

4. Übungsblatt

Embeddings, Vector Databases and Search

Ausgabe: 07.01.25

Abgabe: 24.01.25 (Mitternacht, Moodle)

Wintersemester 2024/2025

Patrick Schäfer, patrick.schaefer@hu-berlin.de

Agenda

- Fragen zur Vorlesung?
- Vorstellung 4. Übungsblatt

Fragen zur Vorlesung?

4. Übungsblatt: Embeddings, Vector Databases & Search



Teilaufgaben

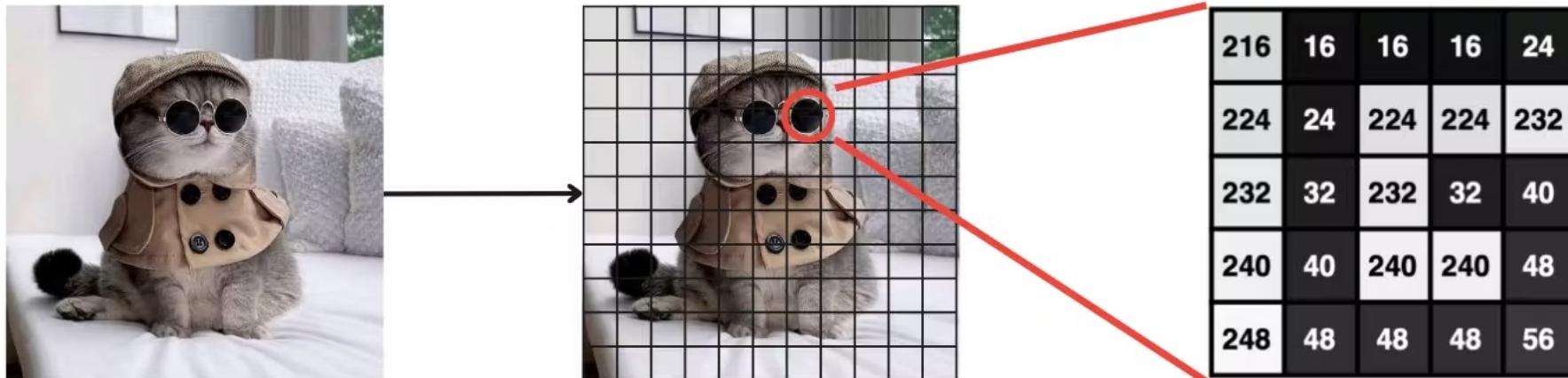
1. Exakte Bilder-Suche
 - a) Indexierung
 - b) Suche
3. Approximative Bilder-Suche mittels HNSW
4. Hyper-Parameter-Optimierung
5. Dogs that look like you
6. Text-To-Image Suche

Embeddings und Vector-Search

- In dieser Übung werden wir FAISS und CLIP nutzen, um Vektor-Embeddings aus Bildern zu generieren, diese zu indexieren und zu suchen.
 - FAISS [1]: Vektor-Suche Bibliothek von Facebook
 - CLIP [2]: Bibliothek zur Erstellung von Embeddings aus Bildern
-
- [1] <https://faiss.ai>
 - [2] <https://openai.com/index/clip/>

Bilder

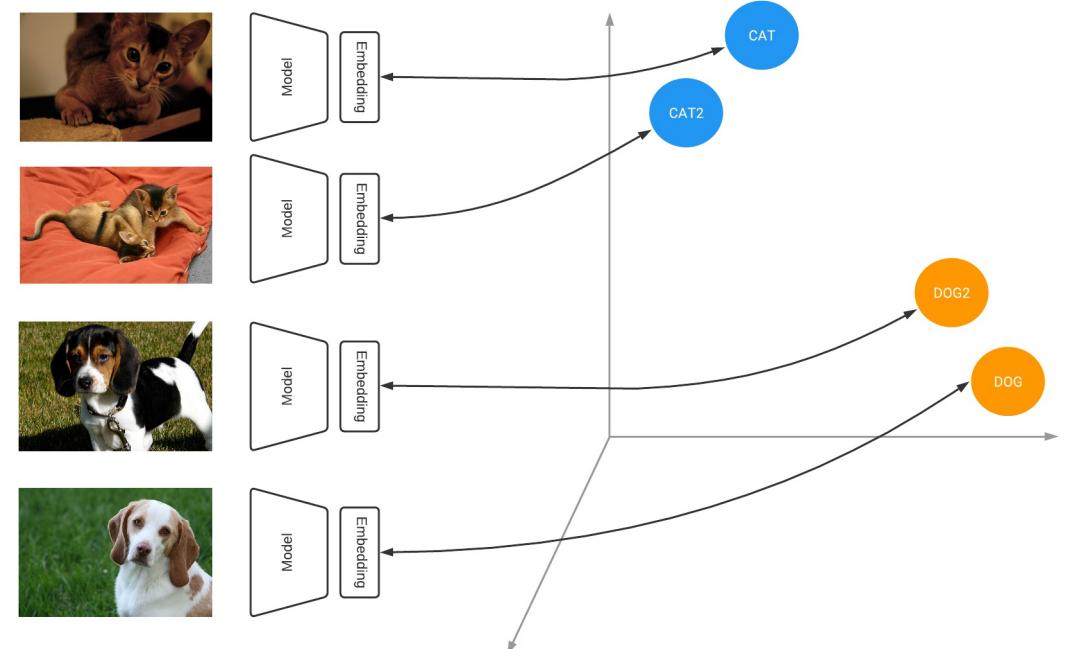
- Bilder werden durch Pixel dargestellt
- Jeder Pixel enthält 3 Channel für die Farbinformation: Rot, Grün, Blau
 - Je 256 Werte



