besser verteilt werden. Dies trug dazu bei, Engpässe zu vermeiden und die Berechnungen effizienter zu gestalten.

**Optimierung des Plots:**

- **Vorschlag:** Das Plotten der Bilder könnte durch die Verwendung schnellerer Visualisierungsmethoden oder durch eine Reduzierung der Bildauflösung optimiert werden, um die Laufzeit weiter zu verkürzen.

### Laufzeit und Benutzererfahrung:

- Ziel: Die Laufzeit für den Vergleich der Eingabebilder soll weiter verkürzt werden, um eine möglichst reibungslose und schnelle Benutzererfahrung zu gewährleisten.

- **Derzeitiger Stand:** Der Vergleich mit normalen Embeddings dauert etwa 20.34 Sekunden, der Vergleich mit PCA-Embeddings dauert etwa 7 Sekunden, während der Vergleich mit Color Histogramms etwa 17 Sekunden benötigt. Das Plotten der Bilder bleibt ein relevanter Faktor, der weiter optimiert werden könnte in dem wir die Bilder vor dem Plotten verkleinern.

# 5. Feasibility analysis/Discussion

Die Leistungsanalyse zeigt, dass die Software vielversprechende Ansätze für die Skalierbarkeit bietet, insbesondere durch die Verarbeitung in Batches, die Nutzung von GPUs und die Anwendung der PCA-Dimensionsreduktion. Diese Techniken machen die Software gut geeignet für große Datensätze, wie den von über 400.000 Bildern, mit dem wir gearbeitet haben. Dank der Optimierungen ist der Vergleich der PCA-Embedding-Ähnlichkeiten mittlerweile so schnell, dass die Software bereits für Echtzeitanwendungen genutzt werden kann.

Skalierbarkeit und Eignung:

Echtzeitfähigkeit durch PCA-Embeddings: Der Vergleich der PCA-Embeddings dauert etwa 5-7 Sekunden, was die Software für Echtzeitanwendungen nutzbar macht. Diese Geschwindigkeit ermöglicht es, Bilder in nahezu Echtzeit zu vergleichen und relevante Ergebnisse zu liefern.

Eignung für große Datensätze: Die Software kann effizient mit sehr großen Bilddatensätzen, wie unserem Datensatz mit über 400.000 Bildern, umgehen, indem sie die verfügbaren Ressourcen

optimal nutzt. Die Verwendung von GPU-Beschleunigung und PCA-Reduktion verringert die Wartezeiten für den Benutzer erheblich.

Herausforderungen bei großen Datensätzen: Obwohl die Software bereits gut mit großen Datensätzen zurechtkommt, könnten bei großen Datensätzen, die weit über die 400.000 Bilder hinausgehen, weiterhin Engpässe auftreten, insbesondere wenn die gesamte Datenbank auf einmal durchsucht werden muss.

Wichtige Einschränkungen:

Rechenintensive Prozesse bei vollständiger Verarbeitung: Die Berechnung von Embeddings und Farbhistogrammen für sehr große Datensätze bleibt zeitaufwendig. Diese Prozesse wurden bisher nicht vollständig optimiert, da der Fokus zunächst auf der Funktionalität und der Erprobung auf kleineren Datenmengen lag.

Speicherbedarf: Die Speicherung großer Mengen an Bilddaten, insbesondere in Form von Farbhistogrammen und Embeddings, erfordert erhebliche Speicherressourcen. Dies kann bei großen Datensätzen zu Speicherengpässen führen.

Verwendung von Color Histogramms: Die Farbhistogramme werden mit einer Bin-Größe von 16x16x16 berechnet, was mehr Details bewahrt, aber die Geschwindigkeit beeinträchtigt. Diese Methode ist gut für präzise Ergebnisse, könnte aber bei sehr großen Datenmengen zu langsam sein.

Möglichkeiten zur Verbesserung:

Optimierung durch spezialisierte Algorithmen: Der Einsatz spezialisierter Algorithmen könnte die Effizienz weiter steigern. Beispielsweise könnten Algorithmen zur schnelleren Approximation von Ähnlichkeitsmaßen verwendet werden, um die Rechenzeit zu verkürzen.

