10-1-2021

Content-Based Image Retrieval using Hierarchical Decomposition of Feature Descriptors

Eisa Adil
University of Windsor

Content-Based Image Retrieval using Hierarchical Decomposition of Feature Descriptors

By

Eisa Adil

A Thesis
Submitted to the Faculty of Graduate Studies
through the School of Computer Science
in Partial Fulfilment of the Requirements for
the Degree of Master of Science at the
University of Windsor

Windsor, Ontario, Canada

Content-Based Image Retrieval using Hierarchical Decomposition of Feature Descriptors

by

Eisa Adil

APPROVED BY:

M. Hlynka
Department of Mathematics & Statistics

B. Boufama
School of Computer Science

I. Ahmad, Advisor
School of Computer Science

October 13th, 2021

Declaration of Originality

I hereby certify that I am the sole author of this thesis and that no part of this thesis has been published or submitted for publication.

I certify that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis and have included copies of such copyright clearances to my appendix.

I declare that this is a true copy of my thesis, and that this thesis has not been submitted for a higher degree to any other University or Institution.

Abstract

Due to modern technological advancements, the pervasiveness and complexity of images have remarkably increased. Searching databases for similar visual content, i.e., Content-Based Image Retrieval (CBIR), remains an open research problem. In this thesis, we propose a novel CBIR approach, in which each symbolic image has a quadtree representation consisting of SIFT-based orientational keypoints. Every quadrant node in the tree represents the dominant orientation of a region in the image. The quadtree image representation is used for bitwise signature indexing and image similarity measurement. Also, we convert each quadtree image representation to a trainable feature vector for use in the K-Nearest Neighbour algorithm and Siamese Deep Neural Networks. The proposed approaches are evaluated using mean average precision (mAP), precision, recall, f-score and contrastive loss on three different image datasets. Our results indicate that, for complex images, orientational quadtrees are significantly more accurate than spatial quadtrees. Further, the derived feature vectors can be used in other machine learning or deep learning methods for training, ensemble, boosting, aggregation or embedding.

Acknowledgements

I would like to express my deepest gratitude to Dr Imran Ahmad for his continuous feedback and guidance. His experience and expertise in the field have proven invaluable for my work.

Also, I'm grateful to Dr Boubakeur Boufama, who played a critical and constructive role in several aspects of my research. Special thanks to Dr Myron Hlynka and Dr Saeed Samet for being on my committee at short notice and always being easy to access and willing to help.

Further, I'd also like to thank Dr Jimcymol James (from Manipal Institute of Technology, India), for bolstering my research aptitude and helping me throughout my Bachelor's. Moreover, I'd like to express appreciation to Mr Clint Olsen (from BlackBerry Limited, Canada), who mentored me very patiently and rigorously throughout my internship.

Finally, I am indebted to my parents, Murtuza Adil and Asna Badar, whose constant love and support kept me motivated throughout my academic career. I'd also like to mention my friends Hardik, Mirza, Nimish, Jenny, Rijin, Shabih, Sami, Yaqub, Shaik, Chinmaya, Prerana, Pallavi, Ali, Balsharan, Priyanka and Rida for making university life an unforgettable experience for me.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Overview

Exponentially larger digital picture collections are being created due to the pervasiveness of image-capturing equipment such as mobile phones, digital cameras, webcams and image scanners. Images are used in a wide range of fields – social media, remote sensing, fashion, crime prevention, publishing, medical, entertainment, architecture, etc. Most applications require efficient image searching, browsing, and retrieval tools to complete their tasks. Numerous general-purpose image retrieval frameworks have been developed for this purpose. There are two types of image retrieval frameworks: text-based and content-based [1]. The text-based framework originated in the 1970s. In systems based on this framework, images are manually tagged with text descriptors. These descriptors are then used to conduct image retrieval using a database management system (DBMS). There are two drawbacks to consider – manual annotation necessitates a significant amount of human effort and, the accuracy of the annotation depends on subjective human perception [2]. In the early 1980s, to address the shortcomings of text-based retrieval systems, content-based image retrieval (CBIR) was introduced [3]. CBIR indexes images according to their visual content, which includes colour, texture, shape and space relationship of objects [4]. The semantics of image contents are automatically identified and understood by the system [5]. In 1984, Chang and Liu released a seminal study in which they described an image indexing and abstraction technique for retrieval in pictorial databases. The pictorial database was made up of image objects and image relationships [6].

To create image indexes, abstraction procedures were defined to conduct clustering and classification of image objects [7]. Since then, virtually all CBIR systems employ image abstraction techniques for efficient, quick and accurate retrieval. Spatial similarity-based image retrieval is a major subclass of CBIR, and it deals with important structural information contained in the image [8] [9]. Abstract or symbolic images are almost always used to speed up and enhance the efficiency of the search and retrieval process.

Ahmad et al. [10] and Khan et al.'s [11] method of CBIR uses the idea of hierarchical decomposition of image space to provide a spatial arrangement of different features. For image retrieval, a two-level indexing technique can help decrease the search space for a given query image, where a computationally efficient first-level index lowers the search space for a more intricate second-level index. In [10] and [11], images are indexed with bitwise signatures and quadtree structures for the two levels, using the spatial positions of image corner points.

It is critical to design techniques that fully exploit global features — whether as standalone scene categorizers, as "context" modules inside larger object recognition systems, or as tools for assessing biases in freshly acquired datasets [12]. One such global feature is "dominant orientation" [13]. Dominant orientation is used to identify areas in an image containing specific directional structure types, which can then be used to improve object detection and image retrieval. Also, the dominant global orientation information can help enable a fast determination of the similarity of two images, before using a more computationally intensive and involved image retrieval approach. Gorkani et al. [13] establish that dominant orientation is not designed to address the "high-level" problem; rather that it will be in response to a high-level request and subsequently integrated with other features.

In this thesis, we combine the concepts of spatial similarity indexing and dominant orientation by revamping Ahmad et al.'s quadtree-based image representation scheme to use global and quad-level dominant orientations. As the dominant orientation goes from global to local, our quadtree decomposes from root to leaf.