- Damit maschinelle Lernmodelle ein Bild effizienter verarbeiten können, wird Bild als Embedding dargestellt werden.

Embedding

- Ein Embedding ist ein Mapping das ein Bild in einen hoch-dimensionalen Vektorraum
- Ziel des Mappings ist es, dass Objekte, die ähnlich sind, nahe beieinander liegen
- Nähe/Ähnlichkeit wird über ein Distanzmaß gemessen
- Ziel ist also Minimierung der Distanz im Embedding-Raum



Quelle: <https://blog.tensorflow.org/2021/09/introducing-tensorflow-similarity.html>

Beispiel-Embedding

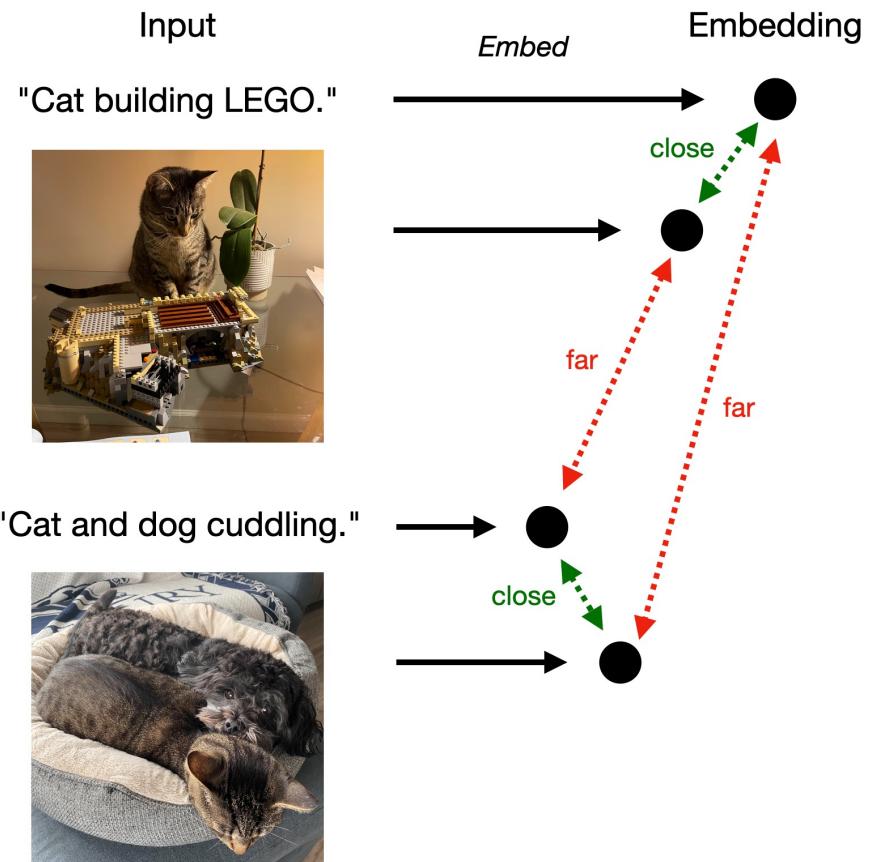


CLIP

- Entworfen, um gemeinsam Bild und Text-Embeddings zulernen
 - Erlaubt Suche von Bildern anhand von Texteingaben
 - Anhand von 400 Millionen Bild- und Textpaaren selbstüberwacht trainiert
 - Embeddings von Text oder Bildern sind 512 dimensional
- <https://openai.com/index/clip/>

Open AI's CLIP

- CLIP mappt sowohl Text als auch Bilder in den gleichen Embedding-Raum



CLIP

- Besteht aus zwei Komponenten:
 - Image Encoder: ResNet-50 und Vision Transformer (ViT)
 - Text Encoder: Transformer
- Contrastive Learning wird verwendet, um Model zu lernen (Fine-Tuning)
- Wir gehen hier nicht auf Details von Transformern und ViT ein.