Auslagerung auf leistungsfähigere Hardware oder Cloud-Server: Eine Migration der Software auf leistungsfähigere Hardware oder die Nutzung von Cloud-Servern könnte die Verarbeitungszeit weiter reduzieren und die Skalierbarkeit erhöhen. Cloud-Lösungen bieten auch die Möglichkeit, Ressourcen flexibel zu skalieren, je nach Bedarf.

Wechsel zu einem leistungsfähigeren Datenbanksystem: Für eine bessere Skalierbarkeit und Leistungsfähigkeit könnte es sinnvoll sein, von SQLite zu einem robusteren Datenbanksystem wie PostgreSQL zu wechseln. PostgreSQL bietet bessere Unterstützung für große Datensätze, parallele Abfragen und erweiterte Funktionen, die für den Einsatz in großen und wachsenden Datenbanken von Vorteil sind.

Einsatz von Klassifizierungstechniken: Eine Klassifizierung der Bilder könnte die Effizienz der Suche weiter steigern. Dabei könnten die Bilder vorab kategorisiert und in gelabelten Pickle-Dateien gespeichert werden. Dies würde die Suche effizienter machen, da nur in den relevanten Kategorien nach ähnlichen Bildern gesucht werden müsste.

Kombination von Ähnlichkeitsmaßen: Abhängig vom Anwendungsfall könnte eine Kombination aus Klassifizierung und feinerer Ähnlichkeitsanalyse, beispielsweise durch Farbhistogramme, die Genauigkeit der Suchergebnisse verbessern. Diese Kombination würde sicherstellen, dass die Software sowohl für allgemeine Bildvorschläge als auch für die Suche nach möglichst identischen Bildern geeignet ist.

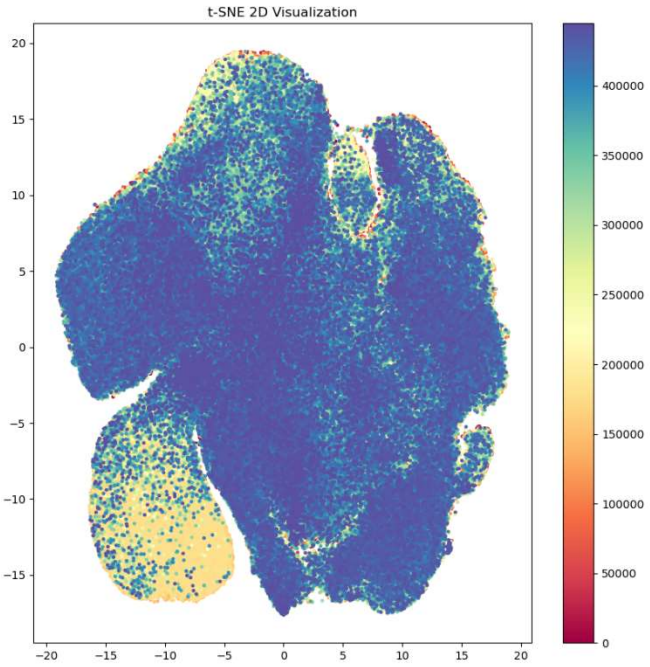
Fazit:

