Universität
Münster

Advancing Large-Scale Data Retrieval: A Co-Design Approach of Machine Learning and Indexing

Inauguraldissertation

zur Erlangung des akademischen Grades
eines Doktors der Wirtschaftswissenschaften
durch die Wirtschaftswissenschaftliche Fakultät der
Universität Münster

vorgelegt von

Christian Lülf

Münster, den 23.05.2024

Dekan	Prof. Dr. Thomas Langer
Erstgutachter	Prof. Dr. Fabian Gieseke
	Universität Münster
	University of Copenhagen
Zweitgutachter	Prof. Dr. ir. Arjen P. de Vries
	Radboud University Nijmegen
Disputation	09.07.2024

To my family.

Abstract

The ever-growing volume of data across diverse domains presents significant challenges for efficient exploration and analysis. A fundamental task within data exploration is the search for relevant objects in large datasets, which is crucial for subsequent downstream tasks. Traditional data exploration methods often rely on nearest neighbor search, a technique where users provide a relevant example to find the most similar instances in the entire dataset. However, these techniques often fall short in scenarios demanding high accuracy and completeness of the results.

Machine learning offers a promising alternative, enabling users to define a query by providing both relevant and non-relevant example objects to train a model that identifies all relevant objects in the entire data catalog. However, this approach has a critical drawback: to find all relevant instances, such a model needs to be applied to the entire dataset. This necessitates scanning every data point, making it prohibitively expensive for large-scale, interactive use cases serving many users and queries simultaneously.

This cumulative dissertation proposes a novel search framework that leverages the power of machine learning models to enable rapid and accurate searches on massive datasets. Unlike previous approaches that require full data scans, our framework achieves efficient search through an innovative co-design strategy that integrates machine learning models with efficient index structures, allowing for fast retrieval of relevant data points. Through several joint works, the underlying theoretical framework is presented and extended. Moreover, its practical applicability is demonstrated in various real-world scenarios.

Contents

Abstract *v*

1 Introduction **1**
- 1.1 Motivation . . . 1
- 1.2 Thesis Outline . . . 3

I Foundations **5**

2 Machine Learning Background **7**
- 2.1 Supervised Learning . . . 7
  - 2.1.1 Decision Trees . . . 9
  - 2.1.2 Random Forests . . . 11
  - 2.1.3 Artificial Neural Networks . . . 12
  - 2.1.4 Transformers . . . 16
- 2.2 Beyond Supervised Learning . . . 19

3 Content-Based Retrieval **21**
- 3.1 Problem Definition . . . 21
  - 3.1.1 Query Formulation . . . 22
  - 3.1.2 Framework . . . 22
- 3.2 From Raw Data to Embeddings . . . 24
  - 3.2.1 Embeddings . . . 24
  - 3.2.2 Deep Feature Extraction Methods . . . 25
- 3.3 Index Structures for Efficient Search . . . 30
  - 3.3.1 Similarity Search . . . 31
  - 3.3.2 Range Search . . . 43
  - 3.3.3 Search Refinement . . . 44
  - 3.3.4 Limitations . . . 47
- 3.4 Performance Evaluation . . . 47

II Contributions **49**

4 Efficient Search-by-Classification **51**
- 4.1 Search-by-Classification . . . 51
- 4.2 Framework . . . 54
- 4.3 Index-Aware Classifiers . . . 56

4.3.1 Decision Branches 56
4.3.2 Hyperbox-Based Neural Network 61
4.4 Index Structure 64
4.5 Concluding Remarks 66

5 Applications 69
5.1 Open Search-by-Classification Platform 69
5.2 Geospatial Image Search Engine 70
5.3 Text-Image Search Engine 72

6 Conclusion and Outlook 77

Bibliography 79

Appendix: Contributed Publications 89

A Efficient Search-by-Classification 91
A.1 Fast Search-by-Classification for Large-Scale Databases Using Index-Aware Decision Trees and Random Forests 91
A.2 End-to-End Neural Network Training for Hyperbox-Based Classification 92
A.3 Training Neural Networks End-to-End for Hyperbox-Based Classification 93

B Applications 95
B.1 RapidEarth: A Search-by-Classification Engine for Large-Scale Geospatial Imagery 95
B.2 CLIP-Branches: Interactive Fine-Tuning for Text-Image Retrieval . . 96

Chapter 1

Introduction

In recent years, there has been a significant increase in the amount of data collected in a multitude of fields. From high-resolution satellite imagery [30] to the massive databases of genome sequences [86] and the ever-expanding universe of web-based media [112], data is being produced at an exponential rate across all scientific disciplines. In the field of remote sensing, for instance, upcoming Earth observation missions such as *Landsat Next* [116] are projected to generate petabytes of data on a daily basis.

While the capacity to collect data has grown exponentially [90], the ability to navigate these vast repositories and extract meaningful insights has not kept pace. Traditional retrieval methods, such as structured queries in relational databases or nearest neighbor searches, often encounter difficulties when confronted with the sheer volume and complexity of large-scale datasets. These approaches either yield unsatisfactory results or necessitate complex queries that are difficult to formulate [81]. Consequently, these limitations hinder the ability to unlock the true potential of the data and translate it into actionable knowledge.

Machine learning presents a promising solution to this challenge. The field of machine learning has witnessed remarkable advancements, particularly with the rise of deep learning. However, the focus has often been on developing increasingly deep models to solve highly complex problems, including decoding protein structures [62] and constructing large language models [107]. While these advancements are significant, a crucial gap remains: developing efficient machine learning methods specifically designed to be applied to large-scale datasets.

1.1 Motivation

This thesis aims to bridge this gap by developing machine learning models that are not only accurate in their predictive capabilities but also efficient when applied to large datasets. This is crucial for addressing the inherent challenges presented by the growing volumes of data. While this endeavor could extend to numerous tasks and applications, this thesis specifically focuses on improving the efficiency of data retrieval. In these scenarios, users want to retrieve all relevant data to their query from such massive data catalogs without long waiting times. A common request is the search for “interesting” objects, as illustrated in Figure 1.1. For instance, an environmental researcher may want to identify all wind turbines in a large catalog of high-resolution satellite imagery in order to calculate the potential electricity production from wind energy. Meanwhile, another user might be interested in the areas of deforestation using the same dataset. These differing queries highlight the

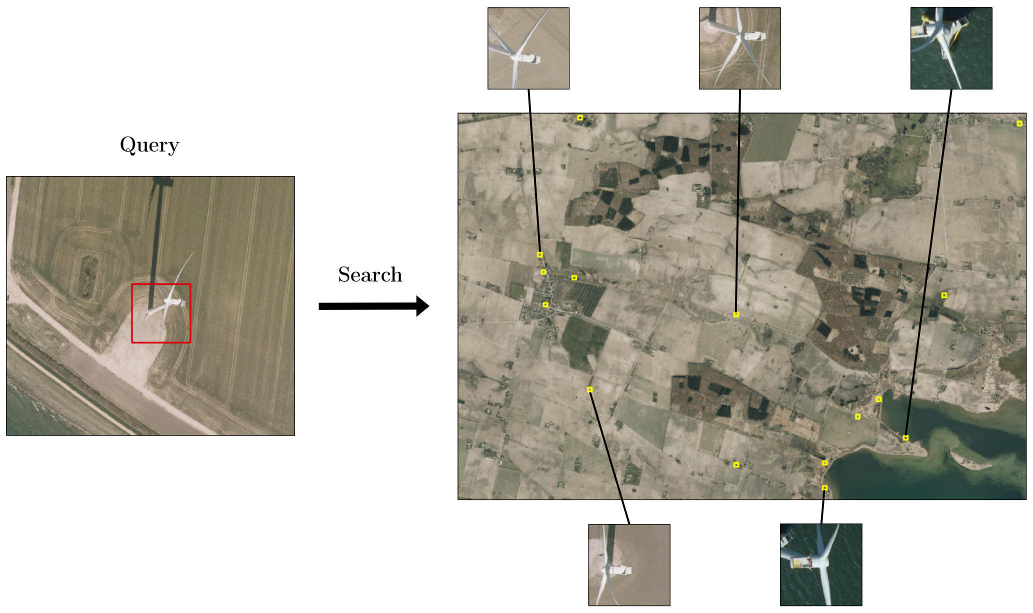


Figure 1.1: Example of a typical search problem for large data catalogs. A user wants to find all instances given a particular example. In this scenario, the search deals with large catalogs of aerial imagery to find all wind turbines in the region. The yellow rectangles on the map visualize all found instances of wind turbines on the map.

challenges these search engines face, which must not only accommodate such diverse needs but also handle them efficiently.

In the domain of database and information retrieval, a common strategy known as *query-by-example* [40] has been widely adopted to address the aforementioned challenges. This method involves providing the search engine with examples of the objects of interest, thereby instructing the search engine as to what it should look out for. In our example, the user would provide the search engine with an image of a wind turbine to tell how the desired objects should look. This led to a wide research branch that found particular attention in the context of media data, such as images, video, audio, and text, collectively referred to as *content-based retrieval* [132].[1]

Traditional search engines primarily rely on nearest neighbor algorithms for efficient retrieval [129]. These techniques typically utilize efficient data structures known as index structures that heavily speed up the query execution. Nevertheless, in certain scenarios, these techniques may yield sub-optimal results, including a high number of false positives or incomplete answer sets. In settings where the results' accuracy is critical, such as in our previous examples, more advanced techniques are required to identify the desired instances correctly and completely.

Machine learning approaches, such as classification, offer a compelling alternative to traditional search engines. By providing both relevant and irrelevant examples, users can train a classification model to identify all desired instances within the entire dataset accurately. A significant drawback, however, is that applying the model to the dataset necessitates scanning each data point, which becomes prohibitively expensive for large-scale datasets.

[1]Well-known implementations of this method include Google Reverse Image Search and the underlying search engines used by platforms like Flickr and Pinterest.

This thesis proposes a novel method that utilizes machine learning to achieve satisfactory query results despite the existing limitations. This method employs a co-design approach, integrating machine learning techniques with efficient index structures to ensure high-quality results and rapid response times. Furthermore, the practical relevance of this framework is demonstrated in the context of two real-world scenarios.

1.2 Thesis Outline

This cumulative dissertation summarizes the findings of multiple publications [73, 74, 81, 82, 83] that have contributed to achieving the overarching research objectives. The remainder of this thesis is divided into two parts.

Part I: Foundations

In this part, we will introduce the underlying concepts that our contributions build upon. Our research unifies ideas from two main areas, which are presented in Chapters 2 and 3.

In Chapter 2, we delve into the realm of machine learning, focusing specifically on supervised learning. We will describe the relevant models such as decision trees or artificial neural networks, which are important for our work. In Chapter 3, we explore the field of information retrieval, with a focus on content-based retrieval as a sub-domain. We will provide an overview of existing approaches for content-based retrieval that utilize various methods of nearest neighbor search, giving context to the current state of the field.

Part II: Contributions

This part presents our contributions, which are in line with our research objectives. It is divided into three chapters.

In Chapter 4, we present our novel search framework, which leverages machine learning models to address content-based retrieval more efficiently and accurately than traditional methods. This chapter introduces not only a new search framework but also novel machine learning models specifically developed for content-based retrieval. In Chapter 5, we build on the newly established search framework and demonstrate its effectiveness in real-world applications. Furthermore, we also address some drawbacks of the original framework. In the final chapter of the thesis, Chapter 6, we present a summary of our findings and provide an outlook for future research. We critically evaluate the outcomes of our contributions concerning our research objectives and offer guidance for researchers who wish to continue exploring this area of study.

Part I
Foundations

Chapter 2

Machine Learning Background

First, we look at the foundations of machine learning that serve as the basis for the algorithms discussed in the remainder of this thesis. Due to the complex nature of machine learning, we have limited our scope to selected topics related to our contributions. More precisely, we focus on the task of supervised learning, in particular classification, and introduce tree- and neural-network-based machine learning models in more detail.

2.1 Supervised Learning

Imagine estimating the value of a new home based on its features, filtering spam from the email inbox, or even predicting disease outbreaks given personal health data. These are just a few of the examples of supervised learning. In supervised learning, a model is trained to learn a mapping between inputs and desired outputs, as shown in Figure 2.1 [53]. The model is provided with labeled examples, wherein each input (x) has a corresponding known outcome (y), such as the price of a house or the label "spam" for an email. By analyzing these examples, the model can identify patterns that allow it to predict the outcome for entirely new, unseen inputs.

More formally, supervised learning involves a training set T of N pairs of input-output pairs, $T = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$ where $\mathbf{x}_i \in \mathbb{R}^D$ corresponds to a D-dimensional feature vector and each corresponding output y_i is a label within a defined label space $\mathcal{Y}$ for $i \in \{1, \ldots, N\}$ [34]. The goal of supervised learning is to learn a function $f^* : \mathbb{R}^D \to \mathcal{Y}$ that accurately predicts the output for any new input $\mathbf{x} \in \mathbb{R}^D$. The task is to estimate ("learn") the function f^* given the training set T of inputs. The estimated function f, referred to as the model, is used to infer a label $\hat{y} = f(\mathbf{x})$ for a new, unseen data point $\mathbf{x}$ as well as possible. The ideal outcome is that the model's prediction matches the true output, that is, $f(\mathbf{x}) = y$. Supervised learning is typically categorized based on the type of output variable. For continuous outputs, where $\mathcal{Y} = \mathbb{R}$, this is referred to as regression. Conversely, for categorical outputs, where $\mathcal{Y} = \{0, \ldots, C-1\}$ and $C \in \mathbb{N}^+$ is the total number of classes, it is named classification. A specific type of classification is referred to as binary classification, where $\mathcal{Y} = \{0, 1\}$, while cases with $C > 2$ are called multi-class classification. This thesis specifically focuses on classification problems.

Performance Metrics in Classification

In the domain of supervised learning, the evaluation of model performance is crucial for understanding how well the model can generalize to unseen data, that is, how well it estimates the function f^*. To assess the generalization capabilities of the

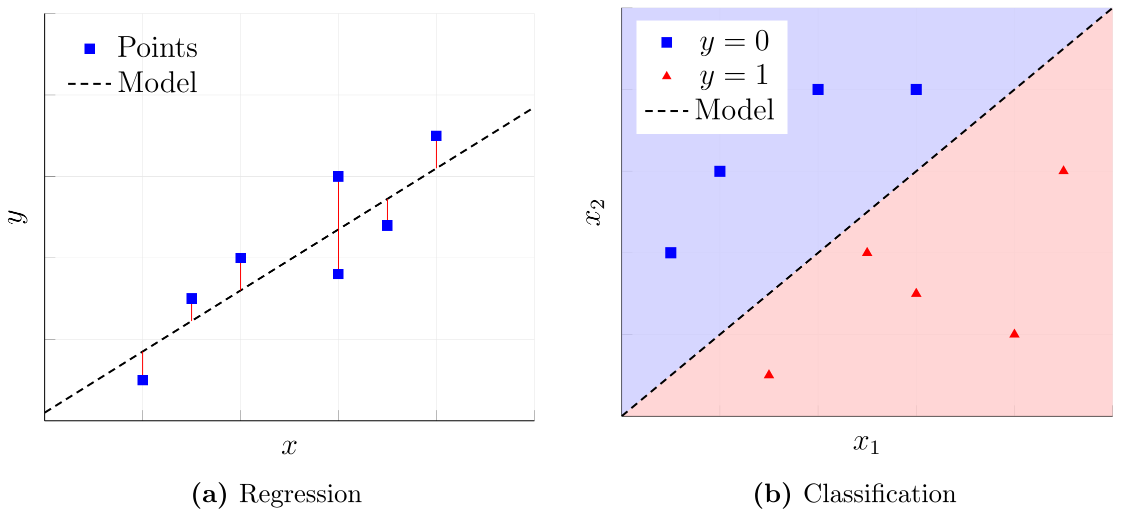


Figure 2.1: Visualization of supervised learning tasks. In regression, a model is learned to predict continuous values, while in classification, it identifies discrete categories.

trained model on unseen data, the model predictions $\hat{y}$ are typically compared to the ground-truth labels y using a designated test set [45]. The test set, denoted as $T_{test} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_M, y_M)\}$, comprises M instances that were not part of the training data. Different performance metrics offer insights into various aspects of the model's capabilities. For classification tasks, metrics such as accuracy, precision, recall, and F-score play an important role. The most intuitive metric is the accuracy of a model, defined as:

$$\text{Accuracy} = \frac{1}{M} \sum_{i=1}^{M} \mathbb{1}(\hat{y}_i, y_i). \tag{2.1}$$

Here, $\hat{y}_i$ represents the predicted label for the i-th instance of T_{test}, y_i is the true label, and $\mathbb{1}(\hat{y}_i, y_i)$ is an indicator function that is 1 if the prediction is correct and 0 otherwise. Accuracy is the fraction of predictions the model got right and is often sufficient to indicate the overall model performance. Other metrics are used when the classification task has some additional constraints. Their underlying concepts are best explained in a binary classification setting, although they also apply to multi-class scenarios. In binary classification scenarios, we distinguish between positive ($y = 1$) and negative labels ($y = 0$). A model prediction $\hat{y}_i$ for element i can be grouped into four categories when compared to the ground truth y_i, as represented by a confusion matrix shown in Table 2.1.

Given the notation in Table 2.1, we can define various classification metrics:

- **Accuracy** can also be denoted as:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}. \tag{2.2}$$

- **Precision** is a measure of the ratio of correctly predicted positive observations to the total predicted positives. It is particularly useful in scenarios where the cost of false positives is high:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}. \tag{2.3}$$

	$\mathbf{y} = \mathbf{1}$	$\mathbf{y} = \mathbf{0}$
$\hat{\mathbf{y}} = \mathbf{1}$	True Positive (TP)	False Positive (FP)
$\hat{\mathbf{y}} = \mathbf{0}$	False Negative (FN)	True Negative (TN)

Table 2.1: Confusion matrix in a binary classification scenario.

- **Recall** is a measure of the ratio of correctly predicted positive observations to all the observations in the actual class. It is a crucial metric in scenarios where the cost of false negatives is high:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}. \tag{2.4}$$

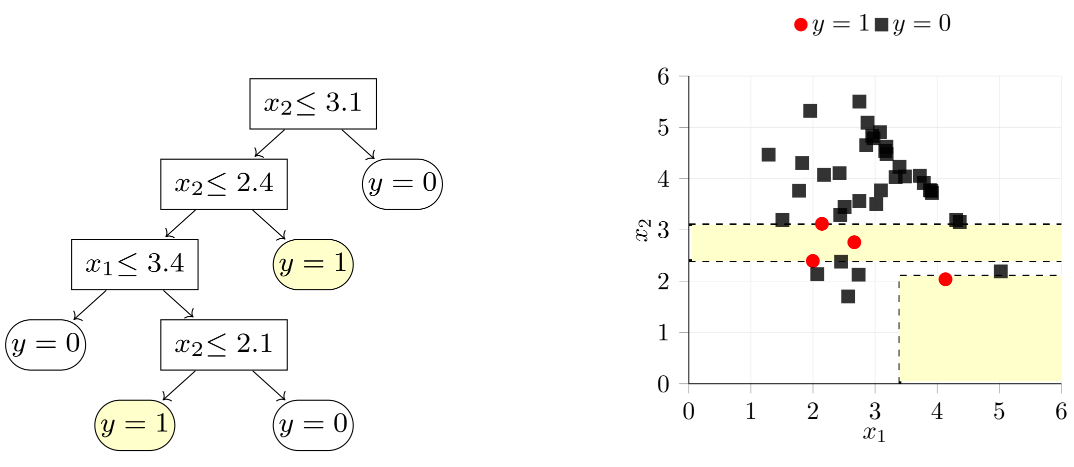
- $\mathbf{F_1}$**-score** is the harmonic mean of precision and recall, considering both false positives and false negatives. It is a useful metric in cases where an equal balance between precision and recall is desired:

$$F_1\text{-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}. \tag{2.5}$$

In scenarios where the distribution of the dataset is dominated by one class, additional metrics can be particularly useful in providing a more comprehensive understanding of model performance. For instance, a model that assigns all instances to the dominating class would yield a very good accuracy, but would not be capable of identifying any other classes. In such scenarios, considering additional metrics like recall and precision and balancing them in F_1-score can provide a more nuanced indication of the generalizability of the model.

2.1.1 Decision Trees

Decision trees, particularly *classification and regression trees* (CARTs) [21], are a simple but powerful machine learning model. Their simplicity and interpretability make them a popular choice for tasks such as classification. They recursively partition the dataset into a set of rectangles. This set is defined by a tree structure, as shown in Figure 2.2. In the context of classification, where the dataset is defined by $T = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\} \subset \mathbb{R}^D \times \{0, \ldots, C-1\}$, the goal of the partitioning is to end up with subsets of the data T that are homogeneous concerning the label y. A decision tree is constructed in a top-down manner, as described in Algorithm 1, that is, from the root node to the leaves, where at each internal node (all nodes except the leaves) a binary split is made that divides the set $S \subseteq T$ into two subsets. These splits are made along a feature axis. To find the optimal split, all possible splits defined by a splitting dimension $\phi \in \{i_1, \ldots, i_\mu\}$ and a splitting threshold $\theta \in \mathbb{R}$ are evaluated, where the hyperparameter $\mu \in \{1, \ldots, D\}$ defines the number of splitting dimensions that are considered per split. In most cases, for single decision trees, $\mu = D$ guarantees that the best current split is identified in a greedy manner. This implies that the construction algorithm is always looking for the current best option, yet at subsequent split points, it may be the case that the preceding splits did not result in the optimal global solution.



(a) Decision tree structure **(b)** Decision surface

Figure 2.2: Decision tree with the corresponding decision surface. The yellow areas in the decision surface correspond to the leaves that contain points of the class $y = 1$, while the white areas belong to the class $y = 0$. Figure adapted from our work [81].

From an optimization perspective, to find the best split (ϕ^*, θ^*), one typically aims at maximizing the so-called information gain $G(S_L, S_R)$. This metric quantifies how well the classes are separated by a split (ϕ, θ) of set S into subsets $S_L = \{(\mathbf{x}, y) \in S \mid \mathbf{x}^{(\phi)} \leq \theta\}$ (i.e., left leaf) and $S_R = S \setminus S_L$ (i.e., right leaf). More precisely, the information gain is defined as:

$$G(S_L, S_R) = Q(S) - \frac{|S_L|}{|S|} Q(S_L) - \frac{|S_R|}{|S|} Q(S_R), \tag{2.6}$$

where $Q(S)$ denotes an impurity function. Then, to find the best split (ϕ^*, θ^*) for S, we solve the following optimization problem:

$$(\phi^*, \theta^*) = \underset{\phi,\theta}{\operatorname{argmax}}\, G(S_L^{(\phi,\theta)}, S_R^{(\phi,\theta)}), \tag{2.7}$$

where $\phi \in \{\phi_1, \ldots, \phi_\mu\} \subseteq \{1, \ldots, D\}$ and the subsets are defined as $S_L^{(\phi,\theta)} = \{(\mathbf{x}, y) \in S \mid \mathbf{x}^{(\phi)} \leq \theta\}$ and $S_R^{(\phi,\theta)} = S \setminus S_L^{(\phi,\theta)}$.

The definition of $G(S_L, S_R)$ relies on the impurity function $Q(S)$, which measures the degree of class mixing within a subset. In literature, numerous impurity functions have been proposed. For classification scenarios, a common choice is the *Gini index*, which is given by:

$$Q_{gini}(S) = \sum_{c=0}^{C-1} p_c(1 - p_c). \tag{2.8}$$

Here, $p_c \in [0, 1]$ is the fraction of points in S belonging to class $c \in \{0, \ldots, C-1\}$. The Gini index is minimal with $Q_{gini}(S) = 0$, which occurs when all instances belong to a single class. This is commonly referred to as pure. Conversely, the Gini index reaches its maximum value, $Q_{gini}(S) = (C-1)/C$, when the C classes are distributed evenly across the dataset.

Algorithm 1 Constructing a decision tree

```
Require: Point set $S \subseteq \mathbb{R}^D \times \{0, \dots, C-1\}$, stopping criterion $\lambda$, hyperparameter
         $\mu \in \{1, \dots, D\}$ ▷ Common choices for $\lambda$ are to stop after a certain depth
         of the tree or until the set is pure.
Ensure: Decision tree $\mathcal{T}$ build for $S$
 1: function BuildDecisionTree($S, \lambda, \mu$)
 2:     if $\lambda$ is met then
 3:         return leaf node storing majority class as a label
 4:     Find optimal split $(\phi^*, \theta^*)$ for $S$                ▷ See Equation 2.7
 5:     $S_L = \{(\mathbf{x}, y) \in S \mid \mathbf{x}^{(\phi^*)} \leq \theta^*\}$
 6:     $S_R = S \setminus S_L$
 7:     $\mathcal{T}_L \leftarrow$ BuildDecisionTree($S_L, \lambda, \mu$)
 8:     $\mathcal{T}_R \leftarrow$ BuildDecisionTree($S_R, \lambda, \mu$)
 9:     Add node storing $(\phi^*, \theta^*)$ and pointers to $\mathcal{T}_L$ and $\mathcal{T}_R$ to the tree $\mathcal{T}$
10:     return $\mathcal{T}$
```

Next to the Gini index, there exist two other common impurity functions for classification scenarios [53]: *entropy* and *misclassification error* (MCE) denoted as:

$$Q_{entropy}(S) = -\sum_{c=0}^{C-1} p_c \log_2(p_c), \text{ and} \tag{2.9}$$

$$Q_{MCE}(S) = 1 - max(p_c), \tag{2.10}$$

respectively. The tree construction stops after each leaf node ends up with only pure subsets of the data or when a stopping criterion λ is met (e.g. all leaves are pure or the maximum depth of the tree is reached). In the majority of cases, fully grown trees are not recommended as they also learn the included noise of the training data, a phenomenon known as *overfitting* [53] the data. This results in a lack of generalization on new, unseen data. Hence, it is generally recommended to employ early stopping or pruning of the tree in a post-processing routine.

To predict the class for a new data point $\mathbf{x} \in \mathbb{R}^D$ using a decision tree $\mathcal{T}(\mathbf{x})$, the majority class in the leaf to which the new point is assigned determines the predicted class. This assignment is based on the feature values of $\mathbf{x}$, which navigate the path from the root to the appropriate leaf of the tree.

One significant challenge with decision trees is their tendency to exhibit high variance. Minor variations in the data can lead to drastically different split decisions, making interpretation somewhat complicated. This instability primarily stems from the tree's hierarchical structure, where an error in the top split cascades down to all subsequent nodes. To mitigate this problem, the concept of *bootstrap aggregating* (bagging) [20] is often employed. Bagging is a method that combines multiple trees to reduce variance. This is also the underlying concept for *random forests* explained next.

2.1.2 Random Forests

Despite their simplicity, random forests remain among the most effective machine learning models [41]. In essence, random forests construct an ensemble of L inde-

Algorithm 2 Constructing a random forest

```
Require: Point set $S \subseteq \mathbb{R}^D \times \{0, \dots, C-1\}$, number of tree models $L$, stopping
         criterion $\lambda$, hyperparameter $\mu \in \{1, \dots, D\}$
Ensure: Random forest $\mathcal{E}$ build for $S$
1: Empty set $\mathcal{E} \leftarrow \{\}$ for storing individual trees
2: for $i = 1$ to $L$ do
3:     Draw a bootstrap sample $Z_i$ of size $|S|$
4:     $\mathcal{T}_i \leftarrow$ BuildDecisionTree$(Z_i, \lambda, \mu)$
5:     $\mathcal{E} \leftarrow \mathcal{E} \cup \{\mathcal{T}_i\}$
6: return $\mathcal{E}$
```

pendent decision tree models $\mathcal{T}_1, \dots, \mathcal{T}_L$ on randomly sampled subsets of the training data T [20]. The sampling technique is called *bootstrapping*, which uniformly draws N instances from the training data with replacement. Each tree is then trained on one of the individual bootstrap samples $Z_1, \dots, Z_L \subset T$, as shown in Algorithm 2. Notably, at each split during the tree construction, a new random subset with μ features is considered to find the best split.

For classification, random forests employ a majority vote (also known as "wisdom of the crowd") to predict the class for a new, unseen $\mathbf{x}$ with:

$$\mathcal{E}(\mathbf{x}) = \underset{c \in \{0, \dots, C-1\}}{\operatorname{argmax}} \sum_{i=1}^{L} \mathbb{1}(\mathcal{T}_i(\mathbf{x}), c), \tag{2.11}$$

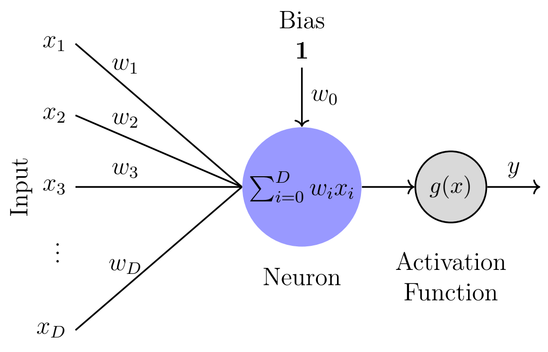
where the indicator function $\mathbb{1}(\mathcal{T}(\mathbf{x}), c)$ outputs 1 if $\mathcal{T}(\mathbf{x}) = c$ and 0 otherwise. The aggregation of the outcomes of multiple individual trees trained on bootstrap samples is referred to as bagging, which was initially proposed by Breiman [20]. The strength of bagging lies in its ability to reduce variance and mitigate overfitting by inducing randomness. The introduction of randomness, both in the data sampling and in the split feature selection, introduces diversity into the trees, preventing them from becoming overly dependent on specific features or patterns in the data. This diversity promotes generalizability to unseen data, leading to robust final predictions of the ensemble. This principle will be of great importance for the development of our tree-based models in Chapter 4.

Another tree-based ensemble model called *extremely randomized trees* or simply *extra trees* [46] takes the induced randomness to an even higher level by introducing randomness also to the split point selection.[1] For each randomly selected feature, it picks a random split point instead of the best. While bagging averages diverse, independent trees, another ensemble technique called *boosting* builds trees sequentially, with each tree trying to correct the errors of the previous one. One of the most popular boosting algorithms is *XGBoost* [26].

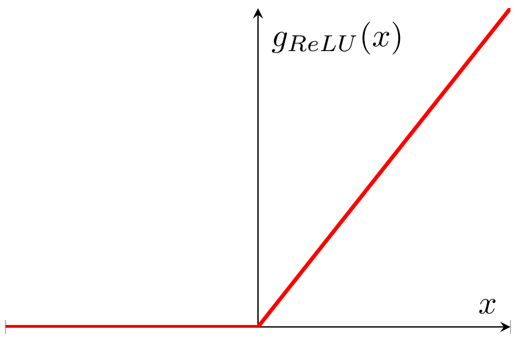
2.1.3 Artificial Neural Networks

Artificial neural networks belong to a class of machine learning models inspired by the neural structures of the human brain [48]. At their core, they consist of fun-

[1]However, each tree is trained on the entire training set T rather than on bootstrap samples, only.