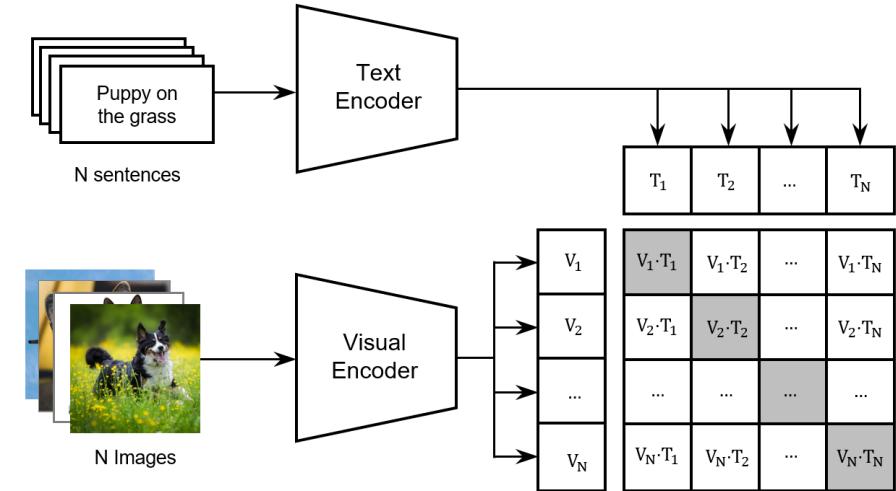
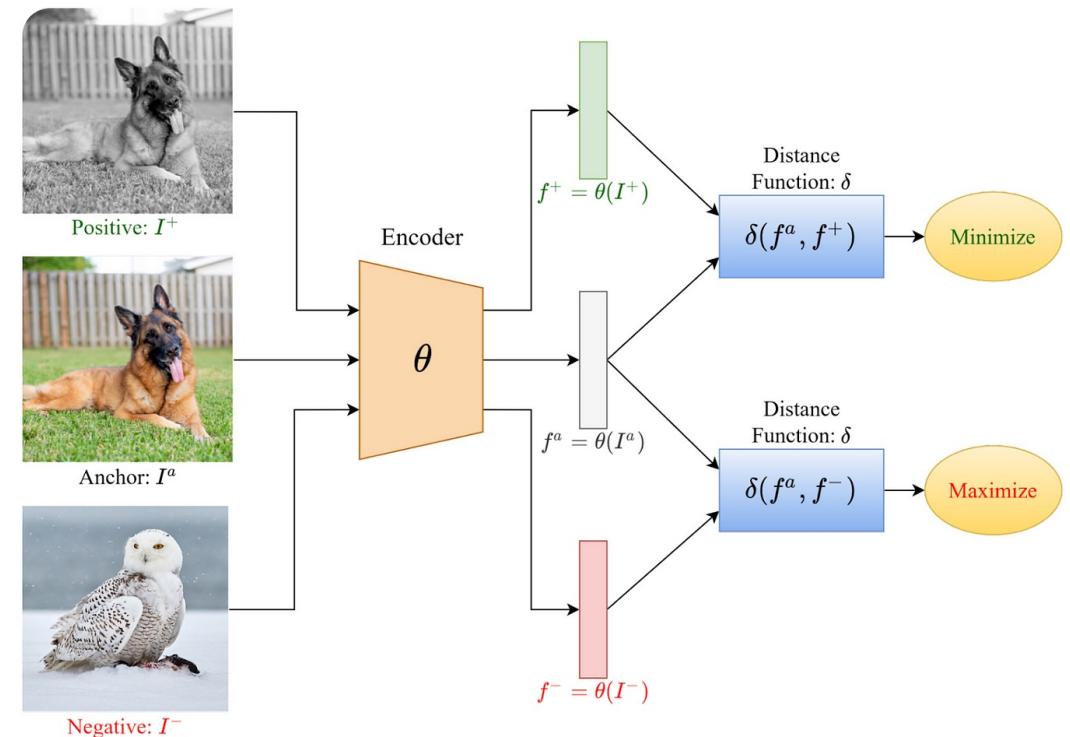


Figure 2: CLIP consists of a visual encoder \mathbb{V} , a text encoder \mathbb{T} , and a dot product between their outputs. It is trained to align images and texts with a contrastive loss. The dot product is used as an alignment score.