CBIR methods based on local descriptors (such as SIFT) have been extensively studied for over a decade due to their advantage in dealing with image feature orientations [14]. We use these state-of-the-art feature descriptors to determine the orientation of each quadtree node during recursive decomposition. Image retrieval tasks use this quadtree image representation with a modified orientation-based signature indexing scheme and quadtree comparison algorithm.

Machine Learning algorithms such as K-means clustering and its variations are widely used for symbolic image clustering [15]. Also, Deep Siamese Neural Networks have recently shown successful results in complex CBIR applications such as face verification [16] [17], to predict if an input pair of images are similar or not [18]. This thesis proposes K-means Clustering and Siamese Deep Neural Networks for image retrieval based on our novel orientation-based quadtree image representation.

1.2 Research Objective

Our primary objective is to improve the retrieval metrics of existing CBIR methods introduced by Ahmad et al. and Khan et al. [10] [11]. To achieve this, the signature scheme and quadtree comparison algorithm need to be modified. Rather than using just spatial locations of corner points as features, we use feature descriptors to yield orientations of each corner point to derive more thorough quadtree-based symbolic image representations. Moreover, we use the derived symbolic image representation to train Machine Learning and Deep Learning models to bolster the metrics.

Our secondary objective is to use this methodology for multi-object similar image retrieval based on image segmentation techniques. The details of this framework are noted in Appendix A.

1.3 Research Motivation

After extensively reviewing literature about the latest CBIR methods, we found that the methodology by Ahmad and Grosky [10] is flexible enough to incorporate newer innovations. While the existing methodology uses feature points, most modern CBIR methods use oriented feature

descriptors. Further, in the past few years, there has been a revolution in artificial intelligence. Thus, we realized that by accommodating some of these advancements into existing schemes, we could potentially improve the image retrieval results.

1.4 Research Contribution

Some of the main contributions of this thesis are summarized below:

1. Novel orientation-based quadtree representation for image retrieval, based on previously introduced spatial similarity-based retrieval techniques in [10] and [11], and on the concept of "dominant orientation" [13]. The orientations are determined by feature descriptors such as SIFT.

2. Novel 64-bit image signature scheme based on quadrant orientations, and modified quadtree distance algorithms.

3. Design of a new algorithm for representing these image quadtree representations as feature vectors, which can be fed into ML & DL image retrieval training pipelines.

4. Application of K-Means Clustering and Siamese Deep Neural Networks to train models for image retrieval using the previously computed image representation feature vectors.

5. Quantitative and qualitative comparison of the proposed system with the ones introduced in [10] and [11].

6. Outline of a framework for multi-object similar image retrieval, along with preliminary implementation and custom metrics. This is useful for future researchers who'd like to extend these CBIR techniques to images with multiple objects.

1.5 Thesis Organization

The chapters of this thesis are organized in the following manner:

Chapter 1 outlines the background information, motivation, and contributions of the research work.

Chapter 2 describes literature in the field of CBIR, with a focus on methods that use quadtree recursive decomposition and feature descriptors.

Chapter 3 details the proposed framework and algorithm for feature extraction and representation, the process of incorporating SIFT features in a quadtree, and representing it for use in an ML or DL pipeline.

Chapter 4 explains the proposed image retrieval methods using signature indexing, quadtree distance comparisons, and ML & DL techniques.

Chapter 5 discusses the implementation details and results obtained for the proposed methods of image retrieval.

Chapter 6 concludes the thesis and provides future research directions.

Appendix A drafts a framework for multi-object similar image retrieval, based on the aforementioned methodology.

Chapter 2

Literature Review

Gorkani et al. [13] examined a criterion for "dominant perceptual orientation". The use of low-level textural orientation to an image retrieval challenge is discussed. To estimate the local orientation and strength at each pixel of the image, a series of directional filters are combined and convolved onto the image, with which an output magnitude is calculated. The uniqueness of this implementation is that it collects orientation information across several scales using a steerable pyramid, then aggregates the orientations from the various scales and calculates which are perceptually dominant, as assessed by a human study. The original picture is at the base of the steerable pyramid (level O), and each subsequent level is generated by filtering and subsampling the preceding level. At each level, orientations are approximated using a combination of directional filters. The number of pyramid levels is set to the maximum possible, based on textural density of the picture or subregion for which the orientation is to be determined.

Chen et al. [19] presented a technique in which a query picture is first split into smooth and high-detail blocks using quadtree segmentation and then categorized. Following that, a range of visually significant areas that have a high edge density are extracted. After encoding and pre-training all the image's blocks, compressed domain indices are produced. Finally, the index histogram is constructed as a feature of the query picture by counting the index frequencies. However, this approach does not use more discriminative and robust feature descriptors like SIFT and requires pre-trained models with considerable known data.

For effective image indexing, Jomier et al. [20] employed a quadtree data structure. A multi-level feature vector is used to represent an image. It is generated by recursively decomposing the image into four quadrants and then saved as a complete fixed-depth balanced quadtree. Each node in the quadtree stores a feature vector for the image quadrant it corresponds to. This method examines the picture for both global and local (sub-region) characteristics. This approach does not need indexing (e.g., signatures), as each node in the quadtree includes a high number of feature vectors. A drawback of this approach is that since each node in the quadtree has a lot of information, each comparison takes significantly longer.

Emerson and Chinniah [21] suggested an algorithm that allows a user to search for a particular object by identifying it with a region quadtree decomposition of the image. Then, various spatial & geometric characteristics (e.g. fractal dimension and wavelets) of the object are calculated and checked against a database of all previously calculated object values. Therefore, there are two steps involved in matching grayscale indices to existing grayscale indices. The first step involves performing a quadtree decomposition of all images in a database. The second step uses a previously calculated quadtree decomposition of all images in a database as a reference to match grayscale indices to one another. The sum of squared differences between the indices obtained for the quads that comprise the item of interest and those in the database produces a sorted list of pictures with comparable features. The disadvantage of this is that the spatial characteristics are far too complicated and particular to be of general use. Furthermore, we must specify weights and parameters for individual datasets ahead of time; additionally, this technique is only effective for single-band pictures. Also, images compare just corresponding quadrants rather than all quadrants.

