


Politehnica University of Bucharest
Faculty of Automatic Control and Computers
Department of Automatic Control and Systems Engineering

MASTER THESIS

From Pixels to Meaning: Organizing visual data with semantic embeddings

Absolvent
Vlad-Ștefan TUDOR

Advisor
Prof. dr. ing. Bogdan DUMITRESCU

Bucharest, 2023

Contents

1. Introduction

1.1. Abstract

In the present day, visual data, in the form of large collections of images, is unquestionably valuable - a sentiment shared by both large tech conglomerates and individuals alike. However, this wealth of data presents a paradox. While its intrinsic value is widely recognized, the lack of effective and efficient structuring mechanisms means that the information within remains largely inaccessible and underutilized. As visual data continues to proliferate at exponential rates, the necessity for methods that can infer meaningful structure within such collections has never been more critical.

While traditional methods for image categorization largely rely on manual tagging or simplistic metadata, such approaches are often cumbersome, time-consuming, and lack the ability to capture the nuanced relationships that may exist between images. For this reason, there is a noticeable paradigm shift towards self-supervised methods that aim to mitigate the need for large volumes of labeled data, offering a more scalable and dynamic way to understand visual content. However, most of the current systems for organizing image collections commonly depend on explicit metadata such as time stamps, geospatial information, or user-provided tags. These methods generally fall short when tasked with capturing the nuanced semantic relationships between images, particularly in scenarios where explicit metadata is either unreliable or entirely absent. In recognition of this limitation, the present work explores a different approach: inferring structure within a collection of visual data based solely on the images' semantic content.

1.1.1. The Need for Semantic Structure in Visual Data

The fundamental concept for this method is the idea of converting each image into a numerical representation, commonly referred to as an "embedding." These embeddings are generated using specific Machine Learning models. Once the images are projected into this numerical space, the topology allows for various distance metrics that can be used to define a measure of "similarity" between the source images. This approach offers the flexibility to dynamically discern semantic affiliations without relying on predefined hard-coded rules, making it particularly useful for tasks such as organizing large personal photo collections - a challenge that has proven cumbersome for most individuals.

In addition to solutions that can convert data to numerical representations, novel approaches introduce the concept of multi-modal Machine Learning models that can simultaneously process multiple types of input, specifically in this case, image and text. Similar to the image-to-embedding transformation, textual data can also be converted into compatible numerical representations. These text-derived embeddings share the same dimensional space and topology as the image-derived embeddings, enabling a unified framework for semantic analysis. This multi-modal capability enriches the search functionality, allowing queries to be formulated using either images or text. The unified embedding space thereby offers a more versatile and comprehensive approach for discovering semantic affiliations, opening up new avenues for user interaction and data structuring.

1.2. Objectives

The present work aims to develop an end-to-end solution for inferring relationships between samples of an unstructured collection of images by analyzing the semantic content of the data. Three main objectives can be outlined:

Robust Embeddings

The initial objective focuses on constructing a pipeline to produce reliable image embeddings. This entails selecting a suitable Machine Learning model from available candidates, then refining and fine-tuning it with custom data sets. The custom data-driven fine-tuning aims to validate the hypothesis that the model can be adapted to specific use-cases, such as recognizing particular individuals or locations. Additionally, auxiliary strategies will be investigated to potentially boost system performance, including the segmentation of images into relevant sub-components based on object detection via a separate model. In the context of a multi-modal approach, the topology shared between image and text embeddings will be examined to ensure that it facilitates intuitive relationships.

1.2.1. Data Structuring Methods

Leveraging the generated embeddings, this phase aims to explore methods for uncovering inherent patterns in the data. Given that the images are now adequately represented by numerical arrays, a range of computational operations can be applied. For instance, 'similarity' can be quantified as the inverse of geometric distance between data points, thereby enabling functionalities such as search (identifying the top-k most similar items) and clustering (grouping similar items in a manner distinct from other groups). The overarching objective is to assess how well these numerical measures align with the semantic or human-interpretive attributes of the images.

1.2.2. End-to-End System

Finally, all the insights from the previous stages have to be synthesized into a cohesive, user-friendly platform. This entails developing a robust backend infrastructure capable of conducting the image processing and analysis, managing the embeddings, and executing tasks like searching and clustering. Complementing this, a user-centric frontend will be designed to provide end-users with an intuitive interface for seamless navigation through their image collections, thereby achieving a transparent, efficient, and intuitive end-to-end solution.

2. Theoretical aspects

This section explores the fundamental concepts supporting the ideas presented in his thesis. The core of the project consists of the Machine Learning model able to process the images, so this can constitute an intuitive starting point for the theoretical introduction in the subject. In brief, this model is capable of extracting relevant features and semantic content and compress this knowledge into a representative encoding, a n-dimensional numerical array. The intuition behind this will be explored first.

Second, the model itself will be discussed, focusing on the architecture. Even though the model of choice is a Transformer-based architecture, several general concepts about the counter-part candidate, the convolutional neural network, are needed. Then the discussion will follow on how the model was effectively trained for the task of generating representative numerical encodings for the image inputs - more precisely, the definition of the criteria and the loss function to be optimized.

Following on the natural steps of this reasoning, once the matter of the numerical representations of the image is clarified, it is a question of how these numerical representations can be used. Various general algorithms can be applied, with one of two goals: compressing the numerical representations even more, if possible and the inference of possible patterns thought the samples encodings. Similarity between groups of samples can introduce the definition of clusters within the data, so proper definition of the theoretical aspects of such algorithms is also required.

2.1. Convolutional Neural Networks

Convolutional Neural Networks (CNNs) have emerged as a pivotal advancement in the field of image processing. They have also found applications in various domains beyond visual tasks, such as speech recognition and natural language processing. At the heart of CNNs lies the operation of convolution, defined as:

$$(f * g)(t) := \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau$$

Intuitively, this acts like a sliding window, called a kernel or filter, traversing over an input data matrix, multiplying corresponding elements, and summing the results to create a feature map. The weights of the kernel represent the parameters within a CNN network, which through the process of training, are optimized for specific patterns in the data, such as edges, textures, or shapes. Through a series of convolutional layers, CNNs gradually build a hierarchy of abstract features, enabling them to capture intricate patterns in the data.

Several legacy CNN architectures have significantly influenced the field of image processing. AlexNet[10], introduced in 2012, is often considered a pioneer, showcasing the potential of deep neural networks. ResNet[6], short for Residual Network, presented in 2015, introduced skip connections to address the vanishing gradient problem, allowing for the training of extremely deep networks. These networks have achieved remarkable performance on benchmark datasets, with ResNet in particular demonstrating impressive results in image classification tasks, notably surpassing human-level performance on the ImageNet dataset[18].

2.2. Transformers and Attention

The Transformer architecture, a groundbreaking innovation in the realm of deep learning, has revolutionized various machine learning tasks, particularly in natural language processing . At the heart of this architecture lies the mechanism of attention[23], a powerful concept that facilitates the model's ability to weigh and combine information from different parts of the input sequence. The term "attention" refers to the ability of the model to dynamically learn and emphasize specific parts of the input data, enabling it to focus on relevant information and extract meaningful patterns during processing. Mathematically, this is formulated as:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

considering that:

- Q represents the query matrix, which contains information from the input that is used for attending to other parts of the input
- K stands for the key matrix, which contains information from the input that is compared with the query to determine the relevance of different parts of the input
- V represents the value matrix, which contains the information that is ultimately retrieved or used based on the attention scores
- $\frac{QK^T}{\sqrt{d_k}}$ computes the scaled dot product between the query and key matrices. The dot product measures how much the query and key vectors align with each other
- $\sqrt{d_k}$ helps stabilize the gradients during training, with d_k being the dimension of the key vectors

This can be further extended to the concept of Self-Attention by having the key, query, and value matrices all derived from the same input sequence. The self-attention mechanism allows a sequence element to attend to other elements within the same sequence, capturing relationships and dependencies between different positions in the sequence.

Moreover, multiple Attention operations can be used in parallel, allowing the model to focus on different aspects of the input data simultaneously. This can be defines as:

$$MultiHead(Q, K, V) = Concat(head_1, ..., head_h)W^O$$

where an individual head is defined as the basic Attention:

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

Leveraging these mechanisms, the Transformer architecture proved a very powerful tool for interpreting sequential data, which incorporate dynamic relationships between different positions of the sequence. Unlike traditional recurrent or convolutional architectures, Transformers do not rely on fixed window sizes or sequential processing, allowing them to handle sequences of varying lengths and capture global context. Moreover, the self-attention mechanism enables Transformers to consider all positions in the input sequence simultaneously, making them highly parallelizable and efficient for training on large datasets.

2.3. Text processing with Transformer models

Even though the current project focuses on image processing, understanding how the Transformer architecture processes text, at least in broad terms, is significant. One of the approaches considered and then selected as a suitable candidate in the current project relies precisely on a model able to interpret text (as will be elaborated in 3.2.4. Therefore his section provides some relevant insights.

2.3.1. Tokenization

In order for a Machine-Learning model to process text inputs some pre-processing is required to convert the strings into a numerical representation. The text input is *tokenized*, which means that it is chunked down in smaller parts, which are larger than individual letters but smaller than words, in most of the cases. Choosing an efficient method for this splitting into smaller chunks is not a trivial matter and there are several methods such as Byte Pair Encoding (BPE) [20] or WordPiece [25]. These chunks will function as a base dictionary of all possible expressions in the input, therefore it is crucial to strike a balance between chunks that are too small, producing a short dictionary but constraining the model to more computation and chunks that are too big, creating redundancy in the input and output spaces of the model.

2.3.2. Encoder and Decoder

The Transformer architecture consists of two main components: the Encoder and the Decoder. These components work together in sequence-to-sequence tasks like machine translation and text generation. Both incorporate attention mechanisms and feed-forward layers in order to processes the input. In the middle of this architecture, the output of the encoder and simultaneously, the input to the decoder is a dense numerical representation that ideally holds as much information from the original input sequence as possible. The efficiency for this compression of the original data directly impacts the accuracy of the model output.

The Transformer introduced breakthroughs in the context of Natural Language processing with models such as GPT (Generative Pre-Training) [15], gradually making the architecture a standard in the industry. The versatility of the paradigm allowed for even more interesting results with application in other areas of Machine-Learning such as computer vision and image processing, as will be further discussed.

2.4. Vision Transformer

The Vision Transformer (ViT) architecture represents a breakthrough in computer vision by applying the Transformer architecture, directly to image data [3]. The key idea is to treat an image as a sequence of fixed-size patches and then use a standard Transformer model to process these patches as if they were tokens in a sentence. This approach eliminates the need for traditional convolutional neural networks in computer vision tasks, challenging the dominant paradigm.

In the ViT model, an image is divided into patches, and each patch is linearly embedded to create a sequence of patch embeddings. Position embeddings are added to convey spatial information, and a learnable ”classification token” is included in the sequence. The Transformer encoder is

then applied to this sequence of patch embeddings to capture relationships and features in the image.

ViT's remarkable success comes when it is trained at a large scale, often on millions of images. With sufficient training data, ViT surpasses state-of-the-art performance on various image recognition benchmarks, including ImageNet and CIFAR-100. ViT's power lies in its ability to leverage massive datasets and learn representations directly from images, challenging the traditional reliance on CNNs for computer vision tasks. While CNNs encode specific inductive biases related to image structure, ViT with its reliance on self-attention mechanisms, offers a more flexible and scalable approach to image understanding.

2.5. Embeddings and similarity

Embedding vectors, often referred to simply as embeddings, have emerged as a versatile tool for representing input data in various domains. These vectors are numerical arrays of arbitrary dimension, usually in the range of hundreds or thousands of float (real) values. Encoding input data in such manner can either be an emergent property of a classical model (trained for classification tasks, for example) or can be explicitly optimized as such. What is important is that they usher in a paradigm where data can be organized and manipulated organically, without the need for predefined categories or labels.

One of the most notable implementations of such a paradigm came with Word2Vec [13], a popular word embedding model introduced in 2013. It transforms words into high-dimensional numerical vectors, capturing semantic relationships between them based on their co-occurrence patterns in a large text corpus. In Word2Vec, words with similar meanings end up with similar vector representations, allowing for vector operations that reveal remarkable semantic insights. For example, by subtracting the vector for "man" from "king" and adding the vector for "woman," you indeed obtain a vector very close to "queen" in the embedding space. This demonstrates that Word2Vec captures not only the syntactic but also the semantic relationships between words, making it a valuable tool for natural language understanding and a foundation for many subsequent developments in word embeddings and NLP.

A similar approach in the field of Computer Vision same with a face recognition task. In such a scenario, the challenge lies in accommodating a variable number of potential categories (each representing a unique identity). This would not work in a standard classification paradigm, as it would require a retraining of at least part of the model with each change in the space of possible classes. In the paper introducing triplet loss [19], the solution to this problem emerged through the use of embeddings, trained to minimize a distance metric between samples of the same identity and to maximize the distance metric for the rest of samples. The name stands for the use of triplets, consisting of two same-class samples and a third sample as negative.

Drawing on this, it becomes clear that great potential lies in the paradigm of using embeddings and optimizing similarity relationships between them. In order to achieve this, a proper definition of similarity needs to be introduced. Similarity between two numerical arrays can be defined as various operations:

- Euclidean Distance: This metric measures the straight-line distance between two points in the embedding space. Smaller Euclidean distances intuitively indicate that the points are closer together.

$$\text{Euclidean Distance} = \sqrt{\sum_{i=1}^{n}(A_i - B_i)^2}$$

- Manhattan Distance: Also known as the L1 distance, this metric calculates the sum of absolute differences along each dimension between two points, similar to how a car would travel in a grid-like city, making it sensitive to changes in multiple dimensions.

$$\text{Manhattan Distance} = \sum_{i=1}^{n} |A_i - B_i|$$

- Cosine Similarity: Cosine similarity measures the cosine of the angle between two vectors. It quantifies the cosine of the angle formed when the vectors are placed tail-to-tail. Values close to 1 indicate high similarity, while values close to -1 indicate dissimilarity.

$$\text{Cosine Similarity} = \frac{\sum_{i=1}^{n} A_i \cdot B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \cdot \sqrt{\sum_{i=1}^{n} B_i^2}}$$

2.6. Clustering

Clustering, in the context of vector embeddings, is a powerful technique used to discover hidden structures and patterns within data. It is well-suited candidate for processing the output of an embedding model which can be viewed in this context as a mechanism for feature extraction, while the clustering algorithm then uncovers similarity patterns in the resulting space. It is particularly valuable when dealing with unstructured data, as it can reveal meaningful groupings without the need for predefined categories or labels. Clustering, in essence, is an unsupervised form of learning that allows for making sense of complex datasets by organizing data points into clusters of similar items.

Two commonly used clustering algorithms are K-Means and K-Nearest Neighbors (KNN)[9]. K-Means is an iterative algorithm that partitions data into "K" clusters, with each cluster center representing a cluster's centroid. Data points are assigned to the nearest centroid, and the centroids are then updated based on the mean of the points assigned to each cluster. KNN, on the other hand, identifies clusters by finding the "K" nearest neighbors for each data point and grouping them together. These algorithms are straightforward and effective, making them popular choices for clustering tasks.

Another noteworthy clustering algorithm is Density-Based Spatial Clustering of Applications with Noise (DBSCAN)[5]. DBSCAN identifies clusters by assessing the density of data points in their proximity. It groups together data points that are closely packed while marking isolated points as noise. DBSCAN is advantageous in discovering clusters of arbitrary shapes and sizes and is robust to noise in the data. However, it has limitations in handling data with varying densities, and determining the optimal parameters (like the minimum number of points to form a cluster) can be challenging.

2.7. Dimensional reduction

Dimensionality reduction is a fundamental technique in data analysis and machine learning, often employed to handle high-dimensional numerical data by projecting it into a lower-dimensional space. This process is particularly useful when dealing with data that has a large number of features or dimensions, as it helps simplify it while retaining its essential characteristics. It ties into the concept of embeddings discussed earlier, as the embedidngs generated by a machine learning model can be further used as inputs for such methods. However, it's essential to

acknowledge that dimensionality reduction can lead to information loss, and the choice of which dimensions to retain is critical.

One of the key challenges in dimensionality reduction is deciding which dimensions to keep and which ones to discard. When reducing dimensions, the goal is to maintain as much of the meaningful information as possible while eliminating redundant or noisy dimensions. The concept of linear independence is central here. If data dimensions are linearly independent, meaning that they do not exhibit strong relationships or dependencies on each other, then dimensionality reduction may be efficiently applied. However, embeddings generated by machine learning models will probably show greater linear independence as the models themselves optimize for relevant features. However, even with a moderate degree of information loss, such algorithms may still be useful in visualizing the embedding space intuitively.

Two well-known dimensionality reduction algorithms are Principal Component Analysis (PCA)[4] and t-distributed Stochastic Neighbor Embedding (t-SNE)[12]. PCA, in essence, seeks orthogonal axes (principal components) in the data space that maximize the variance of the data when projected onto them. By selecting a subset of these principal components, PCA effectively reduces dimensionality while preserving some of the variance. It is particularly effective when the data's intrinsic structure aligns with the principal components.

On the other hand, t-SNE focuses on preserving the pairwise similarities between data points. It models high-dimensional data as points in a lower-dimensional space, adjusting their positions to emphasize the similarity relationships. Data points that are similar in the high-dimensional space are encouraged to be close to each other in the lower-dimensional space, while dissimilar points are pushed apart. This makes t-SNE highly effective at visualizing clusters and revealing local structures in the data. Its non-linearity and ability to capture complex relationships make it a popular choice for tasks like visualization and exploratory data analysis. However, it's essential to note that t-SNE does not necessarily preserve global structures as effectively as PCA.

3. Methodology

In order to transform the broad challenges and objectives outlined in the introduction into actionable solutions, a detailed methodology is further detailed. This section discusses the various approaches and techniques employed to build an end-to-end system capable of inferring relationships in unstructured collections of images based on their semantic content. Three main pillars of the methodology correspond directly to our objectives: first, generating robust image and text embeddings; second, devising effective methods for data structuring like search and clustering; and third, synthesizing these components into an operational, user-centric system. Each pillar involves a blend of computational methods applied to a candidate model for processing images into embeddings, benchmarking strategies, and techniques designed to ensure both robustness and usability. This section addresses each of these areas, providing a blueprint for the project's technical architecture and insights into the choices made during the development process.