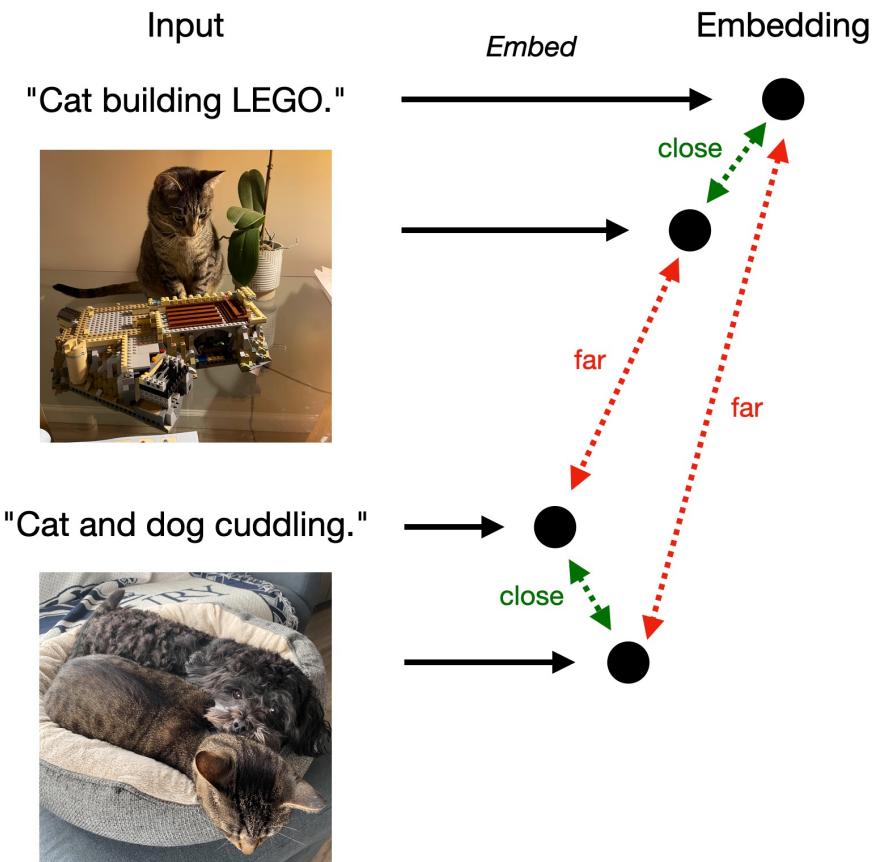
Contrastive Learning

- Grundprinzip: Modell trainiert mit Triplets:
 - Anker, positives Beispiel (gleiche Klasse), negatives Beispiel (andere Klasse)
- Ziel: Ähnliche Objekte sollen ähnliches Embedding lernen, unähnliche Objekte unterschiedliche
- Trainingsziele:
 - Minimiere Distanz zwischen Anker und positivem Beispiel
 - Maximiere Distanz zwischen Anker und negativem Beispiel
- Ergebnis: Modell lernt, ähnliche Objekte näher zusammenzubringen und unähnliche weiter zu trennen



Contrastive Learning – Text und Image

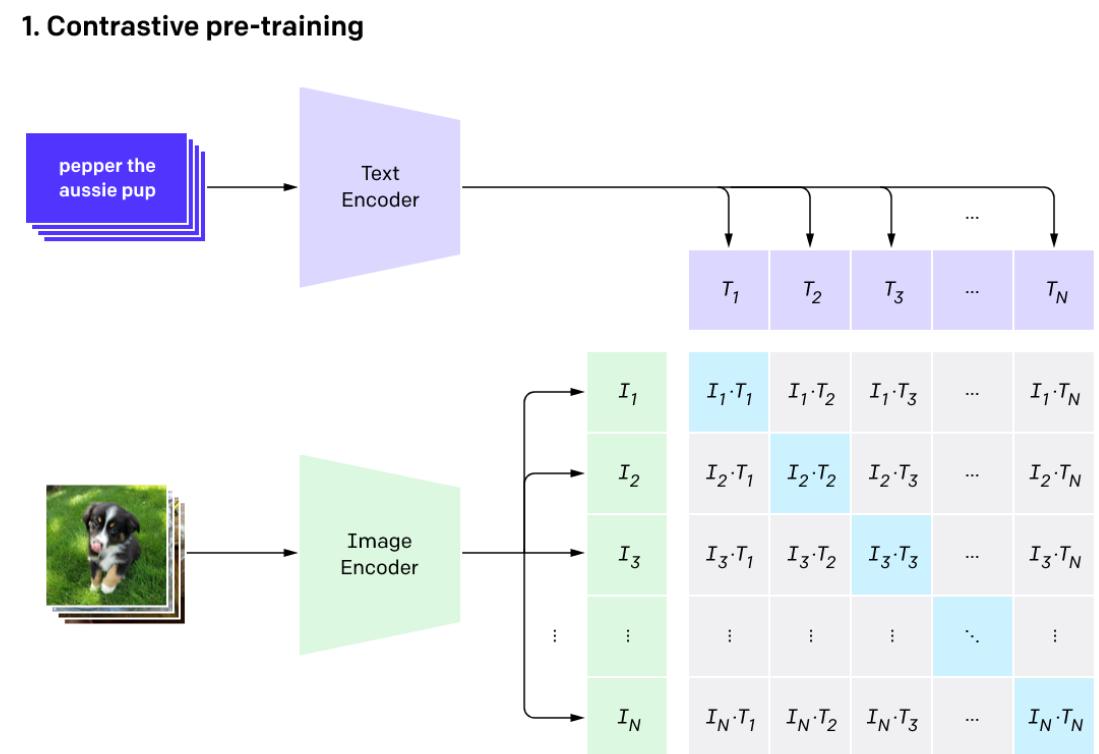
- Ansatz funktioniert auch mit Text oder Text-Bild-Kombinationen:
 - Beispiel (CLIP): Anker ist Bild einer Katze.
 - Positives Beispiel ist Bildbeschreibung „a cat building LEGO“
 - Negatives Beispiel ist Bildbeschreibung „Cat and dog cuddling“