Chen and Huang's research [22] took into account two shape features that are retrieved using a shape decomposition approach. The first feature is the proportion of contour points in each of the four quadrants. The second feature is the number of contour points intersecting with algorithmically generated line segments around the image's centroid. However, one drawback of this method is that pictures must have a very clearly defined outline for the shape to provide results. It is incapable of work-

ing against a dense background. Furthermore, the shape representation in each quadrant has multiple values and, as a result, cannot be readily indexed.

Vikram et al. [23] proposed a spatial similarity-based face indexing method. The spatial dispersion of physiologically significant dominant points on faces is stored in the KD-tree index structure for fast retrieval. The approach is linear transformation insensitive and resistant to changes in posture and emotion. However, this approach is only applicable to face datasets or datasets with a pre-defined structure; therefore, it is necessary to identify critical points in the dataset (i.e. based on the anatomy of the face).

Karami et al. [24] compared several image matching algorithms, including SIFT, SURF, and ORB, with various transformations and deformations, including scaling, rotation, noise, fish-eye distortion, and shearing. They demonstrated that ORB is the quickest method, but SIFT outperforms ORB in most cases examined. Their comparisons do not make use of indexing. As a result, it is significantly slower than index-based techniques.

Mongkolnam et al. [12] described an approach for extracting structural shapes from images, with a particular emphasis on using a multi-scale shape representation from a convex hull to control polygons at various sizes. In addition to the hierarchical shape features, this makes use of regional colour and spatial information and their connections to create a more complex image feature representation. The approach makes use of the information included in the pictures at the component level, and a B-spline approximation is utilized to arrive at the first level control polygons, which are then used to generate the matching convex hulls from the resulting first level control polygons. To accelerate and improve comparisons, the technique repeatedly applies Chaikin's algorithm to the first level control polygons to get finer control polygons. The comparison might begin at the convex hull level, progress to the first level control polygon, and so on. This approach does not employ indexing to represent its spatial or hierarchical features.

In their article, Lazebnik et al. [12] presented an approach that works

by dividing the picture into progressively fine sub-regions and producing histograms of the local features present inside each sub-region. The resultant "spatial pyramid" image format is a straightforward and computationally efficient modification of an orderless bag-of-features image representation that considerably improves performance on difficult scene categorization challenges. The primary contribution is a reconsideration of "global" non-invariant representations based on statistical aggregation of local features over defined subregions. Using an efficient approximation methodology adopted from the pyramid matching scheme, they provide an improved kernel-based recognition method that computes approximate geometric correspondence on a global scale and uses it to recognize objects. This technique entails continually subdividing the picture and calculating histograms of local features at ever-finer resolutions. Additionally, they employ higher-dimensional "strong features" for enhanced discriminatory power (SIFT descriptors).

Torralba et al. [26] demonstrated how to conduct strong place identification, classification of unfamiliar places, and object priming using a holistic, low-dimensional representation of the image. They demonstrate a context-aware vision system for item and place recognition. The objective is to recognize known places, classify novel surroundings, and utilize this knowledge to generate contextual priors for object identification (e.g., tables are more likely in an office than a street). The study proposes a low-dimensional global image representation that contains contextual information necessary for place recognition classification. It demonstrates how such contextual information introduces strong priors that facilitate object recognition. The technique predicts the scene using global picture features and then utilizes the scene as a prior for the local detectors. They achieve object recognition and localization exclusively using global features, and describe how their technique may be coupled with more standard object localization approaches that rely on local features.

El-Qawasmeh [27] proposed an organization for picture databases and an algorithm for image search by example. The proposed organization makes use of quadtrees to divide the database into subsets while also including some additional fields to make image searching easier. They suggest that the search query be processed using the centroid partial match method—the method checks for matching images by selecting ran-

dom points from an image in a circular, uniform movement. Rather than searching the entire picture database, the suggested organization searches a portion of it. It is adaptable because of the changing number of subgroups in the database.

Hsieh and Hsu [28] offered a novel approach for determining the similarity of symbolic images based on both the objects' properties and their spatial relationships. With the suggested technique CPM (Common Pattern Method), which integrates a novel data structure CP_DAG (Common Pattern Directed Acyclic Graph), it is possible to quickly and efficiently retrieve similar symbolic images. The similarity of two symbolic images is determined by the shared objects' characteristics and the spatial relationships between them, and the semantically similar patterns are encoded in the induced CP_DAG networks. It is important to note that the CP_DAG graphs are created in such a concise manner that the number of vertices and routes is effectively restricted; as a result, the efficiency polynomial in terms of the number of objects is maintained for virtually all the observed situations. Additionally, the creation criteria and properties of CP_DAG graphs have been explained and demonstrated. In contrast, there has been no investigation into the hierarchical structure of object classes, nor have the fuzzy measurements of the similarity degrees of attributes and spatial connections been investigated.

Amory et al. [29] proposed a novel Content-based Medical Image Retrieval (CBMIR) technique based on the Hungarian algorithm that compares a single block from the query image to all blocks from each image in the dataset and delivers the image with the closest match. This comparison is based on a feature vector of each block's gray-level intensity. Each window is divided into K groups using the K-mean clustering method. Then, the histogram of each cluster is transformed into a Gaussian distribution. The mean, variance, and skewness of the distribution are calculated based on this histogram. The suggested CBMIR achieves satisfactory results in most cases. The method consists of two phases: feature extraction and retrieval. During the feature extraction step, the feature vector for the query picture is extracted and saved in a meta-feature database – a database of feature vectors. In the retrieval phase that follows, the vector of the query picture is compared to all other feature vectors that have been recorded in the meta-feature database. The

result is then returned, and the nearest images are found using Euclidean distance.

Chapter 3

Proposed Feature Extraction and Representation

3.1 Image Feature Extraction

Each image $\mathcal{I}_i$ in a pictorial database is unique and may contain one or more objects $\mathcal{O}_k, k \geq 1$. In most retrieval techniques, an image is described as a set of representative feature components, defined as $F_k^j = \left\{F_k^1, F_k^2, \ldots, \mathrm{F}_k^{r_k}\right\}$ [11]. These features can be physical, logical, global or local [13].