3.1. Choosing an embedding generation model

The selection of an appropriate model for generating embeddings serves as the cornerstone of this project. This choice has far-reaching implications for the system's performance and capabilities: it dictates the dynamic behavior of the embeddings in relation to input variability, sensitivity to noise, and how they can be used downstream in search and clustering. Several candidate architectures are available for generating image embeddings, and each presents its own attributes and limitations.

3.1.1. Architecture of the model (CNNs and Transformers)

The conventional choice in computer vision applications is the Convolutional Neural Network (CNN)[21]. A typical CNN consists of a series of layers that perform convolutional operations, interspersed with pooling and normalization steps, designed to incrementally abstract the input image into a set of features. These features are then flattened into a one-dimensional array that serves as the numerical representation or "embedding" of the image. Usually, subsequent to this layer, there is a densely connected neural layer that represents the actual output of the model. In most classification tasks, this output directly correlates with the number of possible classes, with each neuron acting as a placeholder for that corresponding label. In that case, its activation value for a given input passed to the network represents the "confidence" the model has towards that class.

A paradigm shift in the field came with the introduction of Transformer architectures [23]. Initially designed for Natural Language Processing, Transformers have now been successfully adapted to a wider range of tasks, including computer vision tasks [3]. Unlike CNNs, Transformers offer greater flexibility in capturing long-range dependencies and complex patterns. Since the general Transformer architecture can be adapted to the same training tasks and criteria as regular CNNs, the comparison here stands in the capability of the network to process the latent patterns in the data. However, while Transformers have demonstrated state-of-the-art performance on a wide array of complex challenges, their computational and data requirements

can be prohibitive. Moreover, not all problems necessitate the flexibility and complexity that Transformers provide.

A solution where the model accounts for more complexity than the data itself can lead to overfitting and overall lower performance. While Transformers are undeniably powerful, their utility depends on the specific demands of a project. In the present context, it's crucial to understand that complexity doesn't always correlate with better performance. CNNs, although more rudimentary, can offer competitive results, particularly for tasks that don't require the understanding of long-range dependencies [14].

3.1.2. Criteria for Model Training

Arriving at a model capable of generating embeddings for input images can happen both as an emergent property or as direct explicit criterion. This distinction is pivotal for the present work, as the topology of the embedding space is intrinsically linked to the specific loss function being optimized. The training objectives are largely agnostic to the underlying architecture, be it Convolutional Neural Networks (CNNs) or Transformers, so the discussion holds irregardless of the choice of architecture.

One prevalent training objective in the machine learning landscape is Classification. This method offers both practicality, since a wide array of problems can be reformulated as binary or multi-class classification tasks, and robustness, as the loss is calculated against a fixed label representing the target class. A common technique for extracting embeddings from a classification-trained model involves leveraging the layer immediately preceding the output. While the output layer is configured to represent the potential classification categories, the preceding layer is considered to encapsulate the salient information needed for such classifications.

Another approach to generating embeddings involves the use of Autoencoders[2], which consist of two primary components: an encoder that condenses the input data into a lower-dimensional form, and a decoder that aims to reconstruct the original input from this compressed representation. The training focuses on minimizing the error between the original and reconstructed data, thus creating an incentive for the model to capture essential features in its internal representation [26]. However, this method may be limited if the features optimized for reconstruction are not inherently discriminative for subsequent tasks, occasionally yielding less-than-ideal embeddings.

Lastly, an alternative strategy is to make the embedding itself the primary output and to tailor the training criteria to reflect the relationships within the training data directly onto the structure of these embeddings. This is frequently achieved through the use of contrastive losses[24], where distance metrics are applied during training to ensure that similar inputs yield similar embeddings, while dissimilar inputs are spread apart in the embedding space. This strategy effectively accentuates the distinctive features of the input data, thereby capturing its nuances in a focused manner. However, it's worth noting that imbalances in the dataset and variations in input formats could introduce challenges, such as causing the embeddings to collapse or diverge.

3.1.3. Aligning architecture and criteria with project objectives

It is crucial to reconcile these insights with the specific aims of the project. Recent research demonstrates that Transformers surpass Convolutional Neural Networks (CNNs) in performance on standard benchmarking datasets [1]. As for the training criteria, theoretical considerations suggest that a hybrid approach may be most effective. Specifically, the incorporation of contrastive loss appears to be well-suited for distinguishing subtle differences in the inputs, which

is one of the core points of the project [7]. At the same time, a secondary objective that counterbalances the first might be useful in focusing on what the input "is" rather than what "it is not". In essence, this can be thought of as a dynamic interplay of opposing forces striving for equilibrium: one criterion aims to tease apart dissimilarities, while the other aims to recognize inherent similarities.

3.2. The CLIP model

Taking into account the considerations discussed earlier, a promising model candidate is OpenAI's CLIP (Contrastive Language-Image Pretraining)[16]. It is a model trained for the task of image captioning — generating descriptive text captions for visual inputs. This criteria aligns well with the principles of balancing out contrast and similarity in the input image through nuances in the caption text, rather than rigid labels. At the same time, the CLIP model comes in multiple architecture variants, comparing performance between traditional convolutions and Transformer. It is also worth noting that the results in performance achieved by the model show that CLIP is competitive even on the traditional classification tasks, as will be discussed below.

Utilizing pre-trained foundational models like CLIP is a pragmatic strategy that addresses the large computational costs associated with training cutting-edge machine learning models from scratch [27]. Such models provide both a robust baseline for immediate deployment and allow a stable starting point for fine-tuning on task-specific data. This approach is especially valuable when extensive hardware resources are limited, as it offers an achievable route to competitive performance.

3.2.1. Architecture

The CLIP model utilizes two distinct architectures for its image and text encoders. For image encoding, it employs either a modified ResNet-50[6] or a Vision Transformer, with adjustments like attention pooling and layer normalization to improve performance. The text encoder is based on a Transformer architecture and operates on a byte pair encoding (BPE)[20] representation of the text. It features a 63-million parameter 12-layer model with 8 attention heads and has a maximum sequence length capped at 76 tokens. Both the image and text feature representations are projected into a joint multi-modal embedding space, where their pairwise similarities are calculated. The model employs a symmetric loss function to learn from both image and text data, enabling it to generalize across various tasks effectively.

3.2.2. Training Dataset

The CLIP model was trained on a dataset consisting of 400 million (image, text) pairs collected from the internet [16]. This training data was used to pretrain the model by predicting which caption corresponds to each image. This approach leveraged a vast source of supervision provided by natural language. The dataset's size and diversity allowed the model to learn rich image representations. After pre-training, CLIP demonstrated its ability to reference learned visual concepts using natural language, enabling zero-shot transfer to downstream tasks.

3.2.3. Reported performance

CLIP's performance in zero-shot learning scenarios makes it a highly promising candidate for tasks that require identifying similar patterns in input images. The model demonstrates impressive versatility across a range of datasets, often outperforming or matching the benchmarks

set by fully supervised or few-shot models [16]. In particular, it excels on fine-grained classification tasks and general object classification datasets such as ImageNet and CIFAR10/100, while also showing state-of-the-art results on STL10 without any task-specific training. Its zero-shot classifier not only outperforms a ResNet-50 on action recognition datasets like Kinetics700 but also roughly matches the performance of a 16-shot classifier using a BiT-M ResNet-152x2[8] on ImageNet-21K, emphasizing the generality of its input features. Such robust performance across varied tasks and datasets strongly suggests that CLIP can effectively capture a broad range of visual concepts, making it well-suited for projects requiring the identification of complex patterns in images.

3.2.4. Multi-modal capabilities

Finally, leveraging CLIP's capability to create multimodal embeddings that link visual and textual information opens up a new and valuable feature for the project: the ability to search for images in using natural language queries. By converting both the text queries and the images into a shared embedding space, the most similar pairs can be identified, similar to the evaluation method on image classification tasks in the CLIP paper. This feature essentially transforms the unsorted image collection into a searchable database, enabling users to effortlessly retrieve specific images or sets of images that closely match their textual queries. The adaptability of CLIP's zero-shot learning across various domains suggests that this feature will be broadly applicable, offering a versatile and powerful tool for image search and organization.

3.3. Fine-tuning the model candidate

While making use of the generality of the model's image analysis capabilities, one particular use case of interest remains the system's ability to properly handle custom data. In the context of organizing a collection of images, it can be relevant to tailor the model to recognize specific persons, objects, or places that might not have been included in the original training dataset. By taking the pre-trained embedder model and subsequently training it on a custom dataset, it can effectively be adapted to these specific scenarios.

To fine-tune the CLIP model for specialized tasks, a tailored dataset featuring target images and associated captions will be assembled. However, creating a dataset that is too narrowly focused, for example consisting of a single entity, therefore making the captions also very similar, could result in unintended model behavior. Such behaviour might include collapse of the model's learnable weights. Even when using a diverse set of training images, the fine-tuning process carries the risk of diminishing the model's general performance on the broader dataset it was originally trained on [22]. To counteract potential overfitting, a balanced training approach is proposed: integrating the custom dataset with examples from CLIP's original training data in a consistent ratio. This strategy aims to both preserve the model's broad utility and adapt it to the specific demands of the project.

The goal of the fine-tuning experiments is to develop a universal methodology that can be applied irrespective of the type of custom data being used. These tests will focus on a facial recognition dataset to evaluate the model's ability to differentiate between visually similar inputs—namely, portrait images. To determine the success of the experiments, performance metrics will be compared between the original and fine-tuned models across both the general-purpose dataset and the specialized training dataset. A successful outcome would be indicated by the model maintaining comparable performance on the original dataset while showing marked improvement on the specialized dataset.

3.4. Benchmarking the topology of the embedding space

The main consideration of the project is that the similarity between two images can be determined by examining the geometric proximity of their respective embeddings, typically using a standard distance metric such as euclidean or cosine distance. However, a practical implementation that successfully delivers on this goal requires more in-depth analysis of the topology of thai embedding space.

One of the primary concerns is establishing numerical benchmarks that define what constitutes the distance of two ”similar” samples and what should be considered as indicating two ”distinct.” Moreover, the relative behavior of text and images has to be studied. Is there a bias towards matching embeddings of text-to-text or image-to-image pairs rather than the corresponding text-to-image pairs? This needs to be evaluated through empirical testing on data distributions that fit the target scope, in this case, common objects and persons.

Secondly, the linearity and behavior of the embedding space itself warrant scrutiny. Specifically, how well does the geometric distance between two embeddings reflect the variations in the corresponding image inputs? Are there any discernible patterns that suggest certain kinds of images, such as two animals, would have closer embeddings compared to a more diverse pair, like an animal and a car? Given that these embeddings were originally trained in a caption-based task, the notion of image similarity is intrinsically linked to the similarity of their potential captions. Therefore it is important to assess how these distinctions came to be represented through the training process.

Lastly, on the semantic front, the model’s prioritization criteria remain somewhat ambiguous. For instance, it’s unclear whether the model gives more weight to the identity of the subject over the format of the image. For example, if provided with three samples—two portraits of individuals A and B, and a full-body image of individual A—which pair would the model deem the closest in the embedding space?

These questions form the basis for a nuanced understanding of the model’s behavior, offering valuable insights that could inform further fine-tuning and application. The corresponding analysis results and conclusions will be discussed in the Results section.

3.5. Inferring structure from embeddings

In the following section, methods for leveraging the generated embeddings to uncover latent relationships between images are explored.

3.5.1. Search

The first practical application under consideration is the implementation of a ”search” feature. This involves querying an indexed database of images, each paired with its corresponding embedding, to find the most similar match based on minimal distance between the query and the existing embeddings. The model’s capability to handle both image and text adds another layer of versatility to this function, as the query can be represented by either one of the two input types. However, a key question that needs to be addressed, as highlighted in the previous section, is whether text queries yield similar distance results when compared to their image-based counterparts.

3.5.2. Clustering

Extending the concept of search from a one-to-many to a many-to-many framework naturally evolves into a clustering task. Utilizing the embeddings from the image data pool, various clustering algorithms can be applied to identify potential groupings. This functionality would serve as a cornerstone for the end-to-end project, allowing for the dynamic and organic organization of what was initially unstructured data. This not only improves manageability but also enhances the utility and user experience of the system.

DBSCAN

In the context of the project, the DBSCAN (Density-Based Spatial Clustering of Applications with Noise)[5] algorithm presents a particularly well-suited approach for clustering image embeddings. Unlike algorithms such as k-means, which assumes clusters to be convex-shaped, DBSCAN identifies clusters as areas of high density separated by regions of low density. This makes it flexible enough to capture clusters of various shapes and sizes. The algorithm operates on two primary parameters which dictate what constitutes a 'dense' region. Essentially, a core sample is an element in the dataset with at least `min_samples` neighbors within an `epsilon` distance, highlighting its location in a high-density region of the vector space. Clusters are formed by iteratively exploring the neighbors of these core samples. Additionally, DBSCAN distinguishes between core samples and non-core samples—the latter being data points that are close to core samples but are not dense regions themselves, often considered as the peripheries of a cluster.

The choice of `min_samples` and `epsilon` is critical: while `min_samples` affects the algorithm's noise tolerance, `epsilon` influences the size of the clusters. An inappropriately small `epsilon` value might result in a large portion of the data being categorized as noise, whereas a too-large `epsilon` could lead to the merging of distinct clusters. Given the nuanced nature of image embeddings, where slight differences can have significant semantic implications, fine-tuning these parameters will be crucial for achieving optimal clustering results.

3.5.3. Projections

The final analytical feature to be introduced is the "Projections" feature. By leveraging the generated embeddings the data can be mapped into a three-dimensional space that, ideally, retains some of the geometric and relational properties inherent in the higher-dimensional embedding space. Given that our CLIP embeddings are 512-dimensional arrays, a primary challenge lies in dimensionality reduction while preserving the intrinsic variance in the data. Unlike traditional feature sets that can often be reduced in dimensionality without much loss of information, embedding spaces pose a challenge due to their linear independence across dimensions. While the focus is largely on visualization for the end user, our approach aims to find a compromise by associating the 3D projections with clusters identified through our clustering algorithms.

t-SNE and PCA

In considering algorithms for this task, two main candidates emerge: t-SNE[12] and PCA[4]. Although both are popular techniques for dimensionality reduction, they serve different purposes and have different strengths and weaknesses. PCA (Principal Component Analysis) is a linear method that seeks to explain the maximum amount of variance in the data using orthogonal

components. It is computationally less intensive and quicker, but it often falls short in capturing nonlinear relationships within the data.

On the other hand, t-SNE (t-Distributed Stochastic Neighbor Embedding) has several advantages that make it more suited to the project's objectives. It excels in retaining local data structures and is sensitive to the scaling of different manifolds or clusters within the dataset. This is particularly useful for visualizing data that lies in multiple, different clusters or manifolds, as is the case with the current image embeddings. While t-SNE is computationally more expensive and generally limited to two or three-dimensional embeddings, its ability to reveal structures at multiple scales and reduce the crowding of data points makes it a powerful tool for this application. Furthermore, any shortcomings concerning the preservation of global structures can be mitigated by initializing the algorithm with PCA, thereby combining the strengths of both methods.

Therefore, given the need to capture and visualize the complex, nonlinear relationships within our high-dimensional data, t-SNE offers a more advantageous solution, albeit with trade-offs in computational efficiency and consistency across multiple runs.

3.6. Enhancing system performance by further image analysis

The potential for augmenting the overall system performance through additional image processing is explored next. One promising avenue involves generating embeddings for specific subregions within an image, effectively narrowing the embedding model's focus to select elements, such as crops of persons in the image. This nuanced approach would enrich the capabilities of search and clustering algorithms, enabling a more detailed recognition process. For instance, a single image could now belong to multiple clusters based on its different components. Extending this idea further, a full-scale object detection model can be incorporated into the system's image processing pipeline.

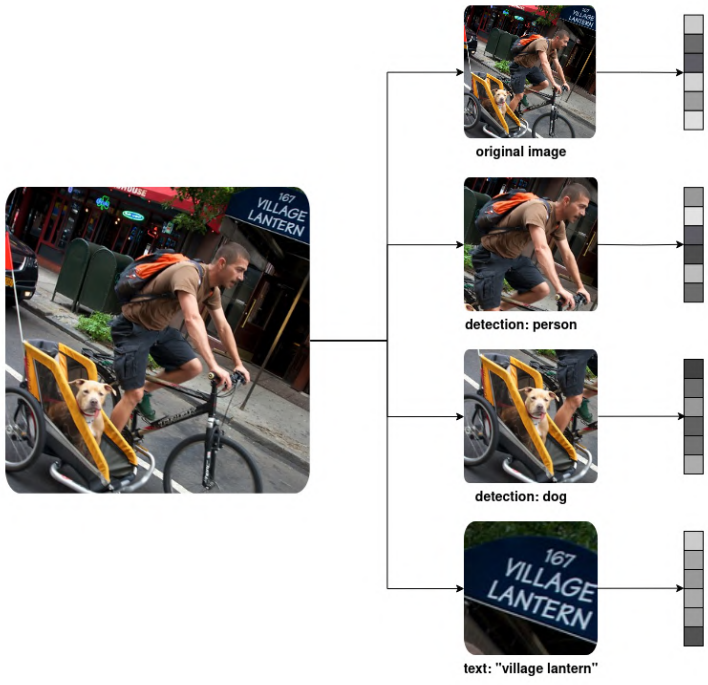


Figure 3.1.: Identifying key elements within the source image and generating representative embeddings for their corresponding crops

The object detector of choice is the Faster R-CNN Resnet model[17], trained on the COCO (Common Objects in Context) dataset, thereby broadening the range of elements the system can recognize in a generalist manner. To prevent sample overload due to excessive detection, thresholds can be established to consider only elements occupying a significant portion of the image, effectively filtering out minor or irrelevant objects.

Moreover, a second layer of processing can be considered for handling text within images, a common feature in many scenarios relevant to this project. While Optical Character Recognition (OCR) is a complex task in its own right and outside the scope of this endeavor, some textual content within images may still offer valuable insights. Words like ”Cafe” or ”Restaurant” in a storefront image, for instance, can serve as useful query points. In cases where the text is sparse and succinct, a straightforward OCR integration could capture this additional layer of information, which could then be processed by the text encoding component of the model.

While these additional processing steps significantly increase the diversity of samples that the system can handle, they also introduce a heightened risk for false positives. This trade-off merits close scrutiny but, within the defined scope of the project, is considered an acceptable compromise for the enhanced analytical depth it brings to the image collection.