(a) Single neuron



(b) ReLU function

Figure 2.3: Figure (a) shows a representation of a signal neuron as a linear function that forwards its output through a non-linear activation function. Figure (b) depicts the ReLU activation function.

damental units known as neurons, each representing a basic linear function $y = \mathbf{w}^\top\mathbf{x} + b$, where $\mathbf{w} \in \mathbb{R}^D$ and $b \in \mathbb{R}$ denote the learnable weights and biases, respectively, and $\mathbf{x} \in \mathbb{R}^D$ is the input. A schematic representation of a single neuron, also called perceptron, is shown in Figure 2.3a. To allow artificial neural networks to model nonlinear relationships, the output of each neuron is passed through a non-linear activation function $g : \mathbb{R} \to \mathbb{R}$. A common choice is the function *rectified linear unit* (ReLU), denoted by $g_{ReLU}(x) = \max(0, x)$ and shown in Figure 2.3b. Conceptually, the ReLU function can be understood as a threshold that only forwards the signal of the neuron once the signal strength exceeds the value zero.

Feedforward Neural Networks

The true potential of artificial neural networks is revealed when multiple neurons are combined into neural networks. This thesis will focus on *feedforward neural networks* (FNNs) [48], a subset of artificial neural networks where the signal flow is unidirectional, moving from input $x \in \mathbb{R}^D$ to output y. The architecture of FNNs is organized into layers: input, one or more hidden, and output layers. As shown in Figure 2.4, the input layer receives the input data, the hidden layers perform computations and transformations through weighted connections of the neurons, and the output layer produces the final prediction. Each layer's output can be considered as the input for the next layer, leading to a concatenation of functions that produces the final output $f(\mathbf{x}) = f^{(L)}(\dots f^{(2)}(f^{(1)}(\mathbf{x}))$, where $f^{(i)}$ represents the function computed by the i-th layer and L is the number of hidden layers, which is also called the depth of the network. When considering neural networks with multiple neural networks (which are commonly used with hundreds of layers), we refer to them as *deep neural networks* (DNNs) or deep learning. The configuration of the individual hidden layers varies according to the specific model architecture, which is dependent on the number of neurons, the number of connections, the type of neurons, and other factors.

The output layer is modified in accordance with the specific learning task at hand, ensuring that the model output aligns with the task objective. In the context of classification, we distinguish between binary classification and multi-class classifica-

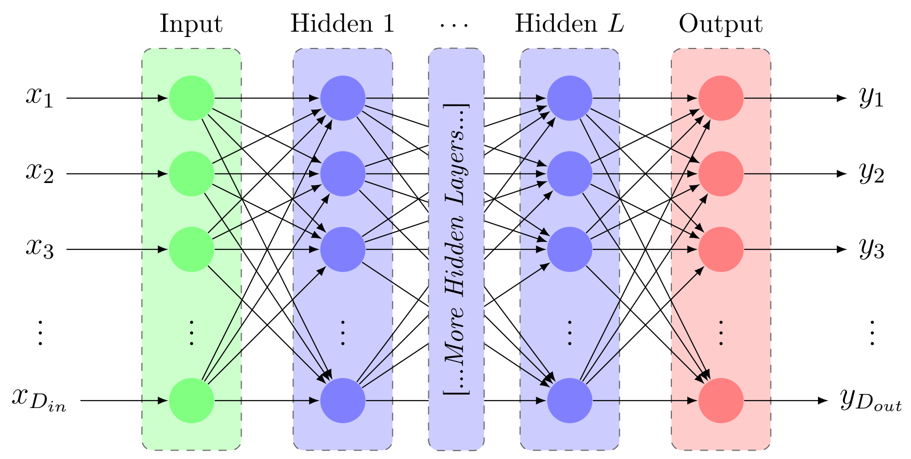


Figure 2.4: Architecture of a feedforward neural network. It consists of one input layer with D_{in}-dimensional input and one output layer of the size D_{out}. In between are L hidden layers with varying input and output connections.

tion. For binary classification, it is sufficient to have a single output neuron followed by a sigmoid activation function, which is given by:

$$g_\sigma(x) = \frac{1}{1+e^{-x}} = \frac{1}{1+\exp(-x)}. \tag{2.12}$$

The activation function outputs values between 0 and 1 that can be interpreted as probabilities of predicting class 1. In the case of multi-class classification, one output neuron is associated with each class C. The softmax activation function $g_{softmax} : \mathbb{R}^C \to (0,1)^C$ is employed that takes all the neurons' output, also called logits, as input $\mathbf{x} \in \mathbb{R}^C$. It maps the logits to probabilities that sum up to 1, formally defined as:

$$g_{softmax}(\mathbf{x})_j = \frac{e^{\mathbf{x}_j}}{\sum_{c=1}^{C} e^{\mathbf{x}_c}}, \qquad j \in \{1, \dots, C\}. \tag{2.13}$$

Model Training

Training FNNs involves a two-step iterative process: a forward pass and a backward pass [67]. During the forward pass, the network calculates the predicted output $\hat{y}$ for a given input $\mathbf{x}$, using the current weights and biases. The model's performance is assessed by computing the loss between the predicted output $(\hat{y})$ and the actual target output (y). To accurately assess the performance in classification tasks, a loss function defined as $\mathcal{L} : \{0,1\}^C \times (0,1)^C \to \mathbb{R}$ is employed to quantify the discrepancy between the predicted outputs and the actual target outputs. This necessitates that the vector of true class labels, y, be represented in a one-hot encoded format, where a class label y is transformed into a binary vector $\mathbf{y} \in \{0,1\}^C$. In particular, if a specific instance belongs to class c, then the c-th element of the vector $\mathbf{y}$ is 1 and all other elements are 0. A common choice for this purpose in classification tasks is the *cross-entropy* (CE) loss, which is defined for a single sample as follows:

$$\mathcal{L}_{CE}(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{c=1}^{C} y_c \log(\hat{y}_c), \tag{2.14}$$

where $\mathbf{y}$ contains the true class labels and $\hat{\mathbf{y}}$ the predicted class probabilities.

For binary classification tasks, the function $\mathcal{L}$ is specifically defined for binary outputs and takes the form $\mathcal{L} : \{0, 1\} \times (0, 1) \to \mathbb{R}$. For CE, we refer to the binary loss variant called *binary cross-entropy* (BCE), which is defined as:

$$\mathcal{L}_{BCE}(y, \hat{y}) = -(y \log \hat{y} + (1 - y) \log(1 - \hat{y})), \tag{2.15}$$

where y is the true binary label and $\hat{y}$ is the predicted probability of the class with label 1.

In the backward pass, the network weights are modified to reduce the calculated loss. To properly adjust the weights, one generally makes use of gradient-based learning methods. For this purpose, the overall gradient $\nabla\mathcal{L}(\mathbf{w})$ of the loss function w.r.t. the weights[2] is computed. This gradient elucidates how changes in weights influence the overall error. The exact computation of the gradients of each weight is non-trivial as this requires resolving the network's chained nature. For this reason, an effective algorithm, named *backpropagation* [101], has been developed to iteratively traverse the network from the final layer to the initial layer, computing the gradient of each layer using the multi-dimensional chain rule.

The gradient descent method is employed to perform weight updates, whereby weights are adjusted in the direction opposite to the gradient in order to minimize the loss. Formally, if $\mathbf{w}_t$ represents the weights at iteration t, the update rule is:

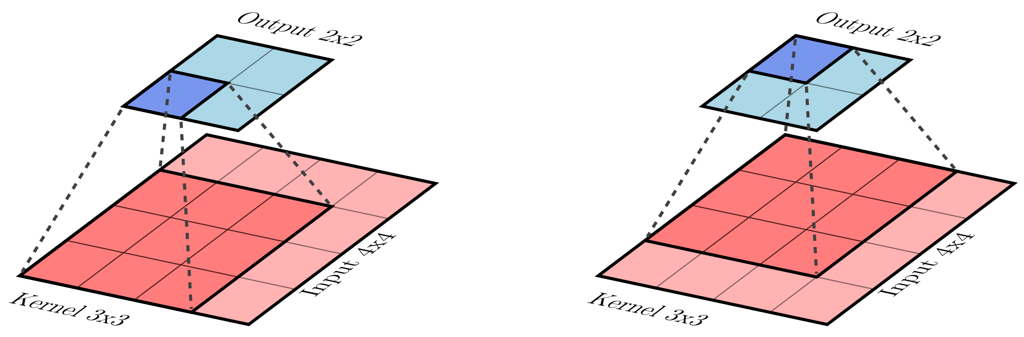
$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \cdot \nabla\mathcal{L}(\mathbf{w}_t), \tag{2.16}$$

where η denotes the *learning rate*, a crucial hyperparameter that influences the magnitude of weight updates. To be more precise, this is done by computing the gradient based on the entire training set T. Since the computation of the partial derivatives can become quite expensive, especially for large datasets, in practice the weight updates are performed on random subsets of the training set $S \subset T$, commonly referred to as *batches*. The size of a batch, denoted by $|S|$, is an important hyperparameter for the model training. The process of updating weights using batches of data and computing gradients iteratively over multiple *epochs* (complete passes through the dataset) continues until the loss converges to a local minimum or another stopping criterion is met. The learning rate must be carefully chosen; a too high rate may cause overshooting of minima, while a too small rate can slow down convergence significantly.

Convolutional Neural Networks

Convolutional neural networks (CNNs) [67] represent a specialized variant of FNNs that are particularly adept at processing data with a grid-like topology, such as images. CNNs differentiate themselves from traditional FNNs by incorporating convolutional layers as layers that perform convolutional operations. These employ learnable filters that slide across the input data, capturing spatial patterns, as shown in Figure 2.5. The essence of the convolution operation involves sliding a small window (or kernel) across the input features and computing the weighted sum of the filter

[2]In this context, the term weights is used to refer to both weights and biases for the sake of simplicity.



(a) Step 1 **(b)** Step 2

Figure 2.5: Sliding window convolving a 3x3 kernel over a 4x4 input producing a 2x2 output feature map. Each output value is calculated by the weighted sum of the input data window and the filter weights of the kernel.

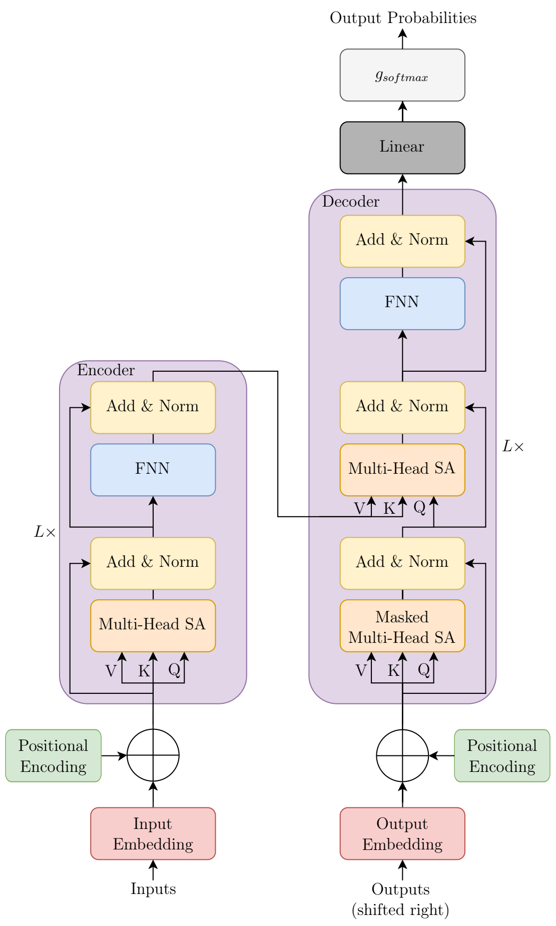
weights and the input data encapsulated by this window [47]. This process assists the network in capturing local structures, from basic edges and shapes to more intricate combinations in images as the number of layers is increased. The typical architecture of CNNs combines convolutional layers with pooling layers. Pooling layers serve to compress the spatial dimensions of the data representations, primarily through methods such as maximum pooling or average pooling. This compression significantly decreases the number of parameters and computational load required by the network, enhancing its efficiency and reducing overfitting. Compared to fully-connected FNNs, CNNs have a relatively low number of weights due to the shared weights in the convolutional layers and the pooling operations. However, they still have stronger classification performances for grid-like data (e.g. images) due to their ability to learn spatial features [102].

2.1.4 Transformers

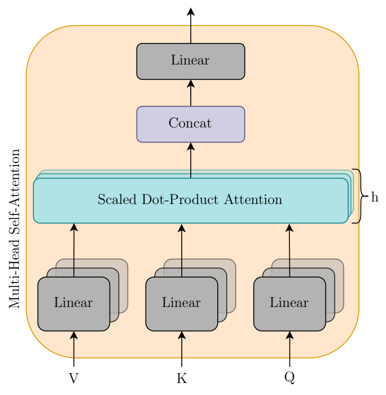
Although CNNs demonstrate strong performance with grid-like data such as images, a novel neural network architecture known as *transformer* [118] has shown considerable promise. While originally stemming from the domain of natural language processing, other works have also shown their power for other data types such as in the area of computer vision [37]. These networks are designed to handle sequential data without the need for recurrence or convolution, leveraging a mechanism called *self-attention* (SA) to draw global dependencies between input and output.

The transformer model consists of two main components: the encoder and the decoder. The encoder maps the input data to a continuous intermediate representation. Given the intermediate representations, the decoder generates the output. In the context of natural language processing, the transformer accepts a text sequence in the form of a sentence, tokenizes it, and outputs the probabilities for the next token at each position in the sequence. The entire transformer architecture is shown in Figure 2.6, which illustrates the high-level encoder-decoder structure and the details of the attention mechanism.

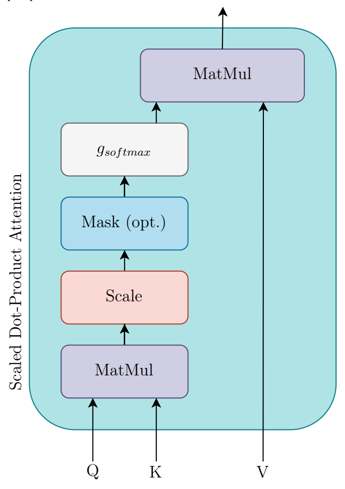
Both, the encoder and decoder comprise a stack of L identical layers ($L = 6$ in original paper [118]). In the encoder, each layer consists of two sublayers where the first is a multi-head self-attention (MSA) block and the second a position-wise FNN.



(a) Encoder-Decoder architecture



(b) Multi-head self-attention



(c) Scaled dot-product attention

Figure 2.6: Transformer architecture going from a high-level representation of the encoder-decoder architecture into the details of the underlying multi-head self-attention mechanism. Figure adapted from Vaswani et al. [118].

All sublayers as well as the embedding layers, produce outputs of dimensionality D ($D = 512$ in original paper [118]). At the end of each sublayer, *add & norm* blocks are employed to stabilize the training. Here, the sublayer's input is added to the output of the sublayer, a technique known as residual connections that was originally proposed in the ResNet [55] model to make the training of very deep FNNs more stable. On top, the sum of both is normalized along the feature dimension, called *layer normalization* (LN) to accelerate the training [9]. The FNN consists of two fully-connected layers with a ReLu activation in between that is applied across all input positions with the same linear transformations. The FNN can be denoted with:

$$\mathrm{FNN}(\mathbf{x}) = g_{ReLU}(\mathbf{x}\mathbf{W}_1^\top + \mathbf{b}_1)\mathbf{W}_2 + \mathbf{b}_2, \tag{2.17}$$

where $\mathbf{W}_1, \mathbf{W}_2 \in \mathbb{R}^{D_m \times D}$ correspond to the weight vectors of the first respectively second layers and $\mathbf{b}_1, \mathbf{b}_2$ are bias vectors where $\mathbf{b}_1 \in \mathbb{R}^{D_m}$ and $\mathbf{b}_2 \in \mathbb{R}^{D}$ with D_m being the inner-layer dimensionality ($D_m = 2048$ in original paper [118]). The decoder is of the same size as the encoder. In addition to the two sublayers in the decoder, another MSA layer is added over the outputs of the encoder. Moreover, the first attention block in the decoder is masked such that the next token prediction at position i is only influenced by the previous tokens at positions less than i. Finally, the output of the decoder is processed by a final linear layer and a softmax layer $g_{softmax}$ that outputs the probabilities for the next token.

Self-Attention

At the core of the transformer model is the self-attention mechanism shown in Figure 2.6c. This mechanism enables the model to dynamically focus on different parts of the input sequence as it processes data. The underlying mechanism of the self-attention technique is the so-called *scaled dot-product attention*, where at each unit three weight matrices are learned for the query $\mathbf{W}_Q \in \mathbb{R}^{D \times D_Q}$, the keys $\mathbf{W}_K \in \mathbb{R}^{D \times D_K}$, and the values $\mathbf{W}_V \in \mathbb{R}^{D \times D_V}$, respectively. For each token i of the sequence, the input representation $\mathbf{x}_i$ is multiplied with each of the corresponding weight matrics to produce the query $\mathbf{q}_i = \mathbf{x}_i\mathbf{W}_Q$, key $\mathbf{k}_i = \mathbf{x}_i\mathbf{W}_K$ and the value vector $\mathbf{v}_i = \mathbf{x}_i\mathbf{W}_V$. Let $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ be the matrices, where the i-th row are vectors $\mathbf{q}_i, \mathbf{k}_i, \mathbf{v}_i$ respectively, then the scaled dot-product attention is defined as:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = g_{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{D_K}}\right)\mathbf{V}, \tag{2.18}$$

where $\sqrt{D_K}$ is used as a scaling factor. This form of attention mechanism is called self-attention, as it considers only the distinct positions of a given sequence, in contrast to previous approaches that considered the entirety of earlier sequences. In the context of a sentence as an input sequence, the self-attention mechanism compares each token in the sentence to all other tokens, allowing the model to understand how the meaning of each token is related to the others and focus on the most relevant ones for a specific task.

Instead of performing single attention, in each MSA block, attention is computed with different learned linear projections of the queries, keys, and values in h different attention heads ($h = 8$ in original paper [118]), as shown in Figure 2.6b. The D_V-dimensional outputs are then concatenated and linearly projected as follows:

$$\begin{aligned} \text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) &= \text{Concat}(\text{head}_1, \dots, \text{head}_h)\mathbf{W}_O, \\ \text{where head}_i &= \textit{Attention}(\mathbf{Q}\mathbf{W}_Q^{(i)}, \mathbf{K}\mathbf{W}_K^{(i)}, \mathbf{V}\mathbf{W}_{\mathbf{V}}^{(i)}). \end{aligned} \tag{2.19}$$

Here, $\mathbf{W}_O \in \mathbb{R}^{hD_\mathbf{V} \times D}$ are the weights of the final linear layer.

The transformer model processes data in parallel rather than in sequence, which means it lacks an inherent understanding of the order of tokens. To address this, positional encoding is added to the input embeddings at the base of both the encoder and decoder. This encoding provides information about the position of each token in the sequence, ensuring that the model can consider the order of tokens when processing text.

Vision Transformer

While the original transformer architecture is designed for one-dimensional sequences of text, the *vision transformer* (ViT) [37] allows for the processing of image data by implementing minor modifications to the architecture. In essence, the input image data must be transformed into a compatible format for the ViT. Let $\mathbf{x} \in \mathbb{R}^{\mathcal{H} \times \mathcal{W} \times \mathcal{C}}$

be an image input where $\mathcal{H}$ is the height, $\mathcal{W}$ the width and $\mathcal{C}$ the number of channels, e.g. $\mathcal{C} = 3$ for RGB. The image is then split into $\mathcal{N} = \frac{\mathcal{H}\mathcal{W}}{\mathcal{P}^2}$ image patches $\mathbf{x}_p \in \mathbb{R}^{\mathcal{N}\times(\mathcal{P}\times\mathcal{P}\times\mathcal{C})}$ where $\mathcal{P}$ is the height and width of the patches. These image patches are flattened by alinear projection with a learnable matrix of $\mathbf{E} \in \mathbb{R}^{(\mathcal{P}\times\mathcal{P}\times\mathcal{C})\times D}$ resulting in D-dimensional embeddings called *patch embeddings*. Similar to the original transformer, positional embeddings are added to the patch embeddings as the initial input to the encoder denoted as $\mathbf{E}_{pos} \in \mathbb{R}^{(\mathcal{N}+1)\times D}$. Additionally, a learnable class token $\mathbf{x}_{class}$ is incorporated into the model, which serves as the overall image representation at the end of the encoder. The input of the encoder $\mathbf{z}_0$ can be described as follows:

$$\mathbf{z}_0 = [\mathbf{x}_{class}, \mathbf{x}_p^1\mathbf{E}, \mathbf{x}_p^2\mathbf{E}, \ldots, \mathbf{x}_p^{\mathcal{N}}\mathbf{E}] + \mathbf{E}_{pos}. \tag{2.20}$$

In the encoder, only minor architectural modifications are made. These include the relocation of the layer normalization to the initial stage of each sublayer and the replacement of the ReLU activation function in the FNN with a GELU non-linearity [56]. The output of the encoder $\mathbf{z}_l$ at layer l with $l = 1, \ldots, L$ can be defined by:

$$\mathbf{z}'_l = \mathrm{MSA}(\mathrm{LN}(\mathbf{z}_{l-1})) + \mathbf{z}_{l-1}, \tag{2.21}$$

$$\mathbf{z}_l = \mathrm{FNN}(\mathrm{LN}(\mathbf{z}'_l)) + \mathbf{z}'_l. \tag{2.22}$$

Unlike the original transformer, the ViT lacks a decoder, as it was originally designed for classification tasks. To make the final prediction, the final state of the class token $\mathbf{z}_L^{(0)}$ is layer normalized and run through another FNN that outputs the class probabilities.

With these modifications, the ViT has the potential to outperform existing state-of-the-art CNN-based approaches on image classification tasks. However, in its original version, there are some drawbacks [64]. Unlike CNNs, which inherently capture local spatial hierarchies due to their convolutional structure, the ViT does not have this built-in bias towards image data.[3] While this may be advantageous in terms of model flexibility, it also implies that the ViT may not effectively capture local features when not trained on large datasets. This, coupled with the computationally intensive nature of self-attention mechanisms, translates to models with higher computational demands. Despite these drawbacks, ongoing research is addressing many of these issues, enhancing the efficiency and applicability of the transformer.

2.2 Beyond Supervised Learning

This section will briefly introduce other well-known learning paradigms and delineate them from supervised learning. This overview is not meant to be complete and only focuses on categories relevant to the thesis. In general, we classify them based on their learning objective and degree of supervision [45].

Unsupervised Learning

Unlike supervised learning, models in unsupervised learning identify hidden patterns and relationships within the data without explicit labels for the data [53].

[3]This is known under the term *inductive bias* [13].

Well-known examples are clustering algorithms like k-means [79] that group similar data points based on specific characteristics. This helps uncover natural categories within the data. Other methods aim to reduce the number of features in a dataset while preserving the most important information such as *principal component analysis* (PCA) [61]. This simplifies data visualization and analysis as users can visualize the data in human-interpretable two- or three-dimensional space.

Semi-Supervised Learning

Semi-supervised learning involves training models on a small amount of labeled data supplemented by a large amount of unlabeled data [117]. This approach leverages the structure or distribution of the unlabeled data to improve learning accuracy. This can be particularly beneficial in scenarios where the labeling process for new data is costly.

Self-Supervised Learning

In recent years, *self-supervised learning* (SSL) [78] has emerged as a new learning paradigm, driven by the limitations of conventional supervised learning methods. Supervised learning relies heavily on expensive manual labeling and is prone to generalization errors and adversarial attacks related to the labels. In contrast to that, self-supervised learning presents itself as a valid alternative without the need for pre-labeled data. Essentially, self-supervised learning operates by extracting labels directly from the input data through the definition of a pretext task. This task enables the model to infer structure and knowledge from the data itself. According to Liu et al. [78], self-supervised techniques can be classified based on their type of pretext task, which can be contrastive, generative, or generative-constrative:

- **Generative**: These methods train an encoder to convert the input x into a vector z and a decoder to reconstruct x from z. One example are autoencoders [101].
- **Contrastive**: These methods train an encoder to convert the input x into a vector representation z, which is then used to measure the similarity between different inputs. One well-known example is *SimCLR* [27] for learning image representations.
- **Generative-Contrastive**: This approach combines the generation of synthetic samples with the discrimination between real and fake samples. An example are *generative adversarial networks* (GANs) [32].

Chapter 3

Content-Based Retrieval

3.1 Problem Definition

Content-based retrieval (CBR) is a subfield of information retrieval that focuses on retrieving data based on its intrinsic content, rather than relying on metadata like tags or keywords [111]. This approach is particularly used in multimedia retrieval where the retrieval should be based on the actual contents of the media data (e.g. images, audio, video, text, etc.). For instance, in image retrieval, CBR systems analyze the visual features of an image such as color, texture, and shape, rather than relying on manually entered keywords. This enables the retrieval of images that are visually similar to a query image, offering a more intuitive and efficient search experience than providing a potentially imprecise keyword as a query item.

Instead of searching the raw data directly, in CBR the data catalog is first transformed into compact vector representations, called embeddings [119]. These embeddings capture low-level feature descriptors of the content and offer a more efficient representation for searching. Given a catalog of embeddings, a CBR search aims to find the most similar embeddings to the query embedding in the data catalog using distance measures. These measures can be efficiently computed leveraging pre-built data structures. Items with embeddings that are closer in the feature space are considered more relevant than distant ones. It is important to note that the quality of the embeddings is crucial for the search. The better the embeddings represent the inherent characteristics of the data, the more precise the results. This is why, over the past decades, numerous methods have been developed to generate embeddings that capture different characteristics of the data, such as texture, edges, or color in images. One well-known method is *scale-invariant feature transform* (SIFT) [80], which has been utilized for CBR in the image domain for a long time.

Despite these advancements, a significant challenge in CBR remains the *semantic gap* [130], which refers to the difficulty of using low-level feature embeddings to respond to high-level semantic queries. For instance, a user may search for images containing the Eiffel Tower, but the embeddings might only detail the shapes and colors present in the images, not the semantic concept of the Eiffel Tower itself. Recent developments in deep learning [119] aim to bridge this gap. By replacing traditional feature extraction techniques like SIFT with advanced DNNs, these modern approaches strive to minimize the gap between user queries and the characteristics captured by the embeddings. This thesis will focus on the recent advancements in the field of CBR leveraging DNNs for extracting the embeddings, which are summarized under the term *deep feature extraction* [28].

3.1.1 Query Formulation

The variety of CBR approaches allows for the support of different types of queries. This is determined by the type of searches that should be enabled by the search engine [134]. The architecture of a retrieval system is fundamentally designed around these query functionalities. CBR distinguishes between two query types, differentiated by their capacity to capture the semantics of user intentions according to Faloutsos et al. [40]:

- **Query-by-Example (QBE):** The most straightforward method and the most commonly used in CBR is query-by-example, which takes an example instance as a query and searches for the most similar items in the data catalog. The simplicity of this approach lies in the fact that the creation of the embeddings of the query is the same as the creation of the embeddings of the instances in the database, which therefore only requires one feature extraction method. Typically, a single example is provided in the query. However, there are also approaches [115, 122] that accept multiple instances as a query, with the intention of more precisely defining the user's intent. These approaches primarily rely on fusing the embeddings of the individual examples into a joint embedding that is then used for the search.
- **Query-by-Content (QBC):** In cases where the creation or provisioning of an example for the user is non-trivial, alternative approaches exist that facilitate the construction of a query for the user. These approaches do not rely on a concrete example but rather on an abstract concept that describes the object of interest. One straightforward strategy is the use of textual descriptions to define the content of interest. In the case of images, this can be achieved through the use of sketches of images, which are known as *query-by-sketch* [22, 125]. While this approach simplifies the creation of a query for the user, it introduces new challenges for the search engine, which must transform the query into the same embedding format as the database instances.

3.1.2 Framework

Once the query type has been defined, the CBR system can be set up. Essentially, a CBR system must provide two primary functionalities [19]. Firstly, it must include a robust feature extraction component that is capable of deriving compact embeddings from the input data. Secondly, the system needs a search component responsible for conducting efficient searches given a query and a data catalog. This process typically involves the utilization of pre-built index structures to facilitate efficient data retrieval. According to Zhou et al. [134], the workflow of a CBR system can be grouped into two distinct phases: the offline and the online stage. A more detailed representation of the CBR workflow is presented in Figure 3.1, which is further explained below.

Offline Stage

The offline stage represents the preprocessing phase before the user can start querying the system [19]. Initially, the data catalog on which the searches are to be performed is collected. Then, the data is processed during the feature extraction phase

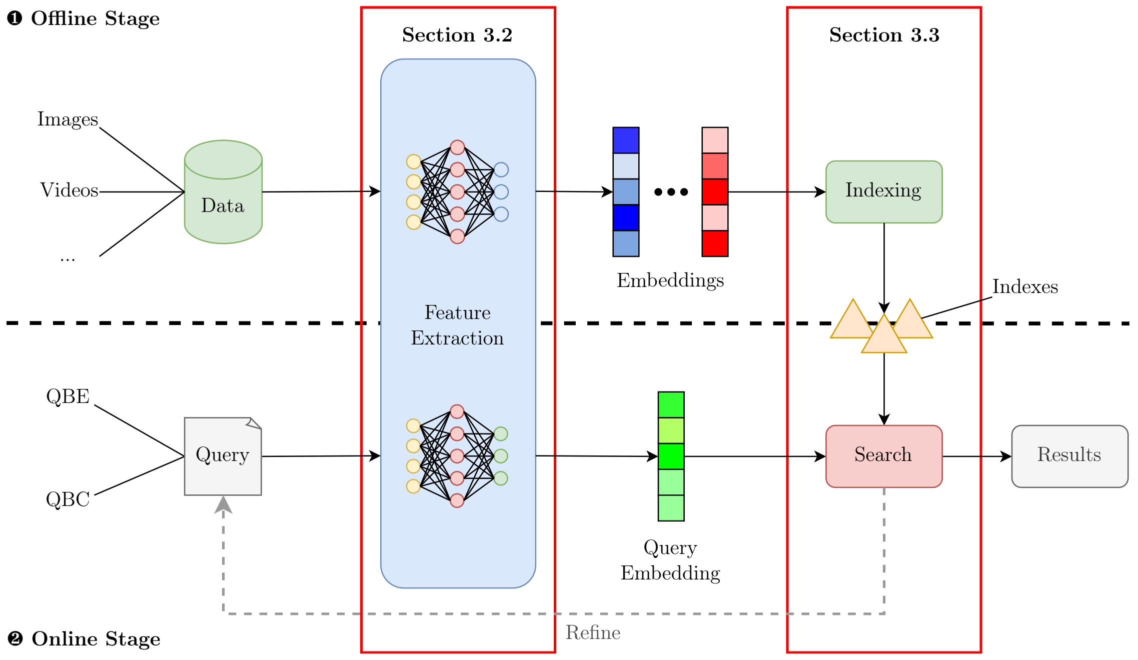


Figure 3.1: Workflow of a CBR system. The gray, dashed arrows indicate optional steps. Section 3.2 will delve into methods for extracting the embeddings through the use of deep neural networks. Section 3.3 will cover the efficient search using index structures.

to transform the data into vector representations. In CBR systems, which predominantly handle complex multimedia data such as images, this process involves transforming raw data into compact embeddings. During the feature extraction phase, the relevant semantics of the data are extracted into compact representations typically using DNNs. Further post-processing steps within the embedding phase can be the fusion of multiple representations into a single, more informative embedding. Alternatively, to further reduce the storage requirements of the retrieval system, the extracted embeddings can be compressed through dimensionality reduction, quantization, or aggregation techniques. In the next step, the index structures are constructed that represent efficient data structures for rapidly searching and retrieving relevant instances from extensive data catalogs.