Quelle: <https://www.rtealwitter.com/deeplearning2023/contrastive.html>

CLIP – Contrastive Pre-Training

- CLIP trainiert parallel (Gewichte des) Text und Image Encoders
- Maximiert Cosine Similarity zwischen Bild- und Text-Paaren mittels Contrastive Learning
 - Positiv-Paar: in Blau
 - Negativ-Paar: in Grau



FAISS

- Vektor Search Library von Facebook
- Implementiert eine Vielzahl von exakten und approximativen Nächste-Nachbar-Index-Strukturen:
 - <https://github.com/facebookresearch/faiss/wiki/Faiss-indexes>

Datensatz

- Stanford Dogs Dataset:
 - <http://vision.stanford.edu/aditya86/ImageNetDogs/>
 - Number of categories: 120
 - Number of images: 20,580
 - Annotations: Class labels, Bounding boxes

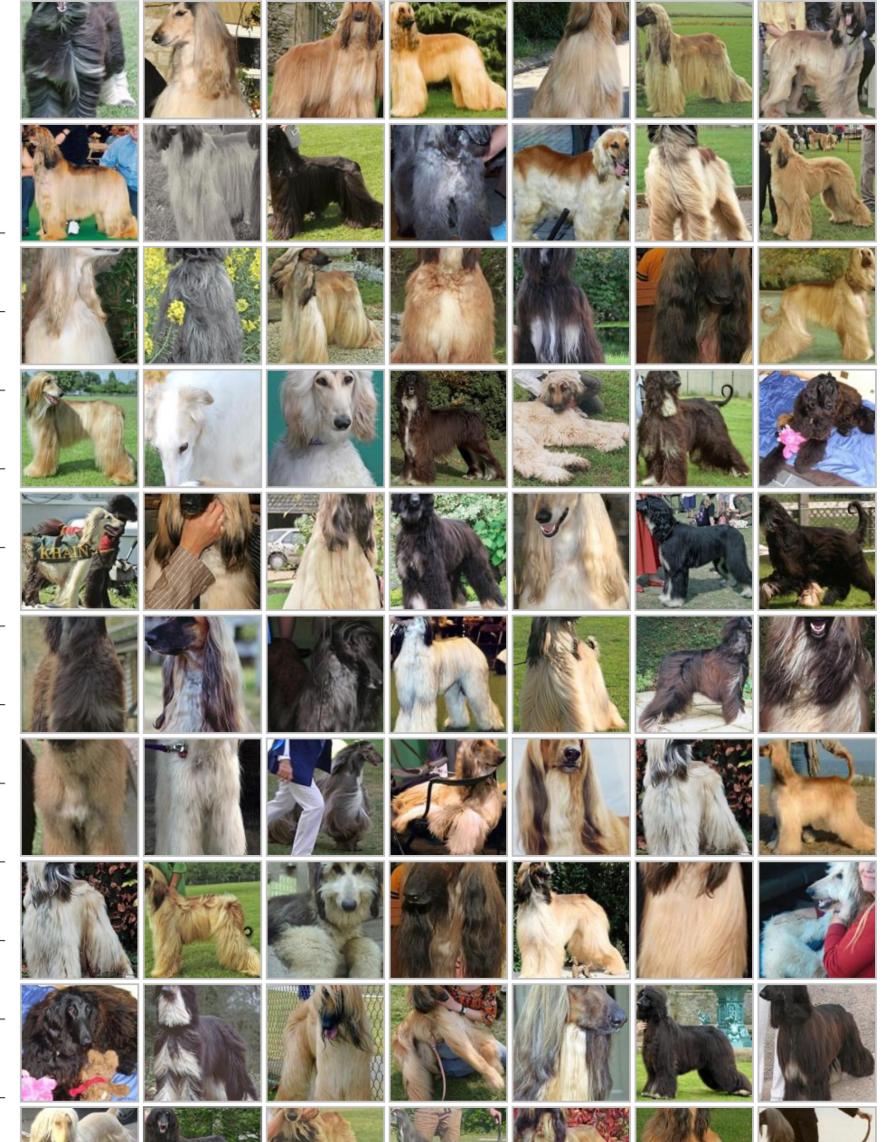
Stanford Dogs Dataset

Afghan hound (239 images)

Summary:

- 120 dog breeds
- ~150 images per class
- Total images: 20,580

[Download dataset](#)



Affenpinscher (150 images)

ImageNet synset: [n02110627](#)

Afghan hound (239 images)

ImageNet synset: [n02088094](#)

African hunting dog (169 images)

ImageNet synset: [n02116738](#)

Airedale (202 images)

ImageNet synset: [n02096051](#)

American Staffordshire terrier (164 images)

ImageNet synset: [n02093428](#)

Appenzeller (151 images)

ImageNet synset: [n02107908](#)

Australian terrier (196 images)

ImageNet synset: [n02096294](#)

Basenji (209 images)

ImageNet synset: [n02110806](#)

Basset (175 images)

ImageNet synset: [n02088238](#)