3.7. Implementation

The methodology regarding the final objective is to design and develop the end-to-end system in an user-centric approach. The problem in question, that of being able to sort through vast amounts of valuable information that otherwise would be of little use, is tackle-able only as much as the potential solution is intuitively accessible. This paradigm, with the end-user at the forefront poses valuable practical challenges at each stage of the implementation. As a result, while the actual user interface is not a main feature in itself, the entire system can be thought of as being a sort of interface for the disorder in the raw input data.

The development of the codebase focused on the principles of generality and expandability to ensure a robust system that can adapt to various usage scenarios. The development approach was functionality-driven, starting with the identification of target functionalities that the system needed to perform. Modules were then incrementally constructed to meet these requirements. Throughout this process, particular attention was paid to elements that could be common across different pipelines. This allowed us to create a modular and extensible codebase that not only fulfills its immediate objectives but is also well-positioned for future enhancements. Adherence to well-established software development principles further ensures that the codebase remains maintainable and scalable as the project evolves.

4. Experiments

4.1. Benchmarking embeddings

To delve deeper into the structure of the embedding space, several experiments are conducted on a selected candidate from the CLIP (Contrastive Language-Image Pre-training) model collection. While high-level performance metrics measured from large-scale datasets offer a broad understanding of a model's capabilities, they often fail to translate these abstract evaluations into actionable insights for specific use-cases. The experiments described herein aim to bridge this gap by examining the model's performance on specific data patterns, focusing particularly on the actual similarity or dissimilarity of embeddings generated for images that are deemed to be similar or different according to human judgment.

To establish a robust ground truth, two distinctive datasets are used:

- COCO (Common Objects in Context)[11]
- LFW (Labeled Faces in the Wild) [**LFW**]

The COCO dataset provides annotations for images, defining individual objects present in the image as bounding boxes coordinates and corresponding labels ('person,' 'car,' 'dog,' etc). This information can be used to extract the exact image crop for the target object thus creating a subset of the data which alligns with a classical categorization scenario - 1-to-1 pairing of input images with well-defined labels. A selection can be made to filter out very small detections which can introduce noise in the system without the visual context surrounding them (there are very small object in the COCO dataset, identifiable as such only by their surrounding space which is not neccessarily included in the actual annotation).

The LFW dataset consists of facial images grouped by individual identities, for example, photos of Arnold Schwarzenegger, Selena Williams, or George Bush and so on. These datasets are valuable for the current experiment as they allow for the creation of well-defined groups of instances against which the model's embeddings can be rigorously evaluated in the context of face recognition. However, it is important to consider that some of the data employed may have been part of the model's training set (which is not explicitly detailed in the original paper). This is particularly relevant for the LFW dataset, as the model has shown to not only distinguish between the faces but also associate them with distinct identities, when the captioning task was evaluated with the names of the respective persons. Such behavior is expected given the public nature of the figures in the LFW dataset.

Additionally, it should be acknowledged that the chosen data represents a best-case scenario in terms of the quality of crops and minimized background interference. Such clean data can serve as a baseline for evaluating similarity metrics but may also set a higher standard than would be experienced in more cluttered, real-world scenarios.

The evaluation will be computed on each dataset individually. Specifically, the similarity of embeddings within and between the predefined groups will be computed evaluated —e.g., how similar are embeddings for all 'car' images amongst themselves and when compared to 'bus'

images, and so on, For computational efficiency, the smallest available model in the CLIP collection is chosen. This choice serves a dual purpose: it not only keeps computation times within a reasonable limit but also provides a conservative benchmark for performance, as larger models in the collection are expected to yield even better results. Importantly, it's worth noting that since all models in the collection share a common training regime, the underlying structure of the embedding space is likely to remain consistent across different model sizes.

4.1.1. Similarity between objects

Different object-pair similarities are evaluated using some of the classes (selected by their corresponding number of samples) of the COCO validation subset. In order to maintain a relatively balanced distribution between classes, a hard limit for the number of samples of each class is applied. The following data distributon is used:

Table 4.1.: Number of Samples per Label

Label	Samples	Label	Samples	Label	Samples	Label	Samples
apple	20	zebra	79	pizza	97	horse	80
person	500	clock	13	sheep	56	skateboard	11
bird	38	motorcycle	102	dog	97	laptop	82
donut	28	skis	10	banana	48	car	77
dining table	342	surfboard	17	cat	130	teddy bear	77
bus	116	truck	81				

From this experiment, several conclusions can be drawn. First, a threshold to distinguish between same-object similarity can be chosen around the value of 0.75. We can see that lower values are obtained for classes that intrinsically hold greater variability both in terms of the depicted objects themselves and in terms of representation in the dataset used for the evaluation. Second, a certain pattern emerges that these similarity scores can be used to infer upstream super-classes, like for example is the case here with the “animals” classes: “sheep”, “dog”, “horse”, or ”vehicles”. This showcases the learned generalization capabilities of the model which would subsequently be of great importance in the project.

4.1.2. Similarity between images of persons

This experiment focuses in the scenario of distinguish between generally similar images (the format of a portrait), nuanced amd subtle differences that account for recognizing a face. Since the model's training contained a various amount of data with diverse content, it is neccessary to assess the capabilities for this fine-grained task. In the context of the project, where the model is integrated into a search engine, the ability to accurately identify individual persons distinctively from others holds significant relevance.

Several instances from the LFW dataset are selected based on the umber of available samples. In order to maintain consistency throughout the experiment, all the present classes are sampled to a number of 25 images.

An important observation is that the model maintains a consistent similarity threshold to that of the previous experiment on images of objects. This suggests that the model is capable of nuanced analysis, beyond a simplistic hierarchical classification of the data. One might expect that all images in this experiment, which predominantly can be described as 'personal portraits', would inherently have a high baseline similarity. However, that is not the case. This observation

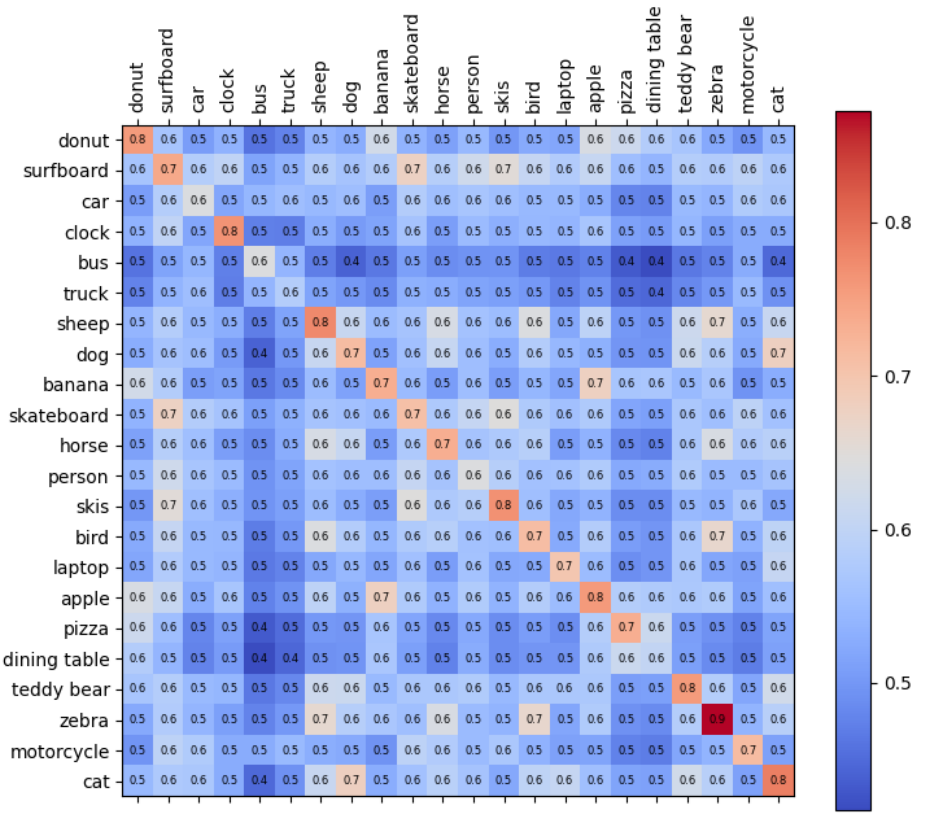


Figure 4.1.: Average cosine similarity between samples from several classes of the COCO dataset [11]

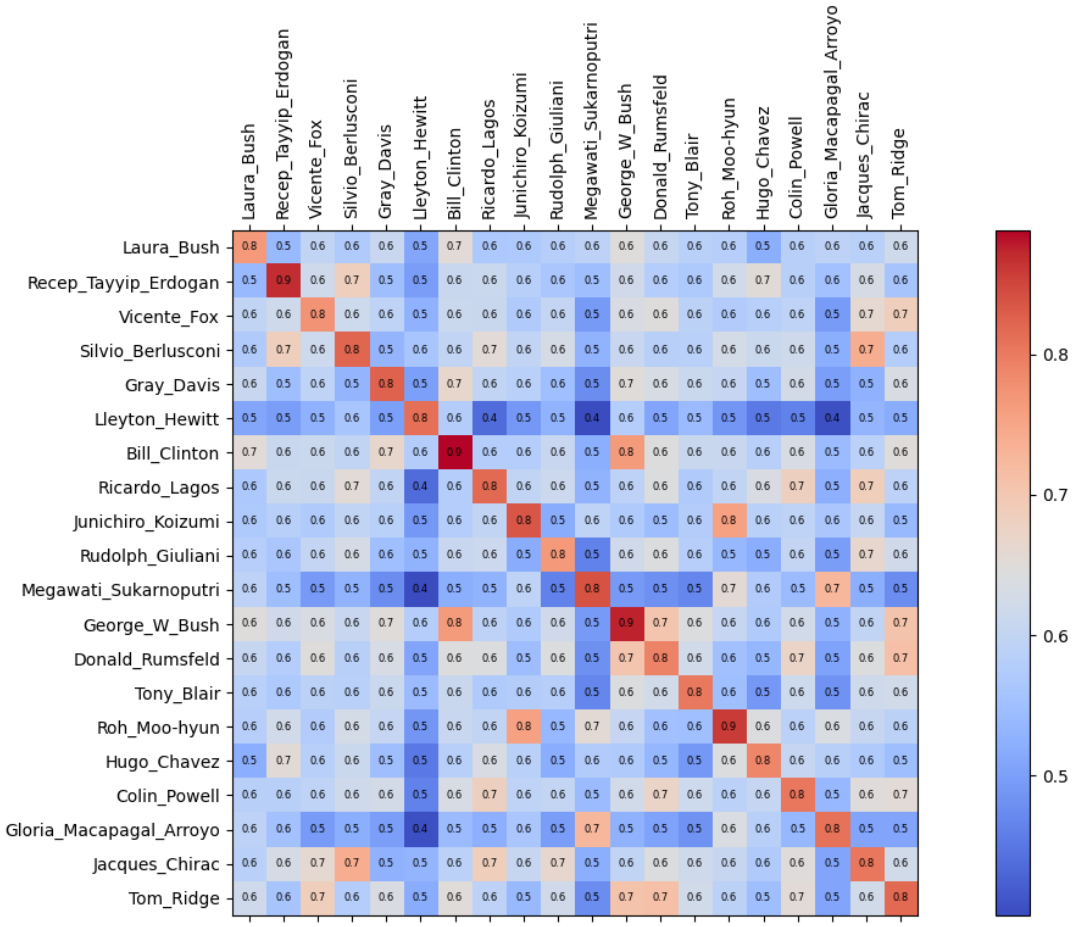


Figure 4.2.: Average cosine similarity between samples from several entities of the LFW dataset [**LFW**]

reveals that the model can discern subtleties, and leverage them to produce sufficiently distinct embeddings, even within a category that is seemingly homogeneous, affirming its robustness.

This insight may contradict traditional expectations or intuition, yet it offers a unique advantage for the scope of this project. It enables the establishment of a uniform threshold for similarity, regardless of the input type. This characteristic implies that the geometry of the embedding space is far from trivial and defies simplistic categorization or interpretation.

In terms of the project's focus, the choice of datasets, comprising common objects and recognizable faces, has proven to be highly relevant. The performance of the model across these different types of data suggests its potential as a universal image encoder. The geometric relationships forming in the embedding space provide meaningful insights, emphasizing the model's capability to convert various images into a form that captures essential features. This makes the model particularly promising for a wide array of applications that require an understanding of image content.

4.1.3. Image to text similarity

Image captioning is the training criterion for the model, yet its practical implications are not immediately obvious. To enhance the empirical understanding of the model's behavior, the

experiment employs selected samples from the COCO dataset, which offers both images and their corresponding captions. Given that multiple captions are available for each image, the dataset provides a rich ground for comparing not just different instances, but also multiple captions for the same image.

One key inquiry is the degree of similarity between the embeddings of an image and two or more differing, but equally valid, text descriptions. This aspect is crucial, as the nature of the captioning task is inherently variable. The experiment aims to assess the model's robustness in accommodating different textual descriptions for the same image.

Furthermore, the experiment seeks to understand the model's behavior when pairing unrelated text and image inputs. A well-performing model should not arbitrarily match any given text with any image, and vice versa. To this end, a similarity matrix is computed to evaluate the relationships between various pairs of inputs.

Additionally, the experiment explores the similarity between text-to-text embeddings. Given that different sub-models handle image and text inputs respectively, there may exist a bias favoring either same-type or different-type inputs. This facet will also be analyzed.

As this is an empirical assessment of the functionality of the model, the specific images and captions used for the results are provided in Figure 4.3. Other experiments were conducted, on different sample but with similar conclusions, therefore in the present section, the focus will remain on this particular set of data and the discussion of their corresponding outputs, presented in Figure 4.4.

The similarity matrix, presented in Figure 4.4, is symmetrical along its diagonal, but both halves are retained for clarity of display. The plot can be divided into three regions of interest: the top left corner, which shows caption-to-caption similarities; the bottom right corner, which displays image-to-image similarities; and the bottom left corner, which highlights caption-to-image similarities. For equivalent caption pairs (e.g., 1 and 2, 3 and 4, etc.), the same reference image is used. This is further corroborated by the presence of a similarity value of 1 on the double diagonal in the bottom right corner of the figure, which represents image-to-image comparisons.

One notable point is that the model may not be entirely robust to variations in text inputs. Text processing is inherently complex, and generating meaningful embeddings from even short pieces of text is not trivial. Despite this, the model maintains decent values for similarity between corresponding captions and images, relative to the other samples. This suggests that an activation function could be employed to equalize the higher similarity scores, where even significant differences may pertain to similar objects, and to expand the lower similarity scores, where the objects are likely to be different.

Input type bias

One other observation reveals that the similarity scores between image and text inputs are significantly lower than those between same-type inputs. This pattern is consistent across multiple experiments and indicates a model bias when bridging the gap between the two distinct sub-models: the image encoder and the text encoder. Despite both encoders outputting embeddings of the same size, their input/output dynamics are not completely aligned.

Within the scope of the current project, this observation yields two important implications. First, if the system aims to seamlessly integrate text and image data through direct pairwise similarity comparisons within a mixed-type data pool, the model will not behave as desired. Ideally, clustering should not be biased by the type of input but by its semantic content. This objective is not directly achievable given the observed model behavior.

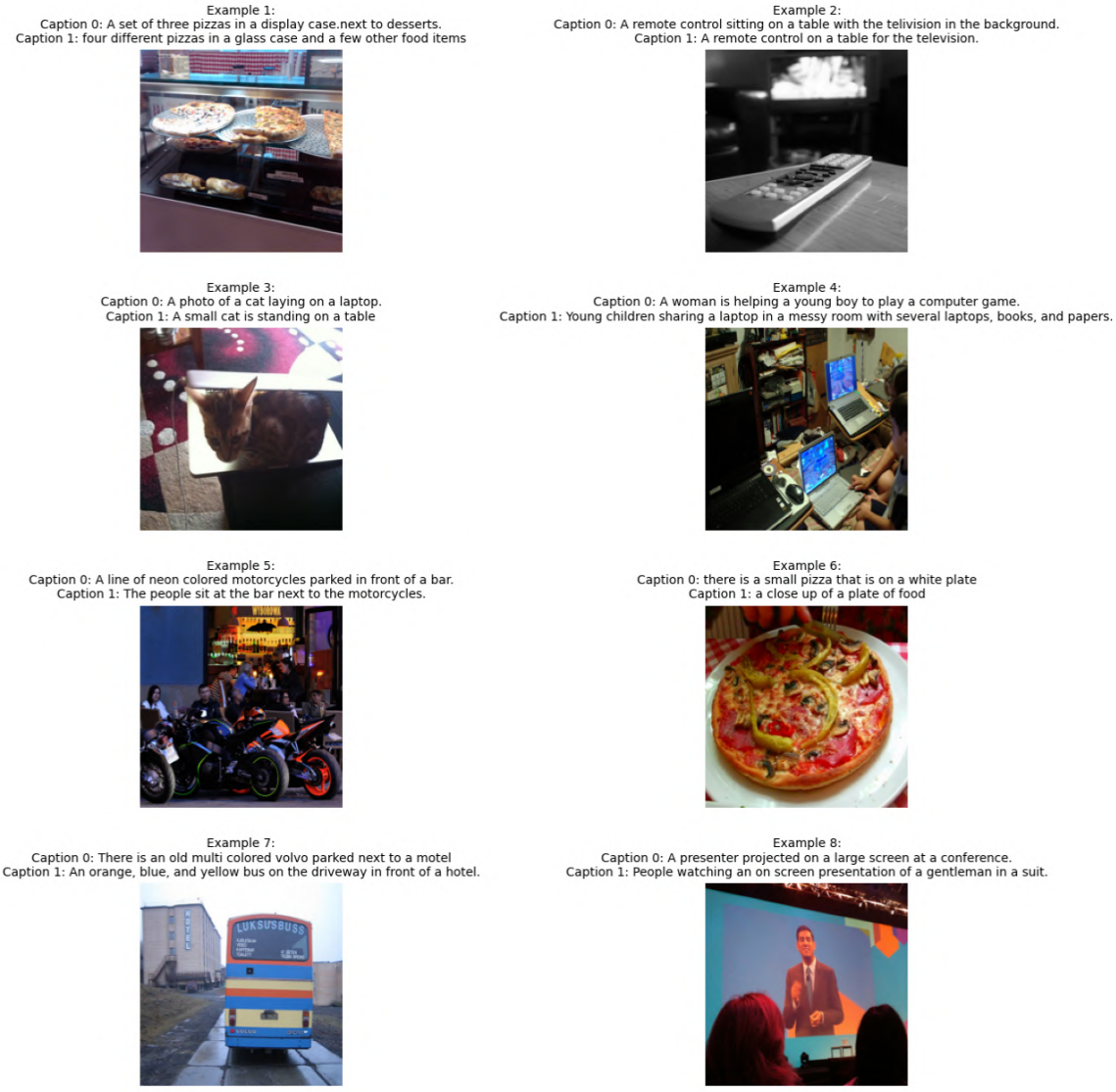


Figure 4.3.: Samples of the captions in the COCO dataset [11]