Online Stage

After the index structures have been built, users can start their searches by formulating queries [134]. Depending on the underlying CBR system, it can accept different types of queries such as QBE or QBC. Each query is converted into an embedding that matches the format used during the feature extraction phase for the data catalog instances. This query embedding is then utilized to find matches by comparing its similarity with the embeddings of the catalog instances. The system efficiently retrieves the most relevant matches, leveraging the pre-built index structures to rank and order them based on their similarity scores. In addition, the initial query results may be refined in order to better align with the user's intentions. Finally, the results are returned to the user.

The following sections will provide a more detailed examination of the two primary components:

(1) **Feature extraction:** The extraction of feature embeddings from raw data is discussed in Section 3.2. This section will particularly focus on feature extraction techniques that leverage DNNs, which are known as deep feature extraction.

(2) **Efficient search**: The search of similar instances within the data catalog for a query is described in Section 3.3. The focus of this study is on the development of efficient retrieval methods that can be applied in large-scale settings, leveraging index structures.

3.2 From Raw Data to Embeddings

Deep feature extraction is a powerful machine learning technique that uses DNNs to transform complex data like images and videos, into compact vector representations called embeddings [38]. These embeddings are crucial for a wide range of applications, including CBR, classification, segmentation, and more, and serve as intermediate representations for further downstream tasks [55, 65, 91]. Before exploring the various deep feature extraction methods, we will introduce the concept and utility of embeddings.

3.2.1 Embeddings

Embeddings are learned by a function $f : X \rightarrow \mathbb{R}^D$. This function translates an input (media) element $x \in X$ into a point in a D-dimensional real vector space. The primary goal of the function $f(x)$ is to encapsulate the semantic meaning (e.g. the visual content of an image) of x in a D-dimensional vector [19]. This vector representation is significantly smaller in dimensionality than the original input x, but should retain the essential attributes necessary for relevant tasks.

The function $f(x)$ is typically embodied by a DNN, often called a *backbone*, that has been pre-trained on specific learning tasks, thereby generating embeddings that spatially encode the semantics of the input data. How these DNNs are pre-trained is discussed in Section 3.2.2. Effective pre-training results in embeddings where vectors of similar objects (e.g. images of monkeys) are located closer together in the vector space, while vectors of dissimilar objects are further apart. This spatial arrangement ensures that the relationships and similarities between the original objects are preserved and represented in the geometry of the vector space. Consequently, the concept of distance in this vector space becomes a practical tool for implementing similarity searches among media items.

Similarity Measures

The similarity between two embeddings $\mathbf{a} = f(x_1)$ and $\mathbf{b} = f(x_2)$ can be measured using various distance measures $\mathcal{D}$ or similarity measures $\mathcal{S}$, both represented by a function $\mathcal{D}, \mathcal{S} : \mathbb{R}^D \times \mathbb{R}^D \rightarrow \mathbb{R}$. Typically, a higher value of $\mathcal{D}(\mathbf{a}, \mathbf{b})$ indicates a lower similarity, while a higher $\mathcal{S}(\mathbf{a}, \mathbf{b})$ corresponds to a higher similarity and vice versa.

- **Euclidean distance** measures the straight-line distance between two points in the embedding space. It ranges from 0 to ∞, where a distance of 0 corresponds to the highest similarity. It is defined as:

$$\mathcal{D}_{euclidean}(\mathbf{a}, \mathbf{b}) = \sqrt{\sum_{i=1}^{D} (\mathbf{a}_i - \mathbf{b}_i)^2}. \tag{3.1}$$

- **Cosine similarity** measures the cosine of the angle between two real-valued vectors $\mathbf{a}$ and $\mathbf{b}$ with values from -1 to 1. A value of 1 indicates the highest similarity. It is defined as:

$$\mathcal{S}_{cosine}(\mathbf{a}, \mathbf{b}) = \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\| \; \|\mathbf{b}\|}, \tag{3.2}$$

  where $\mathbf{a} \cdot \mathbf{b}$ is the dot product between $\mathbf{a}$ and $\mathbf{b}$ and $\|\mathbf{a}\|$ is the L_2-norm of $\mathbf{a}$.

Curse of Dimensionality

When dealing with embeddings in high-dimensional spaces, we encounter specific challenges that are summarized under the term *curse of dimensionality* [19]. In general, this term encompasses several phenomena that arise in the context of high-dimensional data. A key issue is the exponential growth of the volume of the vector space with increasing dimension, leading to data sparsity. In such environments, the relative distance between data points tends to homogenize; that is, differences in distance between points become less pronounced. This complicates the task of efficiently identifying similar points, as the distinction between the closest and farthest points diminishes. As a result, these conditions can significantly slow down query processing, requiring specialized techniques to effectively manage the complexity of high-dimensional data.

3.2.2 Deep Feature Extraction Methods

Extensive research has been conducted on deep feature extraction in the context of CBR [28], with a multitude of approaches developed to extract meaningful embeddings from input data. Commonly, DNNs are utilized that are pre-trained on large datasets, enabling them to capture the essential characteristics of the data and generate informative embedding vectors. Due to the vast number of existing approaches, we will provide a concise overview of those in the image domain, as this is the most extensively researched area for CBR. Although the presented models originate from the field of computer vision, the general concepts of feature extraction and learning paradigms are transferable to other media types. It should be noted that the presented approaches are by no means meant to be complete and should only give an overview of the development of deep feature extraction and some well-known foundation models.

In Figure 3.2 the workflow of deep feature extraction is demonstrated [132], illustrating how DNNs generate embedding vectors from input data. To improve the quality of the extracted embeddings, the DNNs are typically pre-trained on related, large datasets. Alternatively, fine-tuning with task-specific data can further improve performance. Once extracted, embeddings can also undergo post-processing routines before being utilized in subsequent retrieval stages.

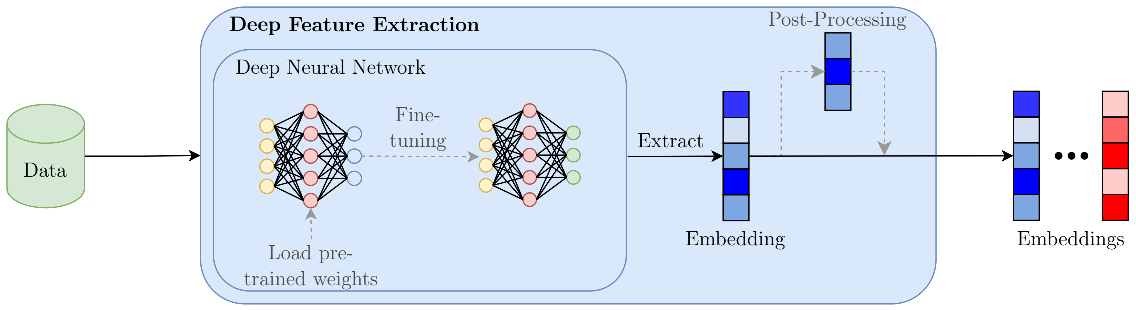


Figure 3.2: Workflow of deep feature extraction. The input data is transformed into embedding vectors utilizing deep neural networks. The gray, dashed arrows indicate optional procedures that may be employed during the feature extraction process.

Generalizable Embeddings

The objective of deep feature extraction is to generate embeddings that not only encapsulate essential information for specific tasks, but also exhibit high generalizability across diverse applications and data content [106]. This is particularly important for the task of CBR, where embeddings are often designed to serve a vast and heterogeneous data catalog. Therefore, ideally, users are interested in highly generalizable embeddings that capture the inherent characteristics of the data, ensuring that the embedding can be applied to a wide variety of use cases.

To achieve this, several strategies are employed [113]. One key approach is increasing the training dataset by introducing a broader array of data, enabling the model to encounter and adapt to a wider spectrum of variations. This helps in developing more robust and universally applicable embeddings. Another important strategy is the expansion of the model architecture through the implementation of larger and more complex models. This enhances their capacity to learn complex relationships and subtle nuances within the data, thereby improving their generalizability.

Transfer Learning

As discussed above, generalizable embeddings require large amounts of data and large models to achieve state-of-the-art performance [135]. However, merely scaling up the model or dataset size presents practical limitations for many users due to several factors. The primary constraint is the computational burden, as training large models on extensive datasets requires substantial computational resources. Furthermore, data acquisition itself presents another hurdle. Manually labeling, collecting, or pre-processing vast data volumes can be time-consuming, expensive, and sometimes infeasible. Fortunately, *transfer learning* provides a powerful approach by leveraging pre-trained models. At its core, it involves training large neural networks on vast, general-purpose datasets. Models such as *DINO* [23, 92], *CLIP* [98], and *BYOL* [51], developed by leading tech companies such as Meta, Microsoft, and Google, are examples of this strategy. These models have been trained on datasets containing millions of objects, utilizing extensive GPU resources in large data centers. The outcome is a set of highly versatile embeddings capable of recognizing fundamental aspects of images without being limited to specific tasks. Remarkably, these embeddings can then be applied to various downstream tasks, even those for which they were not originally trained. This is possible because many of these pre-trained models and their weights are publicly available.

Model Overview

In the early days, the pre-trained models were primarily based on models trained through supervised learning tasks. Prominent examples include the *ResNet* [55] and *VGGNet* [109] models that were trained on the *ImageNet* (ILSVRC2012) [102] classification dataset, which contains more than 1.2 million images of 1000 classes. Due to the considerable size and diversity of the classes, it represented the state-of-the-art in the field of general-purpose embeddings for a wide range of downstream tasks. It is still used today as a benchmark dataset to assess and compare the generalizability capabilities of new models. However, the objective of generalizability contradicts the degree of supervision of the learning task, as the supervision guides the learned embeddings towards a specific direction (the direction of the classification task), potentially sacrificing their generalizability across other downstream tasks [23]. This presents a significant challenge: how can we leverage pre-labeled data to learn generalizable features without restricting the embeddings to the dataset's specific categories? For instance, consider an image dataset labeled for animal classification. The learned embeddings may become adept at distinguishing various animal types, neglecting other relevant information, such as the surrounding environment in which the animal is located.

Consequently, over time, approaches have increasingly employed techniques that require less supervision to learn the inherent visual concepts of the images [23]. In the initial attempts to loosen the supervision, semi-supervised learning approaches were employed that, in addition to a small labeled dataset, utilized large catalogs of unlabeled images. The incorporation of more than a billion unlabeled images into the training process enabled semi-supervised ResNet models to surpass fully-supervised approaches on ImageNet, as demonstrated by Yalniz et al. [127]. The advent of SimCLR [27] paved the way for the development of self-supervised foundation models that did not necessitate the use of labeled data. A plethora of sophisticated models have been published in this domain over the past few years, which either employed a contrastive learning approach [23, 29] or a generative learning approach using autoencoders [54]. A notable advancement in contrastive learning is the use of multiple modalities to learn generalizable representations. In these cases, the data is no longer just represented by a single modality, such as an image, but also by a textual or audio description. The contrastive learning objective is achieved by using separate encoders for each modality. This approach pulls similar objects closer to each other in the joint embedding space and pushes dissimilar objects further apart. The well-known multi-modal model CLIP [98] employs textual descriptions for corresponding images to guide the representation learning towards general visual concepts defined by the text. Describing the visual concepts by text is expected to create embeddings that contain more abstract visual concepts that are better transferable between various datasets.

A list of selected deep feature extraction models mentioned throughout this section is provided in Table 3.1. Here, the models are compared based on their classification performance on the ImageNet dataset. The scores were taken from the reported values in the papers and should be viewed as approximate estimates. Over time, the

Model	Year	Learning Paradigm	FNN Architecture	Top 1 Acc.
VGGNet [109]	2015	Supervised	CNN	0.715
ResNet [55]	2015	Supervised	CNN	0.753
Yalniz et al. [127]	2019	Semi-supervised	CNN	0.767
SimCLR [27]	2020	Contrastive SSL	CNN	0.765
BYOL [51]	2020	Contrastive SSL	CNN	0.796
DINO [23]	2021	Contrastive SSL	CNN, Transformer	0.828
CLIP [98]	2021	Contrastive SSL	CNN, Transformer	0.839
MAE [54]	2022	Generative SSL	Transformer	0.849

Table 3.1: Overview of selected deep feature extraction models for image data. Each model is shown with its year of publication, the underlying learning paradigm according to Section 2.2, the FNN architecture, and the top 1 classification accuracy scores on the ImageNet validation dataset.

models developed more towards the self-supervised learning regime while adopting a more powerful FNN architecture, moving from CNNs to transformer models that resulted in stronger classification performance.

Fine-Tuning

Should the accuracy of the results remain insufficient for a specific retrieval setting, finetuning represents an additional stage where pre-trained models are adapted to specific tasks [28]. Think of a pre-trained model as a master chef with a vast knowledge of various cooking techniques and ingredients. While they may not be experts in every cuisine, they possess a strong foundation to adapt to specific dishes. Similarly, the pre-trained model has learned basic visual concepts that can be adapted to various tasks. In contrast to pre-training, fine-tuning does not necessitate the use of voluminous datasets. Instead, it focuses on making subtle yet impactful adjustments to the model's weights for the specific task at hand. The learning paradigm employed for fine-tuning may vary depending on the specific case. A supervised learning approach is commonly utilized, wherein the model is presented with a reduced classification dataset of the target domain. The fine-tuning process often concentrates on the model's final layers, applying various strategies. These include:

- **Adapting learning rates:** By implementing varying learning rates for different layers or weights, with the highest rates applied to the final layers, the adaptation of the model to the new task is emphasized [99].
- **Freezing layers:** Selectively freezing the weights of certain layers prevents adjustments to their learned features, thereby preserving the general capabilities of the model while focusing training on task-specific adaptations [76].
- **Adding layers:** An alternative approach is to add layers to the network at the end, with configurations tailored to the specific requirements. This allows the remaining layers to be frozen and only the newly added layers to be trained.

Next to the actual fine-tuning for the specific retrieval task, the new re-training of the network also allows for adding some additional auxiliary objectives to the learning process, beyond the primary goal of solving the task at hand. This may entail reducing the dimensionality of the embeddings D, for instance, by reducing the number of neurons in the final layer or enforcing certain geometric properties of the final embeddings, as demonstrated by Sablayrolles et al. [103].

Post-Processing

If the optimization of the embeddings cannot be integrated into the fine-tuning phase, either due to the absence of fine-tuning or because the optimization goal cannot be described by a loss (non-differentiable), it may be addressed in a post-processing stage [28]. In this phase, the initially derived embeddings undergo calibration. The objectives of the post-processing phase include enhancing retrieval accuracy, minimizing the memory requirements of the embeddings, and adapting the embeddings to suit specific indexing structures. This may involve the following methods:

- **Feature fusion**: Multiple extracted embeddings from the same instances, which highlight different aspects of the input, are merged into a joint, more powerful embedding through techniques such as pooling or learned methods [124, 126].
- **Transforming data distribution:** By employing learned-based approaches like *OASIS* [25], the locality in vector space of semantically similar instances can be enhanced.
- **Compression**: By leveraging strategies, such as quantization or dimensionality reduction, the size of embeddings can be reduced. This can be achieved by either storing the individual embedding values in a more compact manner, using fewer bits to represent a single value (quantization) [60], or by reducing the overall dimensionality of the embedding D using dimensionality reduction techniques such as PCA [128].

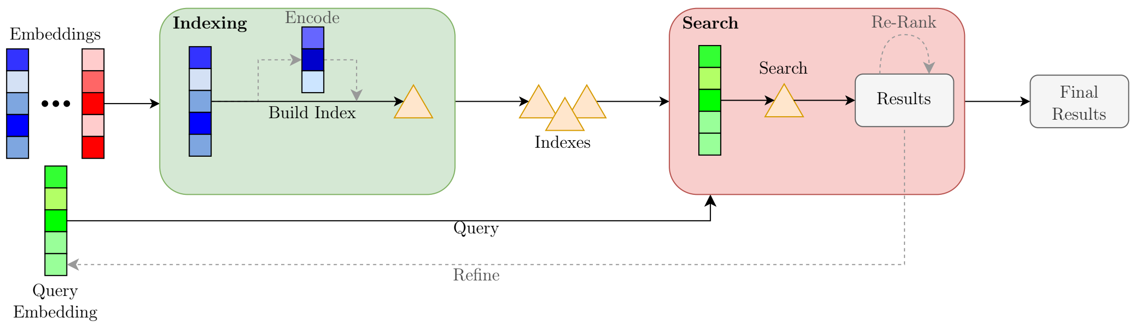


Figure 3.3: Workflow of indexing and search phase. During the indexing phase, the embeddings are used directly to construct the indexes or transformed into more compact representations like hash codes. During the search phase, a search is performed based on the query supported by the pre-built indexes. The retrieved results are either returned directly or some search refinement (either re-ranking or query refinement) is done before. The dashed gray lines indicate optional steps.

3.3 Index Structures for Efficient Search

Once the embeddings have been extracted and post-processed, they are utilized to setup the actual search functionality, which is shown in more detail in Figure 3.3. To speed up the search, index structures [85], or simply indexes, are leveraged in practice. These data structures are designed to organize information in a manner that makes it easier to perform searches and retrieve data quickly, albeit at the cost of pre-processing time and storage. The type of index determines whether the index structures are constructed directly on the embeddings or transformed into more compact representations before the indexes are created [72]. The actual search is then performed using the pre-built index structures. Depending on the accuracy of the results, optional re-ranking steps of the initially retrieved results can be done before returning the result set to the user.

Computational Complexity: The performance of the index structures can be evaluated based on their algorithmic complexity with respect to both time and space, which is articulated through the *Big* $\mathcal{O}$-notation [110]. As a baseline for comparing the runtimes of index structures, one commonly refers to the sequential scan, or "brute force", where each instance of the dataset is compared to the query one after another. The time complexity of this approach is denoted with $\mathcal{O}(N)$, where N resembles the size of the dataset. This time complexity shows a linear relationship w.r.t. the size of the dataset. As the data volume increases, this becomes increasingly inefficient, as illustrated in Figure 3.4. In contrast, advanced index structures are designed to offer sub-linear time complexity, significantly reducing the time required for retrieval. Ideally, these accelerated retrieval capabilities should come with linear storage scaling $\mathcal{O}(N)$, ensuring minimal additional overhead beyond storing the data itself [72].

Nevertheless, not all methods have been subjected to rigorous theoretical analysis, and the boundaries of their complexity may not always be clearly defined. This highlights the importance of benchmarks that assess the presented algorithms in real-world scenarios with large-scale datasets [8].

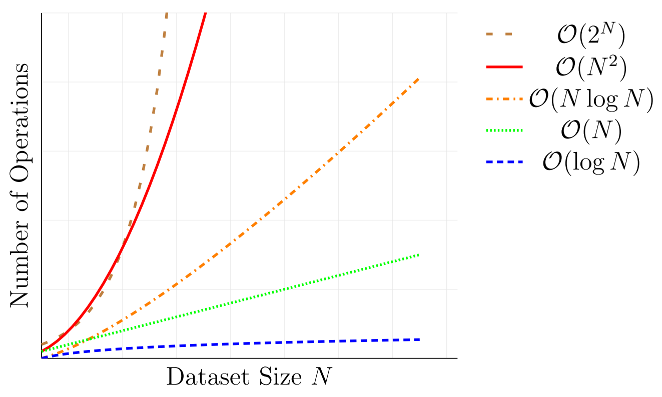


Figure 3.4: A comparison of different time complexity classes concerning the dataset size and the required number of operations.

Search in CBR: In CBR queries, the common task is to retrieve the most similar objects within the dataset [85]. This requirement necessitates the use of index structures specifically optimized for similarity search. While other query types exist, such as range search that finds objects within a defined region (see Section 3.3.2), research in CBR primarily focuses on designing index structures for similarity searches, as presented in Section 3.3.1.

A distinct challenge that arises in CBR is the high dimensionality of the data being processed since the data is commonly represented by high-dimensional embedding vectors [19]. This poses additional challenges to the index structures, as with increasing dimensionality, the performance of traditional index structures typically used in the context of databases strongly degenerates due to the curse of dimensionality. To overcome the limitations imposed by the curse of dimensionality, index structures employed in CBR frequently employ approximation techniques to identify the most analogous instances, albeit at the expense of diminished accuracy.

3.3.1 Similarity Search

As the majority of queries in CBR rely on the concept of similarity, it is necessary to have efficient algorithms to identify the most similar matches within large catalogs of high-dimensional embeddings. This problem is widely studied and is often referred to as the *nearest neighbor* (NN) problem, which aims at identifying the closest point given a query point within a dataset based on a distance measure (see Section 3.2.1) [89]:

Definition 1. *Let* $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\} \subset \mathcal{M}$ *be a set of* N *points in metric space* $\mathcal{M}$ *and* $\mathbf{q} \in \mathcal{M}$ *be a query point, NN search is about finding the point* $NN(\mathbf{q}, X) \in X$*, named nearest neighbor, that is the closest to* $\mathbf{q}$ *concerning a distance metric* $\mathcal{D}$*, that is,* $NN(\mathbf{q}, X) = \operatorname{argmin}_{\mathbf{x} \in X} \mathcal{D}(\mathbf{q}, \mathbf{x})$.

The interest often extends beyond identifying the single nearest neighbor to finding multiple close points within X. This introduces variations such as the k nearest neighbor search (kNN), which finds the top k closest neighbors by distance:

$$k\text{NN}(\mathbf{q}, X, k) = A, \tag{3.3}$$

where A is a set that fulfills the following conditions:

$$|A| = k, A \subseteq X, \quad \text{and} \tag{3.4}$$

$$\forall \mathbf{x} \in A, \mathbf{y} \in X \setminus A, \mathcal{D}(\mathbf{q}, \mathbf{x}) \leq \mathcal{D}(\mathbf{q}, \mathbf{y}). \tag{3.5}$$

Alternatively, in radius nearest neighbor search (RNN), all points within a specified distance threshold r from the query point are retrieved:

$$\text{RNN}(\mathbf{q}, X, r) = \{\mathbf{x} \in X \mid \mathcal{D}(\mathbf{q}, \mathbf{x}) < r\}. \tag{3.6}$$

Approximate Nearest Neighbor Search

The following discussion primarily focuses on NN search in D-dimensional Euclidean space where $\mathcal{M} = \mathbb{R}^D$. While NN search is well-solved for low-dimensional D [39], for high-dimensional D, problems arise that are known under the term curse of dimensionality, which makes exact NN methods impractical (see 3.2.1). To address this issue, researchers have developed approximate formulations of the NN problem that perform well even in high dimensions [3]. These are summarized under the term *approximate nearest neighbor* (ANN) problem. There are several formal definitions of the ANN problem, categorized based on the type of approximation offered [89]. These include error-bound approximations, such as c-approximate nearest neighbors, (c, r)-approximate nearest neighbors) and time-bound approximations, where the search is bound within a predefined time limit. Despite yielding sub-optimal results, these algorithms are notably faster, making them practical for higher-dimensional scenarios. In the widely utilized c-approximate nearest neighbors formulation, the ANN search is defined as:

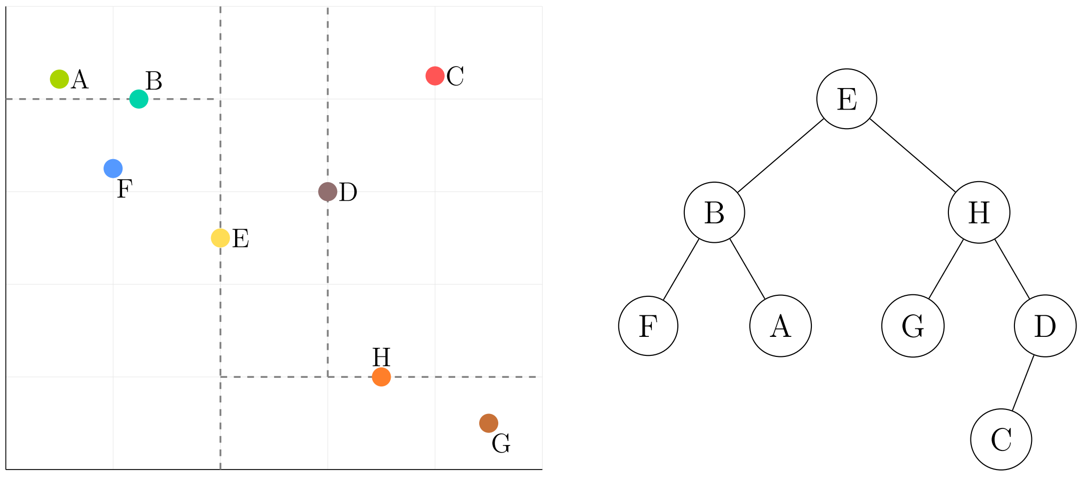
Definition 2. *Let $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\} \subset \mathcal{M}$ be a set of N points in metric space $\mathcal{M}$ and $\mathbf{q} \in \mathcal{M}$ be a query point, the c-ANN problem is about finding any point $ANN(\mathbf{q}, X, c) \in X$ whose distance is at most $c \cdot \mathcal{D}(\mathbf{q}, \mathbf{p}^*)$ for some approximation factor $c \geq 1$, where $\mathbf{p}^* = NN(\mathbf{q}, X)$.*

Similar to the exact NN search, ANN can be extended to find the top k closest approximate neighbors (kANN). Throughout the following discussion of ANN algorithms, the terms ANN and kANN will be used interchangeably to refer to the general concept of finding sub-optimal nearest neighbors with significantly faster search speeds.

ANN Algorithms

In contrast to the use of brute force sequential scans to identify the ANN, over the past decades, more efficient algorithms have been proposed that can be solved in sub-linear time. These algorithms serve as the foundation for the utilized index structures in CBR. As stated by Li et al.[72], the majority of algorithms can be categorized into three classes based on their underlying data structure:

- **Partition-Based:** These structures decompose the feature space into subspaces to perform the search and are best represented by a tree or forest. Examples of such approaches include axis-aligned partitioning techniques such as *k-d trees* [15], pivoting methods like *ball trees* [24], and compact partitioning schemes like *cover trees* [17].



(a) Partitioning schema **(b)** Tree structure

Figure 3.5: Space partitioning of a 2D-space using k-d tree.

- **Hash-Based:** These structures map data elements to compact codes (hashes), allowing for fast lookup. Two common classes differ based on their used hash functions: Data-independent approaches such as *locality sensitive hashing* (LSH) [57] and data-dependent approaches like *learning to hash* (L2H) [120].
- **Graph-Based:** These structures employ graph representations to model proximity. Examples include *hierarchical navigable small world* (HNSW) [84], which is regarded as one of the most efficient ANN algorithms to date [38].

While a general overview of all types of index structures will be provided, this thesis will focus on selected approaches within each category that have been widely researched and applied.

Partition-Based Search

Partition-based algorithms, as the name suggests, involve decomposing the data space into smaller subspaces, primarily using hierarchical data structures such as trees. They operate on the premise of *branch and bound*, where branches of the search tree are systematically explored, and bounds are used to eliminate paths of the tree that do not contain the nearest neighbor.

k-d Tree: A well-known algorithm based on this technique is a k-d tree [15]. A k-d tree is a binary tree that recursively splits the space into two partitions at each non-leaf node. The non-leaf nodes can be thought of as separating hyperplanes that are perpendicular to the axes chosen for the split and which split the points within this current partition into the two subspaces, as visualized in Figure 3.5. There are different ways in which the split and also the split axis are determined for the construction of k-d trees. We will refer to the most commonly used variant, which employs median-based splitting and alternating axis selection.

Construction: Let $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\} \subset \mathbb{R}^D$ be set of N points with $\mathbf{x}_i = \left[\mathbf{x}_i^{(1)}, \ldots, \mathbf{x}_i^{(D)}\right]^\top$ for $i \in \{1, \ldots, N\}$. A k-d tree for X is built recursively, as shown

Algorithm 3 Constructing a k-d tree

```
Require: Set $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\} \subset \mathbb{R}^D$, tree depth $t \in \mathbb{N}$
Ensure: $k$-d tree $\mathcal{T}$ build for $X$
 1: function BuildKdTree($X, t$)
 2:     if $|X| \leq 1$ then
 3:         return node containing $X$
 4:     $\phi \leftarrow (t \mod D) + 1$
 5:     $\mathbf{p} \leftarrow$ median point according to dimension $\phi$
 6:     $X_L \leftarrow \{\mathbf{x}_i \in X \mid \mathbf{x}_i^{(\phi)} \leq \mathbf{p}^{(\phi)}, \mathbf{x}_i \neq \mathbf{p}\}$
 7:     $X_R \leftarrow X \setminus (X_L \cup \{\mathbf{p}\})$
 8:     $H \leftarrow \{\mathbf{x} \in \mathbb{R}^D \mid \mathbf{x}^{(\phi)} = \mathbf{p}^{(\phi)}\}$
 9:     $\mathcal{T}_L \leftarrow$ BuildKdTree($X_L, t+1$)
10:     $\mathcal{T}_R \leftarrow$ BuildKdTree($X_R, t+1$)
11:     Add node storing $H$ and pointers to $\mathcal{T}_L$ and $\mathcal{T}_R$ to the tree $\mathcal{T}$
12:     return $\mathcal{T}$
```

in Algorithm 3. At each level of the tree, a dimension is selected for splitting the data. The dimension chosen for splitting ϕ is determined by an alternating schema based on the current depth of the tree, as given by:

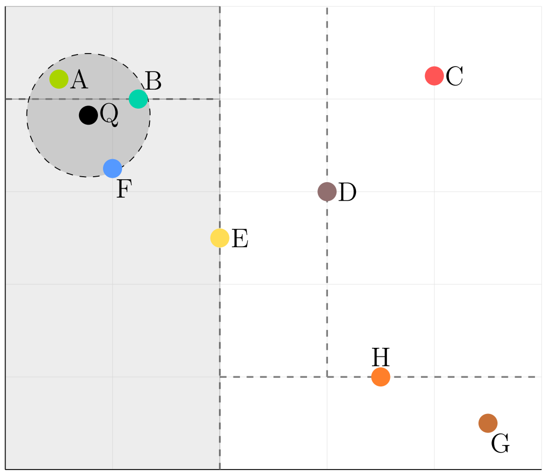
$$\phi = (t \mod D) + 1, \tag{3.7}$$

where $t \in \mathbb{N}$ corresponds to the depth of the tree, starting from zero at the root. To find the splitting point, the points X are sorted by their value in the ϕ-th dimension.[1] Let the sorted points be indexed as $\mathbf{x}_{i_1}, \ldots, \mathbf{x}_{i_N}$ such that $\mathbf{x}_{i_1}^{(\phi)} \leq \ldots \leq \mathbf{x}_{i_N}^{(\phi)}$. The median of the points in the ϕ-th dimension is selected as $\mathbf{p}$, which is $\mathbf{p} = \mathbf{x}_{i_{\lceil N/2 \rceil}}$.