Image features that provide spatial information about the image objects are termed as spatial features. A spatial feature F_k^j of an image object O_k in a 2D image space can be represented as a set of points $P_k^j = \left\{p_k^{j,1}, p_k^{j,2}, \ldots\right\}$, where $p_k^{j,m} = \left(x_k^{j,m}, y_k^{j,m}\right)$ are the (x, y) coordinates of the point in image space.

Image features that provide orientational information about the image objects are termed as orientational features. An orientational feature F_k^j of an image object O_k in a 2D image space can be represented as a set of points $P_k^j = \left\{p_k^{j,1}, p_k^{j,2}, \ldots\right\}$, where $p_k^{j,m} = \left(x_k^{j,m}, y_k^{j,m}, a_k^{j,m}\right)$, (x, y) are the coordinates of the point in image space, and a is the point's angle value between 0° and 360°.

To capture semantic information from the image, we can either annotate these points manually from the representative domain or generate points from an image programmatically. These unique points, referred to

as feature points, correspond to an image object's spatial and/or orientational characteristic(s). For simplicity, it is assumed that a single feature point represents an image object's feature. Thus, a complete image is represented by a collection of representative feature points.

The process of identifying and labelling feature points in a physical image effectively converts it to an analogous symbolic image called a feature image. When designing a computer vision algorithm, the feature extraction techniques chosen are essential. Algorithms for feature extraction can be broadly classified as feature detectors or feature descriptors.

Feature Detectors derive corner points to represent important local features in images. These are the points with a high curvature located at the intersection of the image's illumination areas [30]. Corner points are unaffected by illumination and are rotationally invariant, so we can use them to generate an analogous symbolic image.

Feature Descriptors combine information about the point's spatial location with an orientation and certainty measure, unlike feature points that only give a boolean assertion about the edge's existence [31]. As a result, feature descriptors can attach an orientation value with the point for increased accuracy and representation. However, when a more detailed description of a feature is required to solve the problem, it comes at the expense of additional data and more demanding processing.

Symbolic images, rather than the original image, are used for comparison and determining match quality for retrieval of images against a specified query image. Actual images are retrieved from the database only when provided with the symbolic image retrieval results. Thus, feature image generation is critical in the retrieval process and can significantly impact the system's overall accuracy and performance.

3.1.1 Feature Detectors

A feature detector is an algorithm that selects corner points (also called interest points) from an image, to generate a corresponding symbolic image. It is calculated by mathematically maximizing some kind of "cornerness" function. A point where two edges intersect is called a corner. In

other words, it is a point for which there are two dominant and different edge directions in its local neighbourhood. Harris [32], FAST [33] and Shi Thomasi [34] are the most commonly used corner detectors.

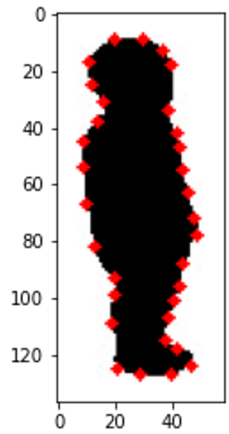


FIGURE 3.1: Feature Detector Example

The Shi-Tomasi method [34][35], also known as Good Features to Track, is a corner detection technique that is commonly used in the field of computer vision to extract specific kinds of features from an image. It is an enhancement of the Harris corner detector [36][37].

The Harris corner detector is one of the most widely used eigenvalue-based feature point detector due to its tolerance to image noise and rotation [38]. A "window" is a fixed square subset of constant-size pixels from the image [39]. Harris detector works by quantifying the sliding window's local changes when patches are moved in various directions by a tiny amount [40][41][42]. The main objective of Harris corner detector is to evaluate the change in the intensity of individual image windows.

The Shi-Tomasi method works in a similar way as the Harris method, but thresholds the final point to a pre-defined value. It generates more stable and precise feature points for tracking. However, it also increases computing requirements [35][41].

3.1.2 Feature Descriptor

A descriptor is a vector of values that characterizes the picture patch surrounding a point of interest. It might be as straightforward as raw

pixel values or as complex as a histogram of gradient orientations.

A combination of an interest point and its description is commonly referred to as a local feature. Local features are used across various computer vision applications, including image registration, three-dimensional reconstruction, object detection, and recognition.

The most commonly used feature descriptor algorithms are SIFT, SURF, ORB and BRISK. For this thesis, we have chosen SIFT because it has proven to give better results [43]. SIFT is composed of two components: a detector and a descriptor [44]. The detector is based on the difference-of-Gaussians (DoG), a Laplacian approximation. The DoG detector identifies blob-like formations' cores. A histogram of gradient orientations is constructed. The descriptor is a binary string that encodes the sign of the difference between specific pairs of pixels in the vicinity of the interest point.