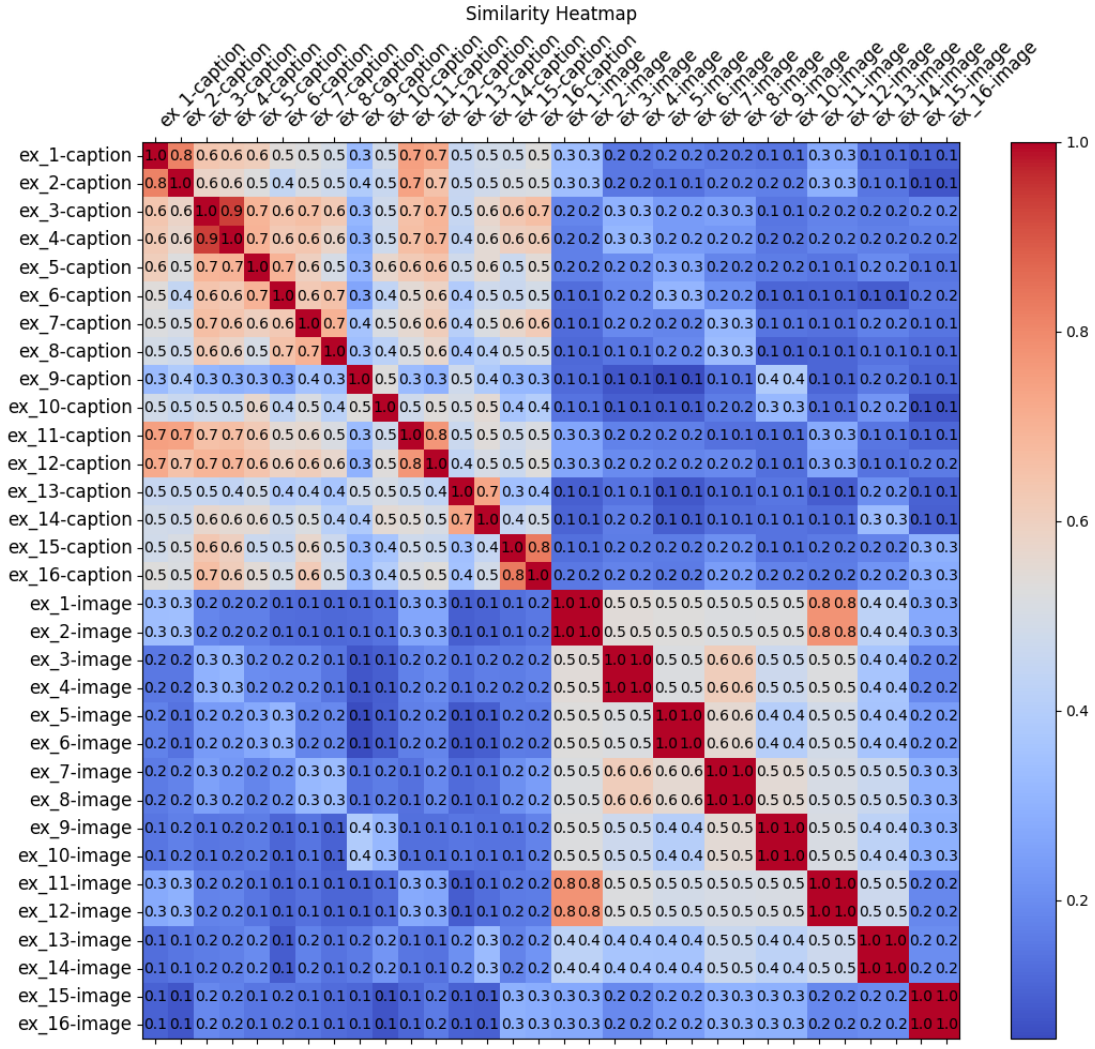


Figure 4.4.: Cosine similarity between the sample images and captions presented in 4.3. Example numbering refers to the captions, therefore sample 1 corresponds with ex_1 and ex_2, etc.

However, this bias appears to be linear, suggesting that a simple scaling factor applied to mixed-type pairs could alleviate the problem. Specifically, in search applications, such as search, which consist of one-to-many comparisons, the type of the query doesn't disrupt the relative ranking of the results, provided that the search pool consists of one consistent type of data.

This observed bias has to be put in the context of the training pipeline. It is at first counter-intuitive that a well performing model on the benchmark datasets would hold such contrasting bias. However, during training, the loss function optimizes the matching inside of a batch, similar to multiple parallel search queries. As described above, these searches, given that they are always performed in the same format (text query on image samples) is not necessarily affected by the bias between data types, thus not being penalized by the loss function.

This insight is crucial for the project, particularly when aiming to establish a consistent confidence threshold for filtering results in both text-based and image-based searches.

Another idea during development of the system was to apply a clustering algorithm on the pool of all data, both original images and their relevant sub-parts. These crops would usually also be images, as extracted by the defining boundaries of the detected objects. However, in other cases these sub-parts would essentially be text inputs extarcted with an OCR algorithm. Thus applying a clustering alogorithm on top of all the cllected samples, regardless of their input type would not work, given the current observations.

4.2. Benchmarking Data Analysis

Methods used in this project for further analyzing the embedding space consist of t-SNE (t-Distributed Stochastic Neighbor Embedding) algorithm for dimensionality reduction and DBSCAN (Density-based clustering Algorithm) for clustering. Both types of methods serve as pivotal tools in the analysis and interpretation of high-dimensional data. Dimensionality reduction techniques aim to simplify the dataset by mapping it to a lower-dimensional space, preserving the essential features and relationships within the data. Clustering algorithms, on the other hand, aim to group similar data points together based on predefined similarity measures.

To evaluate the performance and effectiveness of these algorithms, the COCO (Common Objects in Context) and LFW (Labeled Faces in the Wild) datasets will be used as they provide clear instance labeling, wich allows for straightforward assessment of the algorithms' output. In the case of DBSCAN, it is clear that ideally the identified clusters overlap the known labels and for this, cluster purity can be evaluated. The evaluation of the t-SNE will be done empirically by visual observation. This holds as the application of the algorithm in the project is also strictly visual.

4.2.1. t-SNE

t-SNE analysis is relevant for the project in terms of allowing the intuitive visualization of the indexed data. Since the goal of using the algorithm strictly for display purposes, a general tolerance for the performance is considered. The t-SNE algorithm was run with the following parameters:

```
TSNE(n_components=3, n_iter=1500, perplexity=10, metric='cosine')
```

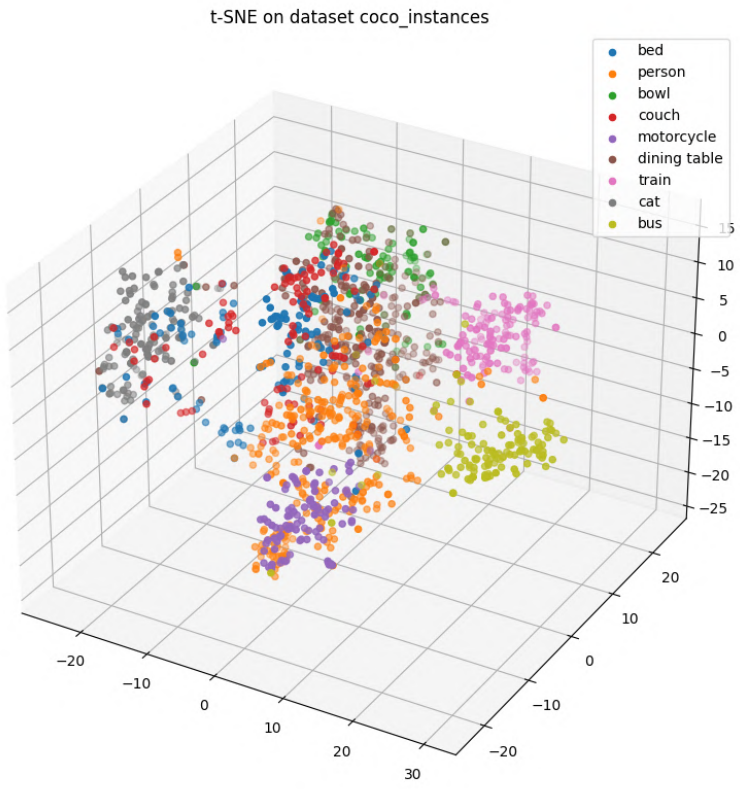


Figure 4.5.: t-SNE projections on samples of COCO instances (1426 samples)

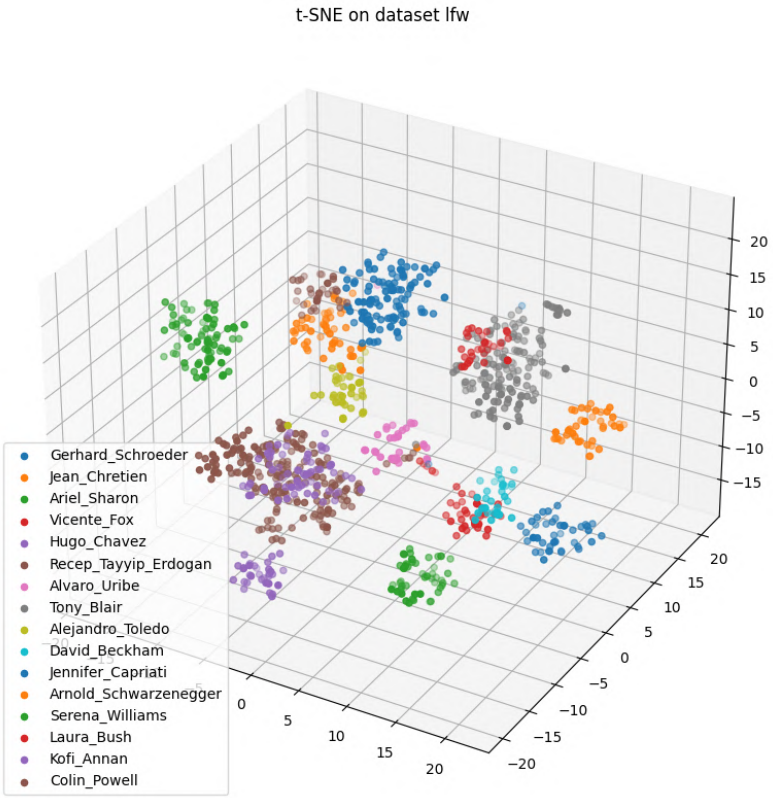


Figure 4.6.: t-SNE projections on samples of LFW instances (1029 samples)

The experiments concluded with the visualization of embeddings in a three-dimensional space, thorugh by plots provided in the Figures 4.5 and 4.6. The original ground-truth labels are displayed with different colors, making it easy to interpret the data with respect to known categories.

It's important to first that achieving an accurate low-dimensional representation of the embeddings is not straightforward. The embedding space likely exhibits a high degree of linear independence between dimensions, making any reduction a challenging task, as compression cannot be applied without the loss of information.

In the case of t-SNE, a comparison between dataset results reveals distinct behavior between the COCO and LFW datasets. COCO samples exhibited greater variability and noise, making them more challenging to project accurately into a lower-dimensional space, as shown in Figure 4.5. Groups of images heavily intersect and overlap and are not defined by clear edges, with some few exceptions. It is also hard to determine whether the spatiality of the embeddings reflects contextual understanding, since semantic parallels can be drawm between the classes.

On the other hand, the LFW dataset presented clearer patterns in the relationships between the samples positions in the lower-dimensional space. It is interesting to individually compare which classes appear geometrically linked in this projection, given that the samples represent images of famous persons. Overall, the plot suggests that when variability is primarily contained within features of interest, like the case of relatively syandardised portraits,t-SNE is more effective at preserving these relationships.

4.2.2. DBSCAN

DBSCAN serves as a central part of the system by identifying and recommending possible clusters or groups of interest within the indexed image data. DBSCAN is an unsupervised learning method that groups data points that are closely packed together based on a distance metric and a density threshold.

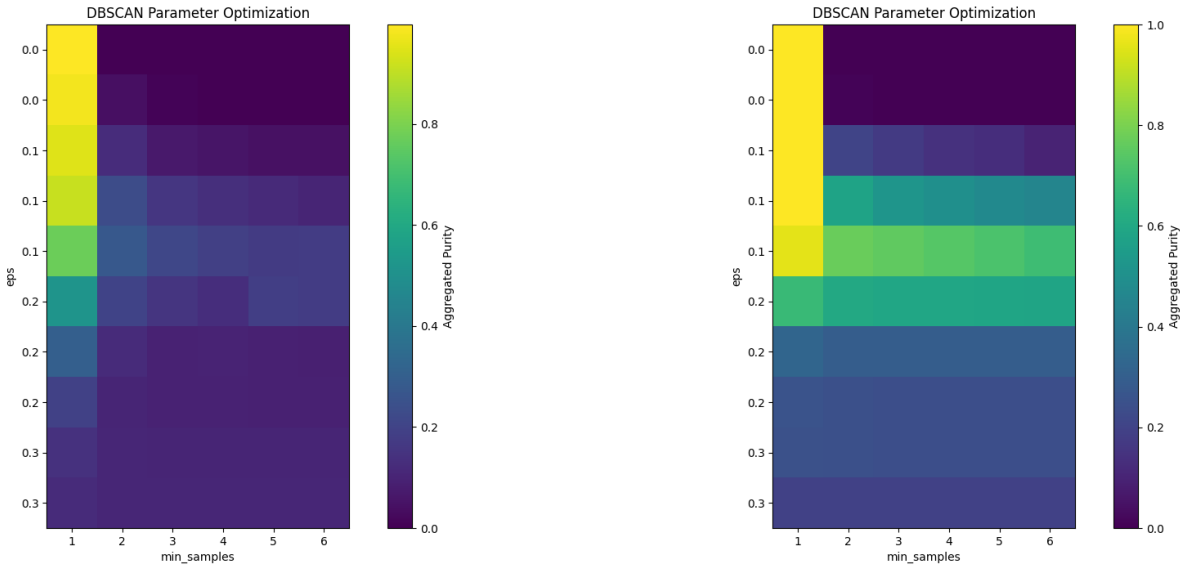
Cluster purity is a key metric for evaluating the performance of DBSCAN in these experiments. Considering that the dataset is labeled with ground-truth categories, a well performing clustering algorithm should idnetify clusters with high overlap over the original labels. Purity is measured by comparing the number of correctly clustered data points with the overall size of the cluster. Essentially, the higher the ratio of data points that share the same label within a cluster, the greater the purity.

Parameter tuning is neccessary for the optimal performance of DBSCAN. In this context, the focus is on finding the best 'epsilon' (distance threshold) and 'min_samples' (minimum number of samples required to form a dense region). These hyperparameters are tuned by running the algorithm iteratively on the dataset with varying combinations of 'epsilon' and 'min_samples' values, then evaluating performance metrics to identify the optimal settings.

The results of this experiment is shown in Figure 4.7, on both COCO and LFW dataset.

Optimal values for DBSCAN parameters

Apart from the empirical results of the hyper-parameter tuning of DBSCAN, ana analytical approach can also be used in order to determine whether the two have conciding conclusions. For instance, the 'min_samples' parameter is stringly dependent on the data distribution. It is a measure for the data points densit as it defines how many "reachable" points are needed for a reference to be considered a core point. However if the dataset itself is under-represented,



(a) Performance on COCO dataset

(b) Performance on LFW dataset

Figure 4.7.: DBSCAN performance for different parameter values of 'eps' and 'min_samples'

this value will negatively impact the results even the considered value corresponds to external known rules in the shape and distribution of the data.

The other parameter, 'epsilon', defining the distance between points that can be considered to represent similarity may be more straight-forward to choose. Given the precedent experiments, a value for the cosine similarity denoting that two sample are in fact equivalent can be estimated around $similarity = 0.8$. Given that DBSCAN operates with distances, not with similarity metrics, this value has to be converted to the corresponding inverse, the *cosine distance*, equal to $distance = 1 - 0.8 = 0.2$. Even though the threshold for similarity in this context is not directly identical to that of the 'eps' parameter, it seems like the two are alligned.

In the benchmark evaluation of different parameter values, some combinations give raise to accuracy scores of 100% which indicate a collapse of the metric into an overfitting scenario. Since the objective is not to apply DBSCAN to this particular set of data, but to a generic distribution, a tradeoff is made towards the value of 0.2. Considering this, 'min_samples' can be chosen equal to 2.

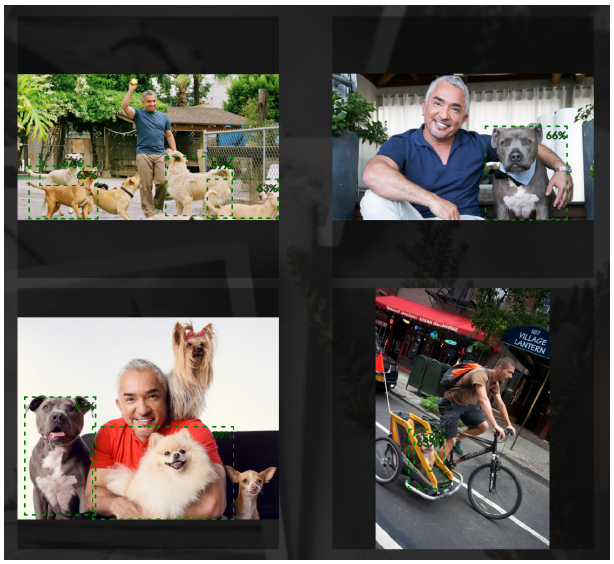
4.3. Search Engine experiments

One of the main final functionalties of the project revolves around the search feature. As, explained in 3.6, the image processing pipeline analyzes each image indexed in the system by employing an Object Detection and an OCR module, respectively. Each image that is added in the system is run through this process of identifying potentially relevant parts. If found, along the original image, these parts are also indexed, meaning that they are processed by the embedding model which generates a corresponding numerical representation.