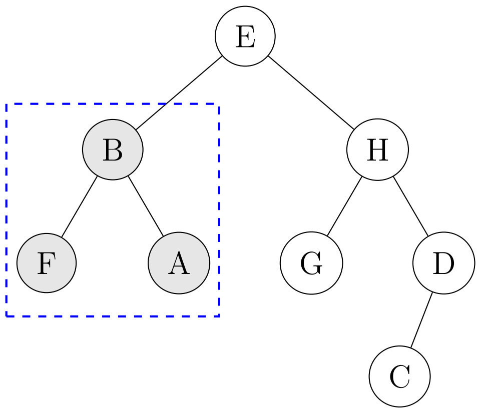
A hyperplane $H = \{\mathbf{x} \in \mathbb{R}^D \mid \mathbf{x}^{(\phi)} = \mathbf{p}^{(\phi)}\}$ is then introduced. The hyperplane splits the set X into two subsets $X_L = \{\mathbf{x}_{i_1}, \ldots, \mathbf{x}_{i_{\lceil N/2 \rceil - 1}}\}$ and $X_R = \{\mathbf{x}_{i_{\lceil N/2 \rceil + 1}}, \ldots, \mathbf{x}_{i_N}\}$. This median-based splitting strategy ensures that the tree remains balanced, with the left child potentially containing at most one more point than the right child. Finally, the node stores the splitting point $\mathbf{p}$ and the splitting hyperplane H and continues with the two child nodes $\mathcal{T}_L$ and $\mathcal{T}_R$ until the size of the point set is less than or equal to 1.

NN Search: Once the tree has been constructed for the point set X, it is possible to initiate NN queries for any point $\mathbf{q} \in \mathbb{R}^D$. The search process starts by traversing the tree from the root to a leaf node. This navigation involves choosing between left or right child nodes based on whether the value of $\mathbf{q}$ is less than or equal to, or greater than the node's value in the splitting dimension. Upon reaching the leaf node, the distance between the leaf and the query point is calculated. This, however, does not conclude the NN search, since the true nearest neighbor can also be located in a neighboring partition, as illustrated in Figure 3.6. Consequently, the recursion is unwound through a backtracking procedure. The backtracking process begins, wherein the algorithm revisits nodes along the path from the leaf node to the root. At each visited node, the algorithm checks whether the hypersphere centered

[1]More efficient methods for finding the median in linear time exist, such as *median-of-medians* [18].



(a) Partitioning schema



(b) Tree structure

Figure 3.6: NN search with k-d tree for query point Q. The blue dashed rectangle indicates which nodes are visited during the backtracking procedure.

at $\mathbf{q}$ with a radius equal to the distance to the current nearest neighbor intersects the splitting plane of the node. If an intersection occurs, the algorithm explores the subtree on the opposite side of the node's splitting plane, which was not visited initially, to find potentially closer neighbors. If the hypersphere does not intersect the splitting plane, the algorithm continues walking up the tree. When the algorithm finishes this process for the root node, the search is complete.

On average, a single nearest neighbor query can be executed in $\mathcal{O}(\log N)$ [42]. However, the backtracking mechanism might result in a worst-case runtime of $\mathcal{O}(N)$ when all nodes must be inspected. Employing k-d trees as an index also comes at the cost of construction time and further space consumption. A k-d tree can be built in $\mathcal{O}(N \log N)$ time using linear-time median finding and occupies $\mathcal{O}(N)$ space.

Approximation Techniques: The performance of k-d trees strongly degrades with increasing dimensionality D due to the exponentially growing number of nodes during the backtracking procedure (due to curse of dimensionality, relative distances between partitions homogenize). That is why approximation techniques have been developed for the k-d tree that enable satisfactory performance in high-dimensional space. Arya et al. [7] proposed an error-bound technique for identifying the c-approximate neighbors, while also time-bound techniques exist, where the search is stopped early after inspecting a fixed number of leaves following an optimized order of leaves during backtracking in a *best-bin-first*-strategy [14].

Hash-Based Search

Hash-based algorithms transform data items into low-dimensional representations. These compact codes are sequences of bits termed hash codes, either in binary or integer form [120]. These hash codes are generated by a hash function $h : \mathbb{R}^D \rightarrow \{0, \ldots, \Omega - 1\}$, where Ω denotes the maximum number of distinct hash codes that can be produced. Formally, for a data point $\mathbf{x}$, the hash code z is defined as $z = h(\mathbf{x})$. In general, multiple hash functions $h_1, \ldots, h_\Lambda$ are employed in the calculation of the compound hash code $\mathbf{z} = \boldsymbol{h}(\mathbf{x})$, where $\mathbf{z} = [z_1, z_2, \ldots, z_\Lambda]^\top$ and

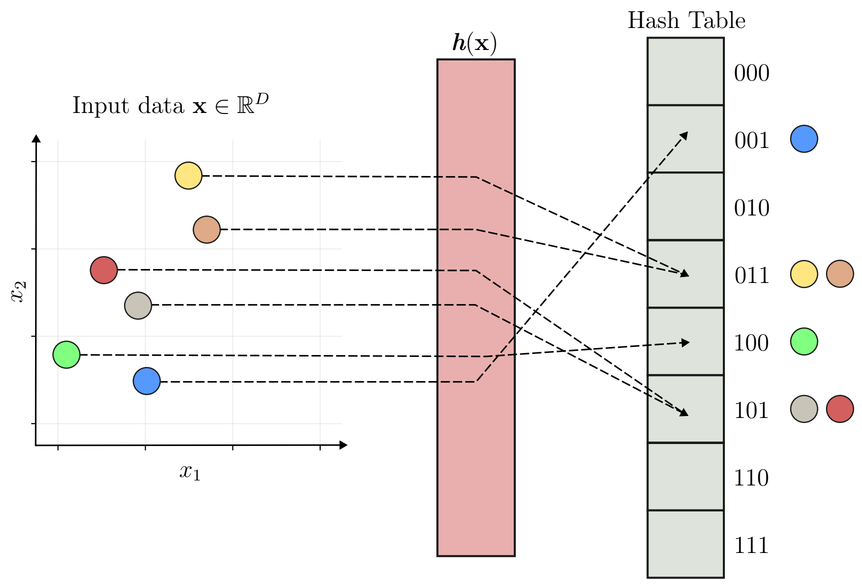

Input data $\mathbf{x} \in \mathbb{R}^D$
x_1
x_2
$\boldsymbol{h}(\mathbf{x})$
Hash Table
000
001
010
011
100
101
110
111

Figure 3.7: The general workflow for hash-based methods involves the encoding of data into compact hash codes using hash functions. This encoded data is then stored in a hash table index structure to facilitate fast lookup. In a straightforward scenario, only the points within the same bucket as the query point, corresponding to identical hash codes, are inspected for ANN search. Figure adapted from Wang et al. [120].

$\boldsymbol{h}(\mathbf{x}) = [h_1(\mathbf{x}), h_2(\mathbf{x}), \ldots, h_\Lambda(\mathbf{x})]^\top$, with Λ being the total number of hash functions used.

Unlike cryptographic hash functions, which prevent different inputs from producing the same output (so-called *hash collisions*), these hash functions are designed to group similar inputs by creating the same output for them [69]. Given a query point $\mathbf{q}$ and a set of points X, the assumption is that the k nearest neighbors of $\mathbf{q}$ in X based on the hash codes will closely resemble the nearest points for the original points. In other words, the hash codes preserve the locality of the original data.

To achieve efficient retrieval with hash-based approaches, data structures such as hash tables [120] are commonly employed, as shown in Figure 3.7. These data structures group the input data into buckets according to their hash code. For a query point $\mathbf{q}$, only those points that are located in the same bucket are retrieved. These are commonly referred to as *candidate pairs*. As a result, the number of distance computations is significantly reduced compared to a full scan. To further increase the recall, multiple buckets can be visited or multiple hash tables can be employed populated by different hash functions. Additionally, to further improve the precision of the results, the retrieved candidates can be re-ranked based on the exact distance from the query. Hash-based methods can be classified into two categories according to the hash functions they utilize [72]. The first category comprises data-dependent methods, which generate hash functions based on the underlying data (e.g. L2H), while the second category includes data-independent methods (e.g. LSH).

Data-Independent: Methods such as LSH employ hash functions that were designed independently of the actual data and were first proposed by Indyk and Motwani [57]. LSH encompasses various methods that use different families of hash

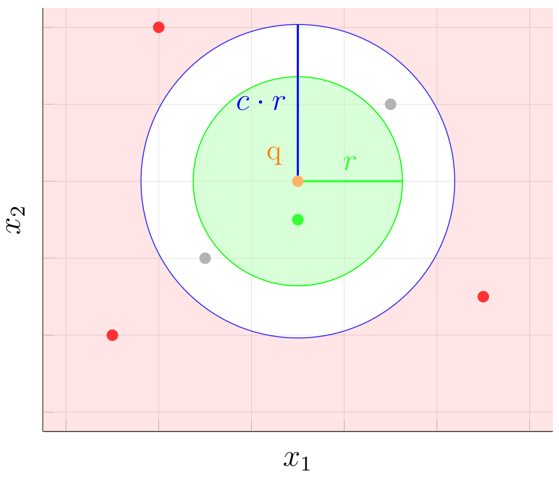


Figure 3.8: The (c, r)-approximate neighbor problem definition. The objective is to find all points that lie within the distance $c \cdot r$ from the query point $\mathbf{q}$, provided that at least one point $\mathbf{p}$ exists with $\mathcal{D}(\mathbf{q}, \mathbf{p}) \leq r$.

functions tailored to specific distance functions and embedding spaces. The design of these hash functions ensures that objects nearby have a significantly higher probability of hash collisions compared to those that are further apart. This notion is formalized in the following definition [57].

Definition 3. *For a metric space $\mathcal{M}$ and a set U, a family $\mathcal{H} = \{h : \mathcal{M} \to U\}$ is called (r_1, r_2, p_1, p_2)-sensitive if the following holds for any $\mathbf{q}, \mathbf{p} \in \mathcal{M}$:*

- *if the distance $\mathcal{D}(\mathbf{q}, \mathbf{p}) \leq r_1$, then the probability $Pr_{\mathcal{H}}[h(\mathbf{q}) = h(\mathbf{p})] \geq p_1$,*
- *if the distance $\mathcal{D}(\mathbf{q}, \mathbf{p}) > r_2$, then the probability $Pr_{\mathcal{H}}[h(\mathbf{q}) = h(\mathbf{p})] \leq p_2$.*

Such a family $\mathcal{H}$ is referred to as an LSH family.

This definition suggests that the probability of two points being hashed to the same value should be at least p_1 when they are close to each other (distance less than r_1), and no more than p_2 when they are considered distant (distance greater than r_2). For an LSH family to be effective, it must fulfill the inequalities $p_1 > p_2$ and $r_1 < r_2$. This setup allows for the solution of the (c, r)-approximate neighbor problem by selecting $r_1 = r$ and $r_2 = c \cdot r$, as depicted in Figure 3.8.

Definition 4. *Given a set X of N points $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\} \subset \mathcal{M}$ in metric space $\mathcal{M}$ and a query point $\mathbf{q} \in \mathcal{M}$, the (c, r)-ANN problem involves identifying any point $ANN(\mathbf{q}, X, c, r) \in X$ whose distance from $\mathbf{q}$ is at most $c \cdot r$, provided there exists a point $\mathbf{p} \in X$ such that $\mathcal{D}(\mathbf{q}, \mathbf{p}) \leq r$. In this context, r is a specified distance threshold and $c \geq 1$ is an approximation factor.*

To amplify the gap between the high probability p_1 and the low probability p_2, multiple hash functions of the family $\mathcal{H}$ are combined to form the compound hash function $\boldsymbol{h}$ [69]. The resulting compound hash function represents a hash function from a new family of hash functions $\mathcal{G} = \{g : \mathcal{M} \to U^{\Lambda}\}$, where Λ is the number of combined hash functions. How these hash functions are combined can result in the creation of various new families of hash functions.[2] We distinguish between *AND*-constructions and *OR*-constructions. In an *AND*-construction, a hash function $g^{\texttt{AND}}$ is formed by combining Λ hash functions from $\mathcal{H}$. For this function, the hash codes for points $\mathbf{q}$ and $\mathbf{p}$ are considered equal, that is, $g^{\texttt{AND}}(\mathbf{q}) = g^{\texttt{AND}}(\mathbf{p})$, only if all corresponding individual hash functions agree, specifically:

$$\forall i\{1, \ldots, \Lambda\} : h_i(\mathbf{q}) = h_i(\mathbf{p}). \tag{3.8}$$

The new hash family $\mathcal{G}$ can be considered as an $(r_1, r_2, (p_1)^\Lambda, (p_2)^\Lambda)$-sensitive family.

In an *OR*-construction where a hash function $g^{\texttt{OR}}$ is constructed by Λ members of $\mathcal{H}$, $g^{\texttt{OR}}(\mathbf{q}) = g^{\texttt{OR}}(\mathbf{p})$ holds true if at least one individual hash function is equal. This is formally expressed as:

$$\exists i\{1, \ldots, \Lambda\} : h_i(\mathbf{q}) = h_i(\mathbf{p}). \tag{3.9}$$

This results in a new hash family that is $(r_1, r_2, 1 - (1 - p_1)^\Lambda, 1 - (1 - p_2)^\Lambda)$-sensitive.

As stated by Leskovec et al. [69], *AND*-constructions result in a reduction of all probabilities, whereas *OR*-constructions have the effect of increasing all probabilities. By cascading multiple *AND*-constructions and *OR*-constructions and judiciously selecting Λ, it is possible to push p_1 close to 1 and p_2 close to 0.

Due to the wide variety of existing LSH approaches for different spaces, the algorithmic complexity varies. In general, it can be said that all approaches provably offer sub-linear query times and sub-quadratic space complexity in high-dimensional spaces [2]. In the case of Euclidean space (on a unit sphere), the optimal provably guaranteed algorithm has a query time of $\mathcal{O}(N^\rho)$ and space complexity of $\mathcal{O}(N^{1+\rho})$, where $\rho = \frac{1}{c^2}$ for an approximation factor $c > 1$ [88]. Despite the existence of known boundaries for space and time complexity, the selection of the optimal LSH family remains challenging due to the discrepancy between theoretical guarantees and practical performance [2]. This discrepancy can be attributed to the data-agnostic nature of the algorithms, which assume that all regions of the data exhibit uniform density (e.g.) to achieve optimal performance. However, such uniformity is rare in real-world applications, leading to inefficient use of resources as the algorithms focus on sparsely populated subspaces [94]. This is the primary motivation for employing data-dependent data structures, as described in the following section.

Data-Dependent: Data-dependent hashing algorithms have been proven to outperform classic data-independent hashing algorithms due to their ability to exploit the underlying structure of the data to construct the hash functions [4]. These methods are often summarized under the term learning to hash, as they learn a hash function $z = h(\mathbf{x})$ that maps an input item $\mathbf{x}$ into a compact code z. The goal is to ensure that the nearest neighbors for a query $\mathbf{q}$ are as close as possible in the coding space compared to the true nearest neighbors in the input space [120]. However, data-dependent algorithms have a significant drawback: they are sensitive to shifts

[2]We assume that the selection of the Λ hash functions from the family $\mathcal{H}$ is performed independently and uniformly at random.

in data and query distributions [1]. Unlike data-independent methods, which guarantee worst-case performance, data-dependent techniques may become less efficient when data distributions shift. When the data distribution changes, the previously optimized mapping may become imbalanced, leading to inefficient searches and necessitating a rebuild of the index.

Typically, the different methods are distinguished based on the optimization objective to preserve similarity, which is: pairwise-similarity persevering, multiwise-similarity persevering, implicitly-similarity persevering and quantization [72]. Given the focus of this thesis, we will exclusively explore quantization-based techniques, as they have become the predominant approach in recent years [38].

Quantization: Recent literature suggests that quantization-based methods offer superior efficiency compared to other learning to hash methods [72]. While quantization has been extensively explored for data compression [50], its application in NN search remained unseen for a long time. Unlike previous hashing methods, which predominantly used binary representations as compact hash codes, quantization-based methods transform the input data into a set of finite values [63]. The following section will focus on the widely used technique of vector quantization [49].

Formally, a vector quantizer is a function h that maps a D-dimensional vector $\mathbf{x} \in \mathbb{R}^D$ to a vector $h(\mathbf{x}) \in \mathcal{C} = \{c_k; k \in \mathcal{I}\}$, where the index set $\mathcal{I}$ is finite $1, \ldots, \Omega$ [63]. The reproduction values c_k are called centroids and the set of reproduction values $\mathcal{C}$ represents the codebook of size Ω. The set of points mapped to a given index i is referred to as a cell and defined as:

$$\mathcal{V}_i = \{\mathbf{x} \in \mathbb{R}^D : h(\mathbf{x}) = c_i\}. \tag{3.10}$$

The Ω cells of a quantizer form a partition of $\mathbb{R}^D$.

In order to obtain centroids that represent the underlying data, they must be learned from the data. In vector quantization, a type of k-means clustering is commonly used to determine the coordinates of the centroids [44]. In the traditional k-means algorithm, the objective is to partition N observations into Ω clusters, with each observation belonging to the cluster with the nearest mean. These cluster centroids are refined over multiple iterations minimizing the within-cluster variance. The cluster centers retrieved by k-means represent the codebook $\mathcal{C}$ and the input data is quantized by assigning it the identifier of the closest cluster centroid:

$$h(\mathbf{x}) = \underset{k \in \{1,\ldots,\Omega\}}{\operatorname{argmin}} \mathcal{D}_{euclidean}(\mathbf{x}, c_k) \tag{3.11}$$

The memory cost of storing the index value without further processing is $\lceil \log_2 \Omega \rceil$ bits [63]. However, the reduced storage resource comes at the cost of lost precision in differentiating single instances. This is quantified with the distortion error E, which is the average distance from all points to their closest centroid:

$$E = \frac{1}{N} \sum_{i=1}^{N} \mathcal{D}_{euclidean}(\mathbf{x}_i, c_{h(\mathbf{x}_i)}) \tag{3.12}$$

For NN search using quantization, a query point $\mathbf{q} \in \mathbb{R}^D$ is first quantized $h(\mathbf{q}) = c_j$. This is followed by the retrieval of all points in the cell $\mathcal{V}_j$ with the same hash code as $\mathbf{q}$ using index structures such as a hash table. An optional re-ranking step based on the original representations of the retrieved points and the query can be performed to enhance the accuracy of the returned approximate nearest neighbors. Typically, increasing the size of the codebook can enhance the accuracy of the returned approximate nearest neighbors by allowing for a more diverse encoding. However, this conflicts with the size of the encoded vector, as the latter grows with each centroid added to the codebook. Especially for high-dimensional data, this can easily result in a significant increase in the codebook size [63]. For instance, to transform a 128-dimensional vector, which is not an uncommon dimensionality in real-world cases, into a 64-bit code, which requires 0.5 bits per feature, $\Omega = 2^{64}$ centroids would be required. This makes it computationally expensive to apply native k-means, as it requires large amounts of data and long training time to learn the quantizer. Moreover, the mere act of storing the vector representations of the individual centroids becomes computationally challenging, as this necessitates the storage of $128 \times \Omega$ floating values.

Product Quantization: *Product quantization* (PQ) [63] represents an efficient solution to this problem. It divides the original vector $\mathbf{x}$ into Λ distinct subvectors $\mathbf{u}_j$, with $1 \leq j \leq \Lambda$ of dimension $D^* = D/\Lambda$, where D is a multiple of Λ. Each subvector is quantized separately using Λ distinct quantizer. The mapping using PQ is defined as follows for input $\mathbf{x}$:

$$\underbrace{\mathbf{x}_1, \ldots, \mathbf{x}_{D^*}}_{\mathbf{u}_1(\mathbf{x})}, \ldots, \underbrace{\mathbf{x}_{D-D^*+1}, \ldots, \mathbf{x}_D}_{\mathbf{u}_\Lambda(\mathbf{x})} \rightarrow h_1(\mathbf{u}_1(\mathbf{x})), \ldots, h_\Lambda(\mathbf{u}_\Lambda(\mathbf{x})), \tag{3.13}$$

where h_j is a vector quantizer operating on D^* dimensions for the j subvector. The compound code for the vector $\mathbf{x}$ is represented as $\boldsymbol{h}(\mathbf{x}) = [h_1(\mathbf{u}_1(\mathbf{x})), \ldots, h_\Lambda(\mathbf{u}_\Lambda(\mathbf{x}))]^\top$. This approach significantly reduces the complexity of the used quantizer by focusing on smaller, manageable subspaces. If the number of centroids is fixed for each dimension with the parameter Ω^*, the total number of centroids for PQ is given by:

$$\Omega = (\Omega^*)^\Lambda. \tag{3.14}$$

This defines the size of the overall codebook $\mathcal{C} = \mathcal{C}_1 \times \ldots \times \mathcal{C}_\Lambda$, which is the Cartesian product of all sub-codebooks. The memory complexity for storing the centroids is given by $\mathcal{O}(\Lambda\Omega^* D^*) = \mathcal{O}(\Omega^{1/\Lambda} D)$, which shows that PQ is also capable of storing codebooks of large Ω, e.g. $\Omega = 2^{64}$ for 64-bit quantization, in memory compared to simple k-means quantization where $\mathcal{O}(\Omega D)$ is the memory complexity. When PQ is coupled with a type of multi-dimensional hash table, so-called inverted multi-index [10], the nearest codewords can be found in $\mathcal{O}(\Lambda\Omega)$ time [44]. This shows that the retrieval does not depend on the dimensionality D and the size of the dataset N, but grows linearly concerning parameter Λ and Ω. This, however, represents a complex trade-off between the codebook size and the accuracy of the retrieved results, which strongly depends on the actual use case and prevalent data properties.

Furthermore, as shown in Figure 3.9, the centroids created with PQ compared to classic k-means might be sub-optimal and occupy space that is not populated by the data. Given that PQ performs optimally when the data exhibits balanced variance across all dimensions, it can create an inefficient encoding for real-world data. To address this issue, advanced PQ approaches [44] have been proposed that contain

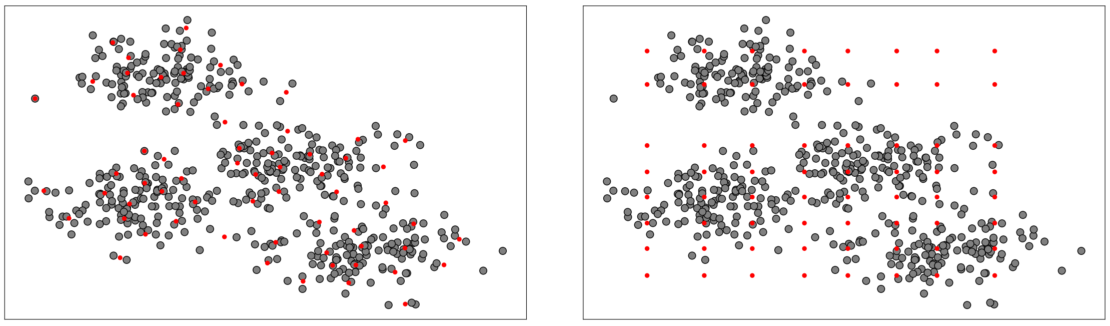

(a) k-means **(b)** PQ

Figure 3.9: A comparison of centroids in simple k-means vector quantization and PQ for $\mathbf{x} \in \mathbb{R}^2$, $\Omega = 64$ and for PQ $D^* = 1$. The red points correspond to the centroids while the gray points represent the underlying data. Despite the fact that the PQ centroids are not optimal, they are considerably more resource-efficient in higher-dimensional cases.

an optimal space decomposition by treating the quantization as an optimization problem minimizing the distortion error (Equation 3.12).

Recent Data-Dependent Approaches: Recent research has focused on improving the mapping from input data to compact codes by building upon the recent advancements of learning-based algorithms. These approaches commonly employ models, such as neural networks, to identify an optimal mapping function [66] or partitioning scheme [36]. Essentially, these models are trained to understand the data's inherent distribution, enabling them to generate a mapping function that is uniquely suited to it. However, a significant limitation of these methods is that they are typically applied to static datasets, where each data point is mapped precisely within the index. As a result, introducing new data necessitates retraining the entire model to update the mapping function and the index structure [123]. Alternatively, neural networks are employed to modify the input data so that existing data-agnostic mapping functions like LSH are working efficiently [103]. This strategy involves modifying the data distribution to optimize the performance of traditional mapping functions, which are most effective when data is uniformly distributed, meaning that points are evenly spread in space. This approach leverages the extensive research and understood theoretical limits of existing mapping functions.

Graph-Based Search

The majority of graph algorithms for ANN that have been studied take the form of greedy routing in kNN graphs [84]. These kNN methods construct a directed proximity graph, where each data point corresponds to a node and edges connecting some nodes define the neighbor relationship. The fundamental principle underlying these algorithms is that a neighbor's neighbor is likely also to be a neighbor [72]. For a given query point, the search is started at a randomly selected node and the graph is iteratively traversed [35]. At each node traversed during the search, the distance between the query point and the adjacent neighbors of the current node is compared. The neighboring node closest to the query is selected for the next step, while the minimum overall distance is stored. The search is stopped when a predefined stopping criterion is met, such as the maximum number of distance

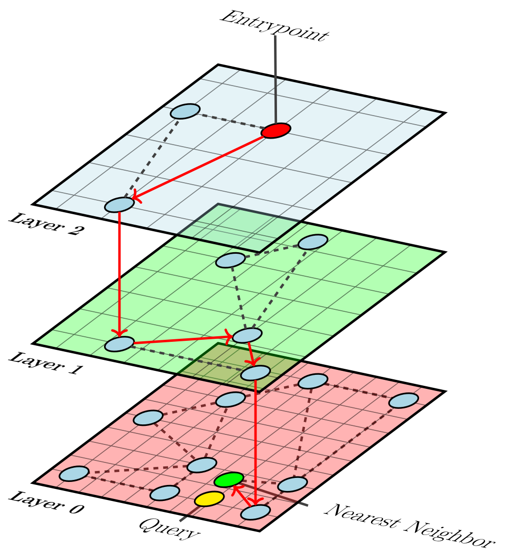


Figure 3.10: Example of NN search using HNSW with three-layered graph structure. The search starts in the top layer, where it searches for the nearest neighbor to the query point Q. Then, it progresses to the next layer, where it refines the search until it reaches the final layer.

calculations. Theoretically, the underlying structure of these graphs, known as the *Delaunay graph* [121], ensures that the greedy traversal will always lead to the true nearest neighbor. However, in practice, the construction of the graph, particularly in high dimensions, becomes complex, which is why it is usually approximated using only the distances between the neighbors. One disadvantage of these graph-based approaches is that they result in the loss of global connectivity. This is because only a limited number of nearest neighbors are connected per node. This can become a significant issue in clustered data, where a substantial number of hops are required to identify the true nearest neighbor.

An alternative approach is the *navigable small world* (NSW) [121] algorithm, which operates on small world graphs to address the loss of global connectivity. The construction of NSW graphs involves the sequential insertion of new elements into the graph in a random order, with each new node connected to the k closest (approximate) neighbors of the previously added point. The long edges formed at the beginning of the construction guarantee global connectivity of the graph (small world), thereby ensuring search efficiency in poly-logarithmic time. The later added nodes form short-range edges, which ensure search accuracy. Nevertheless, due to its poly-logarithmic complexity scaling $\mathcal{O}(\log^2 N)$, the original NSW algorithm is still not performing well for all scenarios, especially for large datasets [121].[3]

A method called HNSW [84] builds upon NSW by introducing a hierarchical structure. The algorithm generates multiple subgraphs (levels) at varying granularities, as shown in Figure 3.10. These layers form a hierarchy, with the topmost layer containing the fewest nodes and each subsequent layer containing progressively more

[3]The poly-logarithmic complexity of a single greedy search in an NSW graph stems from the number of distance computations, which nearly equals the product of the average number of hops taken by the greedy algorithm and the average degree of the nodes encountered along the path.

nodes, culminating in the bottom layer that contains all the data points. The search starts in the upper layer, which only contains the longest links. It then searches for the nearest neighbor until a local minimum is reached. Thereafter, the process continues to the next layer, where the search commences from the node that was the nearest neighbor in the previous layer. This process is repeated until the final layer is reached. The search ends when the nearest neighbor in the final layer has been found. Compared to the poly-logarithmic search complexity of NSW, HNSW achieves logarithmic search complexity $\mathcal{O}(\log N)$ when the number of operations required to find the nearest neighbor on any layer is bounded by a constant [84]. However, in terms of memory consumption, graph-based algorithms tend to allocate more resources than other approaches, which is largely defined by the number of stored graph connections. Considering the construction complexity, building an HNSW index is an iterative insertion of all points, which is merely a sequence of ANN searches at different layers. Therefore, the construction time scales similarly to the search concerning the dataset size $\mathcal{O}(N \log N)$.

Benchmarking

Over the past years, numerous benchmarks [8, 72, 108, 121] have been conducted to compare the different types of ANN search algorithms to assess their performance in real-world settings. he experiments have demonstrated that graph-based algorithms, particularly HNSW, are most effective in terms of query time and the accuracy of the results in high-dimensional cases. However, in settings where computing resources are limited and dealing with datasets with billions of points, quantization approaches like PQ are recommended due to their small memory costs. It was also observed that for low-dimensional cases, e.g. $\mathbb{R}^6$, exact NN search algorithms like k-d trees become a competitive alternative to the existing state-of-the-art methods [8]. In practice, the integration of multiple methods, such as combining HNSW with quantization, is often employed to create composite index structures. These hybrids aim to balance memory efficiency with robust search capability, leveraging the strengths of individual approaches [38].

3.3.2 Range Search

Some of the presented structures for NN search also generalize well for other search tasks. While other search tasks than NN search are not commonly applied in CBR due to the type of queries used, we will still introduce the range search problem, particularly the orthogonal range search problem, that will later be of importance for our CBR search engine. Broadly, a range query involves identifying the subset of points within a predetermined range from a given set. For instance, in a two-dimensional space, a range query might request points falling within a particular circle. In the special case of orthogonal range queries, ranges are represented by axis-aligned rectangles. Orthogonal range searching distinguishes itself from other range search variants by allowing reasoning about each dimension independently due to the axis-aligned boundaries.