Figure 3.2: Feature Descriptor Example

3.2 Quadtree Recursive Decomposition

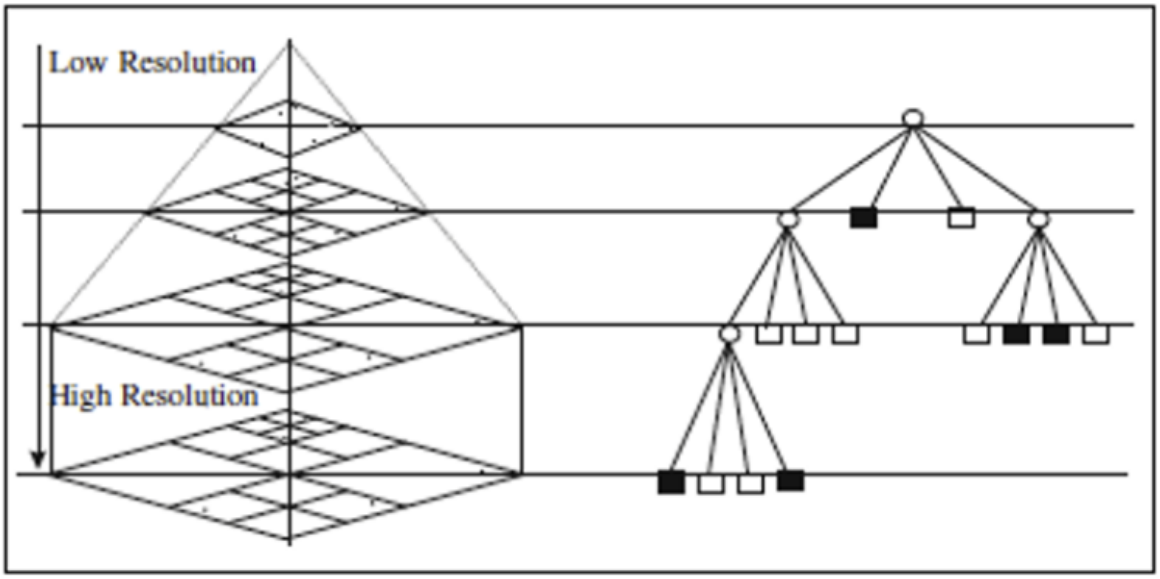


FIGURE 3.3: Quadtree Recursive Decomposition [10]

The quadtree approach is based on the "Divide and Conquer" algorithm. Consider an image that has been divided into four regions. Each of these regions is further subdivided into four more regions. The procedure of partitioning square regions by four will be repeated until the desired level of division is achieved. Quadtrees have the potential to enhance the capability of image processing significantly. Due to quadtrees' intrinsic recursive nature, a typically linear data structure becomes a recursive data structure [27]. As a result, pictures represented by quadtrees have great representational flexibility, making them well-suited for image modification.

Recursive decomposition is a technique for segmenting a feature image into distinct regions to calculate and detect the spatial and semantic connections between individual feature points. It is defined by Ahmad et al. in Definition 3.1 to 3.7 [10]:

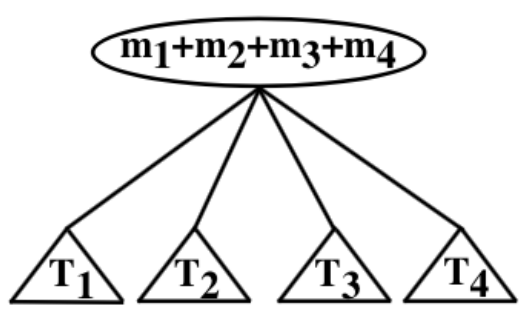
> **Definition 3.1** The process of recursively dividing an image space into four equal size quadrants 0,1,2,3 is termed as the recursive decomposition of an image. The resultant quadrants

are recognized by four directional relations as North-West (NW), North-East (NE), South-West (SW), South-East (SE), respectively. The decomposition process stops only when each and every feature point can be identified by a distinct quadrant and, therefore, each quadrant can contain exactly one point in it. Figure 3.3 is an example of a hierarchically decomposed feature image and its corresponding quadtree representation. In this figure, the root of the tree corresponds to the original non-decomposed image. Each level of the quadtree corresponds to subsequent levels of decomposition. The circles represent internal nodes of the tree, whereas leaf nodes correspond to the smallest quadrants. Black rectangles represent quadrants containing feature points, and white rectangles represent empty quadrants.

Definition 3.2 A quadtree is defined as follows:

- A single node M is a quadtree. This node is a leaf as well as the root of the tree. $m \geq 0$ is called the occupancy of this node. It is meant to capture the number of feature points in the region of the image corresponding to the node. The coordinate sequence of this node is λ, the empty sequence.

- If T_1, T_2, T_3, T_4 are quadtrees whose roots have occupancies m_1, m_2, m_3 and m_4 respectively, where $m_1+m_2+m_3+m_4 \geq 0$, we have that



is a quadtree. The node $(m_1 + m_2 + m_3 + m_4)$ is the root of the resulting quadtree. The leaves of T_1, T_2, T_3 and T_4 are the leaves of the resulting quadtree. The coordinate sequence

of root node is λ. If *seq* is coordinate sequence of a node in T_j, for $1 \leq j \leq 4$ then $j \bullet$ seq is the coordinate sequence of this node in the resulting quadtree, where $\bullet$ is the sequence concatenation operator.

In these definitions, the root node of the quadtree T is denoted by root(T) and the occupancy of the node n of a quadtree is denoted by occupancy (n).

Definition 3.3 The level of a node, n, in a quadtree is the length of the coordinate sequence of that node and is denoted by level (n)

Definition 3.4 The height of a quadtree T, denoted by height ((T), is one more than the maximum level of any node in the quadtree.

Definition 3.5 The i th approximation of a quadtree T, for $i \geq 0$, is the quadtree which results by removing all nodes on level $j > i$. It is denoted by $T^{(i)}$.

Definition 3.6 A quadtree is complete iff each leaf node has an occupancy of 0 or 1.

The quadtree distance functions are described in Section 4.

3.3 Use SIFT features in Quadtree

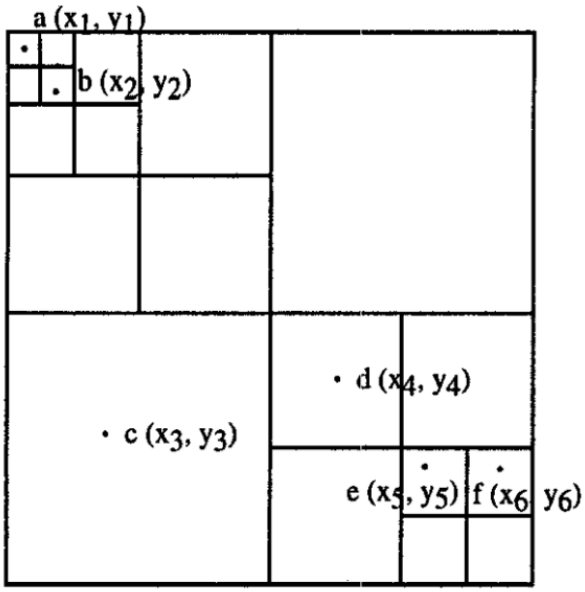


FIGURE 3.4: Spatial decomposition of feature points in an image [10]