The numerical representation of a part is different from that of the original image as the images themselves can be drastically different (even if the part image is contained in the original image). From the point of view of the embedding model, the information in the part is much more focused, as it targets a single subject that is tightly contained in the boundaries of the crop. For

(a) Query image

(b) Results when prioritising 'person'-type detections



(c) Results when prioritising 'animals'-type detections

Figure 4.8.: Different search results depending on label priority. Note that the actual sample that matched the query is shown in the green dotted rectangle

a given image that contains multiple subjects, the process of augmenting the indexed database with each individual subject might allow for better recognition performance. Multiple subjects in the same image might each be individually interpreted as noise by the embedder model, in relation to the others.

A practical experiment to showcase this functionality can be run on the search engine module. With a sufficently large set of images, so that there is plenty of background information to

compare against, the search problem can be formulated as follows: given an input query (image or text), identify the indexed samples (either full images or crops) that maximize the cosine similarity between the sample and the query embeddings.

In this context, an aditional feature can be introduced. Given that each sample is either an original image or was created as a result of the Detection or OCR algorithms, it is therefore labeled with the type of instance that these systems recognized. The Object Detection module, for example, uses the baseline COCO labels, combined in several super-classes, like for example, ”person”, ”animal”, ”vehicle”, etc. This allows for the search engine to be augmented with weights that effectively prioritize between types of samples.

Such a result is shown in Figure 4.8. Note that in the figure, the matching samples are highlighted with a dotted green rectangle. Considering the fact that the res of image data in the system also contains a multitude of individual parts, this experiment showcases a successfully end-to-end search functionality that allows for very specific insights into the indexed data.

4.4. Fine-tuning the CLIP model

The experiment to fine-tune the CLIP model is intended to showcase the possibility of adapting the model to user-specific data. Moreover, it is important to demonstrate the possibility for additional training of the model without damaging the original performance on the benchmarked datasets. In some situations, shifting the input distribution can be a practical use-case, but in general, a fine-tuned model should retain as much of the original capabilities as possible, while learning new specific information.

The scenario for face recognition is considered as this would be a principle feature for an image organization tool designed for general public. The specific use-case would be that the user of such a system may want to filter through the collections of images for those containing themselves or others. In the original model, this can only be accomplished using an image query with the portrait of the target person. This in itself is a practical solution, however it does not leverage the potential for text processing of the model. Thinking in terms of which caption query would match the specific target person, the task becomes complex, as a general description of the features, without the usage of a specific name, can match any number of similar-looking individuals.

Therefore, the goal for a fine-tuning recipe for the base model which would teach the model arbitrary captions is useful. Extending the concept, the same technique can be applied to infuse knowledge about particular places or other image features into the system, thus augmenting the search feature considerably. For the present section, the scope is limited to the task of novel faces recognition.

4.4.1. Choice of training data

The task of identifying a practical dataset for this experiment is not trivial, however, since all the open source captioning datasets were already used during training. Since the interest in person identification task, the LFW (Labeled Faces in the Wild) dataset is naturally of particular interest. A subpart of this dataset can be used, selecting a number of individuals with 30 or more photos each.

However, from an initial experiment, evaluation of the base model on this dataset indicates that the model already has a strong recognition capability for the selected individuals. The datasets consists of images of public figures, so selecting for the most represented classes in this dataset

is clearly biasing the resulting subset of data towards very famous personalities. This in turns extends the challenge as such personalities are probably represented in other datasets, as well, apart from only LFW.

Simulating novel data

Several solutions were considered for arriving at a novel collection of faces data. Given that several samples are needed (initially estimated as 20 images for training and 10 for evaluation, per individual), manually comprising the dataset was not an option. However, a solution arises when considering the problem with the LFW dataset: it is not the actual images that are the issue, but the learnt relationships to the specific captions containing the names.

A solution to this issue is to precisely work against this knowledge. This can be obtained by systematically swapping the samples between the entities. For example the images depicting Arnold Schwarzenegger might now be captioned with "an image of George Bush". As long as this swaps remain consistent, meaning that they are applied to all the samples of the particular entities, this method essentially underlines that names, even in the case of celebrities, are more or less arbitrary combinations of letters.

With this shuffling, the previously learned associations of faces and names becomes virtually useless, so the mdoel has to undergo training in order to learn the new mapping. Even though better performance might be shown in the case of naturally novel data, as the model would not have to un-learn any prevous knowledge) this holds as a practical solution for the goal of the experiment: making the model learn arbitrary faces.

Increase robustness of training with a background dataset

As mentioned above, the main challenge in a fine-tuning task comes with the risk of losing valuable patterns learned in the original extensive dataset. More so, with aggressive hyper-parameters a model might actually collapse in a weight distribution that reduces drastically reduces performance even on the dataset used for fine-tuning.

An idea that would increase the robustness of training is to merge the fine-tuning dataset with samples from a *background dataset* - the original training dataset, so as relatively counter-balance the effects of new information. This way the learning of new information can be stabilized.

In the present experiment, the background dataset will be represented by a subset of the COCO captioning dataset. Random images were selected, along with their corresponding text descriptions, and used in an 1 to 1 ratio with the LFW data used for fine-tuning. This makes sure that the model does not overfit the scope of face recognition and does not forget the general patterns.

In order to evaluate the performance of the fine-tuned model, similarity between corresponding images and captions are measured before and after the training on both the target and the background dataset.

4.4.2. Loss computation

The loss function used for the fine-tuning is a symmetric loss function as described in the original CLIP paper [16]:

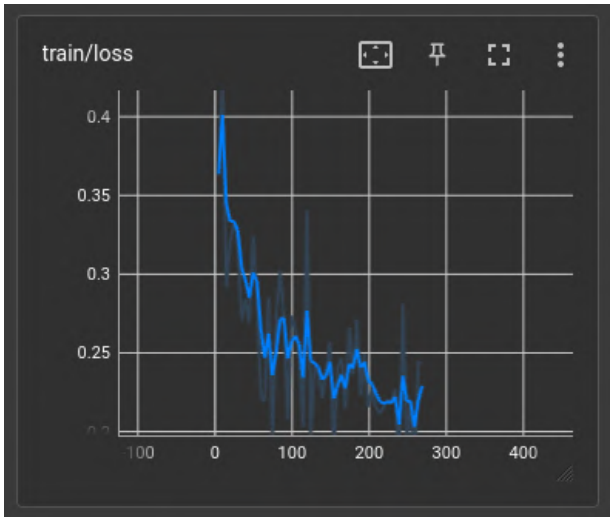


Figure 4.9.: Value of loss function on training data (LFW + COCO)

```
# extract feature representations of each modality
I_f = image_encoder(I) #[n, d_i]
T_f = text_encoder(T) #[n, d_t]
# joint multimodal embedding [n, d_e]
I_e = l2_normalize(np.dot(I_f, W_i), axis=1)
T_e = l2_normalize(np.dot(T_f, W_t), axis=1)
# scaled pairwise cosine similarities [n, n]
logits = np.dot(I_e, T_e.T) * np.exp(t)
# symmetric loss function
labels = np.arange(n)
loss_i = cross_entropy_loss(logits, labels, axis=0)
loss_t = cross_entropy_loss(logits, labels, axis=1)
loss = (loss_i + loss_t)/2
```

This functions similar to a classification task, optimizing the pairing between inputs and labels. In this examples, the labels are neither the text nor the image embeddings, but pairing of the two. In this formualtion the *arrange* function is meant to assign to each individual sample an index so that corresponding images and embeddings are represented as such by being assigned the same index.

4.4.3. Training metrics

The overall loss evolution throughout the training is consistent with the expectations as it decreases steadily, which is shown in Figure 4.9. There is some noise but the overall progress indicates a successful training experiment. Important to consider is the fact that this loss value comprises of both the loss associated with the target dataset (expected to be higher initially as the new information is not yet learnt) and the loss associated with the background dataset.

In order to obtain more detailed look on the training process, a dedicated metric evaluates the performance of the model on the background dataset. Ideally the model maintains a constant evaluation on this subset of data, as this would indicate that the model, while learning from the data in the target dataset does not overfit it and lose performance on its general task. This plot, shown in Figure 4.10, initially follows an upward trend, indicating that the model was starting to perform worse on the background dataset at the start of the training but then begins declining which means that the model actually performs better on the COCO subset, as well. This can be explained by the fact that the COCO dataset was just one of a multitude of training datasets in the original CLIP model pipeline, so this experiment also tilts the distribution of the

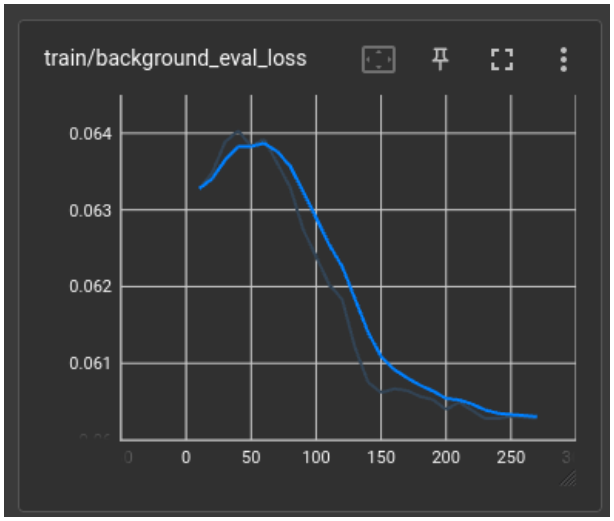


Figure 4.10.: Value of loss function on the background data (COCO)

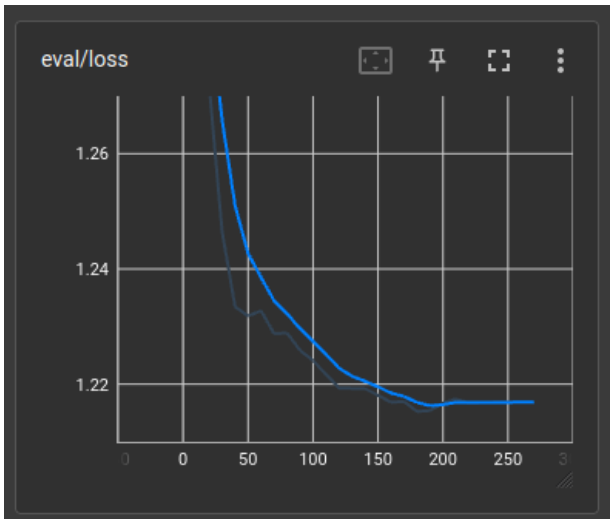


Figure 4.11.: Value of loss function on the evaluation data (LFW)

general features of the model to a degree. However, by inspecting the actual values, it can be concluded that the changes are negligible.

Finally, the most relevant metric for the task is the performance of the model on the target dataset. For all the metrics, the samples used for evaluation do not coincide with the ones used for training so the performance indicated by this evaluation shows the generalization capabilities of the model. The evolution of the loss metric applied to the evaluation data is shown in Figure 4.11. From this, it can be concluded that the model successfully learned the features of interest.

4.4.4. Model evaluation

A subsequent evaluation showcases the exact gains in performance obtained through the fine-tuning process. The base model is first evaluated on both the target and the background datasets. After running the training, the resulting model is evaluated on the same data as the base model. Results are shown in the Table 4.2.

These results show a 13% increase in similarity score on the target dataset. Further training (for more epochs, or on more data) may result in better results but the scope of this experiment was

Model	Target Dataset	Benchmark Dataset
Base model	0.194	0.312
Trained model	0.221	0.341

Table 4.2.: Performance Comparison

to showcase a general recipe for the training on custom data and not optimize any particular performance metric.

5. System Design and Implementation

The system is built primarily in Python. The system is segmented into specific modules that handle various aspects of the pipeline - ranging from data manipulation, embedding generation, indexing for search and clustering and finally user interface. The system operates via a backend server that is decoupled from a browser-based user interface. This setup enables streamlined interactions for tasks such as image uploads or retrievals and the presentation of the algorithms' results. The following section describes each component in detail.

5.1. Tools and technologies

In order to achieve the outlined scope, the codebase implements a number of frameworks and libraries.

5.1.1. PyTorch and Transformers

For handling the Machine Learning models, training and executing predictions, the PyTorch framework and the Transformers library were employed. PyTorch is a highly flexible and dynamic deep learning framework that allows for efficient model building and optimization. Native support for GPU acceleration is particularly useful in optimizing the processing time. The Transformers library provides a rich collection of useful methods and objects and seamlessly integrates with PyTorch, enabling easy importation of models, fine-tuning on the dataset, and making predictions.

5.1.2. EasyOCR and Object Detections (TorchVision)

Image processing is a significant aspect of the project, and for this, EasyOCR for optical character recognition and object detection from the TorchVision library were used. EasyOCR is a deep learning-based Optical Character Recognition (OCR) library built on PyTorch and excels in extracting text from images in various languages. TorchVision is another PyTorch-based package that contains pre-trained models including a suite of object detection. These two libraries allowed for efficient preprocessing of images for subsequent analysis steps.

5.1.3. Scikit-Learn

Scikit-Learn was employed for its vast array of machine learning algorithms and data preprocessing tools. Among the algorithms, Density-Based Spatial Clustering of Applications with Noise (DBSCAN) was selected for its ability to identify clusters of arbitrary shapes within the high-dimensional space of image and text embeddings, as detailed in the previous section. t-Distributed Stochastic Neighbor Embedding (t-SNE) was another powerful tool utilized from the Scikit-Learn library.

5.1.4. Pandas

For data manipulation tasks, the Pandas library was chosen. Fast, flexible, and expressive data structures offered by Pandas are designed to make data analysis in Python both easy and intuitive. Loading, transforming or aggregating large datasets could be effectively manipulated using this library.

5.1.5. FastAPI, Uvicorn, Requests

The backend server was built using FastAPI, a modern web framework built upon Starlette for the web parts and Pydantic for the data parts. FastAPI is known for its high performance and intuitive interface for building APIs. Uvicorn served as an ASGI server to deploy the FastAPI application. On the client-side, the Requests library was used to handle HTTP requests to and from the server, facilitating easy communication between the client and server.

5.1.6. Dash

For the frontend, Dash was chosen, a Python framework for building analytical web applications. Dash is ideal for the use-case as it is designed to integrate seamlessly with data-centric Python libraries like Pandas and Plotly. Highly interactive and visually appealing user interfaces could be created without requiring advanced knowledge in JavaScript or HTML.

5.2. System Architecture

The codebase follows a structure of inter-operating modules that handle the various data flows in the system. In order to outline a general overview of the system, Figure 5.1 is presented, with each module being detailed in the following section.

The architecture is principally designed around the single-responsibility principle, with distinct handler modules taking charge of specific tasks such as database operations, generation of embeddings, and computational analysis. This design ensures that each module serves as a resource for multiple others, streamlining information flow and improving resource utilization. For example, resource-intensive elements like the Embedder and Object Detection models are instantiated only once to optimize resource management.

The architecture also employs a component-based design for object interaction. For instance, if Object A needs to interact with Object B, Object A is instantiated with Object B as a component. The majority of inter-object communication takes place through direct method invocation, which simplifies system complexity and enhances ease of maintenance.

A central orchestrating daemon manages the initialization and coordination of system components. It ensures that interdependent modules are correctly initialized based on their hierarchical relationships.

For persistent storage and data integrity across multiple runs, the system saves processed elements locally as files, as is the case for database snapshots, original image files, and computed embeddings. While this requires periodic validations to confirm the local existence of database items, it offers greater scalability and robustness compared to in-memory caching. Moreover, intermediate computational results for the resource-heavy processes are also stored. This feature

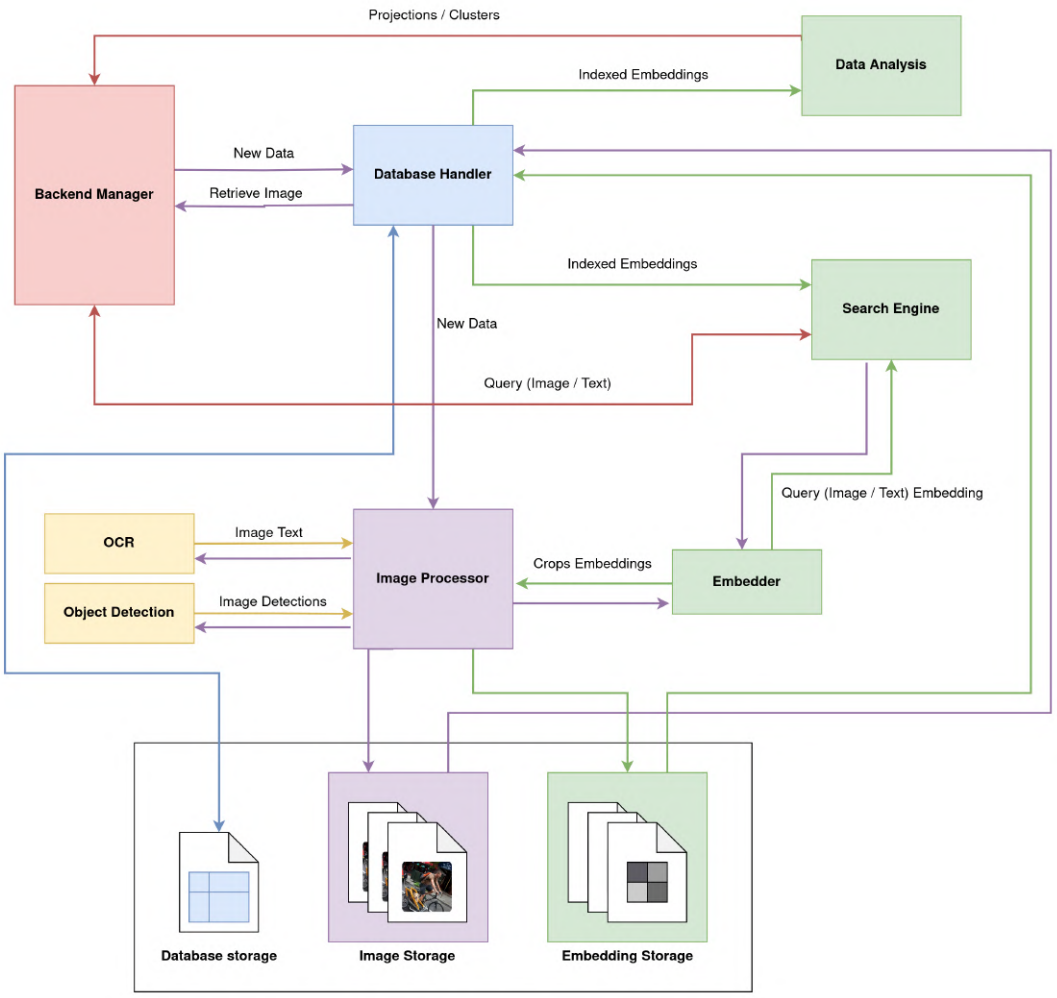


Figure 5.1.: Architecture of the end-to-end system

Figure 5.2.: Relationships between images and embeddings

enhances the system's resilience, allowing processes to resume from their last successful state in case of a system restart or failure.