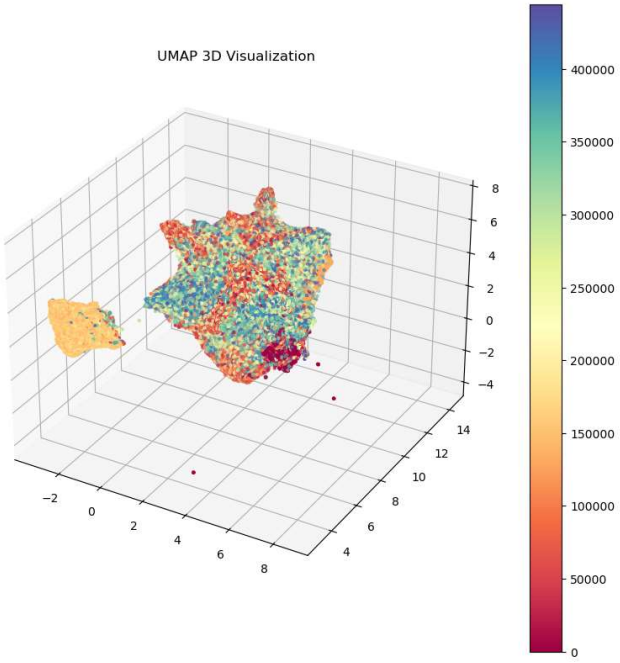
Die Software zeigt großes Potenzial für den Einsatz als Suchmaschine, insbesondere durch die schnelle Verarbeitung der PCA-Embeddings, die sie für Echtzeitanwendungen geeignet macht. Für große Datensätze, wie den von über 400.000 Bildern, ist die Software bereits gut optimiert. Um jedoch auch bei großen Datensätzen optimal zu funktionieren, könnten weitere Optimierungen notwendig sein. Dies umfasst sowohl die Verbesserung der Algorithmen als auch die Anpassung der Hardware. Ein Wechsel zu einem leistungsfähigeren Datenbanksystem wie PostgreSQL könnte die Skalierbarkeit weiter verbessern. Die Integration von Klassifizierungstechniken könnte ebenfalls die Effizienz und Genauigkeit der Suchergebnisse erheblich steigern

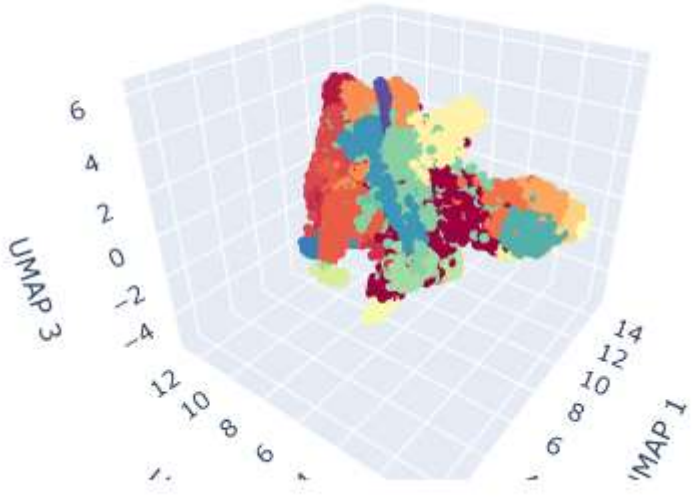
# Teil 2 - Big Data-Bildanalyse

1. Erste Visualisierungen: UMAP mit DBSCAN & K-MEANS

In den ersten Visualisierungen, die UMAP mit dem DBSCAN und dem K-Means -Algorithmus kombinierten, waren die Ergebnisse noch nicht zufriedenstellend. Die Cluster in der Darstellung waren nicht klar getrennt, und es war schwer, sinnvolle Muster oder Gruppierungen innerhalb des Datensatzes zu erkennen. Dies deutete darauf hin, dass die gewählten Parameter für UMAP und die anschließende Clusterbildung durch DBSCAN nicht optimal waren. Die Visualisierungen gaben wenig Aufschluss über die Struktur des Datensatzes und die Verteilung der Bilder.