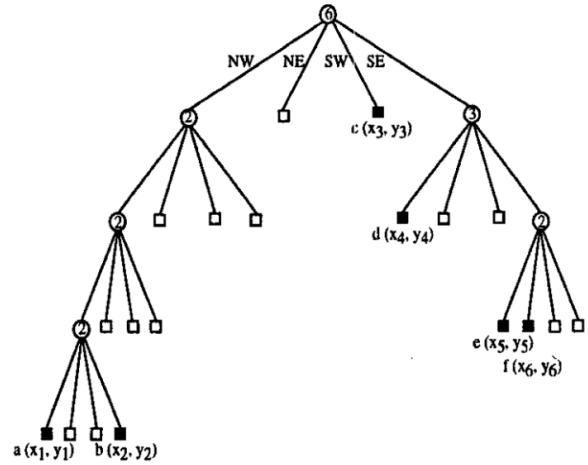


FIGURE 3.5: Quadtree representation of feature points in Figure 3.4 [10]

Contemporary feature extraction techniques use feature descriptors that utilize orientation (rotation of gradient) rather than just the spatial location of points. These feature descriptors are invariant to scale, rotation or lighting difference. In our research work, we will use feature descriptor values in this quadtree representation for image retrieval.

A diagrammatic representation of how image feature points are decomposed into a quadtree representation is shown in Figure 3.3. Based on this methodology, an example with hypothetical values is illustrated in Figure 3.5 and 3.6. In Figure 3.7, we demonstrate with example values how SIFT descriptors can be incorporated into a quadtree just like how Ahmad et al. and Khan et al. [10][11] used the number of spatial points. The orientation values are shown within braces, while the number of spatial points stay the same.

The values in an orientation-based quadtree are averaged up from the leaf nodes. Angles are averaged using Algorithm 1. The averaging algorithm uses a trigonometric approach, because a simple averaging of

angles is inaccurate. For example, the average of 359° and 1° is 0°, and not 180°.

This averaging function is used while building an orientation-based angle quadtree, in Algorithm 2. Initially, the values are in the leaf nodes, as each leaf represents an oriented point. Algorithm 2 calculates averaged values for non-leaf nodes in a recursive bottom-up approach.

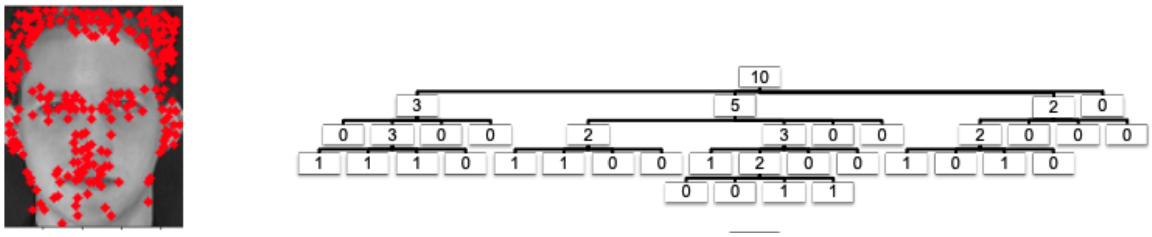


FIGURE 3.6: Quadtree Example with Feature Points

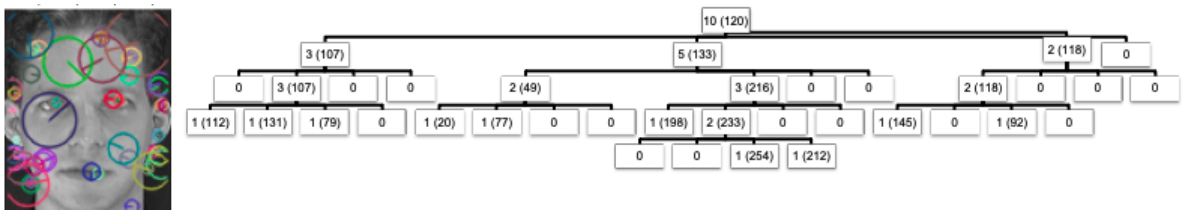


FIGURE 3.7: Quadtree Example with Feature Descriptor

3.4 Cyclical Feature Representation of Quadtree Image

Since we are now storing angle values derived from feature descriptors in a quadtree, we have to deal with cyclical values. For instance, 359 degrees is closer to 1 degree than it is to 355 degrees. It, therefore, becomes challenging to index images using these values without losing crucial information.

We address this problem by converting the 0°-360° numeric values to a concatenated (glued) sin-cos value, as described by Akil [45]. So, for example, 35° is represented as "0.57 : 0.81".

Algorithm 1 *averageAngles*(*angles*)

Input: List of angles, $A_n = \{a_1, a_2, \ldots, \mathrm{a}_n\}$ where: $a_i \geq 0$ and $a_i < 360$, $1 \leq i \leq n$

Output: Average of *angles*

```
x ← 0.0
y ← 0.0
for a in angles do
    x = x + cosine(toRadians(a))
    y = y + sine(toRadians(a))
end for
if x = 0 or y = 0 then
    return −1
end if
result ← toDegrees(arctan2(y, x))
result ← ⌊(result + 360) mod 360)⌋
return result
```

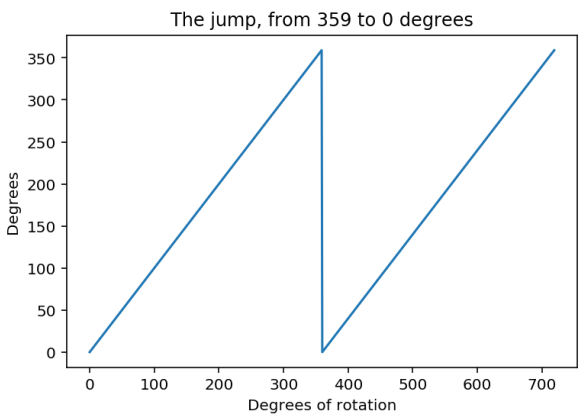


FIGURE 3.8: Illustration of the problem with cyclical values [45]