Configuration settings for the various modules are consolidated within a central 'config' module, which reads from a local dictionary file to determine the operational parameters of the system. This centralized approach governs both external and internal naming conventions, ranging from API endpoint names to database column identifiers, thus simplifying code maintenance and enhancing readability. Directory paths for storing images, embeddings and other artifacts are also specified through this configuration method. Additionally, the config module contains parameters that directly impact the system's output and operational procedures, such as parameters for the clustering and t-SNE algorithms and thresholds for image processing and result filtration.

In this architecture, the process of fine-tuning the embedder model is considered as a separate pipeline, as this process is not considered to be a recurring one. Switching between different embedder models can be achieved by editing the configuration file.

5.3. Components

The system's main components are detailed in this section, highlighting individual features, challenges and solutions with regards to the module's input / output interface.

5.3.1. Database Handler

The Database Handler module is tasked with overseeing database read and write operations, which includes queries and data insertions. When initialized, the module first checks for the existence of a previous snapshot and loads it if available. It then manages new incoming data by calling with the Image Processor module which then handles the processing and embedding generation.

The module utilizes two tables for its operations. The first is the main dataset table, which consists of only two columns: 'image_id' and 'image_path.' This table is used to locate the original image file, as subsequent queries rely on the 'image_id.' The mapping of the entries is one-to-one, however there is a one-to-many relationship between the images and the embeddings as for each image a number of crops are created. The second table is the embeddings table, featuring columns for 'embedding_id,' 'embedding path' coordinates relative to the original image, the sample type, meta-information, and confidence level.

The relationships between entities are illustrated in Figure 5.2.

Image and embedding IDs are generated using a hash function that takes the string encoding of the image as input, ensuring the uniqueness of each sample. In the case of the embeddings the image is the cropped image defined by the coordinates.

The Database Handler module offers various methods that facilitate the retrieval of specific images based on their IDs, as well as the samples corresponding to a particular image. These include the crops and their corresponding embeddings. Other practical methods required by the various system modules were added progressively to the module, whenever querying patterns became apparent.

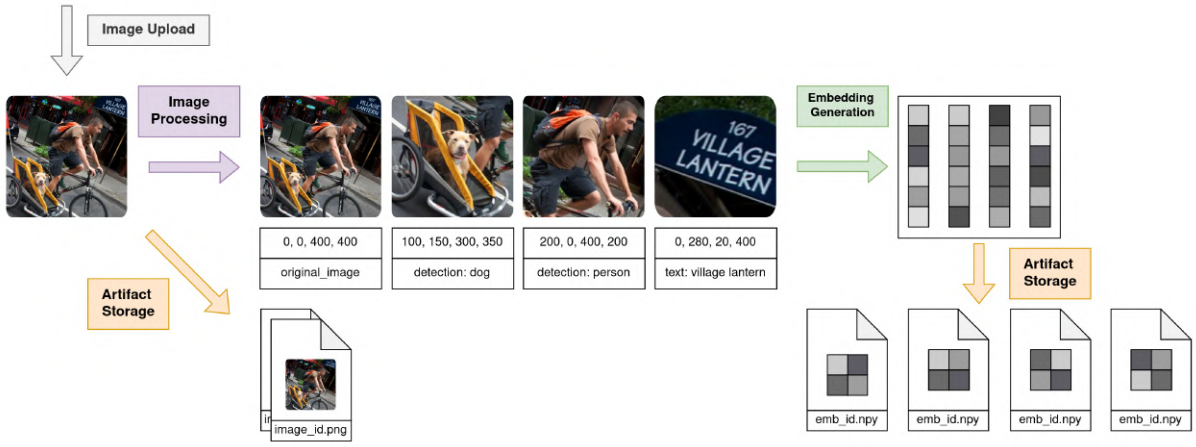


Figure 5.3.: The Image Processor module flow

5.3.2. Object Detector

The Object Detector module takes an image as input and produces a list of identified objects within that image. It also applies all the requisite image pre-processing. The module is capable of identifying various labels, which are essentially super-categories adapted from the COCO dataset labels: person, vehicle, outdoors, animals, sports and objects. It leverages a FasterRCNN ResNet50 model to accomplish these detection tasks. Additionally, the Object Detector provides a confidence metric for each detected object.

5.3.3. OCR

The OCR module scans the input image for any text content. Once detected, it attempts to group adjacent text boxes for more coherent text extraction. The module then provides the spatial coordinates of these text detections, along with their corresponding transcriptions.

5.3.4. Image Processor

The Image Processor module manages the sequence of operations necessary for adding an image to the system, as presented in Figure 5.3 It uses the Object Detection and OCR modules to pinpoint regions of interest within the image and then calls the Embedder module both for the entire image and for each individual segment that has been identified by Object Detection and OCR. The module then saves the image, utilizing the 'image_id' that was previously generated by the Database Handler. Additionally, it stores each resulting embedding, linking it with the 'image_id' and the coordinates that specify the cropped region. Ultimately, the Image Processor creates a small Embeddings table containing this new data associated with the input image, which is then incorporated into the existing dataset by the Database Handler.

5.3.5. Embedder

The Embedder module serves as a functional wrapper around the CLIP model and is tasked with generating both text and image embeddings. It manages the necessary pre-processing for image inputs and handles the tokenization of text inputs. The module outputs a 512-dimensional embedding array corresponding to the given input. For enhanced portability, the Embedder can be initialized with a variety of pretrained models, whether they are sourced from the original

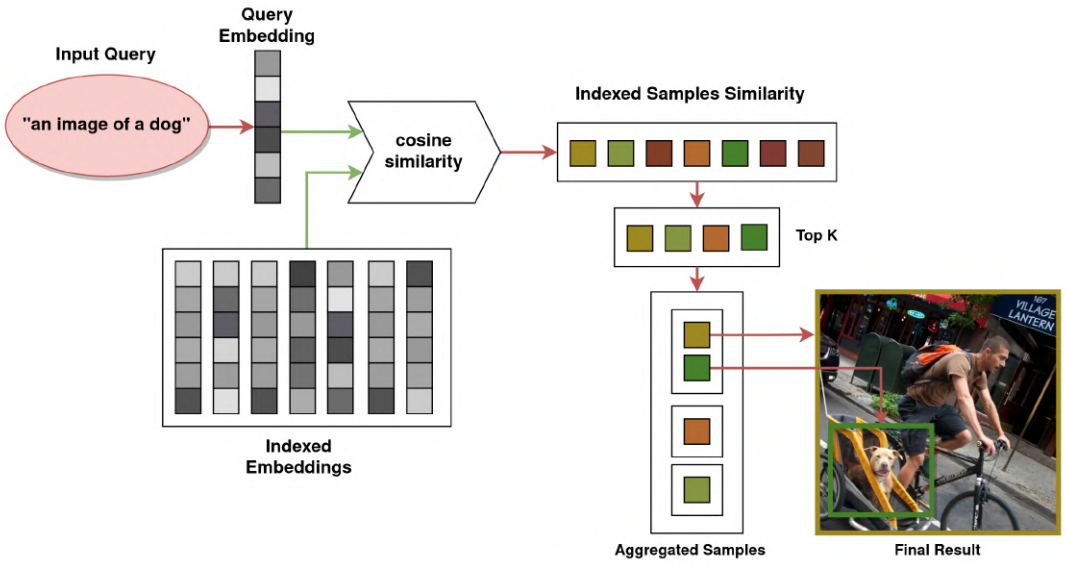


Figure 5.4.: The Search Engine module flow

CLIP repository or loaded from a local file containing a fine-tuned model. The module is also capable of detecting the availability of a GPU and will load the model accordingly.

5.3.6. Search Engine

The Search Engine module executes embedding-based queries within the system's data. It accepts either text or image inputs and calls the Embedder module to generate the corresponding embedding. Following this, it calls the Database Handler to fetch all existing indexed sample embeddings. It computes the similarity between the query and the indexed samples using cosine distance. Thai distance metric was recommended in the CLIP paper [16] as being more effective than euclidean distance.

After this step, a list of all similarities between the query and the indexed samples is obtained. In order to allow dynamic filtration of the results, to these similarities a set of weights is applied. These weights are defined specific to the system types: entire image, person, vehicle, animals, text, etc. This has the effect of tilting the results of the search towards not only the most similar samples but also towards a specific type of samples.

The top-k results are then selected, with k being a parameter of the module. The list of results however consist of image parts crops, therefore the next step is to aggregate them based on the source image they were featured from. The final output is a list of image IDs and for each image ID the list of corresponding samples, along with their coordinates and scores, is also provided.

To accelerate the computation speed, the Search Engine is designed to operate on a GPU if one is available.

5.3.7. Data Analyzer

The Data Analyzer module runs the clustering and t-SNE algorithms on the samples that are indexed in the system. It retrieves the data necessary for running the algorithms by calling the Data Handler, thus ensuring the latest samples are obtained. To reduce unnecessary computation in future runs, the module saves the results locally. It is important to note that adding new data to the system can significantly alter the outcomes of the data analysis, particularly with

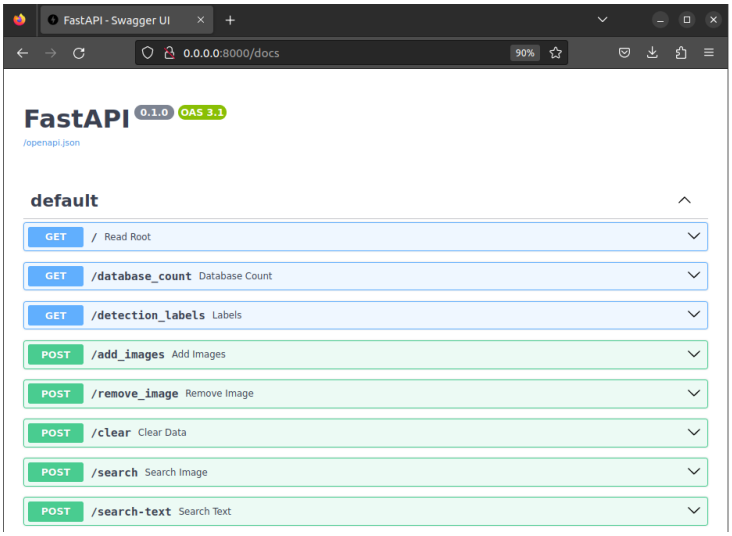


Figure 5.5.: Screenshot of the backend API user interface

clustering algorithms. To address this, the module regularly verifies if there have been updates to the system database. It does this by invoking a method from the Database Handler, which returns the current list of sample IDs. This list is then compared to the data used in the last analysis run. If discrepancies are found, indicating that the previous analysis is now outdated, the algorithms are executed again using the most recent data.

5.3.8. Backend Manager

The Backend Handler serves as the coordinating module that guarantees the proper initialization and operation of all other modules. It adds an extra layer of abstraction by consolidating methods that involve multiple sub-modules into single, streamlined functions. This simplification makes it easier to utilize these functions in the upstream module, that of the API.

5.3.9. API

The API module consists of a FastAPI server that provides all the endpoints needed for frontend interactions. It communicates with the system through an instance of the Backend Manager. This API supports image uploads both as individual files or archives and retrieval of one or multiple images. Endpoints are available to access Data Analyzer results, either in the form of Projections, represented as transformed 3D coordinates, or Clusters, presented as grouped images within a particular or arbitrary cluster. The search functionality is also accessible via an endpoint, accepting either text or image queries as required by the frontend. Several other endpoints deliver real-time information about the system, like the current count of indexed samples. For added convenience, the API is browser-accessible, making it easy to test without requiring a frontend, as shown in Figure 5.5. To enhance security, API key authentication can be implemented, with keys passed through endpoint headers.

5.3.10. Frontend

The Frontend is a user-friendly application designed to facilitate interactions with the system's features. Developed using Dash in Python, it communicates with the API through HTTP requests, deserializing the received payload for further use. The app comprises multiple pages,

each serving distinct functionalities. It features a home page for image uploads, a gallery that displays processed images stored in the database, and menus for text or image search. Additionally, the 'Clusters' page showcases images from a randomly chosen cluster and 'Projections' allows for 3D navigation through the data points. The app aims to provide an intuitive platform for users to explore the data.

Screenshots from the Frontend pages are provided in A

6. Conclusions and Discussion

The primary goal of this project was to create a comprehensive system that could bring structure to an unorganized collection of images by transforming them into semantic representations. This goal required an in-depth exploration through a wide variety of facets of machine learning, data analysis, and software engineering. The multidisciplinary lays in how the research avenues had to be tackled in a close relation to the directions of development with a constant consideration towards implementability and practicality.

In order to rigorously evaluate the success of the initial objectives, let us revisit the list with which this endeavor was started:

- a robust method for extracting information from a collection of unalbeled images into representative semnatic embeddings
- data analysis methids that would uncover relevant and non-trivial patterns of similarity in the set of data embeddings
- a functional implementation of the features into and end-to-end system

With these aspects considered several relevant points are discussed:

6.1. The CLIP model

In this course of the project the CLIP model emerged as the strongest candidate for the goal of in-depth image processing. Going beyond its applicability to the specifics of the project, diving deep into the architecture and inner workings of the CLIP model has both fueled and satisfied curiosity. It provided a valuable opportunity to explore the intricacies of how Transformers, and specifically Vision Transformers, achieve their remarkable performance. This understanding is instrumental not just for this project but for broader applications and future research in machine learning and computer vision.

It's worth noting that understanding machine learning seems to have two distinct phases: before debugging and after debugging. The former phase can be full of theoretical constructs and assumptions, but the latter phase is where real understanding is solidified. The challenges faced and the subsequent (necessary) problem-solving performed during this project have undoubtedly moved me from the former phase to the latter, enriching my comprehension considerably.

Finally, the fine-tuning experiment proved to be an exciting endeavor that confirmed many of the previous assumptions. It also provided an effective method for understanding the subtle nuances of the challanges, such as those associated with crafting an appropriate training dataset. Successfully overcoming these challenges brings a unique sense of satisfaction.

6.2. Uncovering latent patterns

There's something inherently thrilling about bringing order to an otherwise unstructured collection of information. As data is processed through clustering algorithms and projections, unseen patterns begin to emerge, almost as if revealing secrets that were always there but never noticed. This sense of discovery is simply exciting. In this context, the abstract concepts of mathematical methods come into practice to quantifiable, almost physical results.

Successfully harnessing the power of these data analysis techniques, however, required a careful understanding of the methodology. The two methods explored in the project, DBSCAN for clustering and t-SNE for projections, proved effective in meeting the project's objectives as demonstrated in the experiments. While t-SNE has shown some limitations against the high-dimensionality of the embedding space, DBSCAN proved distinctively fit for this challenge.

6.3. Developing the platform codebase

The development of the application served as a valuable context for decisions needed to be taken thought the research process by offering a clear, designated use case for the project's outcomes. At the same time, developing a codebase for an integral system rather than individual experiments is a task that demands organisation and clear overview over the project.

My background predominantly lies in the integration of machine learning solutions, which sometimes requires training and experiments and other times designing architectures of interconnected modules, orchestrated towards a shared goal through extensive codebases. Not at all times is this latter case free of discouragement. However, it was my intention from the start to leverage the benefits of both approaches for my Master Thesis.

One of those challenges was the development of a frontend interface, a domain I had little to no familiarity with prior to this project. Designing the user interface from the ground up was a learning curve, but it was also an enriching experience. Venturing into this previously unexplored territory allowed me to gain a comprehensive view of system design.

6.4. Personal motivation for the project

Lastly, for the final chapter in the present work, I'd like to touch on some personal reflections that were not appropriate for the other sections of this thesis. While the motivation stated in the introduction holds entirely true to itself, it is rather the generic formulation of this, more personal motivation for the project.

Order seems to be a fundamental and existential motivation in the endeavors of Man. What makes us pursuit such idealistic yet seemingly achievable goals cannot be easily explained. It seems that we are not only interested in the physical kind of order, for which there would be some reasonable practical motivations, but also in the abstract kind order, that which manifests in the realm of the abstract.

With this conviction in mind, we are faced with a paradox: the surrounding world does not amount to order, but to chaos. It amounts to entropy, both in the physical realm and in the abstract world of information. With the first, humanity has some experience. But with the second, I argue that we are just discovering how sloppy we have been. Information is no longer a sparsity, and value does not stand in its mere existence, but in its order.

It is not my intention to extend this section into endless philosophical wanderings. Several months ago, I was faced with a very practical definition of all these considerations. The motivation came naturally after some unfortunate manipulation of a hard-drive, that so happened to contain the entire collection of my childhood photos (needless to say, there was no backup). Fortunately, I got in touch with some professionals. I sighed in relief as they informed me that they will probably be able to recover the data - however, they then mentioned, the recovered files will be dumped without any meta-information or the original folder structure.

This lead to me to realize how intrinsically separate the order we claim for our data actually is from the data itself. This is rather counter-intuitive as we ourselves may owe our deep understanding of the world precisely to the ever-present context for each piece of information. Maybe by semantic analysis, unconstrained of conventional labels and tags, our immense volumes of data could be turned into organic knowledge.

This seems to hold in practice as well. Our current state of the art models learned not by optimizing against fixed labels but by iterating through immense corpora of context. Humans, too, learn little by direct instruction, as they do by simple, yet incredibly insightful, observation of the world. Maybe this is the direction forward. And in that case, I hope that this Thesis represents a however small step towards this ambitious goal.