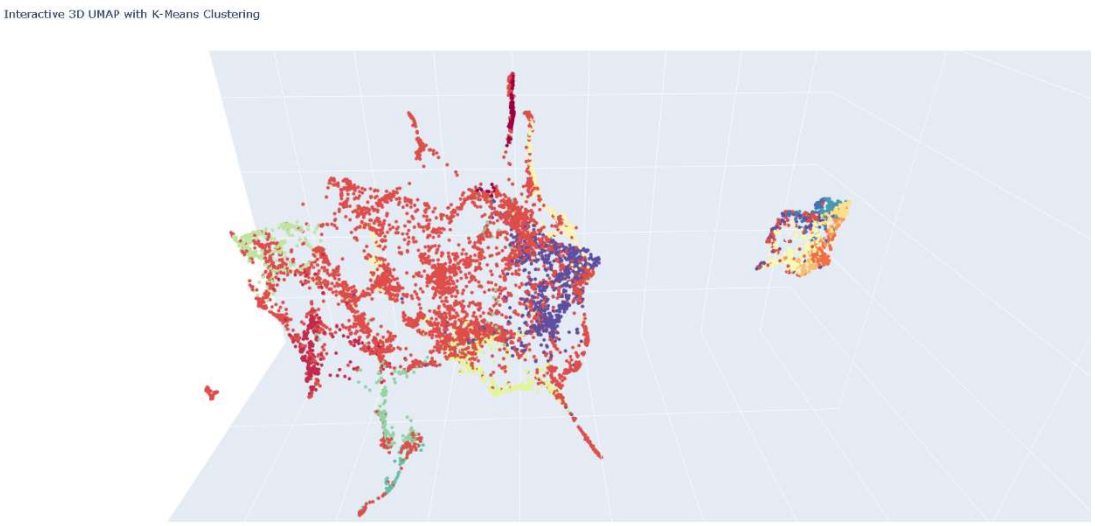


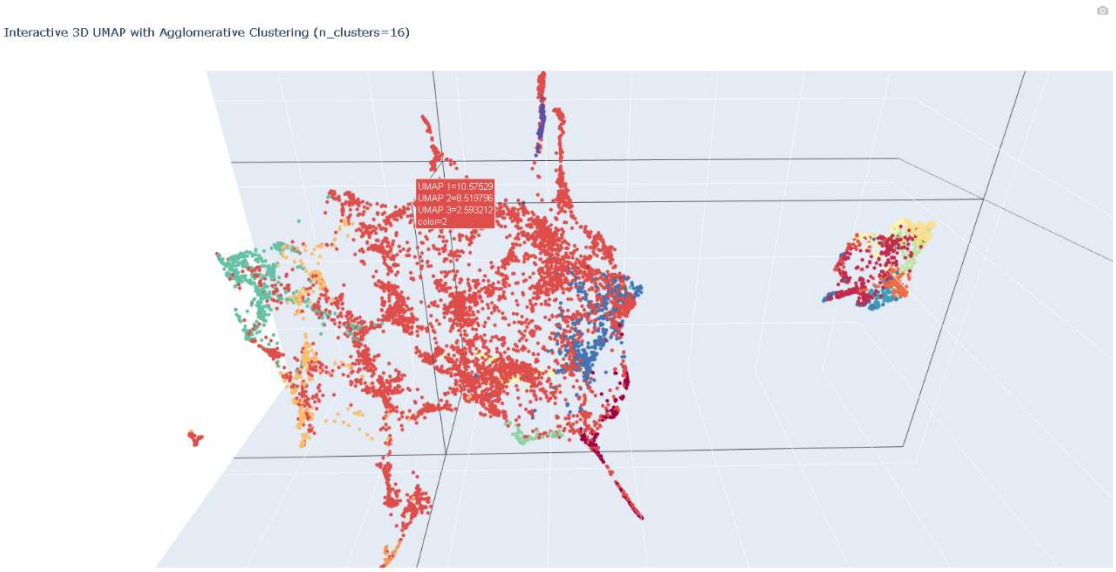

2. Verbesserte Visualisierungen: UMAP mit K-Means und Agglomerative Clustering

In den späteren Visualisierungen wurden die Ergebnisse durch die Kombination von UMAP mit K-Means und Agglomerative Clustering deutlich verbessert. Diese Visualisierungen zeigten klarere und besser getrennte Cluster, was eine wesentlich präzisere Interpretation der Daten ermöglichte. UMAP erwies sich als effektive Methode zur Dimensionen Reduktion, die in Kombination mit geeigneten Clustering-Algorithmen eine bessere Strukturierung der Daten lieferte.

UMAP mit K-Means Clustering: Diese Visualisierung zeigte eine deutliche Verbesserung in der Trennung der Cluster. Bilder, die ähnliche Merkmale aufwiesen, wurden konsistenter gruppiert, und die Cluster waren klar voneinander abgegrenzt.