Beagle (195 images)

ImageNet synset: [n02088364](#)

Bedlington terrier (182 images)

ImageNet synset: [n02093647](#)

Bernese mountain dog (218 images)

ImageNet synset: [n02107683](#)

Black-and-tan coonhound (159 images)

Embeddings

- Mittels CLIP wurden Embeddings aus den Bildern erstellt.
 - Embedding eines Bildes: 512-dimensionaler Vektor
 - Die Embeddings könnten ihr in Moodle herunterladen

	filename	dir	class	pose	xmin	ymin	xmax	ymax	embedding
0	n02097658_26	n02097658-silky_terrier	silky_terrier	Unspecified	41	30	296	398	[0.21719056367874146, 0.2956088483335876, 0.0...
1	n02097658_4869	n02097658-silky_terrier	silky_terrier	Unspecified	43	270	321	498	[-0.3137703835964203, -0.5720250606536865, 0.0...
2	n02097658_595	n02097658-silky_terrier	silky_terrier	Unspecified	82	7	378	355	[0.10297045111656189, -0.08061640709638596, -0...
3	n02097658_9222	n02097658-silky_terrier	silky_terrier	Unspecified	0	12	331	498	[0.20631632208824158, 0.0758294016122818, 0.02...
4	n02097658_422	n02097658-silky_terrier	silky_terrier	Unspecified	146	10	356	332	[0.02929862029850483, -0.1272777659893036, 0....

Task 1: Exakte Suche

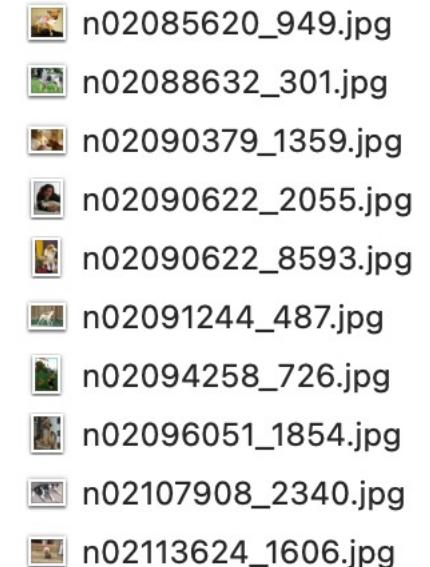
- Erstellt einen **IndexFlatL2 Index** mittels der vortrainierten Embeddings

```
index_flat = faiss.IndexFlatL2(d)  
...
```

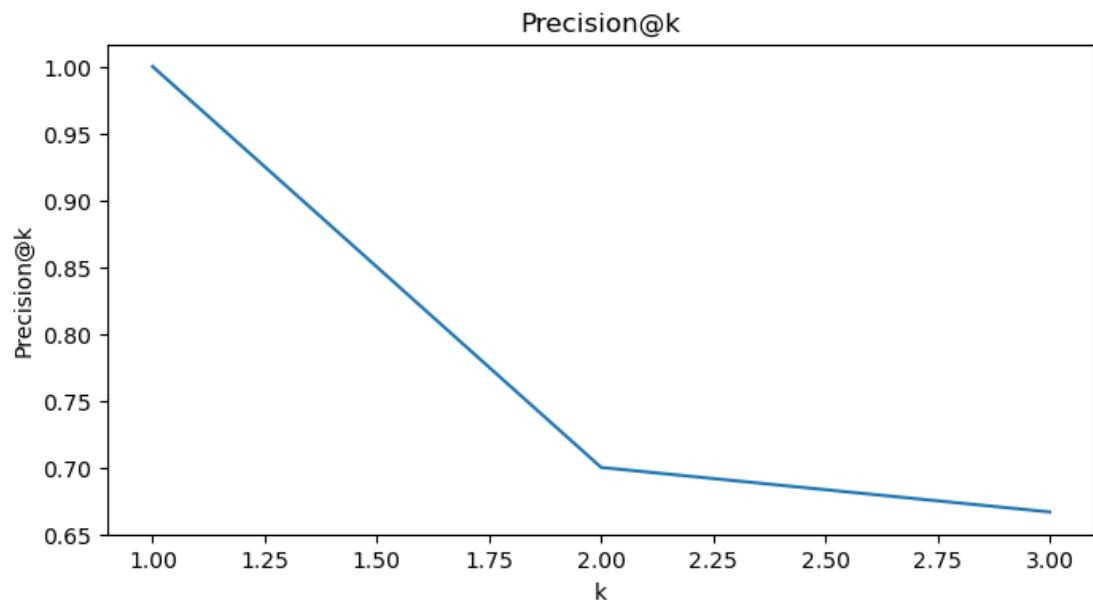
- Siehe auch: <https://github.com/facebookresearch/faiss/wiki>