Definition 5. *Consider a set X of N points in the D-dimensional space $\mathbb{R}^D$. Given an orthogonal range $Q = [l_1, u_1] \times [l_2, u_2] \times \ldots \times [l_D, u_D]$, where $l_i \in \mathbb{R}$ and $u_i \in \mathbb{R}$ represent the lower and upper bounds in the respective dimensions and $l_i < u_i$ holds*

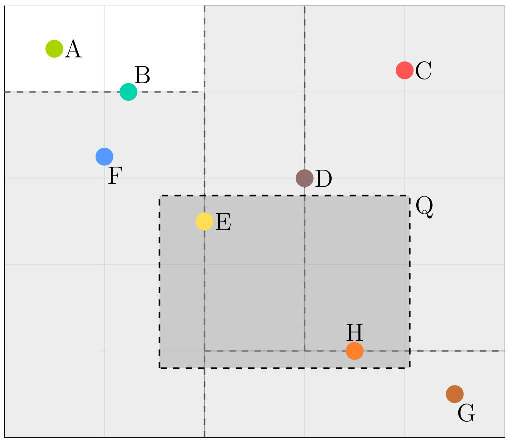


Figure 3.11: Range search with k-d tree. The dark dashed rectangle represents the orthogonal range Q, while the grey partitions correspond to the partitions that are inspected during the tree traversal.

for $1 \leq i \leq D$*, the problem of retrieving data on* $X \cap Q$*, the subset of* X *that is contained in the range* Q*, is called the orthogonal range problem.*

Existing methods for orthogonal range search primarily rely on space-partitioning trees. One method, which is already known from NN search, is the k-d tree. The k-d tree's structure, based on axis-aligned partitioning of the space, can be directly adapted for orthogonal range queries, as described in Section 3.3. The processing of range queries in k-d trees is described in Algorithm 4, where $\mathcal{T}_L$ and $\mathcal{T}_R$ specify the left and right child node, respectively. With regard to algorithmic complexity, the space complexity remains the same, $\mathcal{O}(N)$, exhibiting linear behavior, while the query time complexity for range queries is $\mathcal{O}(DN^{1-1/D}+q)$, where $q = |X \cap Q|$ is the number of returned points [68]. While the number of returned points for NN search problems is typically known beforehand and is usually negligible (e.g. by parameter k for kNN), the number of returned points for range queries is not known and can significantly influence the query time if, for instance, all points are included in the range. Consequently, range queries are considered *output-sensitive* [33].

Another popular index structure for range queries is the so-called range tree, which enables queries to be answered even faster in $\mathcal{O}(\log^{D-1} N + q)$ time. However, this comes at the cost of an increased construction time and space consumption of $\mathcal{O}(DN \log^{D-1} N)$ [16].

3.3.3 Search Refinement

The refinement of search results is a crucial aspect of information retrieval when the initial search results fail to meet the user's expectations, necessitating further optimization. Extensive research has been conducted to investigate methods for enhancing query precision [11, 85]. However, these additional refinement procedures come at the cost of increased query time. This is typically justified when high result accuracy is essential, such as in person re-identification systems [133].

This section delves into techniques that improve queries by analyzing the initially retrieved results. Common approaches involve formulating a new, optimized query based on the initial results. However, some methods bypass this step. These in-

Algorithm 4 Range search using a k-d tree

```
Require: $k$-d tree $\mathcal{T}$ build for set $X \subseteq \mathbb{R}^D$, range $Q = [l_1, u_1] \times [l_2, u_2] \times \ldots [l_D, u_D]$
Ensure: Returned points within the specified range $Q$
 1: function RangeSearchKDTree($\mathcal{T}$,$Q$)
 2:     if $\mathcal{T}$ is a leaf then
 3:         return $X_{\mathcal{T}} \cap Q$
 4:     else
 5:         if $\mathcal{T}_L$ is fully contained in $Q$ then
 6:             return Report all points in subtree $\mathcal{T}_L$
 7:         else if $\mathcal{T}_L$ intersects $Q$ then
 8:             RangeSearchKDTree($\mathcal{T}_L$,$Q$)
 9:         if $\mathcal{T}_R$ is fully contained in $Q$ then
10:             return Report all points in subtree $\mathcal{T}_R$
11:         else if $\mathcal{T}_R$ intersects $Q$ then
12:             RangeSearchKDTree($\mathcal{T}_R$,$Q$)
```

clude learning-to-rank techniques that train models to perform re-ranking[77], utilizing different similarity measures for re-ranking [12], or fusing results from multiple independent searches [131].

In the following section, we will examine techniques that issue an optimized query for search refinement. These include strategies such as relevance feedback, where user input is utilized to fine-tune the results, and query expansion or pseudo relevance feedback, which autonomously reissues and re-ranks the query [85, 132]. Approaches in CBR that exploit specific characteristics of the respective data type, such as the *RANSAC* algorithm [95] for geometric structures in images, are not considered in this discussion as these methods often rely on domain-specific techniques that are beyond the scope of this general overview.

Relevance Feedback

Methods in this category involve the user in the retrieval process to improve the final result set [85]. Typically, the user provides feedback on the relevance of the initially retrieved objects by selecting relevant or non-relevant items that are used to execute an improved query based on the feedback. This user input allows the search system to refine its understanding of the user's requirements, allowing it to present a revised collection of items that better matches the user's needs. Usually, the system may go through one or more iterations until satisfactory results are achieved. One well-known method is the *Rocchio* algorithm [104]. This algorithm modifies the representation of the query in the embedding space by moving the query closer to the embeddings of elements marked as relevant and away from those marked as irrelevant. More precisely, it moves the original query point in the embedding space closer to the centroid of relevant instances and pushes it further away from the non-relevant instances. For an original query point $\mathbf{q}_0 \in \mathbb{R}^D$, the refined query point $\mathbf{q}_m$ is computed by:

$$\mathbf{q}_m = \alpha \mathbf{q}_0 + \beta \frac{1}{N^+} \sum_{i=1}^{N^+} \mathbf{x}_i^+ - \gamma \frac{1}{N^-} \sum_{i=1}^{N^-} \mathbf{x}_i^-, \tag{3.15}$$

where $X^+ = \{\mathbf{x}_1^+, \ldots, \mathbf{x}_{N^+}^+\} \subset \mathbb{R}^D$ and $X^- = \{\mathbf{x}_1^-, \ldots, \mathbf{x}_{N^-}^-\} \subset \mathbb{R}^D$ define the respective sets for relevant and non-relevant instance embeddings and α, β, γ are weights, usually in the range $0 \leq \alpha, \beta, \gamma \leq 1$, used to weight the importance of each term. More recent approaches rely on the use of *support vector machines* (SVMs) that treat the labeled set of relevant and non-relevant instances as a classification problem [114]. The success of relevance feedback methods depends on certain assumptions [85]. First, users must have some idea of what they are looking for that is close to the desired objects. Second, the relevant objects have to be located somewhere close to each other in the embedding space, which means that they have to form a cluster.

Query Expansion

As opposed to relevance feedback, query expansion techniques refine the search without human interaction and have been widely used in the context of CBR [132, 133]. The idea is to employ the top-ranked instances from the initial search, usually an NN search, to reissue a new query to obtain a re-ranked list with better accuracy of the results. Early approaches involved averaging the embeddings of the top k initial results to calculate the new query vector [31]. Over time, methods evolved to use more sophisticated models, such as SVMs, to derive new query embeddings [5].

Certain approaches [59, 97, 133] have extended on the concept of query expansion by redefining neighborhood relationships. This was in response to the observation that the top k results of NN-based searches may include inaccurate matches, which could bias the query expansion process in an undesirable direction. This problem arises from the asymmetric nature of the nearest neighbor criterion. If point $\mathbf{a}$ is the nearest neighbor of point $\mathbf{b}$, it does not guarantee that $\mathbf{b}$ is also the nearest neighbor of $\mathbf{a}$, since there may be a point $\mathbf{c}$ that is closer to $\mathbf{a}$ than $\mathbf{b}$ is. This effect even multiplies as the range of top k results increases and can lead to cases where the top k results contain some false matches. To prevent the expanded query from being biased by false matches, the notion of k reciprocal nearest neighbors ($k\mathcal{R}$NN) can be employed.

Definition 6. *Let $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\} \subset \mathcal{M}$ be a set of N points in metric space $\mathcal{M}$ and given a query point $\mathbf{q} \in \mathcal{M}$, the k reciprocal nearest neighbors of $\mathbf{q}$ are all points among the k nearest neighbors of $\mathbf{q}$ that also have $\mathbf{q}$ among their k nearest neighbors, formally defined by:*

$$k\mathcal{R}NN(\mathbf{q}, X, k) = \{\mathbf{x}_i \mid (\mathbf{x}_i \in kNN(\mathbf{q}, X, k) \wedge \mathbf{q} \in kNN(\mathbf{x}_i, X, k)\}. \tag{3.16}$$

This neighborhood relationship better indicates the similarity of the query item and the found objects. Therefore, some approaches [97] consider the k reciprocal nearest neighbors to form the expanded query set instead of adding all top k results, while others [133] use the set of k reciprocal nearest neighbors to encode them into new embeddings that are used for re-ranking under different distance measures. Nevertheless, re-ranking the results based on the k reciprocal nearest neighbors entails an additional cost for computing the reciprocal neighbors, which is defined by $\mathcal{O}(N^2 \log N)$ for a dataset of size N [105]. This overhead can be partially mitigated by pre-computing the neighborhoods for the dataset [133]. It is important to note, that the asymmetric relationship of nearest neighbors does not necessarily hold in ANN search [6].

3.3.4 Limitations

While indexes significantly speed up query response times, their application in CBR systems frequently encounters overlooked limitations [28]. Common CBR research does not usually consider the dynamic aspect of datasets. Index structures are typically evaluated with static data, ignoring the complexities introduced by data modifications such as updates, deletions, or insertions. Consequently, changing data often requires the index structures to be rebuilt entirely or significantly degrades their efficiency. This limitation presents challenges for real-world CBR systems that must handle evolving datasets.

Moreover, the evaluation of index performance in existing studies occurs primarily in in-memory conditions, assuming that the entire dataset and its index can be fully loaded into memory. While some methodologies, like hash-based approaches, aim to compress data to fit within memory constraints even at large scales, there is a notable gap in research addressing scenarios where datasets exceed memory limits and require secondary storage solutions like SSDs or HDDs.

To address this gap, insights can be drawn from database technologies [71], where index structures are also utilized. In the database context, index structures are designed not only for fast query retrieval but also for their ability to support a range of functions, including similarity and range searches to accommodate a wider variety of use cases. Despite their differences, the underlying index structures often utilize similar techniques, such as partitioning-based algorithms like k-d trees.

Recently, so-called vector databases such as Pinecone[4] or Qdrant[5] have emerged to address these limitations. These databases are specifically designed for storing and querying high-dimensional embedding vectors, which have seen strong growth with the rise of deep learning technologies. Vector databases offer optimized search capabilities tailored for high-dimensional data, utilizing ANN techniques such as hashing, graph-based methods, and partitioning-based strategies. Furthermore, they provide features for more effective data and index management, as well as support for secondary storage.

3.4 Performance Evaluation

The evaluation of the effectiveness of CBR systems is of crucial importance for both their development and application. In the context of information retrieval, performance evaluation is used for distinguishing between the assessment of ranked and unranked results [85].

In settings where results are not ranked, the dataset's instances are classified as either relevant or non-relevant.[6] This classification essentially transforms the evalua-

[4]`https://pinecone.io/`

[5]`https://qdrant.tech/`

[6]The present overview will be limited to metrics designed for retrieval tasks, where relevance is binary; items are either relevant or not.

	Relevant	**Non-relevant**
Retrieved	True Positive (TP)	False Positive (FP)
Not Retrieved	False Negative (FN)	True Negative (TN)

Table 3.2: Confusion matrix for unranked retrieval.

tion into a binary classification task, where the metrics discussed in Section 2.2 apply, but with modified definitions for positive and negative outcomes, as detailed in Table 3.2.

Retrieval tasks usually face the challenge of unbalanced datasets, where non-relevant instances significantly outnumber relevant ones [85]. In such cases, accuracy becomes an unreliable metric, as one could easily achieve high accuracy by categorizing all instances as non-relevant. To address this imbalance, users frequently employ the F_1-score, which allows for more nuanced performance assessment in skewed scenarios.

In the context of ranked retrieval, the ranked results are naturally given by the top k items. Rather than computing precision, recall, and F_1-score for varying levels of k, it is more desirable to obtain a single-figure measure of retrieval quality. One way to achieve this is to consider the ranked set at a specific cutoff point k and then calculate precision and recall based on the cutoff set. These metrics are called *Precision@k* ($P@k$) and *Recall@k* ($R@k$):

$$P@k = \frac{TP_k}{k}, \tag{3.17}$$

$$R@k = \frac{TP_k}{TP_k + FN_k}. \tag{3.18}$$

The *average precision* (AP) score aims at balancing precision and recall [28]. The metric refers to the coverage area under the precision-recall curve and is calculated by:

$$AP = \frac{\sum_{k=1}^{N} P@k \cdot \mathbb{1}(k)}{R}, \tag{3.19}$$

where R is the number of relevant results for the query from the total number of instances in the dataset N and $\mathbb{1}(k)$ is an indicator function defined as:

$$\mathbb{1}(k) = \begin{cases} 1 & \text{if item at } k \text{ is relevant} \\ 0 & \text{otherwise.} \end{cases} \tag{3.20}$$

Often the average AP over multiple queries is used to get a more robust estimate for the retrieval performance, called *mean average precision* (mAP).

Part II
Contributions

CHAPTER 4

EFFICIENT SEARCH-BY-CLASSIFICATION

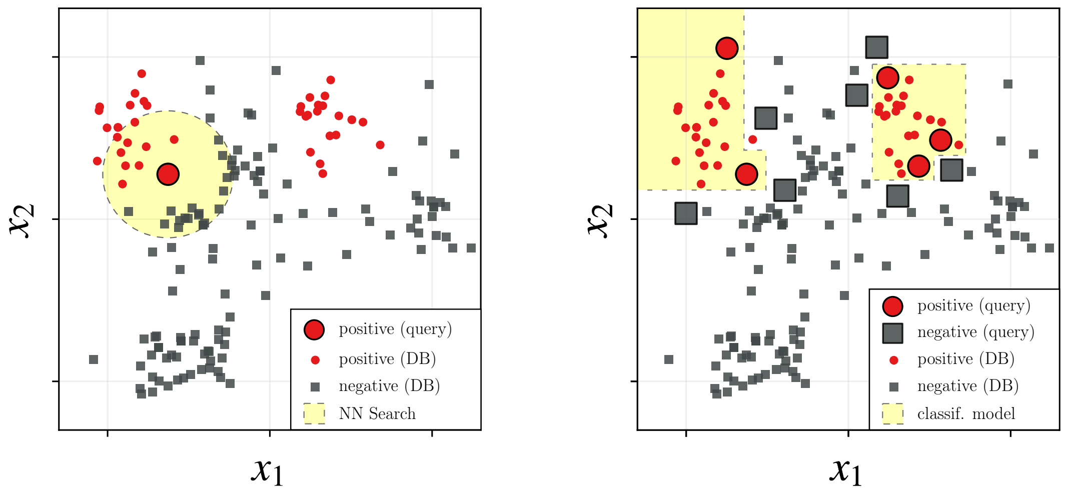
Contributed Material

- **Christian Lülf**, Denis Mayr Lima Martins, Marcos Antonio Vaz Salles, Yongluan Zhou, and Fabian Gieseke. Fast search-by-classification for large-scale databases using index-aware decision trees and random forests. In *Proceedings of the VLDB Endowment*, pages 2845–2857, 2023. (see Appendix A.1)
- Denis Mayr Lima Martins, **Christian Lülf**, and Fabian Gieseke. End-to-end neural network training for hyperbox-based classification. In *31st European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, ESANN*, 2023. (see Appendix A.2)
- Denis Mayr Lima Martins, **Christian Lülf**, and Fabian Gieseke. Training neural networks end-to-end for hyperbox-based classification. *Neurocomputing*. 2024. Under Review. (see Appendix A.3)

4.1 Search-by-Classification

In the preceding part, we examined the utilization of ANN algorithms to enhance search efficiency in CBR systems. However, as previously stated, the outcomes of ANN searches frequently yield unsatisfactory results in terms of accuracy and completeness. As ANN algorithms prioritize the speed of query execution at the expense of reduced accuracy, the results may not represent the optimal NN. Furthermore, as the search is usually restricted to finding the top k nearest neighbors, it might overlook relevant data points residing outside the chosen k neighborhood, particularly for tasks demanding high recall. For instance, when we consider tasks of identifying specific objects, such as wind turbines in satellite imagery databases, to estimate their operational count, ANN search may prove insufficient. Figure 4.1a visualizes this problem for an example scenario with two-dimensional embeddings. In the case of identifying wind turbines, the positive or relevant points would correspond to other instances of wind turbines that are contained in the database (DB). Given a positive query point and a search radius or number of nearest neighbors, a search relying on the exact nearest neighbors can result in an unsatisfactory performance. This is demonstrated in the example, where other relevant points are missed and non-relevant points are falsely included.[1]

[1]This example provides a simplified explanation of the concept. When applied in higher-dimensional spaces with more precisely defined embeddings, the NN-based methods can become more robust and effective.



(a) Nearest neighbor search **(b)** Search-by-Classification

Figure 4.1: Figure (a) illustrates the process of a nearest neighbor (NN) search in a two-dimensional space. For a given query point, the search would return all points within a specified radius, either the radius itself or the radius specified by k. However, this approach may not yield optimal results, potentially missing relevant, positive points and including irrelevant, negative points. Figure (b) depicts the scenario of multiple query points that have been labeled as either positive or negative. In the context of search-by-classification, a classification model is trained to distinguish between the positive and negative instances. This process results in the return of all database instances classified as positive by the classifier, which is represented by the yellow area.

These drawbacks make native NN search an inadequate solution for scenarios where high precision and recall are crucial in the quality assessment of the result set. We have identified two avenues for improvement of NN-based search approaches to mitigate the drawbacks of current approaches. Firstly, to more accurately reflect the user intent of the query, multiple items must be incorporated into the query. Secondly, given that multiple items describe the user intent, methods are required that take into account all query items simultaneously in order to identify similar items in the database.

As previously mentioned in Section 3.1.1, approaches that utilize multiple inputs for NN search are already existing. However, these methods are unable to fully leverage the presence of multiple query items, as they still operate on single-instance NN search. These methods either execute NN search for each query item individually and then re-rank the results, or they combine the embeddings of all items into a single embedding for NN search.

In our work [81], we proposed an alternative solution to the existing NN-based approaches, that reformulates the problem into a binary classification task. This approach is therefore termed *search-by-classification*. The database is considered as an unknown test dataset with instances that are either labeled positive ($y = 1$) or negative ($y = 0$) with respect to a specific user query. To illustrate, if User A wishes to search for wind turbines in the database, all instances containing wind turbines are positive, while all others are negative. Conversely, for a query of User B who is interested in solar panels, all instances with solar panels are treated as positive. In this context, the user query acts as a small training set with multiple positively or negatively labeled items.

A machine learning model can then be trained on the embeddings of the training data to learn to separate the positive and negative instances in the embedding space. It is important to note that the distribution of labels is heavily imbalanced in this scenario, as the number of positives is way smaller than the negatives, which poses challenges for the classification model. The search-by-classification approach utilizes the trained model during the inference phase for the actual search. Here, the model is applied to the entire database, classifying each item. Finally, only the entries classified as positive by the model are returned as search results. A figure that visualizes how search-by-classification approaches implement their search in contrast to the previously discussed NN-based approaches is shown in Figure 4.1b.

It is hypothesized that the search-by-classification approach will deliver more accurate results for a sufficiently large training set than NN-based methods. This is supported by three key arguments:

- **Binary labels:** By allowing users to specify both types of objects the search should return and those it should exclude, users can more accurately express their search intent.
- **Complex model:** A classification model can learn complex rules for identifying positive instances, whereas a NN-based search primarily retrieves the most similar items.
- **Integrated relevance feedback**: As a learning-based approach, search-by-classification can seamlessly integrate user feedback. By incorporating user-designated positive and negative examples into the training set through iterative feedback loops, the classification model continuously improves its accuracy.

However, in order to identify the positive instances in the database, native search-by-classification requires the classification model to pass over all instances in the database, as shown in Figure 4.2. While this is feasible for small datasets, this sequential process becomes impractical for large-scale data and multiple queries as the time grows linearly with the number of instances.

Consequently, we developed a solution for search-by-classification that is efficient even in large-scale data settings [81]. Our proposed method is inspired by the use of index structures in CBR. It includes a classifier whose inference phase is supported by index structures to accelerate the retrieval. This involves finding a mapping between the classification model and the index structures such that the classification model can be translated into an index-supported query. In our work, we exploited the mapping between decision trees and k-d trees to accelerate the inference phase. This is based on the observation that due to the orthogonal nature of the splits in decision trees (see Section 2.1.1), their leaves correspond to orthogonal range queries, as defined in Section 3.3.2. Given this mapping, we can speed up the created range queries by using index structures such as k-d trees that support range queries. However, additional modifications are required to ensure that these range queries remain efficient for high-dimensional data, which is generally affected by the curse of dimensionality.

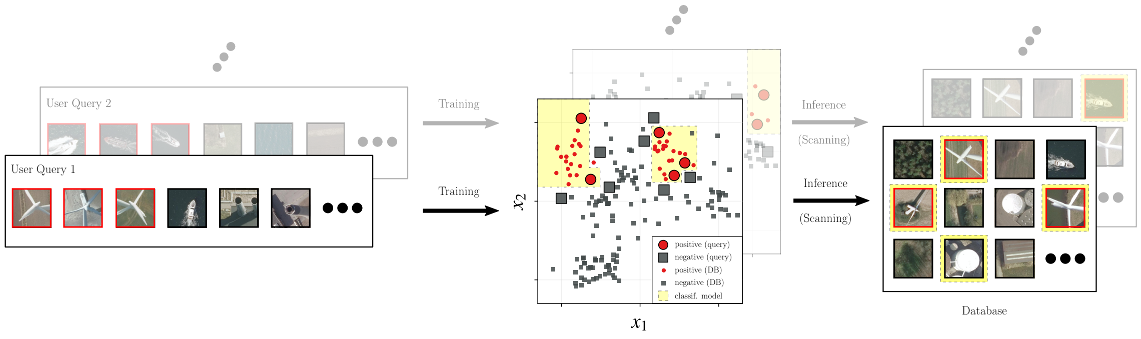


Figure 4.2: Traditional search-by-classification methods necessitate training separate models for each query set. This approach incurs latency when applied to scan the full database for inference. Our framework addresses this limitation by a co-design of indexes and models for efficient retrieval of positive instances, significantly reducing search time. Figure adapted from our work [81].

The following section presents the overall search-by-classification framework. We refer to the new family of classification models constructed in line with their corresponding index structures as *index-aware classifier*. Two distinct models were created, one based on a decision tree and the other one on a neural network. These are explained in more detail in Section 4.3. The variant of k-d trees employed as the underlying indexing structure for our framework is described in Section 4.4. Finally, in Section 4.5, we conclude with remarks on the experimental evaluations of our proposed method and discuss some limitations.

4.2 Framework

Our search-by-classification framework presented in Figure 4.3 has been adapted from the general CBR framework described in Section 3.1.2. Similarly, the workflow can be divided into two phases: 1) Offline Preprocessing and 2) Query Processing.

Offline Preprocessing

This phase is passed once before the actual query processing can be started. The data is extracted by a feature extractor[2], which captures the inherent characteristics of the input data in compact feature embeddings. It is important that the embeddings are generalizable to any search task and not confined to a specific class of queries. Index structures are pre-built on the embeddings for accelerating the search. In order to enable compatibility with the index-aware classifier, we impose some requirements on the construction of the index structures. These requirements include the ability to process orthogonal range queries. For our search engine, we selected a variant of k-d trees due to its wide appliance in research and its straightforward implementation.

Furthermore, we require a set of index structures built on subsets of the extracted embeddings, known as feature subsets. The rationale behind this approach is to

[2]We used a ResNet101 model [55] pre-trained on ImageNet as a feature extractor with a modified final layer that outputs embeddings of $D = 50$.

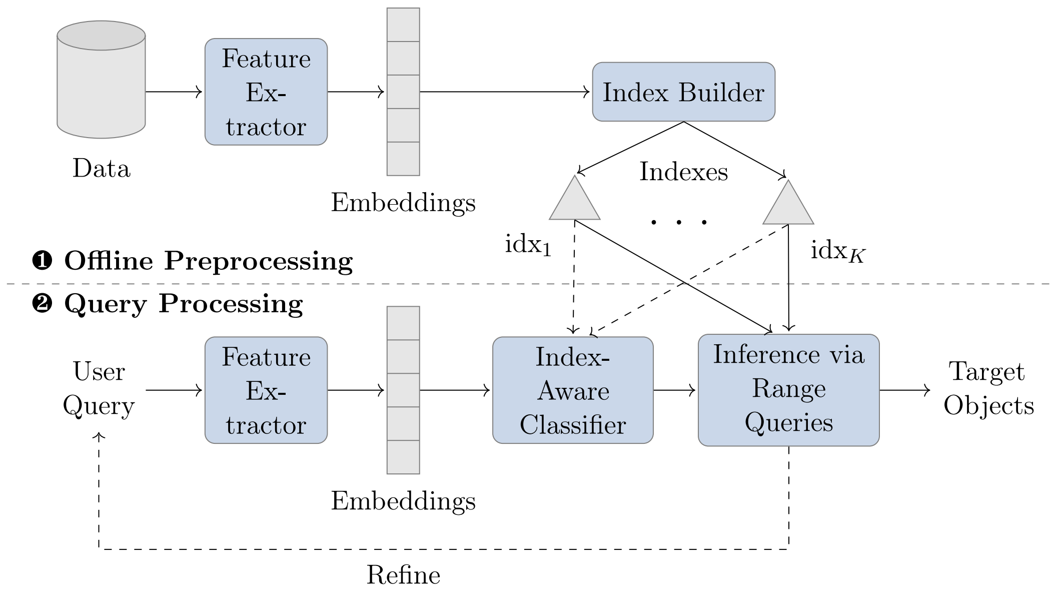


Figure 4.3: Our search-by-classification framework accelerates data retrieval by utilizing an index-aware classifier and pre-built indexes. The classifier leverages efficient range queries on the indexes, eliminating the need for time-consuming full data scans. Figure adapted from our work [81].

decompose the original search query into multiple smaller, more manageable sub-queries that operate on these feature subspaces. By splitting the query into multiple sub-queries, we can benefit from the reduced complexity of each sub-query, which makes it significantly faster to retrieve using the index structures. Still, the sum of all sub-queries must correspond to the same results as the original query.

It is of note that the dimensionality of each subset d should be carefully selected, e.g. $d = 4$, to prevent the curse of dimensionality, as described in Section 3.2.1. Furthermore, depending on the dimensionality of the data D, the number of feature subsets K should also be carefully selected to ensure good coverage of all available features of the embedding space. Conversely, the more index structures on feature subsets are built, the more storage is consumed for storing these additional index structures.

Query Processing

In this phase, the index-aware classifier handles the actual query workload. Unlike the classic CBR framework where the search is performed via NN search, we perform the search via index-supported range queries extracted from the trained classifier. However, users must provide more precise queries by supplying multiple positive and negative examples that define their search intent. As is the case with any classification task, performance is enhanced as more (meaningful) training data is added. The query items are then transformed into embeddings and forwarded to the index-aware classifier to train the model. Not all existing classification models are natively supported to become an index-aware classifier according to our definition, as this requires the fulfillment of the following requirements:

- **Orthogonal decision boundaries:** The learned decision boundaries that separate the positive and negative classes in feature space must be axis-aligned, i.e. orthogonal, such that they can be translated into range queries.
- **Effectiveness in feature subspaces:** Index-aware classifiers must operate in the given feature subspace to build the decision boundaries in order to make use of the pre-built index structures that were built on only feature subsets. The overarching goal is that these models perform as effectively as traditional classification models, which are not confined to specific subspaces.

These constraints significantly limit the types of classification models that can be used. Most existing models struggle to create axis-aligned decision boundaries, leaving tree-based methods such as decision trees as the primary candidates. However, we observed that decision trees exhibited undesirable classification performance on benchmark classification datasets when restricting the tree construction to feature subspaces. This motivated us to develop our own index-aware classifiers that are optimized for the classification under the given constraints. Our first model, *decision branches* [81], draws inspiration from the classic decision tree as it also builds tree-based models that minimize an impurity function. In contrast to decision trees, our model employs a bottom-up approach, whereby individual trees are constructed from one instance and subsequently expanded to include additional positive instances. As a follow-up, we also worked on *HyperNN* [73], an index-aware classifier that is based on neural networks.

Our search-by-classification framework distinguishes itself from native NN-based search systems through its integrated relevance feedback mechanism. Should the initial search results fail to meet the user's expectations, they may enhance the outcome by selecting additional positive and negative items from the response set to incorporate into the training dataset. After this, the classifier utilizes this updated training set to learn and refine its model in subsequent searches. As the model training and inference run at a fast pace due to the pre-built index structures, these extra iterations do not consume a significant amount of time.

4.3 Index-Aware Classifiers

In the following sections, we will delve into the details of our developed index-aware classifiers. We first start with our tree-based variant named decision branches and then will continue with the neural network model HyperNN. To describe our algorithms, we consider a binary classification dataset $T = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\} \subset \mathbb{R}^D \times \{0, 1\}$. An instance $(\mathbf{x}_i, y_i)$ with $y_i = 1$ corresponds to a positive or relevant element while $y_i = 0$ to a negative instance for $1 \leq i \leq N$. We assume K index structures of randomly drawn feature subsets $F_1, \ldots, F_K \subset \{1, \ldots, D\}$ with $|F_\kappa| = d$ for $1 \leq \kappa \leq K$ and $1 \leq d \leq D$ are pre-built.

4.3.1 Decision Branches

Our decision branches model (DBranch) represents a new construction schema for tree-based classification models that builds the bottom parts of decision trees, which we call decision branches. Each decision branch is associated with a D-dimensional

Algorithm 5 BuildDBranch: Constructing DBranch model

```
Require: Dataset $T = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_N, y_N)\} \subset \mathbb{R}^D \times \{0,1\}$, $F_1, \dots, F_K \subset \{1, \dots, D\}$ with $|F_\kappa| = d$, $\kappa = 1, \dots, K$, for some $1 \le d \le D$, $1 \le p \le K$, $\mu \in \{1, \dots, D\}$, and stopping criterion for decision tree $\lambda$
Ensure: Set $\mathcal{T} = \{(B_1, \mathcal{B}_1), \dots, (B_M, \mathcal{B}_M)\}$ of $D$-dimensional boxes $B_1, \dots, B_M$ with $n_b(B_m) \le d$ for $m = 1, \dots, M$ along with associated decision branches $\mathcal{B}_1, \dots, \mathcal{B}_M$
 1: function BuildDBranch($T, \{F_1, \dots, F_K\}, p, \mu, \lambda$)
 2:     $T_0 \leftarrow \{(\mathbf{x}, y) \in T | y = 0\}$; $T_1 \leftarrow \{(\mathbf{x}, y) \in T | y = 1\}$
 3:     $\mathcal{T} \leftarrow \{\}$
 4:     repeat
 5:         Let $(\mathbf{x}', y')$ by any positive instance in $T_1$
 6:         $g_{opt} \leftarrow 0$; $B_{opt} \leftarrow$ None
 7:         $\mathcal{F} = \{F_{i_1}, \dots, F_{i_p}\} \subseteq \{F_1, \dots, F_K\}$
 8:         for each $F \in \mathcal{F}$ do
 9:             $B, g \leftarrow$ GreedyMaxGainBox($T_0 \cup T_1, \mathbf{x}', F$)
10:             if $g > g_{opt}$ then
11:                 $g_{opt} \leftarrow g$; $B_{opt} \leftarrow B$
12:         $T_1, R_1 \leftarrow$ RemoveInstances($T_1, B_{opt}$)
13:         $T_0, R_0 \leftarrow$ RemoveInstances($T_0, B_{opt}$)
14:         $\mathcal{B} \leftarrow$ BuildDecisionTree($R_0 \cup R_1,\ \lambda,\ \mu$)
15:         $\mathcal{T} \leftarrow \mathcal{T} \cup (B_{opt}, \mathcal{B})$
16:     until $T_1$ is empty
17:     return $\mathcal{T}$
```

hyperbox[3] analogous to a leaf node in a decision tree. These boxes are supposed to be translated into multiple orthogonal range queries for retrieving the data. We denote a box B according to the notation of a range query in Definition 5 as:

$$B = [l_1, u_1] \times [l_2, u_2] \times \ldots \times [l_D, u_D], \tag{4.1}$$

where $l_j \in \mathbb{R}$ and $u_j \in \mathbb{R}$ represent the lower and upper bounds in the j-dimension, respectively, for $1 \le j \le D$, and $l_j < u_j$ holds. If a feature j is not restricted in both directions, that is, $l_j = -\infty$ and $u_j = +\infty$, we call it unbounded. Note that these features are not considered for the translation into a range query as they put no constraint on the query. If the dimension j of a box is bounded in both directions, that is, $l_j > -\infty$ and $u_j < +\infty$, we call it bounded w.r.t. dimension j, whereas if it is left- or right-bounded, that is, $l_j > -\infty$ or $u_j < +\infty$, we call it half-bounded w.r.t. dimension j. We specify the number of bounded and half-bounded dimensions with $n_b(B)$.