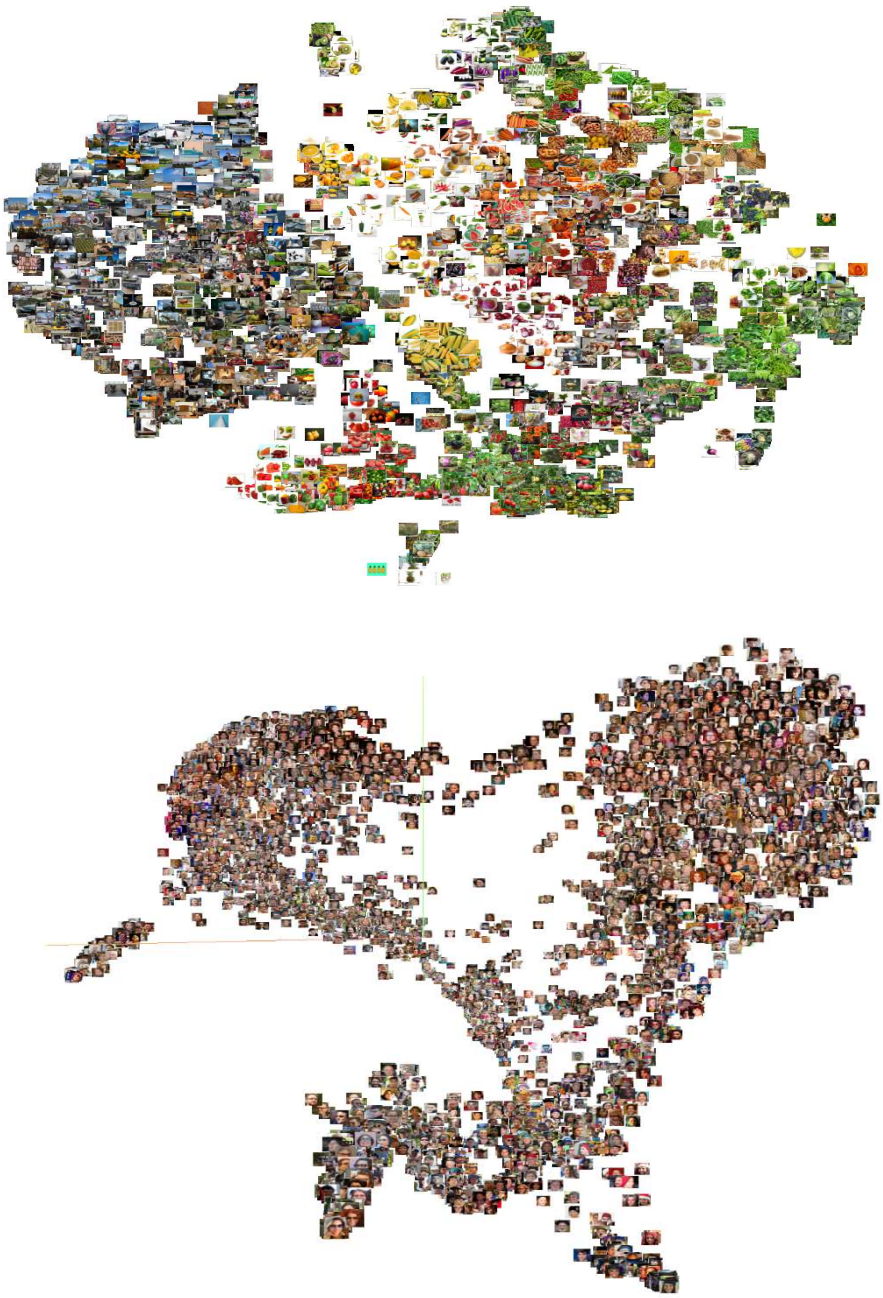



UMAP mit Agglomerative Clustering: Auch hier waren die Cluster gut definiert, und die Anwendung von Agglomerative Clustering ermöglichte eine weitere Verfeinerung der Gruppierungen, was in einer detaillierten und klaren Visualisierung resultierte.