Task 2: 3-NN Anfragebeantwortung

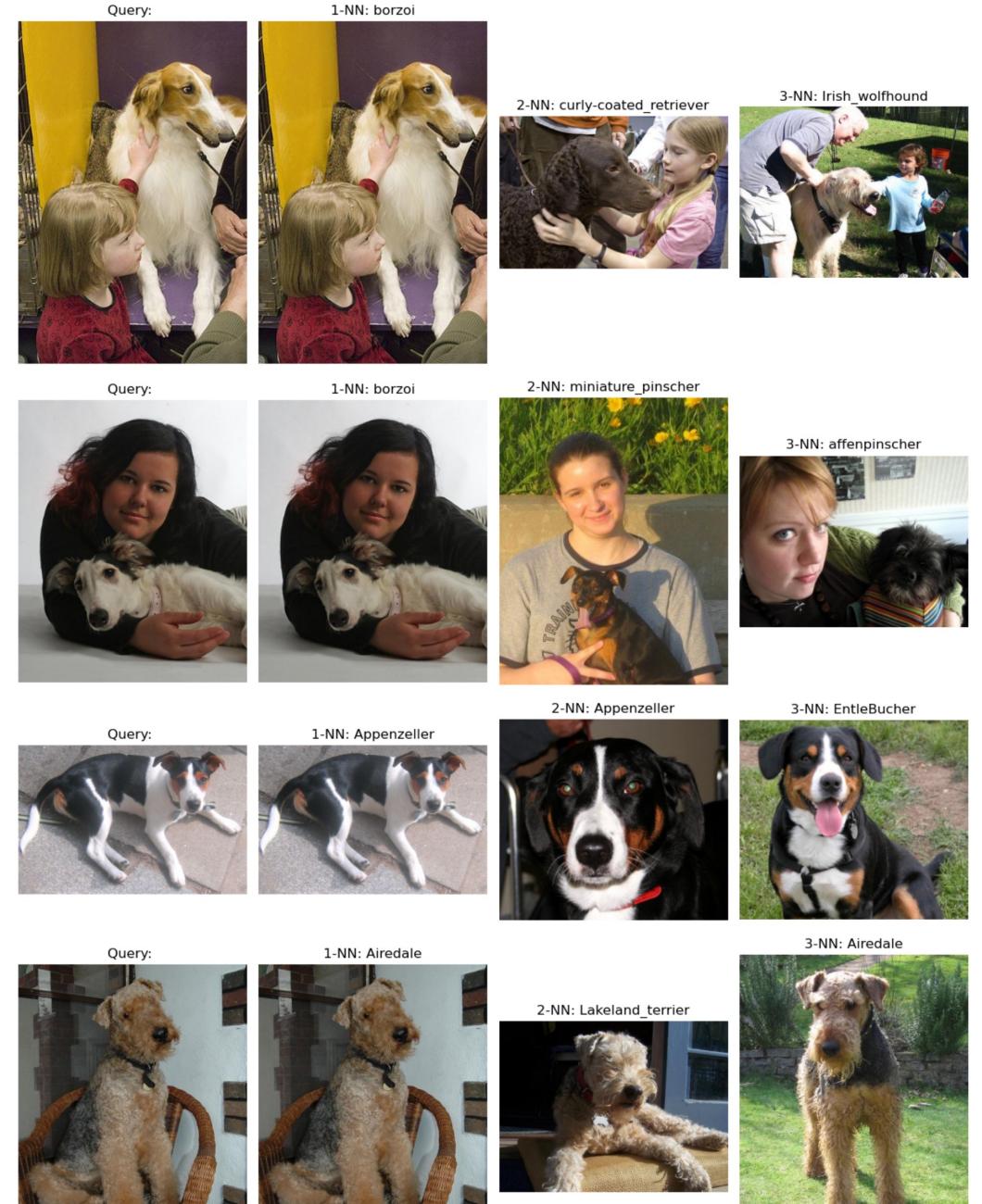
- Ihr sollt nun **3-Nächste Nachbarn Anfragen** beantworten
 1. Verwendet die 10 Bilder im Ordner „queries“
 2. Erstellt CLIP Embeddings
 3. Sucht mittels **IndexFlatL2** die **3-ähnlichsten Hunde** pro Bild.
- Siehe auch: <https://github.com/openai/CLIP>
- Wichtig: der Datentyp der Embedding-Vektoren muss float32 sein - sonst wirft FAISS einen Fehler



Anfrage und 3-NN



Precision@k



Task 3: Anfragebeantwortung mittels HNSW

- Wiederholt nun Task 1&2, aber:
 - Führt eine **3-NN**-Suche durch
 - Verwendet aber den **Hierarchical Navigable Small World index** von FAISS
- `index = faiss.IndexHNSWFlat(d, M)`
- `hswg_index.hnsw.efConstruction = ...`
- `hswg_index.hnsw.efSearch = ...`