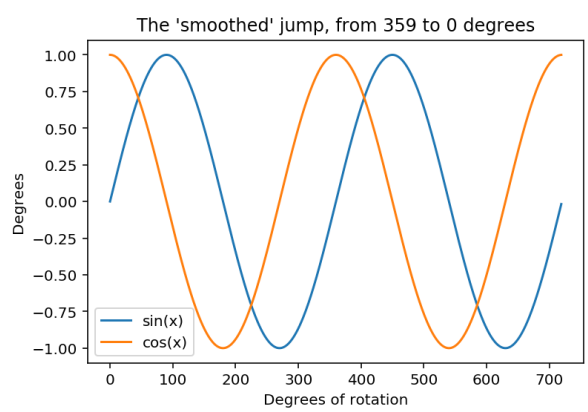


FIGURE 3.9: Why we use sin-cos values instead [45]

First, we do a depth-first traversal (described in Algorithm 3) of the quadtree to derive a flattened list representation of the image quadtree, consisting of an array of angle values. Then, we convert each angle to a glued sin-cos representation, and group them as a feature vector. The resultant file is a collection of feature vectors, each representing an image quadtree. These feature vector files can be fed into an AI model for image training.

Algorithm 2 *buildQuadtreeWithNodeAngles*(t)

Input: Tree Node t, where t is the root node of the quadtree, all leaf nodes have angle values, non-leaf nodes have value -1

Output: Quadtree in which every node has an angle value, averaged up from the leave nodes bottom-up recursively

```
if t = null then                                   ▷ Base Case
    return −1
end if
if t.angle is not −1 then                          ▷ Base Case
    return t.angle
end if

▷ Loop through 4 children of the node and add their respective angles
to anglesForAverageList:

anglesForAverageList ← {}
for i from 1 to 4 do
    q ← buildQuadtreeWithNodeAngles(t.child(i))
    if q ≠ −1 then
        Add q to anglesForAverageList
    end if
end for

▷ Calculate average angle value of the list using Algorithm 1 and assign
to parent angle:

result ← averageAngles(anglesForAverageList)
t.angle ← result

return result
```

Algorithm 3 *writeAngleQuadtreeIntoFile*($t, maxDepth, depth, allAnglesOutput, tParent$)

Input: Tree Node t, where t is the root node of the quadtree, all nodes have angle values.
$maxDepth$, maximum depth of tree we want to write into file.
other parameters are recursive parameters.

Output: $allAnglesOutput$ is a string representation of the Angle Quadtree for training models

```
if depth > maxDepth then                    ▷ Constraint with maxDepth
    return null
end if
tempNode ← tParent

▷ If we don't have angle value for a node, traverse upward (child to
parent) till we find an angle value:

if t = null  or t.angle = −1 then
    while tempNode.angle = −1 do
        tempNode ← tempNode.parent
    end while
end if

▷ Append angle value of node to allAnglesOutput text file and repeat
process for the node's 4 children:

allAnglesOutput = allAnglesOutput + tempNode.angle
for i from 1 to 4 do
    if t = null or t.child(i) = null then
        Add missing child as −1
    end if
    allAnglesOutput+ =
     writeAngleQuadtreeIntoFile(t.getChild(i), maxDepth, depth, "", t)
end for

return allAnglesOutput
```

Chapter 4

Proposed Image Retrieval Methods

4.1 Quadtree Comparison Method

4.1.1 Quadtree Spatial Distance Method

Under the existing scheme, distance between two images is calculated by comparing the number of spatial points in corresponding nodes of both quadtrees.

Definition 4.1 [10] To establish a measure of similarity between the images matched in the first stage of filtering, their corresponding quadtrees are matched. The quadtrees are matched with the help of a distance function. The distance function is defined so that the distance is computed node by node, starting from the root and going down along both of the trees gradually. This results in fewer comparisons, since it allows us to eliminate trees that appear different at the initial stages of processing. The distance between two quadtrees T and $U, d(T, U)$ is:

case 1: Suppose height $(T) =$ height $(U) = 1$ and occupancy (root(T))+ occupancy(root(U)) $= 0$. Then

$$d(T, U) = 0$$

case 2: Suppose height $(T) =$ height $(U) = 1$ and occupancy (root(T))+ occupancy(root(U)) > 0

Also, let $M = \text{occupancy}(\text{root}(T))$ and $N = \text{occupancy}(\text{root}(U))$. Then

$$d(T,U) = \frac{|M-N|}{\max(M,N)}$$

case 3: Suppose height $(T) = 1$ and occupancy $(\text{root}(T)) = 0$ and height $(U) > 1$. Then

$$d(T,U) = 1$$

case 4: Suppose height $(T) = 1$, occupancy $(\text{root}(T)) = 1$, height $(U) > 1$ and at least one child of the root node of U has an occupancy greater than 0. Then

$$d(T,U) = \frac{|N-1|}{N}$$

case 5: Suppose height $(T) = 1$, occupancy $(\text{root}(T)) = 1$, height $(U) > 1$ and at least one child of the root node of U has an occupancy equal to 0. Then

$$d(T,U) = 1$$

case 6: Suppose height $(T) > 1$ and height $(U) > 1$.

For $1 \leq j \leq 4$, let the subtree of T and U determined by the nodes having coordinate sequence j be called T_j and U_j respectively.

Let occupancy $(\text{root}(T)) = M$ and occupancy $(\text{root}(U)) = N$

For $1 \leq j \leq 4$, let occupancy $(\text{root}\,(T_j)) = m_j$ and occupancy $(\text{root}\,(U_j)) = n_j$. Then

$$d(T,U) = \max\left(\sum_{j=1}^{4} \frac{m_j}{M} d\left(T_j, U_j\right), \sum_{j=1}^{4} \frac{n_j}{N} d\left(T_j, U_j\right)\right)$$

Quadtree Spatial Distance Example:

Figure 4.1 illustrates a partial quadtree with hypothetical values. It demonstrates how distance is calculated at the lowest level. A 0 distance value indicates that the nodes are a perfect match, and a value of 1 indicates completely dissimilar nodes.