Our construction algorithm aims to create a set of boxes with their associated decision branches that contain a maximum number of positive examples while keeping the number of negative instances in their boxes minimal by minimizing an impurity criterion. At first glance, it may appear that this does not deviate from the construction of decision trees. However, the construction of these boxes takes place

[3]For simplicity, the term "box" will be used to refer to "multi-dimensional hyperbox" in the following.

Algorithm 6 GreedyMaxGainBox: Growing the boxes

```
Require: Subset $S = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_{N'}, y_{N'})\} \subset \mathbb{R}^D \times \{0, 1\}$, $\mathbf{x}' \in \mathbb{R}^D$, and $F \subset \{1, \dots, D\}$ with $|F| = d$ for some $1 \leq d \leq D$
Ensure: Box $B \subset \overline{\mathbb{R}}^D$ with $n_b(B) \leq d$ and gain $g \in \mathbb{R}$
1: function GreedyMaxGainBox($S$, $\mathbf{x}'$, $F$)
2:     $F_s = (i_1, \dots, i_d) \leftarrow$ RandomSequence($F$)
3:     $B \leftarrow$ InitialEmptyBox($\mathbf{x}'$, $S$, $F_s$)
4:     for $j = 1, \dots, d$ do
5:         $B \leftarrow$ ExpandBox($\mathbf{x}', S, B, i_j$)
6:     $g \leftarrow$ Gain($S, B$)
7:     return $B, g$
```

under the constraints that each box is only bound in a few dimensions and that the set of bounded dimensions corresponds to the feature subsets $F_1, \dots, F_K$ for which index structures are pre-built. For instance, given an index structure constructed for the feature subset $F_\kappa = \{3, 5, 10\}$, a corresponding box B will possess bounds (either half-bounded or fully bounded) exclusively for the dimensions indexed by F_κ, while for all other dimensions, B remains unbounded. This guarantees that the range queries produced by the boxes are compatible with the existing pre-built index structures. Furthermore, they rely on a limited subset of features, thus avoiding the curse of dimensionality, as only the bounded dimensions affect query time. We refer to this alignment of the construction algorithm with the available index structures as being index-aware.

Construction Algorithm

Our construction algorithm is described in detail in Algorithm 5. It takes the training dataset T, the K feature subsets $F_1, \dots, F_K$ and hyperparameters $1 \leq p \leq K$ for the number of feature subsets tested per iteration of the box construction, $\mu \in \{1, \dots, D\}$ for the number of features tested per split in the decision branches construction as well as the corresponding stopping criterion λ. The construction begins with the two subsets $T_1 = \{(\mathbf{x}, y) \in T \mid y = 1\}$ and $T_0 = \{(\mathbf{x}, y) \in T \mid y = 0\}$. Over multiple iterations boxes with their corresponding decision branches are constructed until no positive instances remain in T_1. At each iteration, a random positive instance from T_1 is taken as starting point as well as a random subset of feature subsets $\mathcal{F} = \{F_{i_1}, \dots, F_{i_p}\} \subseteq \{F_1, \dots, F_K\}$. Among all selected feature subsets, the algorithm constructs boxes origin in point $(\mathbf{x}', y')$ that maximize an information gain via function GreedyMaxGainBox (see Algorithm 6). The box with the highest information gain B_{opt} is selected as the final DBranch model candidate. All points, both positive and negative, within the box B_{opt} are eliminated from sets T_1 and T_0 using the RemoveInstances function for following iterations. This function divides the input set into two subsets: those inside the box and those outside. Given the set of positive and negative points included in box B_{opt}, the corresponding decision branch is constructed via the function BuildDecisionTree (see Algorithm 1). The function builds a regular decision tree for the subset, which results in the decision branch $\mathcal{B}$. Both the box B_{opt} and the decision branch $\mathcal{B}$ are added to the set $\mathcal{T}$, which represents the final DBranch model.

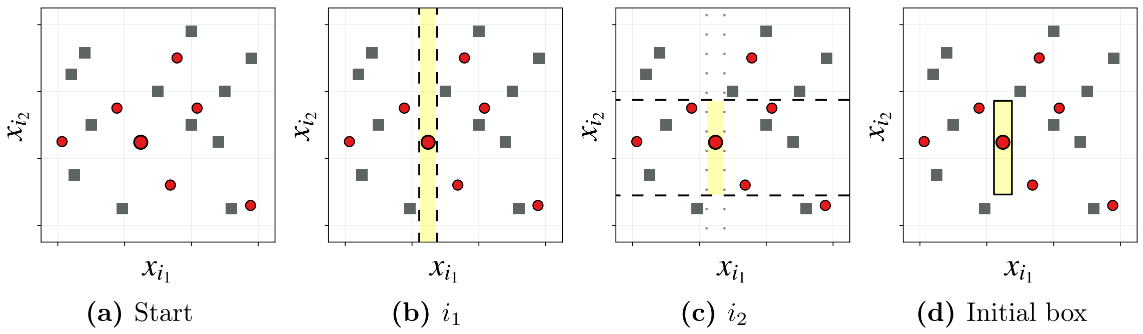
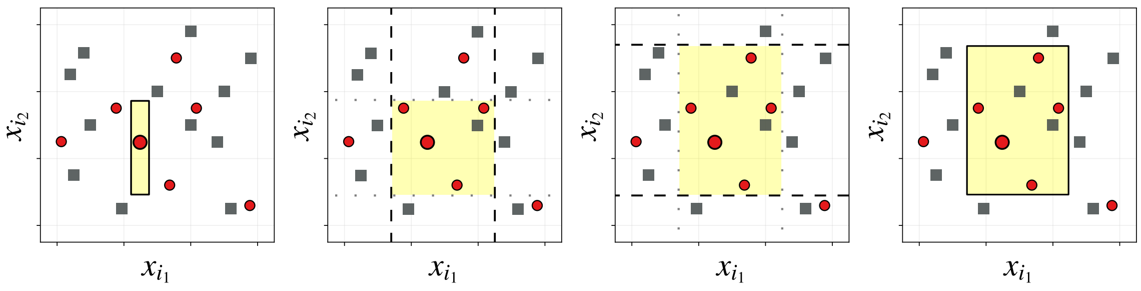


Figure 4.4: Illustration of InitialEmptyBox. The large red points represent the starting point $(\mathbf{x}', y')$. Figure adapted from our work [81].

How we construct the individual boxes is determined in the function GreedyMaxGainBox, which is shown in Algorithm 6. The goal is to create a box B and compute its associated information gain for a given point $\mathbf{x}'$, a feature subset F and a subset of the training instances $S \subset T$ of the size N' such that it is bounded only in the dimensions of the feature subset, which in total $n_b(B) \leq d$. In general, the box construction grows the boundaries from the box in F sequentially one dimension after another. The order of how the dimensions are processed is randomly arranged by the function RandomSequence. The following procedure can be grouped into two stages: 1) the box initialization phase defined by the function InitialEmptyBox and 2) the box expansion phase defined by the function ExpandBox. In the initialization phase (see Line 3), a preliminary box is constructed based on the given point $\mathbf{x}'$. The goal is to create a rectangle that covers as much space as possible while no other points than $\mathbf{x}'$ are included. Thereby, we ensure good conditions for the upcoming expansion phase. The problem at hand is known under the name *maximum empty rectangle containing a query point* [52] and is non-trivial to solve with solutions that require $\mathcal{O}(\log^4 N)$ time complexity. Therefore, we developed a simple heuristical approach that is still efficient to solve and makes use of randomness. We present the technique for a 2D example in Figure 4.4.

Afterwards, the procedure enters the expansion phase (see Line 5), where the box is expanded by maximizing the information gain (see Equation 2.6). The expansion proceeds dimension by dimension $i_1, \ldots, i_d$ expanding the lower and upper boundary, as shown in Figure 4.5. For the expansion of dimension i_j with $j = 1, \ldots, d$, it only considers the subset $\overline{S}_{i_j} = S \cap \overline{B}_{i_j}$ where the box $\overline{B}_{i_j}$ is the same as B except that in dimension i_j the lower and upper boundary l_{i_j} and u_{i_j} are set to $-\infty$ and ∞. This is indicated in Figure 4.5b and Figure 4.5c by the dotted lines. To expand the lower boundary in i_j, all points in $(\mathbf{x}, y) \in \overline{S}$ with $\mathbf{x} < \mathbf{x}'$ are sorted in increasing order w.r.t. their distance $|\mathbf{x}_{i_j} - \mathbf{x}'_{i_j}|$ to point $\mathbf{x}'_{i_j}$ in dimension i_j. Starting with $I = \{(\mathbf{x}', y')\}$, the points are then processed sequentially and added to the set I. The remaining points form the set $O = S \setminus I$. We select the split point that maximizes the information gain $G(I, O)$, as defined in Equation 2.6. Similar to classic decision trees, we use the Gini index as impurity function. After all or a fixed number of points have been added without improvement of the gain, the split point with the highest information gain is chosen as the new lower boundary of B. Analogous to the lower boundary, we repeat the expansion for the upper boundary and as well



(a) Inital box **(b)** i_1 **(c)** i_2 **(d)** Final box

Figure 4.5: ExpandBox: The red points depict positive and the black points negative instances, respectively. The initially covered point corresponds to the positive instance $\mathbf{x}$. Figure adapted from our work [81].

as for all dimensions $j = 1, \ldots, d$. Finally, the optimal box for feature subset F is returned together with its computed information gain.

Models

The final DBranch model $\mathcal{T}$ consists of a set of boxes $B_1, \ldots, B_M$ with their corresponding decision branches $\mathcal{B}_1, \ldots, \mathcal{B}_M$. In this context, a box B results from the bottom-up construction phase (Algorithm 5 Lines 5-13), while $\mathcal{B}$ is a decision branch that corresponds to a small decision (sub-)tree constructed in a top-down manner, which separates the subset $R_0 \cup R_1$ of the points contained in box B (Algorithm 5 Line 14). The entire DBranch model is shown in Figure 4.6 demonstrating how it can be employed as an index-aware classifier. The points included in the boxes can be quickly retrieved via orthogonal range queries supported by the pre-built index structures (indicated by dashed black rectangles). The retrieved point sets are then classified into positive and negative by the corresponding decision branches that return the final set of positive points (indicated by yellow areas). We developed two versions of how the decision branches $\mathcal{B}$ are constructed to separate $R_0 \cup R_1$. Either the model builds a tree considering all D features or it only resorts to the features of the current feature subset F. The first approach offers superior classification performance, as all features can be leveraged to distinguish between positive and negative points. However, this approach necessitates the evaluation of all features, which can be time-consuming for larger datasets. Since the second option operates on the features of the respective feature subset, the required features can be directly retrieved from the index structures without extra loading times. However, this variant has a slightly worse classification performance, as shown in our experiments [81].

The construction of a box B includes elements of randomness, either through the random selection of instances $(\mathbf{x}', y')$ or by expanding the box dimensions in a randomly determined order. Although incorporating randomness is not essential for constructing a single DBranch model $\mathcal{T}$, it becomes crucial for creating ensembles of decision branches, such as those seen in standard tree-based ensemble models like random forests and extremely randomized trees. In these models, randomness enhances the diversity within the ensemble, thereby improving the robustness and accuracy of the predictions. Extending decision branches to ensembles can be done by simply combining the sets of L DBranch models $\mathcal{T}_1, \ldots, \mathcal{T}_L$. The database instances are returned that are classified as positive by a majority vote of the models similar as

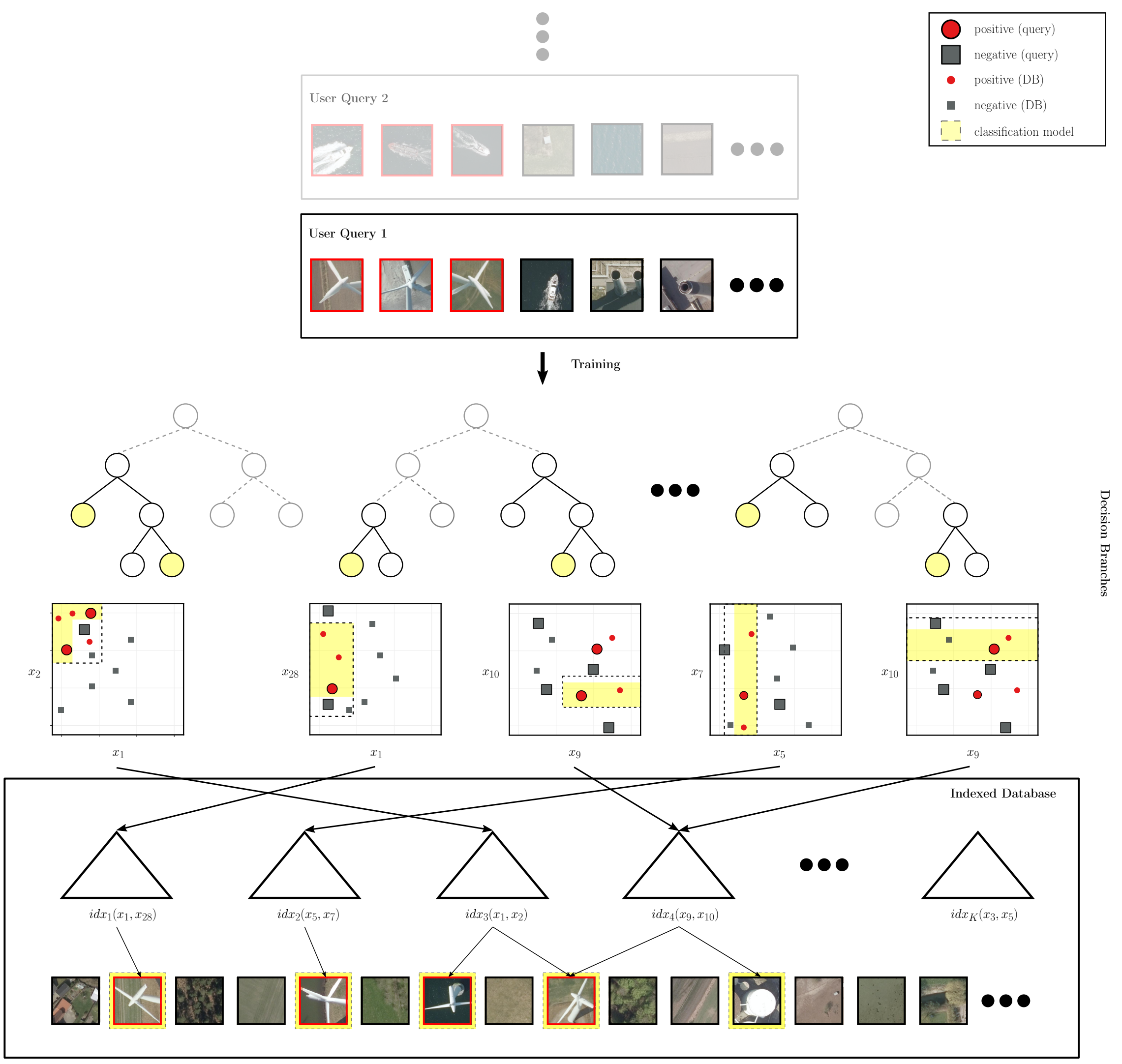


Figure 4.6: Processing user queries via decision branches and pre-built index structures for large databases. The bounding box B associated with a given $\mathcal{B}$ is visualized as a dashed black rectangle. The leaves of the branches corresponding to the positive class are highlighted in yellow. Figure adapted from our work [81].

for the random forest defined in Equation 2.11. However, the increased classification performance of the ensemble models comes at the cost of additional query processing time since more range queries coming from all L models need to be processed.

4.3.2 Hyperbox-Based Neural Network

Next to decision branches, we also examined the potential of learning orthogonal, multi-dimensional boxes using neural networks. This approach, which we named HyperNN [73], was initially developed for model interpretability, as rectangular box representations of the decision boundaries are often favored for human decision-making [43]. However, due to the orthogonal decision boundaries, the underlying algorithm also generalizes to be used as an index-aware classifier for search-by-classification. For the description of the algorithm, we will refer first to the original description and then will transfer it to the use case of index-aware classification.

In this context, we will denote the boxes not based on the lower and upper boundaries as it was done in Equation 4.1, but by the lower boundary and the length span:

$$B = [l_1, l_1 + s_1] \times [l_2, l_2 + s_2] \times \ldots \times [l_D, l_D + s_D], \tag{4.2}$$

where $l_j \in \mathbb{R}$ and $s_j \in \mathbb{R}$ represent the lower bound and length span in the j-th dimensions $1 \leq j \leq D$ with $0 \leq s_j$. The upper bound u_j can be derived by $u_j = l_j + s_j$. For a point $\mathbf{x} \in \mathbb{R}^D$, it is contained by box B if and only if for every dimension j, the following condition is satisfied:

$$l_j \leq \mathbf{x}_j \leq u_j \quad \forall j \in \{1, \ldots, D\}. \tag{4.3}$$

The containment of a point $\mathbf{x}$ in box B can be defined by an indicator function:

$$\mathbb{1}_B(\mathbf{x}) = \begin{cases} 1 & \text{if } l_j \leq \mathbf{x}_j \leq u_j \quad \forall j \in \{1, \ldots, D\} \\ 0 & \text{otherwise.} \end{cases} \tag{4.4}$$

The entire model is represented by a set of boxes $\mathcal{B} = \bigcup_{m=1}^{M} B_m$ of M boxes $B_1, \ldots, B_M$. In a binary classification task, we classify point $\mathbf{x}$ by:

$$\mathcal{T}_{\mathcal{B}}(\mathbf{x}) = max(\mathbb{1}_{B_1}(\mathbf{x}), \ldots, \mathbb{1}_{B_M}(\mathbf{x})), \tag{4.5}$$

which classifies it as 1 if it is contained at least in one box otherwise 0. The goal of the learning task is to find a set $\mathcal{B} = \{B_1, \ldots, B_M\}$ by solving the following optimization task:

$$\min_{\mathcal{B}} \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(y_i, \mathcal{T}_{\mathcal{B}}(\mathbf{x}_i)), \tag{4.6}$$

where $\mathcal{L}$ is a loss function typically represented by the BCE (see Equation 2.15). The loss is minimized by adjusting the weights, which are represented by the vectors of lower bounds and length spans of the boxes, through gradient optimization. Specifically, for a given box B_m for $1 \leq m \leq M$, we denote the weight vector of all lower bounds as $\mathbf{w}_l^{(m)} = \left[l_1^{(m)}, l_2^{(m)}, \ldots, l_D^{(m)}\right]$ and the weight vector of all length spans as $\mathbf{w}_s^{(m)} = \left[s_1^{(m)}, s_2^{(m)}, \ldots, s_D^{(m)}\right]$. Therefore, to update the box parameters in the direction that minimizes the loss, we calculate the gradient of the loss function with respect to each box's lower bounds and length spans.

However, the calculation of the gradient becomes impossible due to the non-differentiable operations included in the model. This is mainly attributed to the indicator functions $\mathbb{1}_{B_1}(\mathbf{x}), \ldots, \mathbb{1}_{B_M}(\mathbf{x})$, which can be seen as a step function according to Figure 4.7, which is non-differentiable. Therefore, we reformulate the containment check by differentiable operations. First, we re-define the condition from Equation 4.4 by differentiable operations. Let $\boldsymbol{\omega}_l^{(m)}(\mathbf{x}) = \mathbf{x} - \mathbf{w}_l^{(m)}$ and $\boldsymbol{\omega}_u^{(m)}(\mathbf{x}) = \mathbf{w}_l^{(m)} + \mathbf{w}_s^{(m)} - \mathbf{x}$, then a point is covered by box B_m when $\boldsymbol{\omega}_l^{(m)}(\mathbf{x})$ and $\boldsymbol{\omega}_u^{(m)}(\mathbf{x})$ only have non-negative values, formally defined as:

$$\mathbb{1}_{B_m}(\mathbf{x}) = \begin{cases} 1 & \text{if } \boldsymbol{\omega}_{l_j}^{(m)}(\mathbf{x}_j) \geq 0 \text{ and } \boldsymbol{\omega}_{u_j}^{(m)}(\mathbf{x}_j) \geq 0 \quad \forall j \in \{1, \ldots, D\} \\ 0 & \text{otherwise.} \end{cases} \tag{4.7}$$

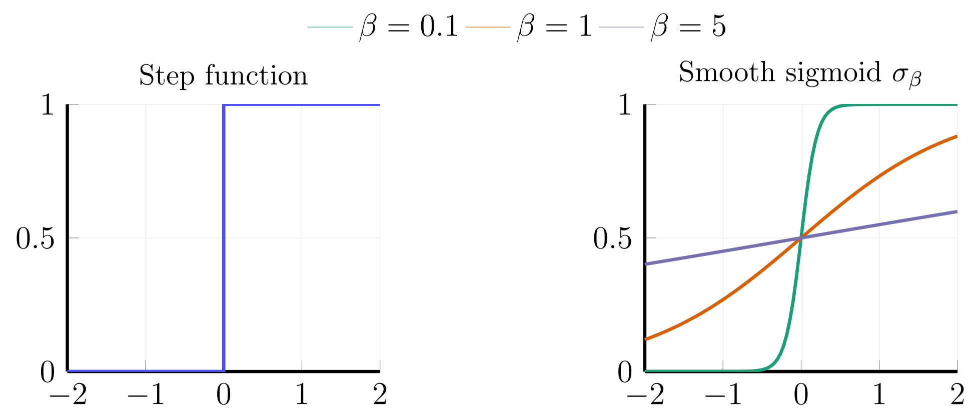


Figure 4.7: HyperNN uses smooth sigmoid as a differentiable alternative to the classic step function. Figure adapted from our work [74].

As before, to obtain meaningful gradient information in the backpropagation phase, we cannot resort to element-wise step functions to check for this property.[4] Instead, we resort to a differentiable surrogate applied to the minimum value (across all D dimensions) of both $\boldsymbol{\omega}_l^{(m)}(\mathbf{x})$ and $\boldsymbol{\omega}_u^{(m)}(\mathbf{x})$. More precisely, for $\boldsymbol{\omega}_l^{(m)}(\mathbf{x})$, we implement this check via a modified version of the sigmoid function [58]:

$$\sigma_\beta(min(\boldsymbol{\omega}_l^{(m)}(\mathbf{x}))) = \frac{1}{1 + \exp(^{-min(\boldsymbol{\omega}_l^{(m)}(\mathbf{x}))}/_{\beta})}. \tag{4.8}$$

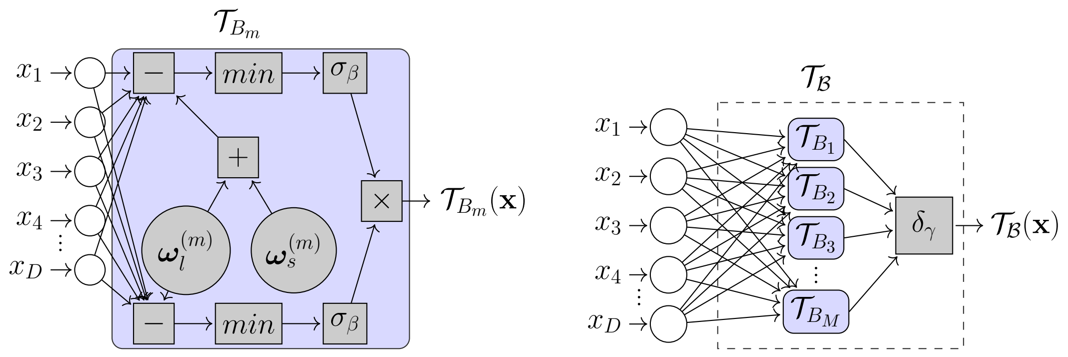
The parameter β controls the smoothness of the function, as shown in Figure 4.7 (top-right). The smaller β the more closely the function resembles the original step function while still providing gradient information. Similar to the check of the lower bounds, we implement the upper bound check with $\sigma_\beta(min(\boldsymbol{\omega}_u^{(m)}(\mathbf{x})))$. Finally, we multiply the lower and upper bound checks for producing the differentiable containment check of box B_m as $\mathcal{T}_{B_m}(\mathbf{x}) = \sigma_\beta(min(\boldsymbol{\omega}_l^{(m)}(\mathbf{x}))) \times \sigma_\beta(min(\boldsymbol{\omega}_u^{(m)}(\mathbf{x})))$. For each box, a value in the range $[0, 1]$ is output for a point $\mathbf{x}$ that indicates the likelihood of the point being contained by the box. In the context of the entire model $\mathcal{T}_{\mathcal{B}}$, the model should indicate whether the point $\mathbf{x}$ is at least contained by one box. This can be checked by obtaining the maximum over all boxes $\mathcal{T}_{B_1}, \ldots, \mathcal{T}_{B_M}$, as shown in Equation 4.5 for the non-differentiable case. However, this approach would only yield gradient information for the box with the maximum score and no gradient information for the remaining boxes, which weakens the training of the model. Therefore, we also employ a smoothened function of the max operation δ_γ:

$$\delta_\gamma(\mathcal{T}_{B_1}(\mathbf{x}), \ldots, \mathcal{T}_{B_M}(\mathbf{x})) = \frac{\sum_{m=1}^{M} \mathcal{T}_{B_m}(\mathbf{x}) \exp(^{\mathcal{T}_{B_m}(\mathbf{x})}/_{\gamma})}{\sum_{m=1}^{M} \exp(^{\mathcal{T}_{B_m}(\mathbf{x})}/_{\gamma})}, \tag{4.9}$$

where values close to 1 denote containment of $\mathbf{x}$ and γ controls the smoothness of δ_γ.

All described operations to create a differentiable box containment check form the neuron in the HyperNN and are visualized in Figure 4.8a. The entire HyperNN model, composed of multiple neurons $\mathcal{T}_{B_1}, \ldots, \mathcal{T}_{B_M}$, is shown in Figure 4.8b. Training

[4]In the forward pass, we still output the discrete values to make the model predictions.



(a) HyperNN neuron **(b)** HyperNN network structure

Figure 4.8: Architecture of HyperNN. Figure adapted from our work [73].

the HyperNN network involves optimizing the weights of each neuron for the weight vectors corresponding to lower bounds, $\mathbf{w}_l$, and length spans, $\mathbf{w}_s$, by minimizing the loss function outlined in Equation 4.6. Notably, we leverage the length spans as trainable parameters instead of the upper bounds. This choice promotes stability during gradient optimization. We hypothesize that optimizing the length span is mathematically more tractable than optimizing the upper bound's location in high-dimensional spaces, which could also be proven experimentally.

To make HyperNN useable as an index-aware classifier, we restrict the feature spaces in which each neuron builds its box. Specifically, for each feature subset $F_1, \dots, F_K$ for which indexes have been preconstructed, we instantiate a neuron. Each neuron learns a box B_κ for $1 \leq \kappa \leq K$ that at maximum has bounded or half-bounded dimensions in F_κ. In the remaining dimensions, the box is unbounded. To reduce the number of range queries, only the boxes containing at least one point of the training set will be used. With this filtering of irrelevant boxes, we can also increase the number of boxes per feature subset to improve the search accuracy without significantly harming the query time. However, this comes at the cost of additional construction time for the entire model since the gradient optimization becomes more computationally expensive when more boxes and weights are considered. This can be partially mitigated by moving the computation to the GPU. Since the underlying neural network structure of HyperNN is built based on the deep learning library PyTorch [93], it benefits from the library's GPU acceleration, enhancing the speed of the compute-intensive gradient optimization. Yet, this requires the search engine to include a GPU, which can substantially increase power consumption.

4.4 Index Structure

For our search engine, K index structures with d-dimensional features of the size N are pre-built. Given the extensive storage demands these indexes can impose, especially with larger datasets, it is critical to address potential memory resource constraints on standard server setups. To mitigate these concerns, we employ a variant of the k-d tree, called *k-d-b tree* [100]. In this variant, the leaf nodes, which contain the embedding vectors of the points and thus account for most of the storage cost, are stored on disk. Conversely, the internal nodes contain only boundary information of the partitions and are kept in memory. The splitting mechanism

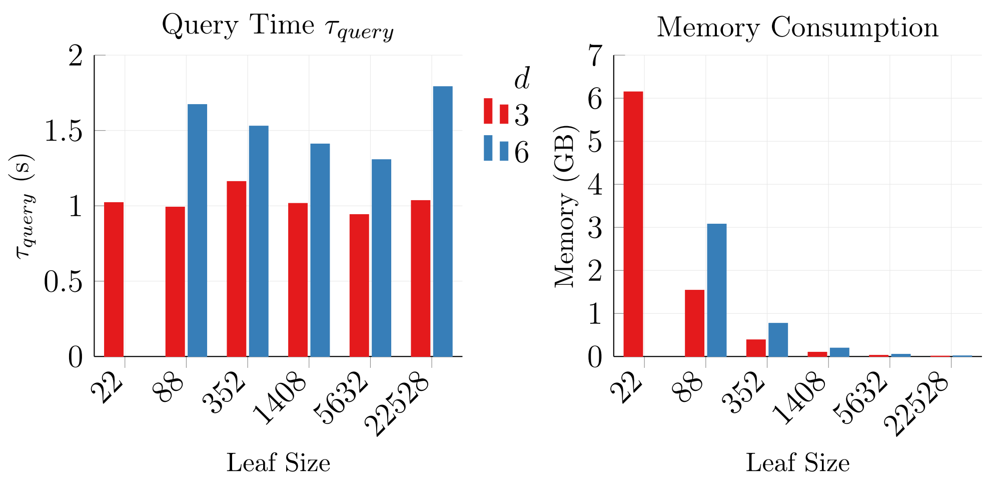


Figure 4.9: Impact of leaf size on query time and memory consumption of a single k-d-b tree. No results for $d = 6$ and leaf size 22 due to exceeding machine memory capacity. Figure adapted from our work [81].

remains the same as in the k-d tree described earlier. Also, the query processing remains similar to that described in Section 3.3.2 for the regular k-d tree, where only the leaves are inspected that intersect with the query rectangle. For the k-d-b tree, this means that only the intersecting leaves are loaded from the disk.