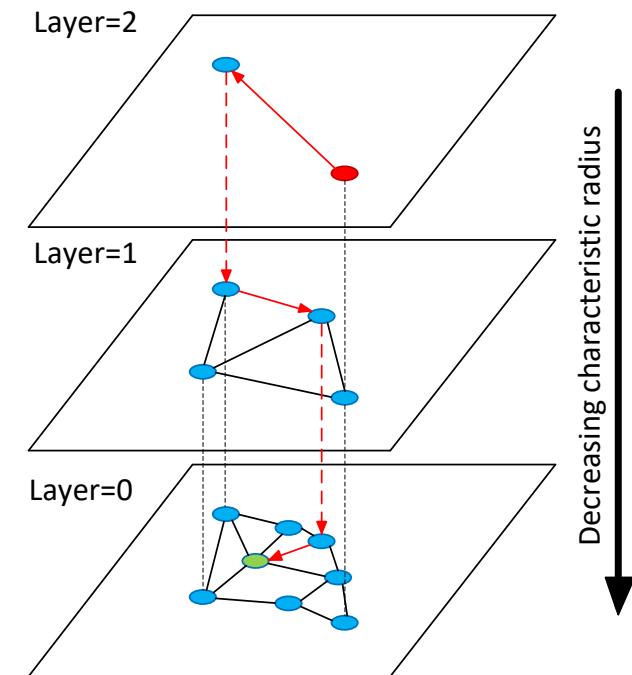


Fig. 1. Illustration of the Hierarchical NSW idea. The search starts from an element from the top layer (shown red). Red arrows show direction of the greedy algorithm from the entry point to the query (shown green).

Tuning des HNSW-Index

- **M:**
 - Anzahl der Verbindungen pro Knoten.
 - Höherer Wert → bessere Genauigkeit.
 - Nachteil: Höherer Speicherverbrauch, langsamere Indexierung.
- **EfConstruction:**
 - Anzahl der der k-NN, die während des Graphaufbaus gesucht werden sollen
 - Größerer Wert → bessere Graphqualität.
 - Nachteil: Längere Konstruktionszeit.
- **EfSearch:**
 - Anzahl der k-NN während der Suche
 - Höherer Wert → höhere Genauigkeit.
 - Nachteil: Langsamere Suche.
- **Metric Type:**
 - Distanz-Metrik: z.B. L1 oder L2

Task 4: Optimierte Hyper-Parameter von HNSW

- Wir wollen die besten Parameter für eine **5-NN Suche** finden.
- Testet folgende Hyper-Parameter des **HNSW Index**:
 - `M_values = [8, 16, 32]`
 - `efConstruction_values = [40, 100, 200]`
 - `efSearch_values = [10, 50, 100]`
 - `metric_types = [faiss.METRIC_L2, faiss.METRIC_L1]`
- Gebt jeweils die besten Parameter für die
 1. schnellste Suche und
 2. genaueste Suche (gemessen als precision@5)
- aus

Task 5: Sucht Hunde, die wie Ihr aussehen

- Verwendet Bilder von euch selbst und sucht nach Hunden, die wie ihr aussehen.
- Alternativ: nehmt 5 Bilder aus dem Internet und sucht nach Hunden, die ähnlich zu diesen Personen aussehen.

Task 6: Text-To-Image Suche

- Ihr sollt nun 3-Nächste Nachbarn Anfragen beantworten.
 1. Ihr habt **textuelle Beschreibungen** gegeben
 - „A dog and a girl“
 2. Erstellt **CLIP Embeddings** des Textes
 3. Erstellt einen **IndexFlatL2** mit **Metric METRIC_L1**
 4. Sucht mittels FAISS die **3-ähnlichsten Bilder**.

a dog and a girl



- Hinweise
 - Text-Embeddings: <https://github.com/openai/CLIP>
 - Wichtig: Datentyp der Embedding-Vektoren muss float32 sein - sonst wirft FAISS einen Fehler

Getting Started – CMS JupyterHub

- Download aus Moodle:
 - Embeddings: vectors.csv.gz
 - Query Images: queries.zip
 - Primer: vector_search-primer.ipynb
- Öffnet: <https://jupyterhub.cms.hu-berlin.de>
- Öffnet das Tensorflow-Environment

Pakete Installieren

- Verwendet das Tensorflow-Environment
- Einkommentieren:
- **!{sys.executable} -m pip install --no-input openai-clip**
- **!conda install --yes --prefix {sys.prefix} -c pytorch -c nvidia faiss-gpu=1.9.0**
- **!conda install --yes --prefix {sys.prefix} torchvision**

Laufzeiten auf JupyterHub des CMS

- Embeddings erstellen: ~4 Minuten
- FAISS Indexierung: ~0.1 sec
- Tipp: speichert eure Embeddings zwischen:

```
query_embeddings_path = "query_embeddings.npy"
if not os.path.isfile(query_embeddings_path):
    # Create new embeddings
    ...
    np.save(query_embeddings_path, query_embeddings)

else:
    # Load embeddings from file
    query_embeddings = np.load(query_embeddings_path)
```

Abgabe: Freitag, 24.01.25

- Upload des Jupyter-Notebooks
 - Jupyter-Notebook
 - HTML-Export des Notebooks
 - Ggf. eigene Bilder, die genutzt wurden (als ZIP)
- Ihr müsst alle Teilaufgaben erfolgreich bearbeiten

Fragen zur Aufgabe?