Appendices

A. Application

This section documents the features of the final User Interface of the system, elaborating on the main functionality of the project showcased in the following pages:

- Home - landing page for the application, allows for image upload and displays the number of currently indeed images.
- Search Image / Text - allows for search in the image database using an image or a text query
- Clusters - visualizes the images in the clusters identified by the DBSCAN algorithm
- Projections - visualizes the t-SNE projections of the embeddings corresponding to the indexed images

A.1. Home

The Home page is where the user lands when opening the application. Data can be uploaded either as separate files or as an archive. The application processes the images synchronously, and checking if they already exist in the system to avoid duplicates and unnecessary processing times. A counter indicates the current number of indexed images.

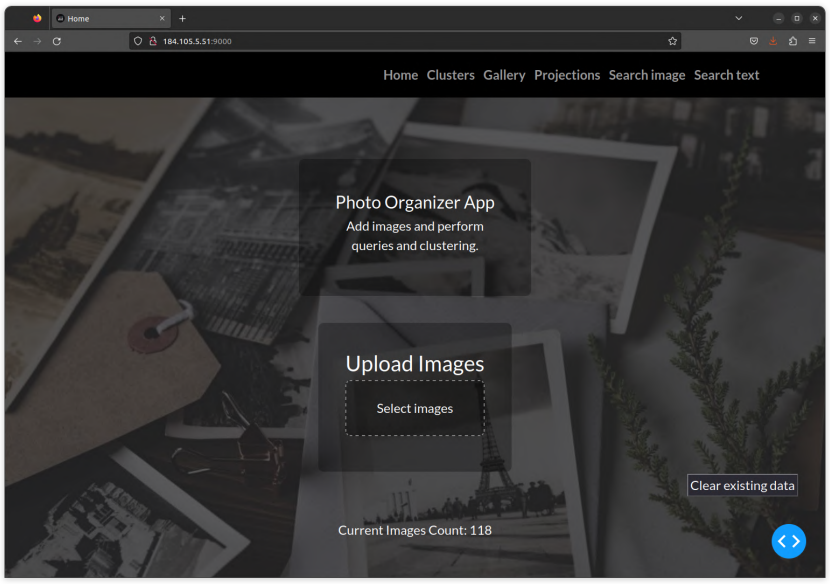


Figure A.1.: The Home page of the application

A.2. Search Pages

Two pages in the platform are dedicated to the search operation by either image or text queries, respectively. After the embedding stage, the main search engine pipeline is the same regardless of the input type.

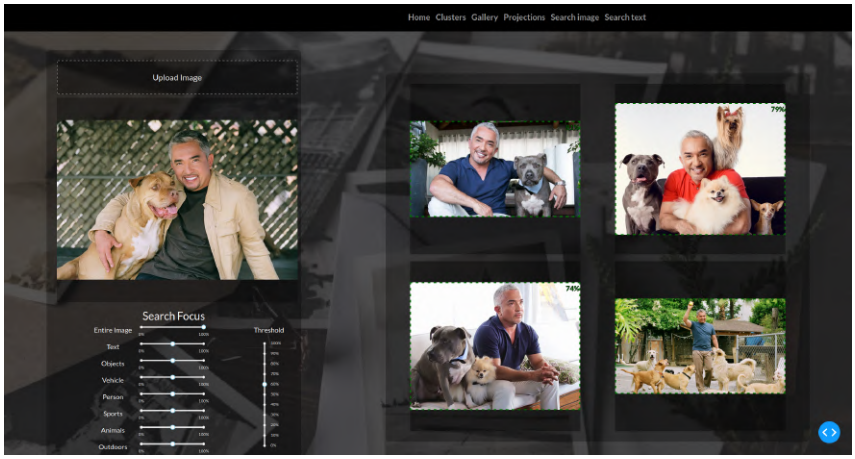


Figure A.2.: The Search page of the application - image query

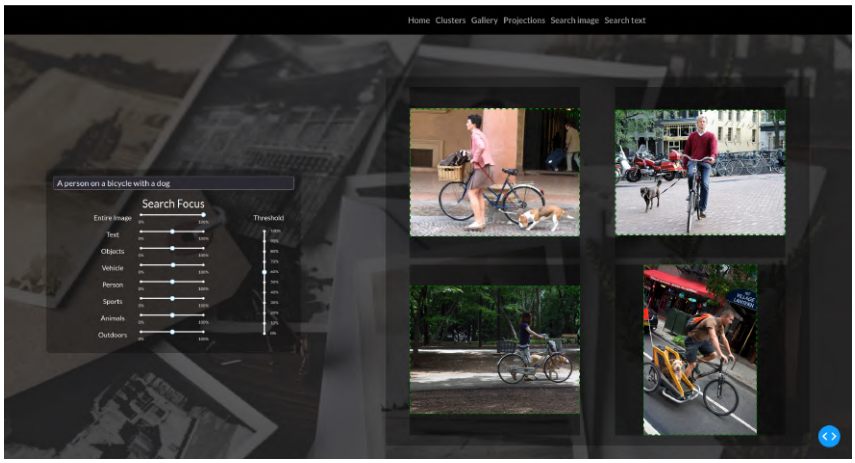
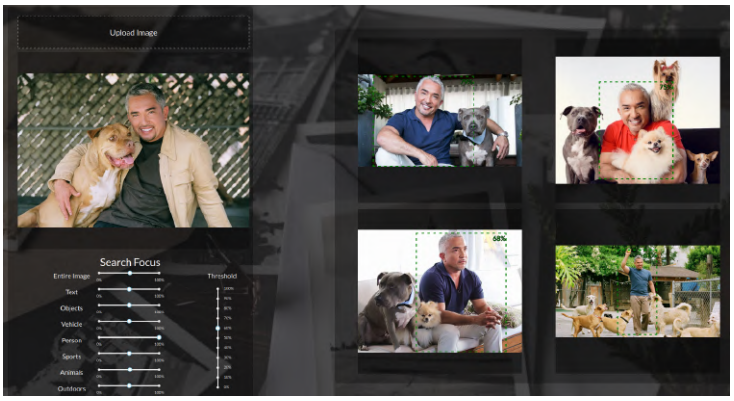


Figure A.3.: The Search page of the application - text query

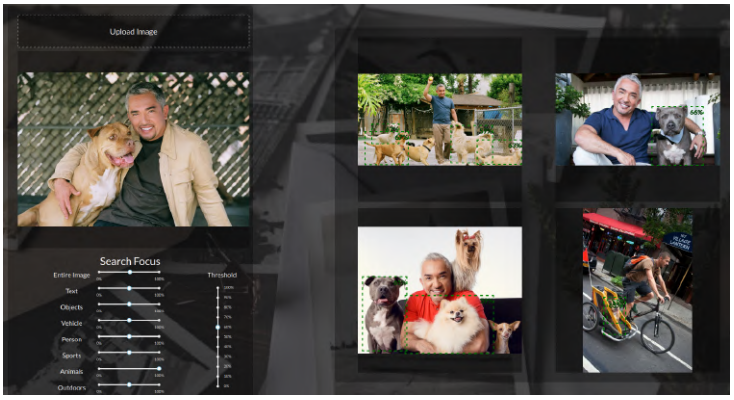
A key point to be noted is the configuration of the Search with the parameters displayed in the left panel:

- The threshold filters only the results with a similarity score higher than the set value, which allows for customizing the false positive - false negative trade-off.
- The label weights prioritize the search results based on the type of the sample. The samples consist either of the original images (as uploaded in the system) or of crops of the originals image as resulted from the Image Processing stage.

To showcase the effects of these parameters, Figure A.4 is provided. In the example presented, it is relevant to note that the results of the search algorithm consist of the highlighted areas and not of the entire images, which are only displayed as a reference. As explained in 3.6, the pipeline for generating the image embeddings uses Object Detection and OCR to decompose the image into sub-parts that are each individually encoded, each of the resulting samples being used in the search algorithm. The search feature of the platform is one of the core functionalities as it allows for an intuitive retrieval method for the indexed data in the system.



(a) Example of search results when ”person” type crops are prioritized



(b) Example of search results when ”animal” type crops are prioritized

(c) Example of search results when ”bicycle” type crops are prioritized

Figure A.4.: Difference in search results based on changes in the parameters. Note that the search match is highlighted with the green dotted rectangle. In these cases the entire image is displayed only as reference

A.3. Clusters Page

Another objective of the system is the possibility for semantic-based exploration of the indexed data. This can be achieved through the Clusters page which displays images grouped together based on the results of the DBSCAN algorithm. Since the algorithm is run on the encoded representation of the images, the clustering criteria is represented by the features of the images and their semantic content. The resulting clustering criterion is not directly explicit, which

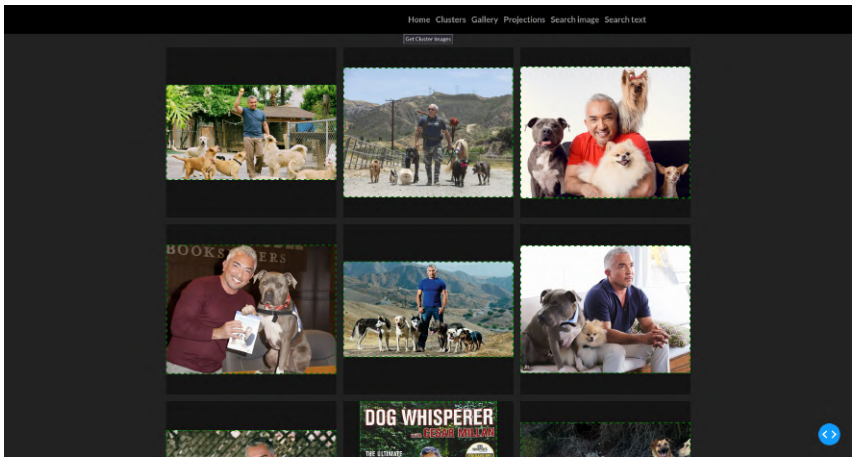


Figure A.5.: The Clusters page of the application, displaying images which belong to a cluster identified by the DBSCAN algorithm

also brings a level of randomness in the system, since the the similarities between samples can drastically change with different sets of data. However, since the embedding model was trained on a captioning task, the most intuitive way the criteria for grouping can be hough of is 'how would someone describe this image'. If two images would have similar descriptions, then the clustering algorithm can be expected to place them in the same category.

A.4. Projections Page

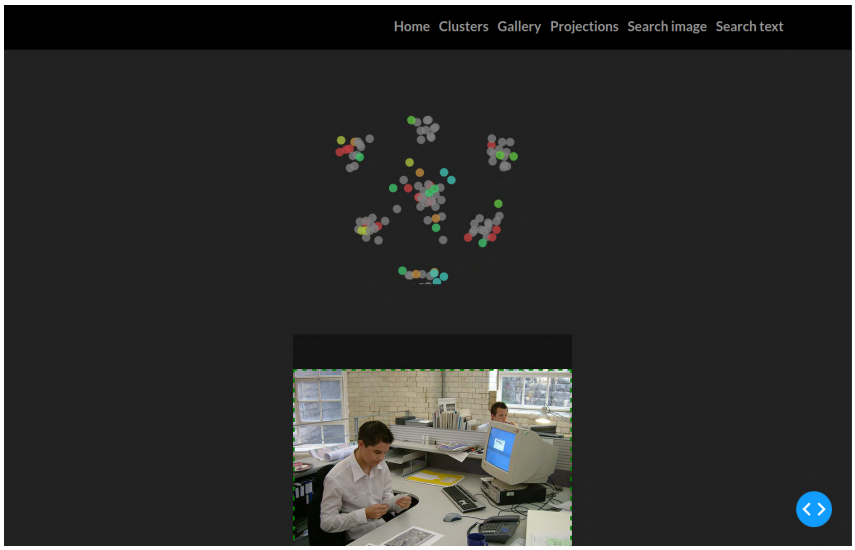


Figure A.6.: Visualizing the t-SNE (t-distributed Stochastic Neighbor Embedding) projections of the image embeddings

Given that he indexed images are converted to n-dimensional arrays (512 in the case of the current implementation), different dimensionaliy reduction algorithms can be applied. Constraining the data to 3 dimensions opens up the possibility for a plot containing the data points in their respective relative positions. This plot is displayed in the Projections page in an interactive window. To be able to interact with the samples, hovering over a specific point shows the image it represents.

Although the t-SNE algorithm is a powerful tool for producing lower-dimensional representations of data, in the present case, the algorithm does not achieve very high accuracy in representing the variability of the data. To summarize, the geometrical distances displayed in the plot represent the relationships between the embeddings only to a certain degree, as the embedding space dimensions are mostly liniarly independent from one oanother, thus little compression can be applied without losing valuable dynamics. However, for the display purposes of this page, the algotihm produces decent results.

B. Source Code

The code base is fully available at: https://github.com/tdrvlad/photo-organizer. This section showcases relevant parts of the codebase with the source code stripped down of some of the methods required for compilation in order to allow for readability.

B.1. Training and Evaluation

This is the pipeline script defining the arguments, loading the dataset and training the CLIP model.

```
# Imports

class InstancesDataset:
    image_id_col = 'image_id'
    image_path_col = 'image_path'
    instance_col = 'instance'

    def __init__(self, data_dir, max_num_samples=None, min_num_samples=None, max_num_instances=None):

        self.data_dir = data_dir
        instances = get_instances_from_dir(self.data_dir)

        df_data = []
        for instance in instances:
            image_paths = get_instance_images(os.path.join(self.data_dir, instance))
            if min_num_samples is not None and len(image_paths) < min_num_samples:
                continue
            if max_num_samples is not None:
                image_paths = image_paths[:max_num_samples]

            image_ids = image_paths_to_ids(image_paths)
            for image_id, image_path in zip(image_ids, image_paths):
                df_data.append(
                    {
                        self.image_id_col: image_id,
                        self.image_path_col: image_path,
                        self.instance_col: instance
                    }
                )

        self.df = pd.DataFrame(df_data)
        instances = list(set(self.df[self.instance_col]))
        if max_num_instances is not None and max_num_instances < len(instances):
            instances = random.sample(instances, max_num_instances)
            self.df = self.df[self.df[self.instance_col].isin(instances)]

        logging.info(f'Extracted {len(self.df)} samples from {len(instances)} instances: {", ".join(instances)}.')

    def __len__(self):
        return len(self.df)

```

```
    def get_dataset(self):
        ds = datasets.Dataset.from_pandas(self.df)
        return ds
```

Listing B.1: Model training script

B.2. Backend

B.2.1. Database Handler

This is the class that handles the main data flows in the system, interfacing the image database.

```
# Imports


class DatabaseHandler:
    _bbox_col = "bbox"

    def __init__(self, image_processor: ImageProcessor = None):

        self.image_processor = image_processor

        if os.path.exists(self.dataset_df_file):
            self.dataset_df, self.embeddings_df = self.load_dataframes()
            logging.info(f"Loaded database with {len(self.dataset_df)} samples.")
            self.check_df()
        else:
            logging.info("Initialized New database.")
            self.dataset_df = DatasetDataframe()
            self.embeddings_df = EmbeddingsDataframe()

    def load_dataframes(self):
        # ...

    def save(self):
        # ...

    def __len__(self):
        return len(self.dataset_df)

    def get_sample_from_image_id(self, image_id):
        # ...
        return result_row

    def get_sample_from_embedding_id(self, embedding_id, return_as_df=False):
        # ...
        return result_row

    def get_embedding_bbox(self, embedding_id):
        embedding_df = self.get_sample_from_embedding_id(embedding_id, return_as_df=
    True)
        embedding_df = self.merge_embeddings_df_bbox_columns(embedding_df)
        image_id = embedding_df.image_id[0]
        bbox = embedding_df[self._bbox_col][0]
        return image_id, bbox

    def get_embeddings_bboxes(self, embedding_ids):
        image_ids_and_bboxes = [self.get_embedding_bbox(embedding_id) for
    embedding_id in embedding_ids]
        image_ids, bboxes = map(list, zip(*image_ids_and_bboxes))
        return image_ids, bboxes

```

```
    def get_image_paths_from_ids(self, image_id_list):
        # ...
        return image_paths

    def get_image_ids_from_embedding_ids(self, embedding_id_list):
# ...
        return image_ids

    def check_df(self):
    # ...

    def clear(self):
        # ...

    def get_image_embeddings(self, image_id) -> EmbeddingsDataframe:
        # ...
        return image_embeddings_df

    def remove_image(self, image_id):
        # ...

    def get_image_embeddings_bboxes_scores_types(self, image_id):
        # ...
        return embedding_ids, bboxes, scores, types

    def get_sample(self):
        # ...
        return image_path, embedding_paths, embedding_bboxes, embedding_crops_type

    def add_images(self, dir_path, generate_embeddings=True):
        image_paths = retrieve_image_files(dir_path)
        logging.info(f"Adding {len(image_paths)} images found in {dir_path}.")

        df = DatasetDataframe(image_paths=image_paths)
        df.image_id = df.image_path.apply(get_image_id_from_path)

        # Filter images where the id could not be generated
        initial_len = len(df)
        df = df[~df.image_id.isnull()]
        if len(df) != initial_len:
            logging.info(
                f"Dropped {initial_len - len(df)} images because they could not be
    loaded."
            )

        # Filter duplicates
        initial_len = len(df)
        df = df.drop_duplicates(subset=df.image_id.name, keep="first")
        if len(df) != initial_len:
            logging.info(
                f"Dropped {initial_len - len(df)} images because they were
    duplicates."
            )

        # Filter images that already exist in the database
        # Id is obtained as hash of the image, so checkup is done by the actual
    image.
        initial_len = len(df)
        df = df[~df.image_id.isin(self.dataset_df.image_id)]
        if len(df) != initial_len:
            logging.info(
                f"Dropped {initial_len - len(df)} images because they already exist
    in database."
            )
        if len(df) == 0:
            return
```

```
111
112         if generate_embeddings:
113             embeddings_df = generate_df_embeddings(
114                 df=df, embedding_fn=self.image_processor.generate_image_embeddings
115             )
116             self.add_df_to_embeddings_df(embeddings_df)
117
118         self.add_df_to_dataset_df(df)
119         logging.info(
120             f"Added {len(df)} samples to the Database. Current Database size: {len(
    self.dataset_df)}."
121         )
122
123     def load_embeddings(self, device='cpu', return_tensor=True, target_labels=None):
124         embeddings_list = []
125         failed_indexes = []
126
127         embeddings_df = self.embeddings_df
128         if target_labels is not None:
129             embeddings_df = embeddings_df[self.embeddings_df.crop_type.isin(
    target_labels)]
130
131         for index, row in embeddings_df.iterrows():
132             embedding_path = row[self.embeddings_df.embedding_path.name]
133             try:
134                 embedding = np.load(embedding_path)
135                 embeddings_list.append(embedding)
136             except Exception as e:
137                 failed_indexes.append(index)
138
139         if len(failed_indexes) > 0:
140             logging.warning(f"Failed to load {len(failed_indexes)} embedding nodes."
    )
141         embeddings_df = embeddings_df.drop(failed_indexes)
142         embeddings_df = self.merge_embeddings_df_bbox_columns(embeddings_df)
143         assert len(embeddings_list) == len(embeddings_df), 'Length of embedding list
     differs from length of embeddings Dataframe.'
144
145         embeddings = np.array(embeddings_list)
146         if return_tensor:
147             embeddings = torch.Tensor(embeddings).to(device)
148
149         return embeddings_df, embeddings
150
151     def merge_embeddings_df_bbox_columns(self, df: EmbeddingsDataframe) ->
    EmbeddingsDataframe:
152         # ...
153
154
155 def test_database_handler():
156     from photo_organizer.backend.embeddings.embedder import ImageProcessor
157
158     embedder = ImageProcessor()
159     database_handler = DatabaseHandler(image_processor=embedder)
160     database_handler.add_images("local_data/source_photos", generate_embeddings=True
    )
161
162
163 if __name__ == "__main__":
164     test_database_handler()
```