3. Visualisierungen mit TensorBoard: Erste Versuche mit t-SNE und UMAP

Als nächstes wurden Visualisierungen mithilfe von TensorBoard erstellt, die sowohl t-SNE als auch UMAP verwendeten. Diese Visualisierungen stellten einen Fortschritt dar, zeigten jedoch immer noch nicht die Klarheit, die für eine tiefergehende Analyse erforderlich ist. Die Cluster waren zwar etwas besser definiert, aber immer noch weitgehend überlappend und boten keine präzise Trennung der Bildgruppen. Die Interpretation der Cluster blieb schwierig, und es war unklar, ob die verwendeten Algorithmen die komplexen Beziehungen innerhalb des Datensatzes angemessen erfassten.

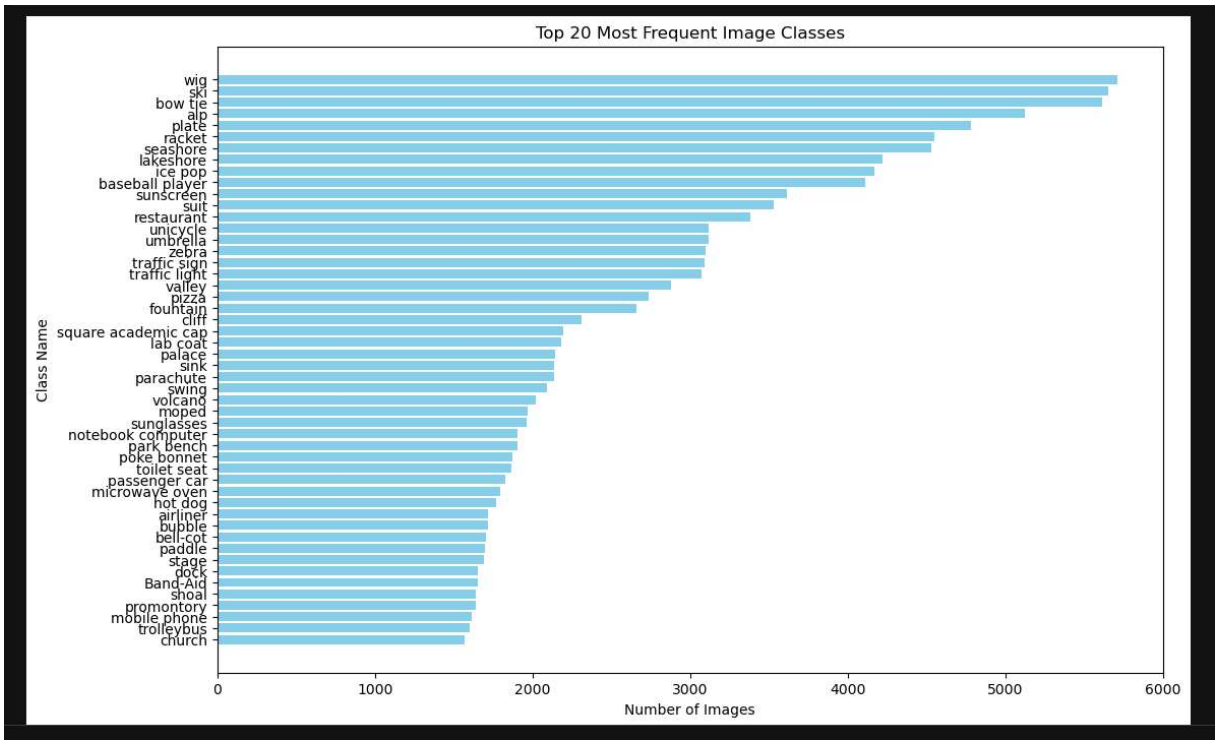


4. Finale Visualisierung: Klar erkennbare Cluster

Die abschließende Visualisierung zeigte schließlich klar erkennbare Cluster und stellte damit den Höhepunkt der bisherigen Entwicklungsarbeit dar. Diese Visualisierung bot nicht nur eine klare Trennung der verschiedenen Bildgruppen, sondern ermöglichte auch eine intuitive Interpretation der Datenstruktur. Die Verwendung optimierter UMAP-Parameter und fortschrittlicher Clustering-Algorithmen führte zu einer prägnanten und aussagekräftigen Darstellung der Daten, die einen tiefen Einblick in die Verteilung und Ähnlichkeiten der Bilder im Datensatz ermöglichte.