The number of data points stored in a leaf node, often referred to as the *leaf size*, plays a critical role in the efficiency of a k-d-b tree. Smaller leaves enable tighter bounding boxes around data points, which in turn translates to a reduction in false positive points loaded during a range search. The search process examines fewer irrelevant data points, potentially leading to faster retrieval times. While smaller leaves improve search efficiency, they can negatively impact I/O performance. Modern storage systems operate with fixed-size data blocks, also called pages. If leaves are smaller than the page size, accessing them incurs the overhead of reading the entire page, regardless of the actual data volume needed. This can lead to inefficient bandwidth utilization. Furthermore, the reduction in leaf size results in the formation of more complex trees, which in turn necessitates the allocation of additional memory to accommodate the individual trees.

These effects were also demonstrated in our experiments [81], in which we evaluated various leaf size configurations for a single k-d-b tree and compared the query times and memory requirements for multiple range queries extracted from our DBranch models.[5] We made the evaluations for feature subset sizes $d = 3$ and $d = 6$ on an aerial imagery dataset comprising 1,441,557,000 instances, which had been preprocessed by a ResNet101 feature extractor into a feature vector. Our experiments focused on "cold" storage queries, ensuring disk and operating system caches were cleared to prevent caching effects on query time. The results are presented in Figure 4.9. It was demonstrated that the leaf sizes are not only influenced by the system specifics but also by the size of the feature subsets d. Notably, even for large-scale datasets, the memory footprint of a single k-d-b tree remains minimal, with sizes as

[5]All experiments throughout this thesis were conducted on an Ubuntu 18.04 server with 24 *AMD EPYC 7402P* cores, 192 GB DDR4-RAM and 30 TB of NVMe storage.

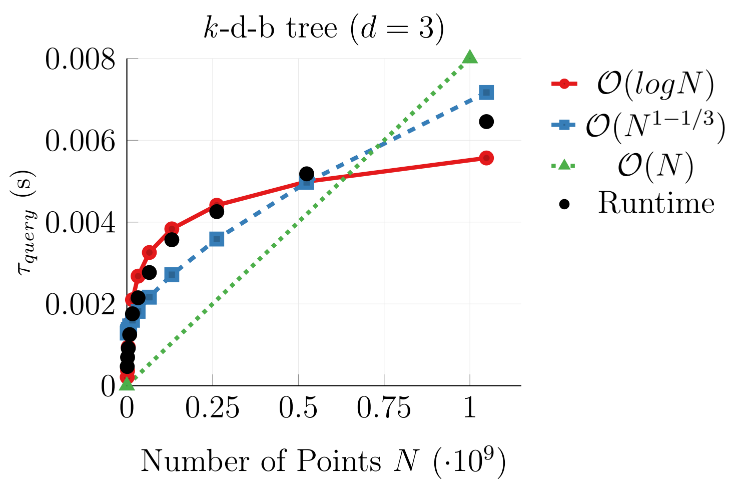


Figure 4.10: Scaling behavior of our k-d-b tree concerning dataset size N compared to asymptotic curves of other complexity classes. Figure adapted from our work [81].

low as 24 MB for $d = 3$ and 48 MB for $d = 6$ for a leaf size of 5,632. This allows us to hold a large number of trees in memory even on commodity hardware.

We further analyzed whether the theoretical sublinear time complexity of range queries over a k-d-b tree can be achieved in practice. To this end, we measured the query time τ_{query} of the k-d-b tree for increasing dataset sizes N using the aerial imagery dataset. These times were then compared to the theoretical asymptotic curves for the logarithmic $\mathcal{O}(\log N)$ and linear $\mathcal{O}(N)$ complexity classes, as well as the time complexity $\mathcal{O}(DN^{1-1/D})$ of a standard k-d tree for orthogonal range search.[6] The results are presented in Figure 4.10, which demonstrates the scaling behavior of the k-d-b tree implementation. Notably, the results fall between the asymptotic curves for $\mathcal{O}(\log N)$ and $\mathcal{O}(DN^{1-1/D})$.

4.5 Concluding Remarks

In the previous sections, we explored the distinct elements that comprise our search-by-classification framework. In the forthcoming discussion, we will present detailed insights from our experimental evaluations and address the observed limitations of the framework.

Experimental Evaluation

In our experiments [81], we demonstrated that our search-by-classification framework performed particularly well on large-scale datasets. The approach was evaluated on the previously described aerial image dataset, comprising 1,441,557,000 instances. The embeddings were extracted with dimensionality $D = 50$ using a fine-tuned ResNet101, resulting in a 600 GB embedding vector. Additionally, a hold-out set of 110,000 instances was labeled for seven classes (chimney, plane, ship, solar panel, storage tank, wind turbine, and others) to evaluate the retrieval quality in terms of F_1-score. Multiple k-d-b tree index structures for $d = 3$ were con-

[6] In our experiments, we kept the number of returned points, q, constant, so q is neglected in the runtime analysis.

structed with different numbers of feature subsets $K \in \{50, 150, 300\}$ and a leaf size of 5,632.

A series of queries was generated for each tested search model on the labeled training set, comprising 30 positive instances of the respective class and 30,000 negative instances. This was done to account for the skew in the class distribution in the dataset, which was largely dominated by the "others" class. In total, we compared different variants of our DBranch models and ensembles (DBEns) with classic scan-based models, including decision trees (DTree), random forests (RForest), and extra trees (ExTrees) as well as a nearest neighbor search baseline (NNB). The NNB reflects the expected classification performance when users utilize a search engine based on NN that accepts only single-instance queries. To ensure a fair comparison, the NNB is treated as a model in which all k nearest neighbors generated in response to a user query are classified as positive. Conversely, all remaining instances are classified as negative. The value of k corresponds to the actual number of positive instances associated with each user query, providing the NNB with a theoretical advantage, since this information is typically not known beforehand. Two methods for NNB were tested, where all D features were used to build a k-d-b tree and only one randomly selected feature subset was pre-built for the decision branches. The search results using all features achieved similar performance in terms of F_1-score at significantly higher query times. Therefore, only the results of the NNB operating on one feature subset were reported. The query time of the NNB search could be even further reduced by employing ANN search at the cost of even lower retrieval quality.

For our DBranch models, we evaluated both versions of the top-down construction phase described earlier, where either all features are considered (denoted with ‡) or only the features where the decision branch is bounded by the features of the associated box (denoted with †). For our ensemble model, we discarded the top-down construction phase as the ensembling itself was already sufficient to achieve satisfactory classification performance, with the advantage of a faster query time. We compared two versions with 5 and 25 models as part of the ensemble, labeled 5t and 25t. In addition to the retrieval accuracy, which was measured by the average F_1-score across all queries, the average time of the entire query execution was also recorded in seconds. This was split into the phases τ_{train} for the training of the classifier, τ_{query} for the retrieval of the positive database objects, and τ_{total} as the sum of both. The results of the experiments are presented in Table 4.1. They demonstrate that our index-aware classifier could achieve similar performance to the classic tree-based classification models while being significantly faster in retrieval. For $K = 300$, we could achieve a speedup in retrieval by a factor $715\times$ for the single decision branches model and $195\times$ for the ensemble model compared to their scan-based counterparts. In comparison to the NNB search, the NNB approach achieved the fastest retrieval performance. However, the query results were significantly worse than those obtained using classification-based approaches.

Limitations and Future Research

The experiments presented in Table 4.1 demonstrate that the set of index structures required for our original search method incurs additional storage costs that exceed the size of the original dataset. It can be observed that, in general, leveraging more

Model	$K = 50 \rightarrow 2.17$ TB				$K = 150 \rightarrow 6.5$ TB				$K = 300 \rightarrow 13$ TB			
	τ_{train}	τ_{query}	τ_{total}	F_1	τ_{train}	τ_{query}	τ_{total}	F_1	τ_{train}	τ_{query}	τ_{total}	F_1
DBranch†	0.310	1.445	1.756	0.801	0.399	1.090	1.489	0.824	0.567	0.892	1.459	0.847
DBranch‡	0.335	16.618	16.953	0.818	0.420	14.685	15.105	0.833	0.672	13.844	14.516	0.854
DTree	0.855	1,043.433	1,044.288	0.829	0.855	1,043.433	1,044.288	0.829	0.855	1,043.433	1,044.288	0.829
NNB	—	0.298	0.298	0.431	—	0.298	0.298	0.431	—	0.298	0.298	0.431
DBEns$_{5t}$	0.529	9.760	10.288	0.895	0.993	5.666	6.658	0.914	1.862	5.156	7.018	0.912
DBEns$_{25t}$	0.891	28.607	29.497	0.915	1.543	22.639	24.182	0.925	2.729	19.716	22.445	0.930
RForest	0.274	1,319.688	1,319.961	0.904	0.274	1,319.688	1,319.961	0.904	0.274	1,319.688	1,319.961	0.904
ExTrees	0.122	1,332.026	1,332.148	0.950	0.122	1,332.026	1,332.148	0.950	0.122	1,332.026	1,332.148	0.950

Table 4.1: Results on the aerial image data set encompassing around 0.6 TB of data. Time is given in seconds. Index size is reported next to each value of K. For each model, we report the measured times of the individual query phases and the retrieval quality in terms of F_1-score. Table adapted from our work [81].

feature subsets for the DBranch models leads to higher retrieval accuracy, but at the cost of increased storage requirements for the pre-built index structures. Therefore, it is necessary to balance the retrieval quality and available storage capacities based on the specific use case. Consequently, we further investigated the potential of reducing the size of the embeddings to lower the storage capacities involved, which amounted to 600 GB of data. To address this challenge, we explored the possibility of reducing the feature vector size while maintaining good retrieval quality through quantization, as described in Section 5.3.

It is important to note that the k-d-b tree index structures that we used are best suited for static data environments. The insertion, update, or deletion of data records would necessitate the rebalancing of each of the K index structures, which would cause additional processing time. Alternative index structures, such as the *bkd-tree* [96], extend upon the original k-d-b tree by optimizing the index structure for bulk updates. This could be relevant for dynamic data scenarios, where the data is constantly changing and minor decreases in the query performance are acceptable. The modular design of our search-by-classification framework facilitates the replacement of the utilized index structure, thereby enabling it to evolve in tandem with the emergence of new developments in multi-dimensional index structures [71].

Chapter 5

Applications

Contributed Material

- **Christian Lülf**, Denis Mayr Lima Martins, Marcos Antonio Vaz Salles, Yongluan Zhou, and Fabian Gieseke. RapidEarth: A search-by-classification engine for large-scale geospatial imagery. In: *Proceedings of the 31st ACM International Conference on Advances in Geographic Information Systems*, 2023. This work was awarded with the Best Demo Award. (see Appendix B.1)
- **Christian Lülf**, Denis Mayr Lima Martins, Marcos Antonio Vaz Salles, Yongluan Zhou, and Fabian Gieseke. CLIP-Branches: Interactive fine-tuning for text-image retrieval. In: *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2024. Accepted (In press). (see Appendix B.2)

Having established a functioning and efficient search-by-classification framework, we proceeded to apply it to real-world scenarios in order to demonstrate its effectiveness. To facilitate its wider applicability to other researchers and their own use cases, we developed a search platform that integrates our search method and is accessible to the public. Based on this platform, we created two prototypes for geospatial imagery and text-based image search. We actively selected use cases, where users are confronted with ever-increasing amounts of data and where approaches to quickly navigate through the data are strongly demanded. Furthermore, while working on the new prototypes, we also addressed the inherent drawbacks of our original proposed search-by-classification framework. This involved reducing the storage overhead of the index structures using quantization and improving the quality of the embeddings by using better feature extractors.

5.1 Open Search-by-Classification Platform

In our work [82], we introduced an open search-by-classification platform. Given a dataset of any multimedia content, such as images or videos, and their corresponding extracted embeddings, the platform provides the necessary software for setting up a search-by-classification system. It consists of three main components namely the *search*, the *data*, and the *web* application. Figure 5.1 shows the overall system architecture of the required components and their interaction during the query processing phase. All of the individual services communicate via pre-defined interfaces using *REST API* calls. These allow for an independent deployment of the individual services on different servers, which makes it suitable to be deployed in cloud environments. Furthermore, since each service is containerized using Docker, each service is highly scalable according to the workload. The required data, consisting of the

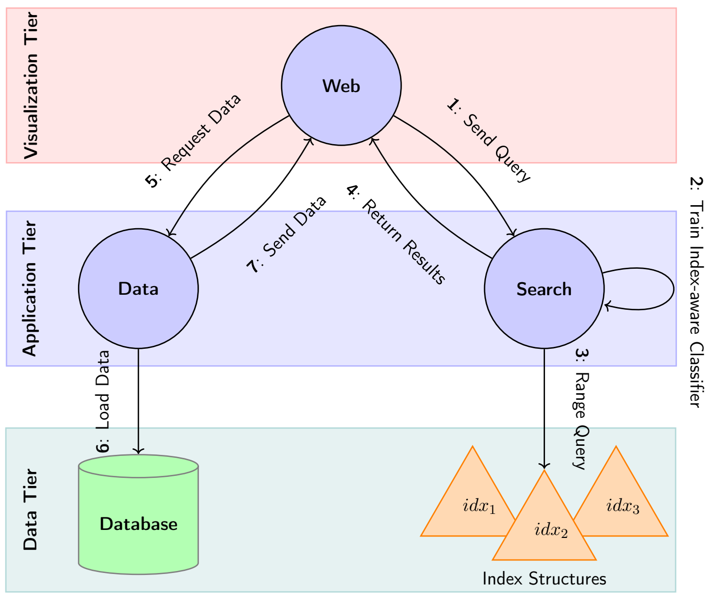


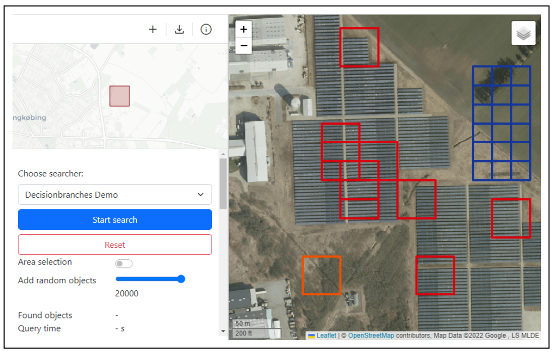
Figure 5.1: System architecture of our search-by-classification platform. It consists of three services: web, data, and search, representing the entire search engine application. The illustrated workflow outlines the internal process followed when a user submits a query to the search engine. Figure adapted from our work [82].

raw data and the index structures, are decoupled from the respective services and can be stored on external storage systems such as network storage, which offer vast storage capacities. The functions of each service within the platform are summarized as follows: the search service contains the actual search logic. It brings all the necessary software required to run the index-aware classifier (either DBranch or HyperNN) and is also responsible for managing the index structures. Prior to queries being processed, the search service must build the index structures. The data service stores and manages the raw multimedia data that is required to return the search results. In the case of an image search engine, it stores the image data and returns the set of images for the query response set. The web service represents the user interface of the search engine, which is provided via a web service. As the web service is use-case specific, it must be customized before it can be set up. For the geospatial search engine, a web map service was used for the web frontend of the search engine while for the text-image search engine, a text interface was presented.

5.2 Geospatial Image Search Engine

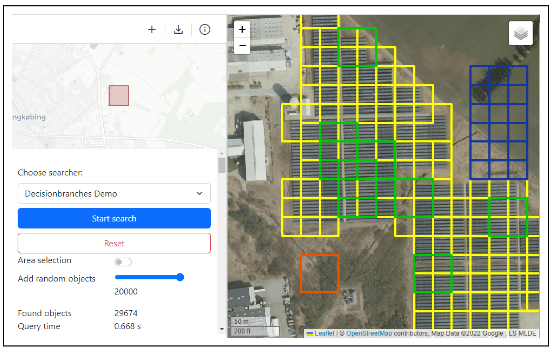
The initial application scenario for our search-by-classification platform was for geospatial imagery. In recent years, there has been a significant increase in the volume of geospatial imagery, a trend that is expected to continue in the future [30]. Earth observation missions such as Sentinel [70] or Landsat [116] provide daily and openly accessible images from the Earth's surface that produce terabytes of data every day. Navigating through these amounts of data becomes a significant burden for users, hindering their ability to generate value from the available data. There-

Legend: 🟥: Positive selected 🟦: Negative selected 🟧: Location of cursor 🟨: Result 🟩: Result + selected


ngkøbing
Leaflet | © OpenStreetMap contributors, Map Data ©2022 Google , LS MLDE
50 m
200 ft

Choose searcher: Decisionbranches Demo
Start search
Reset
Area selection
Add random objects 20000
Found objects -
Query time - s

(a) Define user query


ngkøbing
Leaflet | © OpenStreetMap contributors, Map Data ©2022 Google , LS MLDE
50 m
200 ft

Choose searcher: Decisionbranches Demo
Start search
Reset
Area selection
Add random objects 20000
Found objects 29674
Query time 0.668 s

(b) Retrieve results

Figure 5.2: Demonstration of RapidEarth's web interface. The user defines the query by selecting positive and negative patches on the map. The search returns all patches that fit the query and highlights them on the map. Figure adapted from our work [82].

fore, we proposed *RapidEarth*, a search-by-classification engine for geospatial imagery [82].

As shown in Figure 5.2, the search engine is designed to identify objects of interest in large catalogs of geospatial imagery displayed on a web map. Following the search-by-classification paradigm, users can define their search intent by selecting positive (objects of interest) and negative (objects not of interest) instances to describe their search intent. The web map interface facilitates this process as the user can click on instances on the map to assign a label. Under the hood, the search is then executed using our index-aware classifiers. The resulting instances are then mapped according to their locations on the map, allowing the user to inspect the results and, if necessary, refine them.

Our search engine RapidEarth is built on an aerial dataset from Denmark in the year 2018, with a resolution of 12.5 cm per pixel. The dataset has been divided into a grid of overlapping patches, resulting in 90,429,772 patches for the entire country of Denmark. In order to enhance the feature extraction process, the ResNet-based feature extraction employed in our initial search-by-classification system is replaced with a more sophisticated self-supervised feature extractor, namely DINO [23], which was trained on a random subset of 400,000 image patches.[1]

In conclusion, we were able to establish a functioning prototype search engine for the entire area of Denmark that could respond to queries within seconds due to the pre-built index structures. RapidEarth has demonstrated considerable potential across various applications. For instance, environmentalists seeking to identify areas of deforestation similar to those in the Amazon can leverage RapidEarth to discover previously unnoticed regions. Moreover, the platform could be used for the tracking of animal groups across vast landscapes, such as those found in Africa, by utilizing high-resolution imagery. With all the essential code for RapidEarth made publicly accessible, we encourage the exploration and development of new use cases for this technology.

[1]In this case, leveraging pre-trained weights trained on general-purpose image datasets that are publicly available for DINO did not yield significant benefits during pre-training due to the domain gap between natural images and remote sensing imagery [87].

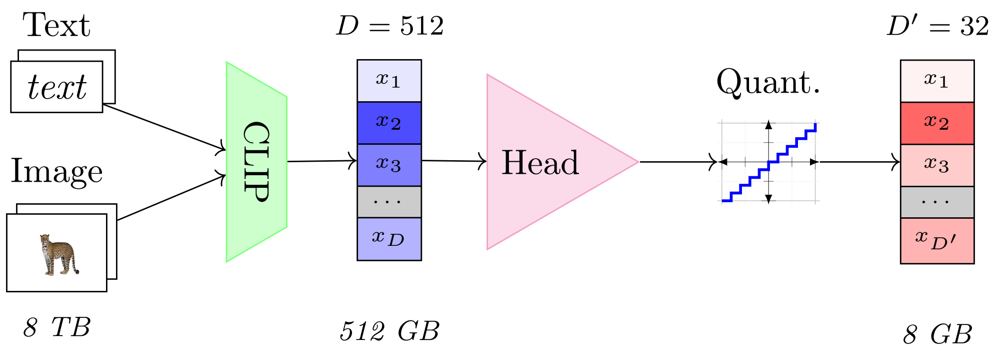


Figure 5.3: Our feature extraction process for CLIP-Branches. The initial text and image data are transformed into 512-dimensional embeddings using CLIP. These embeddings are refined via a custom head module, followed by scalar quantization, resulting in a 32-dimensional 1-byte embedding. Figure adapted from our work [83].

5.3 Text-Image Search Engine

In a second application of our search-by-classification platform, we addressed the text-image search domain. This involves locating images based on textual queries, similar to functionalities seen in platforms such as Google Images, Pinterest, or Flickr. Due to the continually increasing number of images posted on the web, the need for efficient search methods is high. However, we posit that the search quality of the results can even be further optimized to the needs of the user when employing our search method that allows for iterative fine-tuning of the results based on user feedback. To demonstrate a potential application scenario for our platform in text-image search, we proposed our prototype *CLIP-Branches* [83]. As the name implies, we employ the CLIP model (see Section 3.2) as a novel pre-trained foundation for our feature extractor. CLIP is a multi-modal model trained on text-image pairs to create more generalized embeddings. It comprises two individual encoders for images and text, both of which produce embeddings in the same feature space. Thereby, similarities between images and text pairs can be measured by considering only their embeddings. This property provides the foundation for our text-image search engine.

However, our original search-by-classification approach suffered from a large storage overhead, caused by the extracted high-dimensional embedding vectors, which was further multiplied by the utilized index structures. This can become a significant burden for large image databases with billions of instances, where the storage overhead becomes too massive. To reduce the storage requirements while maintaining a satisfactory level of retrieval quality, CLIP-Branches is designed to make the existing embeddings more compact by modifying the inherent CLIP encoders. Our modifications entail two improvements: firstly, the embedding size is reduced by adding a final model head to the encoders with a smaller dimensionality. Secondly, quantization is used to reduce the storage requirements of the embeddings. The improvements to the feature extraction procedure are exemplified in Figure 5.3. The figure also reports the storage savings for the largest employed image dataset, which consists of more than 260 million images and consumes more than 8 TB of storage. The improved feature extraction results in embeddings of the size of 8 GB for all images.

Model Head

To further improve the efficiency of the CLIP feature extractor, custom model heads are appended to the CLIP encoders. These heads consist of new fully-connected layers added at the network's end, effectively reducing the final embedding dimensionality from $D = 512$ to $D' = 32$. To calibrate the new head modules, the models are trained on the MSCOCO Image Captioning (MSCOCO) dataset [75] with 82,612 training samples and 40,438 validation samples containing images with their respective textual descriptions. During this training phase, all the weights of the original CLIP model are frozen to fine-tune the newly added layers exclusively. The training was conducted over 100 epochs under the same settings as the original CLIP model.

Quantization

A specialized regularization term is incorporated into the retraining process of the CLIP model to optimize the embeddings for the intended quantization. This regularization promotes a uniform distribution of feature vectors across the spherical feature space, which enhances the performance of the subsequent scalar quantization process. Scalar quantization is a method of reducing the number of bits required to represent each value in the embedding. This is typically achieved by transforming `float32` values (4 bytes) into `uint8` values (1 byte) by mapping each value to one of 256 possible integer values. To this end, the so-called *Kozachenko-Leononenko* (KoLeo) regularization [103] is adopted. Given a batch $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$ of N embeddings, each within a D-dimensional space, the KoLeo regularization function, denoted as $\mathcal{L}_{KoLeo}$, is formally defined as a mapping $\mathbb{R}^{N \times D} \to \mathbb{R}$:

$$\mathcal{L}_{KoLeo}(X) = -\frac{1}{N} \sum_{i=1}^{N} \log \rho_{X,i}, \tag{5.1}$$

where $\rho_{X,i} = min_{j \neq i} \mathcal{D}_{euclidean}(\mathbf{x}_i, \mathbf{x}_j)$ is the minimal Euclidean distance between $\mathbf{x}_i$ and any other point in the batch X. In total, the embedding size is reduced by a factor of 64 due to the two improvements described above (more precisely, from 512 values with 4-byte precision to 32 values with a 1-byte representation).

Search Engine

The extracted compact text and image embeddings are used to build our new text-image search engine, CLIP-Branches, which is visualized in Figure 5.4. Similar to traditional search engines, users define their search intent by providing a textual description. This text is transformed into an embedding using our feature extractor (Step ❶) to perform an ANN search to retrieve the top k results (Step ❷). However, the initial results presented to the user may not fully satisfy the user's needs (Step ❸), either because they are inaccurate or incomplete. In scenarios where retrieving all relevant results is crucial, and missing even a single important instance is expensive, traditional image search engines with their top k retrieval approach may not be sufficient. Therefore, CLIP-Branches includes a fine-tuning stage where the user can label the initial top k results whether they fit or do not fit the query intent (Step ❹). This trains our index-aware classifiers (Step ❺) to retrieve a more complete and accurate set of instances (Step ❻) efficiently using the pre-built index

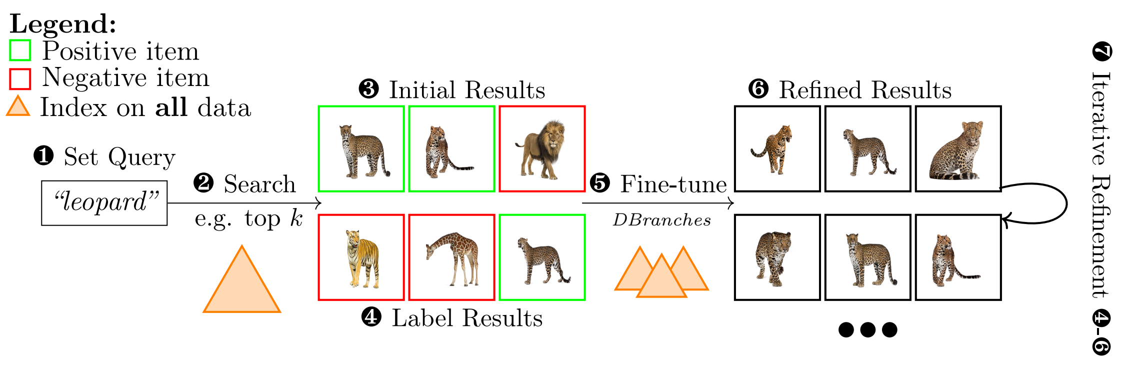


Figure 5.4: Search workflow of CLIP-Branches. Traditional text-to-image search engines typically only employ steps ❶ to ❸ while CLIP-Branches adds a fine-tuning stage to refine the initial search results (Steps ❹-❼). Index structures are employed during the search for faster execution. Figure adapted from our work [83].

structures. This can be further improved by fine-tuning the results over multiple iterations (Step ❼).

Experiments

In our experiments, we demonstrated the effectiveness of CLIP-Branches. Figure 5.5 compares the search accuracy of the fine-tuned results with that of the initial top k results that users would typically obtain from NN-based search engines. Our experiments on multiple benchmark datasets show that our fine-tuned search using index-aware classifiers outperforms typical NN-based search on average after 22 positive instances have been labeled for training for the single DBranch model and 8 for our DBEns model. As a competitor, we used an NN search approach, treating it as a binary classifier. In this context, the k nearest neighbors found in the test set were considered to be positive ($y = 1$), while all other instances were considered to be negative ($y = 0$). Similar to the experiments in Section 4.5, we set k to the true number of positive instances in the test set. Notably, the CIFAR-10 dataset required the most labeled positive examples to outperform the traditional NN search, likely due to the high efficacy of NN search, which presents a substantial hurdle for our method.

Furthermore, we evaluated the performance of various deep feature extraction techniques based on the CLIP model and the impact of quantization on the resulting embeddings. Our experiments utilized the MSCOCO dataset to extract embeddings from an unknown test set using four distinct methods:

(1) **CLIP** ($D = 512$): original CLIP features with dimensionality $D = 512$,

(2) **CLIP + PCA**: PCA is applied to original CLIP features, reducing dimensionality to $D' = 32$,

(3) **CLIP + Head**: embeddings extracted via the CLIP model. The model was extended with a custom model head, which was fine-tuned on the MSCOCO training set, resulting in outputs with a dimensionality of $D' = 32$,

(4) **CLIP + Head + KoLeo**: the custom model head was further fine-tuned to incorporate KoLeo regularization into the loss function.

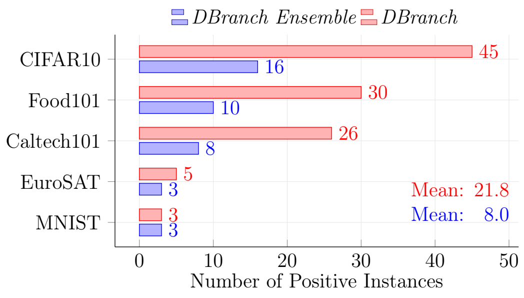


Figure 5.5: Number of positive training instances required for DBranch and DBEns models to surpass NN-based search results among different image benchmark datasets with respect to classification accuracy. Figure adapted from our work [83].

Recall@	MSCOCO 1	10	100	Accuracy
CLIP ($D = 512$)	-	-	-	0.474
CLIP + PCA	0.683	0.757	0.802	0.144
CLIP + Head	0.831	0.890	0.926	0.390
CLIP + Head + KoLeo	0.953	0.970	0.985	0.404

Table 5.1: Comparison of quantization methods applied to raw CLIP features for the MSCOCO dataset, detailing Recall@k scores post-quantization, alongside zero-shot classification accuracy. Each quantization method reduces the feature dimensionality to $D = 32$. Table adapted from our work [83].

To assess the quality of these embeddings, we measured the zero-shot classification accuracy on the test set. In Table 5.1, it is shown that our CLIP + Head + KoLeo embeddings outperformed the other methods with reduced dimensionality in terms of classification accuracy. This indicates that these embeddings retain a substantial portion of the information inherent to the original CLIP features.

Moreover, we evaluated how well the neighborhood structures were preserved after quantization by computing the Recall@k. This metric compares the set of nearest neighbors determined from quantized features to those identified from the same set of unquantized features. A Recall@k value close to one suggests a high overlap, indicating minimal impact from quantization on the corresponding feature set. It should be noted that we report Recall@k values exclusively for the reduced feature sets, as comparing these values with the original CLIP features with dimensionality $D = 512$ would not be appropriate. Table 5.1 illustrates that the features processed with our head and KoLeo regularization preserve neighborhood structures more effectively through quantization compared to those processed using traditional methods (e.g. CLIP + PCA), or those without KoLeo regularization (CLIP + Head).

With the introduction of CLIP-Branches, we successfully demonstrated the effectiveness of our search-by-classification method in text-image search engines. Furthermore, by reducing the size of the embeddings, we have efficiently addressed the inherent storage challenges associated with this technique. These improvements significantly boost performance in resource-constrained environments, positioning our method as a robust alternative to traditional approaches.

Chapter 6

Conclusion and Outlook

The primary objective of this thesis was to develop a co-design approach that integrates machine learning techniques with index structures to enhance the efficacy of large-scale data retrieval. This approach should be further validated through real-world applications to demonstrate its practical relevance.

Summary

The key innovation lies in our search-by-classification method, which uses classification models to frame the retrieval task as a binary classification problem. Instead of relying on a single query item, our approach utilizes multiple positive and negative examples to capture the user's search intent more precisely. With a sufficiently large query set, we could significantly outperform the accuracy of traditional NN-based approaches. To ensure efficient application on massive datasets, we introduced index-aware classifiers. These models seamlessly integrate with pre-built indexes, enabling fast retrieval while maintaining high result accuracy. We developed two different index-aware classification models that can be utilized in our search framework.