Listing B.2: Database Handler module

B.2.2. Image Processor

This is the class responsible for indexing an image added to the system by applying OCR and Object Detection to decompose the image into relevant components and then generating the corresponding embedding.

```
# Imports


class ImageProcessor:
    full_image_label = "Entire Image"
    text_label = 'Text'

    def __init__(self, embeddings_dir=config.paths.embeddings_dir, max_image_size=
    config.backend.max_image_size, embedder_model_id=config.backend.
    embedder_model_id, lowercase=True):
        self.device = "cuda" if torch.cuda.is_available() else "cpu"

        self.clip_embedder = ClipEmbedder(
            embedder_model_id=embedder_model_id,
            lowercase=lowercase
        )

        self.object_detector = ObjectDetector()
        self.ocr = OCR()
        # ...


    def encode_image_path(self, image_path):
        # ...


    def encode_image(self, image: PILImage):
        embedding = self.clip_embedder.encode_image(image)
        return embedding

    def resize_image(self, image: PILImage):
        # ...


    def encode_text(self, text: str):
        embedding = self.clip_embedder.encode_text(text)
        return embedding

    def generate_image_embeddings(
        self, image_path, image_id, save_selection_image=True, ratio_min=config.
    backend.crop_min_ratio, ratio_max=config.backend.crop_max_ratio
    ):
        try:
            embeddings_df = self._generate_image_embeddings(
                image_path=image_path,
                image_id=image_id,
                save_selection_image=save_selection_image,
                ratio_min=ratio_min,
                ratio_max=ratio_max
            )
            return embeddings_df
        except Exception as e:
            logging.warning(
                f"Error generating embedding for image {image_path}: {str(e)}"
            )
            raise e
            return None

    def _generate_image_embeddings(
        self,
```

```
        image_path,
        image_id,
        save_selection_image=True,
        ratio_min=0.25,
        ratio_max=0.9,
    ):
        image_embeddings_dir = get_image_embeddings_dir(
            image_id=image_id, embeddings_dir=self.embeddings_dir
        )

        image = read_image(image_path)
        image = resize_image(image, max_image_size=self.max_image_size)
        w, h = image.size
        original_area = w * h

        # Selections will be type: List[Tuple[left, top, right, bottom, label,
    confidence, meta]
        selections = [(0, 0, w, h, self.full_image_label, 1.0, None)]
        selections += self.object_detector(image)
        selections += self.ocr(image, label=self.text_label)

        # The embedder will process all individual detections (objects of interest)
    plus the entire original image.
        lefts, tops, rights, bottoms, labels, scores, metas = zip(*selections)

        lefts = [left / w for left in lefts]
        rights = [right / w for right in rights]
        tops = [top / h for top in tops]
        bottoms = [bottom / h for bottom in bottoms]

        embeddings_df = EmbeddingsDataframe(
            image_ids=[image_id] * len(labels),
            lefts=lefts,
            tops=tops,
            rights=rights,
            bottoms=bottoms,
            types=labels,
            metas=metas,
            confidences=scores
        )
        embeddings_df.embedding_id = embeddings_df.apply(get_embedding_id, axis=1)
        embeddings_df.embedding_path = embeddings_df.apply(
            get_embedding_path, image_embeddings_dir=image_embeddings_dir, axis=1
        )
        embeddings_df.crop_ratio = embeddings_df.apply(
            get_area_ratio_from_row, original_area=1, axis=1
        )

        # Filter small / large crops. Keep the 'Entire Image' crop.
        embeddings_df = embeddings_df[embeddings_df.crop_ratio > ratio_min]
        embeddings_df = embeddings_df[
            (embeddings_df.crop_ratio < ratio_max)
            | (embeddings_df.crop_type == self.full_image_label)
        ]

        for index, row in embeddings_df.iterrows():
            if os.path.exists(row.embedding_path):
                continue

            selection_image = crop_image(
                image=image,
                left=row.crop_left,
                top=row.crop_top,
                right=row.crop_right,
                bottom=row.crop_bottom,
                coords_relative=True
```

```
            )
            if save_selection_image:
                selection_image.save(f"{row.embedding_path}.png")

            if row.crop_type == self.text_label:
                embedding = self.encode_text(row.crop_meta)
            else:
                embedding = self.encode_image(selection_image)
            np.save(row.embedding_path, embedding)

        embeddings_df.to_json(get_embedding_report_path(image_embeddings_dir))
        return embeddings_df
```

Listing B.3: Image Processor module

B.2.3. Embedder

This is the a wrapper class for the embedding generation model, allowing for interchangeability between models, either from the CLIp repository or from a local training experiment.

```
# Imports

class ClipEmbedder:
    def __init__(self, embedder_model_id=config.backend.embedder_model_id, lowercase
    =False):
        self.lowercase = lowercase

        self.model = CLIPModel.from_pretrained(embedder_model_id)
        self.processor = CLIPProcessor.from_pretrained(embedder_model_id)

    def preprocess_image(self, image: PILImage, return_tensors="pt"):
        image = image.convert("RGB")
        processed_image = self.processor(images=image, return_tensors=return_tensors
    )['pixel_values'][0]
        return processed_image

    def encode_image(self, image: PILImage):
        preprocessed_image = self.preprocess_image(image)
        preprocessed_image = preprocessed_image.unsqueeze(0)
        with torch.no_grad():
            image_features = self.model.get_image_features(preprocessed_image)
            embedding = image_features.float()[0]
        embedding = embedding.detach().cpu().numpy()
        return embedding

    def preprocess_text(self, text: str, return_tensors="pt"):
        if self.lowercase:
            text = text.lower()
        processed_text = self.processor(text=text, return_tensors=return_tensors)['
    input_ids'][0]
        return processed_text

    def encode_text(self, text: str):
        preprocessed_text = self.preprocess_text(text)
        preprocessed_text = preprocessed_text.unsqueeze(0)
        with torch.no_grad():
            text_features = self.model.get_text_features(preprocessed_text)
            embedding = text_features.float()[0]
        embedding = embedding.detach().cpu().numpy()
        return embedding
```

Listing B.4: Embedder module

B.2.4. Search Engine

This is the class that handles all operations related to computing similarity, ordering the results and selecting the top matches for a text or image search query.

```
# Imports

def get_embeddings_tensor(embedding_paths, device="cpu"):
    embeddings = np.array(
        [np.load(embedding_path) for embedding_path in embedding_paths]
    )
    embeddings_tensor = torch.Tensor(embeddings).to(device)
    return embeddings_tensor


class SearchEngine:
    _similarity_col = "similarity"
    _weighted_similarity_col = "weighted_similarity"
    _score_col = "score"

    def __init__(self, database_handler: DatabaseHandler, embedder: ImageProcessor,
    label_weights: LabelWeights, text_label:str, image_label:str):

        self.device = torch.device("cuda") if torch.cuda.is_available() else 'cpu'

        self.database_handler = database_handler
        self.embedder = embedder
        self.label_weights = label_weights
        self.text_label = text_label
        self.image_label = image_label

        self.embeddings_df, self.embeddings_tensor = None, None
        self.reload()

    def reload(self):
  # ...

    def apply_weight(self, df_row, query_label, label_weights_dict,
    include_confidence_weight=False):
        label = df_row[self.embeddings_df.crop_type.name]
        label_weight = label_weights_dict[label] * self.label_weights(label,
    query_label)

        weighted_similarity = df_row[self._similarity_col] * label_weight
        if include_confidence_weight:
            weighted_similarity *= df_row[self.embeddings_df.crop_confidence.name]

        return weighted_similarity

    def group_embeddings_df_by_image_id(
        self, df: EmbeddingsDataframe
    ) -> EmbeddingsDataframe:
        image_id_groups = df.groupby(self.embeddings_df.image_id.name)
        columns_to_aggregate = [
            col for col in df.columns if col != self.embeddings_df.image_id.name
        ]
        aggregated_df = image_id_groups[columns_to_aggregate].agg(list).reset_index
    ()
        return aggregated_df

    def aggregate_results_scores(
        self, result_df, aggregate: Literal["avg", "max"] = "max"
    ):
        if aggregate == "avg":
            result_df[self._score_col] = result_df[self._weighted_similarity_col].
    apply(
```

```
                np.mean
            )
        elif aggregate == "max":
            result_df[self._score_col] = result_df[self._weighted_similarity_col].apply(
                np.max
            )
        else:
            raise ValueError(f"Unknown aggregation {aggregate}.")
        return result_df

    def search_image(self, image: PILImage, label_weights_dict, top_k=4, threshold=0.0):
        image_embedding = self.embedder.encode_image(image)

        image_ids, scores, bboxes = self._search_embedding(
            embedding=image_embedding,
            label_weights_dict=label_weights_dict,
            top_k=top_k,
            threshold=threshold,
            query_label=self.image_label
        )

        return image_ids, scores, bboxes

    def search_text(self, text: str, label_weights_dict, top_k=4, threshold=0.0):
        text_embedding = self.embedder.encode_text(text)

        image_ids, scores, bboxes = self._search_embedding(
            embedding=text_embedding,
            label_weights_dict=label_weights_dict,
            top_k=top_k,
            threshold=threshold,
            query_label=self.text_label
        )

        return image_ids, scores, bboxes

    def _search_embedding(
        self,
        embedding: np.ndarray,
        label_weights_dict: dict,
        query_label: str,
        threshold: float = 0.0,
        top_k: int = 4,
    ) -> (List[str], List[float], List[List[float]]):
        embedding_tensor = torch.tensor(embedding).unsqueeze(0).to(self.device)

        if len(self.embeddings_tensor) == 0:
            logging.warning('cannot perform search without any indexed embeddings.')
            return [], [], []

        # Compute cosine similarities
        similarities_tensor = F.cosine_similarity(
            embedding_tensor, self.embeddings_tensor
        )
        similarities = similarities_tensor.detach().cpu().numpy()

        # Apply weights for the similarity depending on the type of embedding (Entire-image / crop, ect)
        results_df = self.embeddings_df.copy()
        results_df[self._similarity_col] = similarities
        results_df[self._weighted_similarity_col] = results_df.apply(
            self.apply_weight, query_label=query_label, label_weights_dict=label_weights_dict, axis=1
        )
```

```
119
120         # Sort the results and select first top_k * 2.
121         # Some result may refer to the same image, so after this they are merged and
     then top_k are selected.
122         results_df = results_df.sort_values(
123             by=self._weighted_similarity_col, ascending=False
124         ).head(top_k * 3)
125         results_df = self.group_embeddings_df_by_image_id(results_df)
126         results_df = self.aggregate_results_scores(results_df, aggregate="max")
127         results_df = results_df.sort_values(by=self._score_col, ascending=False).
    head(
128             top_k
129         )
130         results_df = results_df[results_df[self._score_col] > threshold]
131
132         image_ids = list(results_df[self.embeddings_df.image_id.name])
133         scores = list(results_df[self._weighted_similarity_col])
134         bboxes = list(results_df[self.database_handler._bbox_col])
135
136         return image_ids, scores, bboxes
```

Listing B.5: Search Engine module

Bibliography

[1] Yutong Bai **andothers**. *Are Transformers More Robust Than CNNs?* 2021. DOI: 10.48550/ARXIV.2111.05464. URL: https://arxiv.org/abs/2111.05464.

[2] Dor Bank, Noam Koenigstein **and** Raja Giryes. *Autoencoders*. 2020. DOI: 10.48550/ARXIV.2003.05991. URL: https://arxiv.org/abs/2003.05991.

[3] Alexey Dosovitskiy **andothers**. *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. 2020. DOI: 10.48550/ARXIV.2010.11929. URL: https://arxiv.org/abs/2010.11929.

[4] Karl Pearson F.R.S. "LIII. On lines and planes of closest fit to systems of points in space". **in***The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*: 2.11 (1901), **pages** 559–572. DOI: 10.1080/14786440109462720.

[5] Michael Hahsler, Matthew Piekenbrock **and** Derek Doran. "bdbscan/b: Fast Density-Based Clustering with iR/i". **in***Journal of Statistical Software*: 91.1 (2019). DOI: 10.18637/jss.v091.i01. URL: https://doi.org/10.18637/jss.v091.i01.

[6] Kaiming He **andothers**. *Deep Residual Learning for Image Recognition*. 2015. DOI: 10.48550/ARXIV.1512.03385. URL: https://arxiv.org/abs/1512.03385.

[7] Prannay Khosla **andothers**. *Supervised Contrastive Learning*. 2020. DOI: 10.48550/ARXIV.2004.11362. URL: https://arxiv.org/abs/2004.11362.

[8] Alexander Kolesnikov **andothers**. *Big Transfer (BiT): General Visual Representation Learning*. 2019. DOI: 10.48550/ARXIV.1912.11370. URL: https://arxiv.org/abs/1912.11370.

[9] Oliver Kramer. *Dimensionality Reduction with Unsupervised Nearest Neighbors*. Springer Publishing Company, Incorporated, 2013. ISBN: 3642386512.

[10] Alex Krizhevsky, Ilya Sutskever **and** Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". **in***Advances in Neural Information Processing Systems*: **byeditor**F. Pereira **andothers**. **volume** 25. Curran Associates, Inc., 2012.

[11] Tsung-Yi Lin **andothers**. *Microsoft COCO: Common Objects in Context*. 2014. DOI: 10.48550/ARXIV.1405.0312. URL: https://arxiv.org/abs/1405.0312.

[12] Laurens van der Maaten **and** Geoffrey Hinton. "Visualizing Data using t-SNE". **in***Journal of Machine Learning Research*: 9 (2008), **pages** 2579–2605. URL: http://www.jmlr.org/papers/v9/vandermaaten08a.html.

[13] Tomas Mikolov **andothers**. *Efficient Estimation of Word Representations in Vector Space*. 2013. DOI: 10.48550/ARXIV.1301.3781. URL: https://arxiv.org/abs/1301.3781.

[14] Francesco Pinto, Philip H. S. Torr **and** Puneet K. Dokania. "An Impartial Take to the CNN vs Transformer Robustness Contest". **in**(2022): DOI: 10.48550/ARXIV.2207.11347. URL: https://arxiv.org/abs/2207.11347.

[15] Alec Radford **and** Karthik Narasimhan. "Improving Language Understanding by Generative Pre-Training". **in**2018: URL: https://api.semanticscholar.org/CorpusID:49313245.

[16] Alec Radford **andothers**. *Learning Transferable Visual Models From Natural Language Supervision*. 2021. DOI: 10.48550/ARXIV.2103.00020. URL: https://arxiv.org/abs/2103.00020.

[17] Shaoqing Ren **andothers**. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. 2015. DOI: 10.48550/ARXIV.1506.01497. URL: https://arxiv.org/abs/1506.01497.

[18] Olga Russakovsky **andothers**. *ImageNet Large Scale Visual Recognition Challenge*. 2014. DOI: 10.48550/ARXIV.1409.0575. URL: https://arxiv.org/abs/1409.0575.

[19] Florian Schroff, Dmitry Kalenichenko **and** James Philbin. "FaceNet: A Unified Embedding for Face Recognition and Clustering". **in**(2015): DOI: 10.48550/ARXIV.1503.03832. URL: https://arxiv.org/abs/1503.03832.

[20] Rico Sennrich, Barry Haddow **and** Alexandra Birch. *Neural Machine Translation of Rare Words with Subword Units*. 2015. DOI: 10.48550/ARXIV.1508.07909. URL: https://arxiv.org/abs/1508.07909.

[21] Neha Sharma, Vibhor Jain **and** Anju Mishra. "An Analysis Of Convolutional Neural Networks For Image Classification". **in***Procedia Computer Science*: 132 (2018), **pages** 377–384. DOI: 10.1016/j.procs.2018.05.198. URL: https://doi.org/10.1016/j.procs.2018.05.198.

[22] Yi Tay **andothers**. *Scale Efficiently: Insights from Pre-training and Fine-tuning Transformers*. 2021. DOI: 10.48550/ARXIV.2109.10686. URL: https://arxiv.org/abs/2109.10686.

[23] Ashish Vaswani **andothers**. *Attention Is All You Need*. 2017. DOI: 10.48550/ARXIV.1706.03762. URL: https://arxiv.org/abs/1706.03762.

[24] Feng Wang **and** Huaping Liu. *Understanding the Behaviour of Contrastive Loss*. 2020. DOI: 10.48550/ARXIV.2012.09740. URL: https://arxiv.org/abs/2012.09740.

[25] Yonghui Wu **andothers**. *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation*. 2016. DOI: 10.48550/ARXIV.1609.08144. URL: https://arxiv.org/abs/1609.08144.

[26] Wenchao Yu **andothers**. "Embedding with Autoencoder Regularization". **in***Advanced Information Systems Engineering*: Springer Berlin Heidelberg, 2013, **pages** 208–223. DOI: 10.1007/978-3-642-40994-3_14. URL: https://doi.org/10.1007/978-3-642-40994-3_14.

[27] Ce Zhou **andothers**. *A Comprehensive Survey on Pretrained Foundation Models: A History from BERT to ChatGPT*. 2023. DOI: 10.48550/ARXIV.2302.09419. URL: https://arxiv.org/abs/2302.09419.