## Diskussion über die Klassifizierung und Labeln

Die anfängliche Klassifizierung des Datensatzes mit ResNet50 war nicht ideal, da die Labels ungenau und die Resultate unbefriedigend waren. Der Versuch, den Datensatz später mit YOLOv5 zu klassifizieren, scheiterte aufgrund der enormen Zeit, die für die Verarbeitung erforderlich war. Dies zeigte die Herausforderungen bei der Arbeit mit sehr großen Datensätzen und die Notwendigkeit einer effizienten Klassifizierungsmethode.




```
YOLOv5  2024-8-9 Python-3.8.19 torch-1.13.1+cu117 CUDA:0 (NVIDIA GeForce RTX 3080, 10240MiB)

Fusing layers...
YOLOv5s summary: 213 layers, 7225885 parameters, 0 gradients
Adding AutoShape...
Total images in the database: 444668

Processing Images:   0%|          | 0/444668 [00:00<?, ?image/s]Processing 32 images in this batch.

Processing Images:   0%|          | 32/444668 [02:09<500:04:14,  4.05s/image]Processing 32 images in this batch.

Processing Images:   0%|          | 64/444668 [04:18<498:59:02,  4.04s/image]Processing 32 images in this batch.

Processing Images:   0%|          | 96/444668 [06:29<502:27:15,  4.07s/image]Processing 32 images in this batch.

Processing Images:   0%|          | 128/444668 [08:39<501:35:02,  4.06s/image]Processing 32 images in this batch.

Processing Images:   0%|          | 160/444668 [10:45<496:06:41,  4.02s/image]Processing 32 images in this batch.

Processing Images:   0%|          | 192/444668 [12:51<493:03:23,  3.99s/image]Processing 32 images in this batch.
```

Fazit:

Die Entwicklung der Visualisierungen zeigt eine deutliche Verbesserung in der Qualität und Klarheit der Datenrepräsentation. Die ersten Plots, die mit Matplotlib erstellt wurden, hatten Schwierigkeiten, klare Einblicke in die Struktur des Datensatzes zu bieten. Diese frühen Versuche zeigten überlappende und unklare Cluster, was es erschwerte, sinnvolle Interpretationen aus den Daten abzuleiten.

Mit der Verfeinerung der Methoden verbesserten sich auch die Visualisierungen deutlich. Die verbesserten Plots, die mit Plotly erstellt wurden, zeigten zwar eine bessere Struktur und begannen, die Vielfalt des Datensatzes zu erfassen, jedoch blieb die vollständige Komplexität des Datensatzes noch unzureichend dargestellt. Diese Plots zeigten eine gewisse Clustering-Struktur, aber die Unterschiede zwischen den Gruppen waren nicht so klar und eindeutig, wie es gewünscht war.

Der Durchbruch kam mit den finalen TensorBoard-Visualisierungen. Durch die Nutzung von t-SNE und UMAP innerhalb von TensorBoard konnten die Plots endlich gut definierte und klare Cluster zeigen, die tiefe Einblicke in die zugrunde liegende Struktur des Datensatzes boten. Die Klarheit und Präzision dieser TensorBoard-Plots machen sie zu den erfolgreichsten Visualisierungsversuchen und zeigen effektiv die Beziehungen und Ähnlichkeiten zwischen den Bildern im Datensatz.

Diese Entwicklung unterstreicht die Bedeutung der Wahl der richtigen Werkzeuge und Techniken für die Datenvisualisierung. Während Plotly eine Verbesserung gegenüber den ersten Matplotlib-Versuchen darstellte, zeigte die überlegene Leistung von TensorBoard bei der Offenlegung klarer und aussagekräftiger Cluster seinen Wert im Umgang mit komplexen Datensätzen. Die finalen TensorBoard-Plots stellen den Höhepunkt dieses Prozesses dar und bieten die genaueste und aufschlussreichste Darstellung der Datenstruktur.