The practical value of our framework is showcased through two functional prototypes, RapidEarth and CLIP-Branches, built upon our open search-by-classification platform. The underlying software framework allows for straightforward adaptation to diverse use cases, fostering broader adoption and facilitating future research endeavors.

Limitations and Further Research

While our proposed method has shown promising results, we also observed some limitations:

User Engagement: Although accurate, our search-by-classification method requires users to create large query sets, which can be time-consuming, especially when data access is limited or the items being searched are rare. To mitigate this, we explored potential solutions in subsequent works. In RapidEarth, users are assisted in setting up initial query sets using a map interface, enabling faster navigation through the data. CLIP-Branches employs a hybrid search engine, combining traditional NN-based search with our search-by-classification framework. The traditional NN search identifies a preliminary set of potential candidates that match a text query, which is then refined by our search-by-classification framework. This hybrid structure, where our search-by-classification framework acts as a refinement option, exemplifies a novel application scenario, particularly advantageous where high precision and recall of results are crucial.

Storage Overhead: The pre-built index structures in our search framework result in some storage overhead. To mitigate this, the storage size of the embeddings extracted from the raw data can be reduced through fine-tuning and quantization. This significantly reduces the storage requirements of the entire search platform, enabling deployment on commodity hardware even for large-scale settings. However, the extra overhead of the index structures has not been entirely eliminated. Further research could explore ways to reduce the storage overhead of the index structures. One approach would be to build smaller sets of index structures while maintaining similar retrieval accuracy. Another approach could involve integrating compression techniques into the index structures to avoid storing the entire embeddings of the points.

Static Datasets: Our framework is currently optimized for static data settings, where modifications in the underlying dataset are not efficiently supported. The k-d-b tree used is not optimized for dynamic scenarios. However, our search platform's modular architecture allows for replacing the utilized index structures with any other existing method that supports orthogonal range queries. One such method is the bkd-tree, which extends the regular k-d-b tree by enabling more efficient processing of database updates.

Outlook

Widening the scope, there is potential to generalize the co-design approach between machine learning models and index structures to further enhance search-by-classification systems. Our research has already demonstrated the synergy between tree-based models and index structures. Future developments could explore the integration of various indexing techniques into the construction of classification models. For instance, an index-aware classifier constrained by the cells produced by product quantization could benefit from the large storage savings due to the compact codes used to store elements instead of full embedding vectors. As a potential machine learning model, we have already demonstrated with HyperNN that neural networks with their adaptable architecture and customizable parameters are promising candidates for embodying these future machine learning models.

The contributions of this thesis provide several insights at the intersection of machine learning and information retrieval, rethinking the idea of efficient application of machine learning models. By employing a co-design approach that integrates machine learning models with index structures, this work not only enhances the efficiency of data retrieval systems but also opens up promising new research directions. Given their high practical relevance, the proposed methods have the potential for implementation in real-world applications in the future.

Bibliography

[1] A. Andoni and D. Beaglehole. Learning to hash robustly, guaranteed. In *International Conference on Machine Learning*, pages 599–618, 2022.

[2] A. Andoni, P. Indyk, T. Laarhoven, I. Razenshteyn, and L. Schmidt. Practical and optimal LSH for angular distance. In *Proceedings of the 28th International Conference on Neural Information Processing Systems*, pages 1225–1233, 2015.

[3] A. Andoni, P. Indyk, and I. Razenshteyn. Approximate nearest neighbor search in high dimensions. In *Proceedings of the International Congress of Mathematicians*, pages 3287–3318, 2018.

[4] A. Andoni and I. Razenshteyn. Optimal data-dependent hashing for approximate near neighbors. In *Proceedings of the Forty-Seventh Annual ACM Symposium on Theory of Computing*, pages 793–801, 2015.

[5] R. Arandjelović and A. Zisserman. Three things everyone should know to improve object retrieval. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2911–2918, 2012.

[6] S. Arya, T. Malamatos, and D. M. Mount. Space-time tradeoffs for approximate nearest neighbor searching. *Journal of the ACM*, 57(1):1–54, 2009.

[7] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM*, 45(6):891–923, 1998.

[8] M. Aumüller, E. Bernhardsson, and A. Faithfull. ANN-bench-marks: A benchmarking tool for approximate nearest neighbor algorithms. In *International Conference on Similarity Search and Applications*, pages 34–49, 2017.

[9] J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[10] A. Babenko and V. Lempitsky. The inverted multi-index. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(6):1247–1260, 2015.

[11] R. A. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval.* Addison-Wesley Longman Publishing Co., Inc., 1999.

[12] S. Bai and X. Bai. Sparse contextual activation for efficient visual re-ranking. *IEEE Transactions on Image Processing*, 25(3):1056–1069, 2016.

[13] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. F. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner,

Çaglar Gülçehre, H. F. Song, A. J. Ballard, J. Gilmer, G. E. Dahl, A. Vaswani, K. R. Allen, C. Nash, V. Langston, C. Dyer, N. M. O. Heess, D. Wierstra, P. Kohli, M. M. Botvinick, O. Vinyals, Y. Li, and R. Pascanu. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

[14] J. S. Beis and D. G. Lowe. Shape indexing using approximate nearest-neighbour search in high-dimensional spaces. In *Proceedings of IEEE computer society conference on computer vision and pattern recognition*, pages 1000–1006, 1997.

[15] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[16] J. L. Bentley. Decomposable searching problems. *Information Processing Letters*, 8(5):244–251, 1979.

[17] A. Beygelzimer, S. Kakade, and J. Langford. Cover trees for nearest neighbor. In *Proceedings of the 23rd International Conference on Machine Learning*, pages 97–104, 2006.

[18] M. Blum, R. W. Floyd, V. Pratt, R. L. Rivest, and R. E. Tarjan. Time bounds for selection. *Journal of Computer and System Sciences*, 7(4):448–461, 1973.

[19] C. Böhm, S. Berchtold, and D. A. Keim. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Comput. Surv.*, 33(3):322–373, 2001.

[20] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[21] L. Breiman, J. Friedman, C. Stone, and R. Olshen. *Classification and Regression Trees.* Taylor & Francis, 1984.

[22] Y. Cao, H. Wang, C. Wang, Z. Li, L. Zhang, and L. Zhang. MindFinder: interactive sketch-based image search on millions of images. In *Proceedings of the 18th ACM International Conference on Multimedia*, pages 1605–1608, 2010.

[23] M. Caron, H. Touvron, I. Misra, H. Jégou, J. Mairal, P. Bojanowski, and A. Joulin. Emerging properties in self-supervised vision transformers. In *Proceedings of the International Conference on Computer Vision (ICCV)*, pages 9630–9640, 2021.

[24] L. Cayton. Fast nearest neighbor retrieval for bregman divergences. In *Proceedings of the 25th International Conference on Machine Learning*, pages 112–119, 2008.

[25] G. Chechik, V. Sharma, U. Shalit, and S. Bengio. Large scale online learning of image similarity through ranking. *Journal of Machine Learning Research*, 11(36):1109–1135, 2010.

[26] T. Chen and C. Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794, 2016.

[27] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton. A simple framework for contrastive learning of visual representations. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119, pages 1597–1607, 2020.

[28] W. Chen, Y. Liu, W. Wang, E. M. Bakker, T. Georgiou, P. W. Fieguth, L. Liu, and M. S. Lew. Deep learning for instance retrieval: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45:7270–7292, 2021.

[29] X. Chen, S. Xie, and K. He. An empirical study of training self-supervised vision transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9640–9649, 2021.

[30] G. Cheng, X. Xie, J. Han, L. Guo, and G.-S. Xia. Remote sensing image scene classification meets deep learning: Challenges, methods, benchmarks, and opportunities. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 13:3735–3756, 2020.

[31] O. Chum, J. Philbin, J. Sivic, M. Isard, and A. Zisserman. Total recall: Automatic query expansion with a generative feature model for object retrieval. In *2007 IEEE 11th International Conference on Computer Vision*, pages 1–8, 2007.

[32] A. Creswell, T. White, V. Dumoulin, K. Arulkumaran, B. Sengupta, and A. Bharath. Generative adversarial networks: An overview. *IEEE signal processing magazine*, 35(1):53–65, 2018.

[33] M. De Berg. *Computational Geometry: Algorithms and Applications*. Springer Science & Business Media, 2000.

[34] M. P. Deisenroth, A. A. Faisal, and C. S. Ong. *Mathematics for Machine Learning*. Cambridge University Press, 2020.

[35] W. Dong, C. Moses, and K. Li. Efficient k-nearest neighbor graph construction for generic similarity measures. In *Proceedings of the 20th International Conference on World Wide Web*, pages 577–586, 2011.

[36] Y. Dong, P. Indyk, I. P. Razenshteyn, and T. Wagner. Learning space partitions for nearest neighbor search. In *International Conference on Learning Representations*, 2019.

[37] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *9th International Conference on Learning Representations*, 2021.

[38] M. Douze, A. Guzhva, C. Deng, J. Johnson, G. Szilvasy, P.-E. Mazaré, M. Lomeli, L. Hosseini, and H. Jégou. The Faiss library. *arXiv preprint arXiv:2401.08281*, 2024.

[39] H. Edelsbrunner. *Algorithms in Combinatorial Geometry*. Springer Publishing Company, Inc., 1st edition, 2012.

[40] C. Faloutsos, R. Barber, M. Flickner, J. Hafner, W. Niblack, D. Petkovic, and W. Equitz. Efficient and effective querying by image content. *Journal of Intelligent Information Systems*, 3:231–262, 1994.

[41] M. Fernández-Delgado, E. Cernadas, S. Barro, and D. Amorim. Do we need hundreds of classifiers to solve real world classification problems? *Journal of Machine Learning Research*, 15:3133–3181, 2014.

[42] J. H. Friedman, J. L. Bentley, and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Math. Softw.*, 3(3):209–226, 1977.

[43] J. H. Friedman and N. I. Fisher. Bump hunting in high-dimensional data. *Statistics and computing*, 9(2):123–143, 1999.

[44] T. Ge, K. He, Q. Ke, and J. Sun. Optimized product quantization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(4):744–755, 2014.

[45] A. Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow.* O'Reilly Media, 2022.

[46] P. Geurts, D. Ernst, and L. Wehenkel. Extremely randomized trees. *Machine Learning*, 63(1):3–42, 2006.

[47] F. Gieseke, S. Bloemen, C. van den Bogaard, T. Heskes, J. Kindler, R. A. Scalzo, V. A. Ribeiro, J. van Roestel, P. J. Groot, F. Yuan, et al. Convolutional neural networks for transient candidate vetting in large-scale surveys. *Monthly Notices of the Royal Astronomical Society*, 472(3):3101–3114, 2017.

[48] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning.* MIT Press, 2016.

[49] R. Gray. Vector quantization. *IEEE ASSP Magazine*, 1(2):4–29, 1984.

[50] R. Gray and D. Neuhoff. Quantization. *IEEE Transactions on Information Theory*, 44(6):2325–2383, 1998.

[51] J.-B. Grill, F. Strub, F. Altché, C. Tallec, P. H. Richemond, E. Buchatskaya, C. Doersch, B. A. Pires, Z. D. Guo, M. G. Azar, B. Piot, K. Kavukcuoglu, R. Munos, and M. Valko. Bootstrap your own latent a new approach to self-supervised learning. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, volume 33, pages 21271–21284, 2020.

[52] G. Gutiérrez and J. R. Paramá. Finding the largest empty rectangle containing only a query point in large multidimensional databases. In *International Conference on Statistical and Scientific Database Management*, 2012.

[53] T. Hastie, R. Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning.* Springer, 2009.

[54] K. He, X. Chen, S. Xie, Y. Li, P. Dollár, and R. Girshick. Masked autoencoders are scalable vision learners. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 15979–15988, 2022.

[55] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[56] D. Hendrycks and K. Gimpel. Gaussian error linear units (GELUs). *arXiv preprint arXiv:1606.08415*, 2016.

[57] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, pages 604–613, 1998.

[58] E. Jang, S. Gu, and B. Poole. Categorical reparameterization with gumbel-softmax. In *International Conference on Learning Representations*, 2017.

[59] H. Jegou, C. Schmid, H. Harzallah, and J. Verbeek. Accurate image search using the contextual dissimilarity measure. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(1):2–11, 2010.

[60] J. Johnson, M. Douze, and H. Jégou. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547, 2019.

[61] I. T. Jolliffe and J. Cadima. Principal component analysis: a review and recent developments. *Philosophical transactions of the royal society A: Mathematical, Physical and Engineering Sciences*, 374(2065):20150202, 2016.

[62] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko, et al. Highly accurate protein structure prediction with AlphaFold. *Nature*, 596(7873):583–589, 2021.

[63] H. Jégou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):117–128, 2011.

[64] S. Khan, M. Naseer, M. Hayat, S. W. Zamir, F. S. Khan, and M. Shah. Transformers in Vision: A Survey. *ACM Comput. Surv.*, 54(10s), 2022.

[65] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo, P. Dollár, and R. Girshick. Segment anything. *arXiv preprint arXiv:2304.02643*, 2023.

[66] T. Kraska, A. Beutel, E. H. Chi, J. Dean, and N. Polyzotis. The case for learned index structures. In *Proceedings of the 2018 International Conference on Management of Data*, pages 489–504, 2018.

[67] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[68] D. T. Lee and C. K. Wong. Worst-case analysis for region and partial region searches in multidimensional binary search trees and balanced quad trees. *Acta Inf.*, 9(1):23–29, 1977.

[69] J. Leskovec, A. Rajaraman, and J. D. Ullman. *Mining of Massive Datasets*. Cambridge University Press, 2nd edition, 2014.

[70] J. Li and D. P. Roy. A global analysis of sentinel-2a/2b and landsat-8 data revisit intervals and implications for terrestrial monitoring. *Remote Sensing*, 9(902), 2017.

[71] M. Li, H. Wang, H. Dai, M. Li, R. Gu, F. Chen, Z. Chen, S. Li, Q. Liu, and G. Chen. A survey of multi-dimensional indexes: Past and future trends. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–20, 2024.

[72] W. Li, Y. Zhang, Y. Sun, W. Wang, M. Li, W. Zhang, and X. Lin. Approximate nearest neighbor search on high dimensional data — experiments, analyses, and improvement. *IEEE Transactions on Knowledge and Data Engineering*, 32(8):1475–1488, 2020.

[73] D. M. Lima Martins, C. Lülf, and F. Gieseke. End-to-end neural network training for hyperbox-based classification. In *31st European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, ESANN*, 2023.

[74] D. M. Lima Martins, C. Lülf, and F. Gieseke. Training neural networks end-to-end for hyperbox-based classification. *Neurocomputing*, 2024. Under Review.

[75] T.-Y. Lin, M. Maire, S. J. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft COCO: Common objects in context. In *European Conference on Computer Vision*, pages 740–755, 2014.

[76] H. Liu, D. Tam, M. Muqeeth, J. Mohta, T. Huang, M. Bansal, and C. A. Raffel. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. In *Advances in Neural Information Processing Systems*, volume 35, pages 1950–1965, 2022.

[77] T.-Y. Liu. *Learning to Rank for Information Retrieval*, volume 3, page 225–331. Now Publishers Inc., 2009.

[78] X. Liu, F. Zhang, Z. Hou, L. Mian, Z. Wang, J. Zhang, and J. Tang. Self-supervised learning: Generative or contrastive. *IEEE Transactions on Knowledge and Data Engineering*, 35(1):857–876, 2023.

[79] S. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.

[80] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60:91–110, 2004.

[81] C. Lülf, D. M. L. Martins, M. A. V. Salles, Y. Zhou, and F. Gieseke. Fast search-by-classification for large-scale databases using index-aware decision trees and random forests. In *Proceedings of the VLDB Endowment*, pages 2845–2857, 2023.

[82] C. Lülf, D. M. L. Martins, M. A. V. Salles, Y. Zhou, and F. Gieseke. RapidEarth: A search-by-classification engine for large-scale geospatial imagery. In *Proceedings of the 31st ACM International Conference on Advances in Geographic Information Systems*, 2023.

[83] C. Lülf, D. M. L. Martins, M. A. V. Salles, Y. Zhou, and F. Gieseke. CLIP-Branches: Interactive fine-tuning for text-image retrieval. In *Proceedings of*

the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval, 2024. Accepted (In press).

[84] Y. A. Malkov and D. A. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Trans. Pattern Anal. Mach. Intell.*, 42(4):824–836, 2020.

[85] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval.* Cambridge University Press, USA, 2008.

[86] V. Marx. The big challenges of big data. *Nature*, 498(7453):255–260, 2013.

[87] M. Mendieta, B. Han, X. Shi, Y. Zhu, and C. Chen. Towards geospatial foundation models via continual pretraining. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 16806–16816, 2023.

[88] R. Motwani, A. Naor, and R. Panigrahy. Lower bounds on locality sensitive hashing. *SIAM Journal on Discrete Mathematics*, 21(4):930–935, 2008.

[89] M. Muja and D. G. Lowe. Scalable nearest neighbor algorithms for high dimensional data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(11):2227–2240, 2014.

[90] M. Naeem, T. Jamal, J. Diaz-Martinez, S. A. Butt, N. Montesano, M. I. Tariq, E. De-la Hoz-Franco, and E. De-La-Hoz-Valdiris. Trends and future perspective challenges in big data. In *Advances in Intelligent Data Analysis and Applications*, pages 309–325, 2022.

[91] H. Noh, A. F. de Araújo, J. Sim, T. Weyand, and B. Han. Large-scale image retrieval with attentive deep local features. *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 3476–3485, 2016.

[92] M. Oquab, T. Darcet, T. Moutakanni, H. V. Vo, M. Szafraniec, V. Khalidov, P. Fernandez, D. Haziza, F. Massa, A. El-Nouby, R. Howes, P.-Y. Huang, H. Xu, V. Sharma, S.-W. Li, W. Galuba, M. Rabbat, M. Assran, N. Ballas, G. Synnaeve, I. Misra, H. Jegou, J. Mairal, P. Labatut, A. Joulin, and P. Bojanowski. DINOv2: Learning robust visual features without supervision. *arXiv preprint arXiv:2304.07193*, 2023.

[93] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. PyTorch: an imperative style, high-performance deep learning library. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, 2019.

[94] L. Paulevé, H. Jégou, and L. Amsaleg. Locality sensitive hashing: A comparison of hash function types and querying mechanisms. *Pattern Recognition Letters*, 31(11):1348–1358, 2010.

[95] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman. Object retrieval with large vocabularies and fast spatial matching. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2007.

[96] O. Procopiuc, P. K. Agarwal, L. Arge, and J. S. Vitter. Bkd-tree: A dynamic scalable kd-tree. In *Advances in Spatial and Temporal Databases*, volume 2750, pages 46–65, 2003.

[97] D. Qin, S. Gammeter, L. Bossard, T. Quack, and L. van Gool. Hello neighbor: Accurate object retrieval with k-reciprocal nearest neighbors. In *Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition*, pages 777–784, 2011.

[98] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, pages 8748–8763, 2021.

[99] Y. Ro and J. Y. Choi. Autolr: Layer-wise pruning and auto-tuning of learning rates in fine-tuning of deep networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 2486–2494, 2021.

[100] J. T. Robinson. The k-d-b-tree: a search structure for large multidimensional dynamic indexes. In *Proceedings of the 1981 ACM SIGMOD International Conference on Management of Data*, pages 10–18, 1981.

[101] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[102] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet large scale visual recognition challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[103] A. Sablayrolles, M. Douze, C. Schmid, and H. Jégou. Spreading vectors for similarity search. In *7th International Conference on Learning Representations, ICLR 2019*, 2019.

[104] G. Salton. *The SMART Retrieval System—Experiments in Automatic Document Processing.* Prentice-Hall, Inc., 1971.

[105] M. S. Sarfraz, A. Schumann, A. Eberle, and R. Stiefelhagen. A pose-sensitive embedding for person re-identification with expanded cross neighborhood reranking. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 420–429, 2018.

[106] M. B. Sariyildiz, Y. Kalantidis, D. Larlus, and K. Alahari. Concept generalization in visual representation learning. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 9609–9619, 2021.

[107] M. Shanahan. Talking about Large Language Models. *Commun. ACM*, 67(2):68–79, 2024.

[108] H. V. Simhadri, G. Williams, M. Aumüller, M. Douze, A. Babenko, D. Baranchuk, Q. Chen, L. Hosseini, R. Krishnaswamny, G. Srinivasa, S. J. Subramanya, and J. Wang. Results of the NeurIPS'21 Challenge on Billion-Scale Approximate Nearest Neighbor Search. In *Proceedings of the NeurIPS 2021 Competitions and Demonstrations Track*, pages 177–189, 2022.

[109] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

[110] M. Sipser. *Introduction to the Theory of Computation*. International Thomson Publishing, 1st edition, 1996.

[111] A. Smeulders, M. Worring, S. Santini, A. Gupta, and R. Jain. Content-based image retrieval at the end of the early years. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(12):1349–1380, 2000.

[112] J. R. Smith. Riding the multimedia big data wave. In *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1–2, 2013.

[113] C. Sun, A. Shrivastava, S. Singh, and A. Gupta. Revisiting unreasonable effectiveness of data in deep learning era. In *Proceedings of the IEEE international conference on computer vision*, pages 843–852, 2017.

[114] D. Tao, X. Tang, X. Li, and X. Wu. Asymmetric bagging and random subspace for support vector machines-based relevance feedback in image retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(7):1088–1099, 2006.

[115] T. Tuytelaars and C. Schmid. Vector quantizing feature space with a regular lattice. In *2007 IEEE 11th International Conference on Computer Vision*, pages 1–8, 2007.

[116] U.S. Geological Survey. Landsat next. Fact Sheet 2024-3005, U.S. Geological Survey, Reston, VA, 2024. Report.

[117] J. E. Van Engelen and H. H. Hoos. A survey on semi-supervised learning. *Machine learning*, 109(2):373–440, 2020.

[118] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, page 6000–6010, 2017.

[119] J. Wan, D. Wang, S. C. H. Hoi, P. Wu, J. Zhu, Y. Zhang, and J. Li. Deep learning for content-based image retrieval: A comprehensive study. In *Proceedings of the 22nd ACM International Conference on Multimedia*, pages 157–166. Association for Computing Machinery, 2014.

[120] J. Wang, T. Zhang, j. song, N. Sebe, and H. T. Shen. A survey on learning to hash. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(4):769–790, 2018.

[121] M. Wang, X. Xu, Q. Yue, and Y. Wang. A comprehensive survey and experimental comparison of graph-based approximate nearest neighbor search. *Proc. VLDB Endow.*, 14(11):1964–1978, 2021.

[122] Y. Wang, X. Lin, L. Wu, and W. Zhang. Effective Multi-Query Expansions: Collaborative Deep Networks for Robust Landmark Retrieval. *IEEE Transactions on Image Processing*, 26:1393–1404, 2017.

[123] C. Wongkham, B. Lu, C. Liu, Z. Zhong, E. Lo, and T. Wang. Are updatable learned indexes ready? *Proc. VLDB Endow.*, 15(11):3004–3017, 2022.

[124] M. Wortsman, G. Ilharco, S. Y. Gadre, R. Roelofs, R. Gontijo-Lopes, A. S. Morcos, H. Namkoong, A. Farhadi, Y. Carmon, S. Kornblith, and L. Schmidt. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162, pages 23965–23998, 2022.

[125] H. Xu, J. Wang, X.-S. Hua, and S. Li. Image search by concept map. In *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 275–282, 2010.

[126] H. Xuan, R. Souvenir, and R. Pless. Deep randomized ensembles for metric learning. In *The European Conference on Computer Vision (ECCV)*, pages 751–762, 2018.

[127] I. Z. Yalniz, H. Jégou, K. Chen, M. Paluri, and D. K. Mahajan. Billion-scale semi-supervised learning for image classification. *arXiv preprint arXiv:1905.00546*, 2019.

[128] A. B. Yandex and V. Lempitsky. Aggregating Local Deep Features for Image Retrieval. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1269–1277, 2015.

[129] A. Zhai, D. Kislyuk, Y. Jing, M. Feng, E. Tzeng, J. Donahue, Y. L. Du, and T. Darrell. Visual discovery at pinterest. In *Proceedings of the 26th International Conference on World Wide Web Companion, WWW*, pages 515–524, 2017.

[130] H. Zhang, Z.-J. Zha, Y. Yang, S. Yan, Y. Gao, and T.-S. Chua. Attribute-augmented semantic hierarchy: towards bridging semantic gap and intention gap in image retrieval. In *Proceedings of the 21st ACM International Conference on Multimedia*, pages 33–42, 2013.

[131] L. Zheng, S. Wang, L. Tian, F. He, Z. Liu, and Q. Tian. Query-adaptive late fusion for image search and person re-identification. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1741–1750, 2015.

[132] L. Zheng, Y. Yang, and Q. Tian. SIFT meets CNN: A decade survey of instance retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(5):1224–1244, 2018.

[133] Z. Zhong, L. Zheng, D. Cao, and S. Li. Re-ranking Person Re-identification with k-Reciprocal Encoding. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3652–3661, 2017.

[134] W. Zhou, H. Li, and Q. Tian. Recent Advance in Content-based Image Retrieval: A Literature Survey. *arXiv preprint arXiv:1706.06064*, 2017.

[135] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He. A comprehensive survey on transfer learning. *Proceedings of the IEEE*, 109:43–76, 2019.

Appendix:
Contributed Publications

Appendix A

Efficient Search-by-Classification

A.1 Fast Search-by-Classification for Large-Scale Databases Using Index-Aware Decision Trees and Random Forests

Christian Lülf, Denis Mayr Lima Martins, Marcos Antonio Vaz Salles, Yongluan Zhou, and Fabian Gieseke. Fast search-by-classification for large-scale databases using index-aware decision trees and random forests. In *Proceedings of the VLDB Endowment*, pages 2845–2857, 2023.

Abstract: The vast amounts of data collected in various domains pose great challenges to modern data exploration and analysis. To find "interesting" objects in large databases, users typically define a query using positive and negative example objects and train a classification model to identify the objects of interest in the entire data catalog. However, this approach requires a scan of all the data to apply the classification model to each instance in the data catalog, making this method prohibitively expensive to be employed in large-scale databases serving many users and queries interactively. In this work, we propose a novel framework for such search-by-classification scenarios that allows users to interactively search for target objects by specifying queries through a small set of positive and negative examples. Unlike previous approaches, our framework can rapidly answer such queries at low cost without scanning the entire database. Our framework is based on an index-aware construction scheme for decision trees and random forests that transforms the inference phase of these classification models into a set of range queries, which in turn can be efficiently executed by leveraging multidimensional indexing structures. Our experiments show that queries over large data catalogs with hundreds of millions of objects can be processed in a few seconds using a single server, compared to hours needed by classical scanning-based approaches.

A.2 End-to-End Neural Network Training for Hyperbox-Based Classification

Denis Mayr Lima Martins, **Christian Lülf**, and Fabian Gieseke. End-to-end neural network training for hyperbox-based classification. In *31st European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, ESANN*, 2023.

Abstract: Hyperbox-based classification has been seen as a promising technique in which decisions on the data are represented as a series of orthogonal, multidimensional boxes (i.e., hyperboxes) that are often interpretable and human-readable. However, existing methods are no longer capable of efficiently handling the increasing volume of data many application domains face nowadays. We address this gap by proposing a novel, fully differentiable framework for hyperbox-based classification via neural networks. In contrast to previous work, our hyperbox models can be efficiently trained in an end-to-end fashion, which leads to significantly reduced training times and superior classification results.

A.3 Training Neural Networks End-to-End for Hyperbox-Based Classification

Denis Mayr Lima Martins, **Christian Lülf**, and Fabian Gieseke. Training neural networks end-to-end for hyperbox-based classification. In: *Neurocomputing.* 2024. Under Review.

Abstract: Modern decision-making requires the use of powerful algorithms to make sense of a variety of data. In this context, hyperbox induction has been seen as a promising technique in which decisions on the data are represented as a series of orthogonal, multidimensional boxes (i.e., hyperboxes) that are often interpretable and human-readable. However, existing hyperbox induction methods are no longer capable of efficiently handling the increasing volumes of data many application domains are confronted with. Moreover, current methods offer little to no control on specific properties of the induced box models, such as the number or the sizes of the hyperboxes. In this work, we propose a novel, fully differentiable framework for hyperbox induction that makes use of recent advancements in neural networks. In contrast to existing approaches, our hyperbox-based models can be trained in an end-to-end fashion, which leads to significantly reduced training times and superior classification results.

APPENDIX B

APPLICATIONS

B.1 RapidEarth: A Search-by-Classification Engine for Large-Scale Geospatial Imagery

Christian Lülf, Denis Mayr Lima Martins, Marcos Antonio Vaz Salles, Yongluan Zhou, and Fabian Gieseke. RapidEarth: A search-by-classification engine for large-scale geospatial imagery. In: *Proceedings of the 31st ACM International Conference on Advances in Geographic Information Systems*, 2023. This work was awarded with the Best Demo Award.

Abstract: Data exploration and analysis in various domains often necessitate the search for specific objects in massive databases. A common search strategy, often known as search-by-classification, resorts to training machine learning models on small sets of positive and negative samples and to performing inference on the entire database to discover additional objects of interest. While such an approach often yields very good results in terms of classification performance, the entire database usually needs to be scanned, a process that can easily take several hours even for medium-sized data catalogs. In this work, we present RapidEarth, a geospatial search-by-classification engine that allows analysts to rapidly search for interesting objects in very large data collections of satellite imagery in a matter of seconds, without the need to scan the entire data catalog. RapidEarth embodies a co-design of multidimensional indexing structures and decision branches, a recently proposed variant of classical decision trees. These decision branches allow RapidEarth to transform the inference phase into a set of range queries, which can be efficiently processed by leveraging the aforementioned multidimensional indexing structures. The main contribution of this work is a geospatial search engine that implements these technical findings.

B.2 CLIP-Branches: Interactive Fine-Tuning for Text-Image Retrieval

Christian Lülf, Denis Mayr Lima Martins, Marcos Antonio Vaz Salles, Yongluan Zhou, and Fabian Gieseke. CLIP-Branches: Interactive fine-tuning for text-image retrieval. In: *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2024. Accepted (In press).

Abstract: The advent of text-image models, most notably CLIP, has significantly transformed the landscape of information retrieval. These models enable the fusion of various modalities, such as text and images. One significant outcome of CLIP is its capability to allow users to search for images using text as a query, as well as vice versa. This is achieved via a joint embedding of images and text data that can, for instance, be used to search for similar items. Despite efficient query processing techniques such as approximate nearest neighbor search, the results may lack precision and completeness. We introduce CLIP-Branches, a novel text-image search engine built upon the CLIP architecture. Our approach enhances traditional text-image search engines by incorporating an interactive fine-tuning phase, which allows the user to further concretize the search query by iteratively defining positive and negative examples. Our framework involves training a classification model given the additional user feedback and essentially outputs all positively classified instances of the entire data catalog. By building upon recent techniques, this inference phase, however, is not implemented by scanning the entire data catalog, but by employing efficient index structures pre-built for the data. Our results show that the fine-tuned results can improve the initial search outputs in terms of relevance and accuracy while maintaining swift response times.