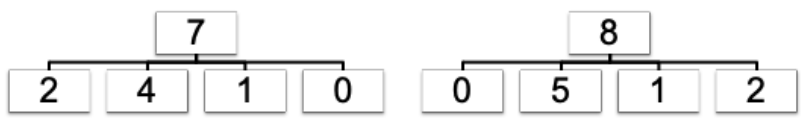


FIGURE 4.1: Quadtree Spatial Comparison Example

$$
\begin{aligned}
&d(M,N) = |M-N|/max(M,N)\\
&d(2,0) : 2/2 = 1\\
&d(4,5) : 1/5 = 0.2\\
&d(1,1) = 0/1 = 0\\
&d(0,2) = 2/2 = 1
\end{aligned}
$$

4.1.2 Quadtree Angle Distance Method

With the proposed quadtree distance function, the tree node matching is done by comparing the angle of the respective nodes.

Definition 4.2. The distance between two quadtrees T and U, $d(T,U)$ is defined as follows:

case 1: Suppose height (T) = height $(U) = 1$ and occupancy (root(T))+ occupancy(root(U)) $= 0$. Then

$$d(T,U) = 0$$

case 2: Suppose height (T) = height $(U) = 1$ and occupancy (root(T))+ occupancy(root(U)) > 0

Also, let M = angle(root(T)) and N = angle(root(U)). Then

$$d(T,U) = \frac{angleDistance(M,N)}{180^\circ}$$

case 3: Suppose height $(T) = 1$ and occupancy $(\text{root}(T)) = 0$ and height $(U) > 1$. Then

$$d(T,U) = 1$$

case 4: Suppose height $(T) = 1$, occupancy $(\text{root}(T)) = 1$, height $(U) > 1$ and at least one child of the root node of U has an occupancy equal to 0. Then

$$d(T,U) = 1$$

case 5: Suppose height $(T) > 1$ and height $(U) > 1$.

For $1 \leq j \leq 4$, let the subtree of T and U determined by the nodes having coordinate sequence j be called T_j and U_j respectively.

Let $\text{angle}(\text{root}(T)) = M$ and $\text{angle}(\text{root}(U)) = N$

For $1 \leq j \leq 4$, let angle $(\text{root}\,(T_j)) = m_j$ and angle $(\text{root}\,(U_j)) = n_j$. Then

$$d(T,U) = \max\left(\sum_{j=1}^{4} \frac{m_j}{M} d\left(T_j, U_j\right), \sum_{j=1}^{4} \frac{n_j}{N} d\left(T_j, U_j\right)\right)$$

Quadtree Angle Distance Example:

Figure 4.2 illustrates a partial quadtree with hypothetical angle values. It demonstrates how distance is calculated at the lowest level. A 0 distance value indicates that the nodes are a perfect match, and a value of 1 indicates completely dissimilar nodes. It uses Algorithm 4 $angleDistance(x,y)$ to calculate the distance between the angles, and

then it's normalized by dividing 180°, because the maximum angle distance possible is 180°.

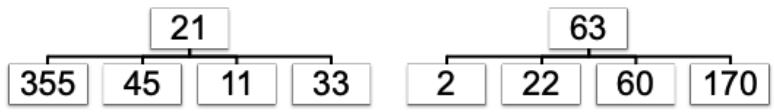


FIGURE 4.2: Quadtree Angle Comparison Example

$$
\begin{aligned}
&d(a,b) = angleDistance(a,b)/180\\
&d(355,2) : 7/180 = 0.03\\
&d(45,22) : 23/180 = 0.12\\
&d(11,60) = 35/180 = 0.19\\
&d(33,12) = 101/180 = 0.56
\end{aligned}
$$

Algorithm 4 $angleDistance(x, y)$

Require: $x, y \geq 0$ and $x, y < 360$ ▷ Input angles

```
result ← 0
phi ← floorMod(abs(x − y), 360)
if phi > 180 then
    result ← 360 − phi
else
    result ← phi
end if
result ← abs(result)
return result
```

4.2 Signature Indexing Method

4.2.1 Existing Methods

The existed methods proposed by Ahmad et al. [10] and Khan et. al [11] only use number of spatial feature points to formulate the signature. The former utilizes the standard deviation of the number of points in Level 1 (4 quadrants) to the total points, tree depth and average points in each decomposition. The latter takes the % of points in each quadrant, divided

into 16 sub-quadrants. In the following subsections, we summarize Ahmad et al.'s 32-bit signature scheme, Khan et. al's 32-bit signature scheme and also an updated version of Khan et. al's signature scheme, which is 64-bit. We have introduced this 64-bit spatial signature for using it as a baseline for our experimental comparisons.

Ahmad et al.'s signature scheme [10]

In Ahmad et al.'s signature scheme, a signature is defined as a 32-bit number with two disjoint fields. The first one of these fields is based on population standard deviation of the number of feature points in the four quadrants after only the first level of quadtree decomposition. It can be represented by the following equation [10]:

$$S_1^i = \sqrt{\frac{\sum_{j=0}^{3} (a_j - \mu)^2}{\sum_{j=0}^{3} a_j}}$$

where $\mu = \sum_{j=0}^{3} a_j/4$ and a_j is the number of feature points in each of the four quadrants after the first level of decomposition.

The second field is the average number of feature points at each level of decomposition of the image and can be represented by the following equation:

$$S_2^i = \frac{\sum_{j=0}^{3} a_j}{h}$$

where h is the height of the quadtree.

After building the signature, the notion of Tolerance Factor is used to relax the signature to make it more flexible while comparing two images.

The main problem with this signature representation is that it generalizes both of its disjoint fields. Using only the number of spatial points in the first level's four quads, is not representative enough for most images. As a result, this signature matching scheme results in too many false positives [11].

Khan et. al's signature scheme [11]

Khan et. al's original signature S of an image quadtree T is 32-bit number, and consists of four 8-bit disjoint fields $S_i, 1 \leq i \leq 4$. Each of these fields corresponds to one of the four quadrants in the recursive decomposition.

We have updated the existing 32-bit signature scheme to introduce a 64-bit scheme with the same underlying hypothesis. The 64-bit scheme helps the signature become far more representative of the image. Also, it makes the comparison between Khan et. al's existing signature scheme and our proposed angle signature scheme fairer, as both signatures will be 64-bit. We will elaborate on both the original 32-bit scheme and our updated 64-bit scheme, in the following sub-sections.

- if height $(T) = 1$, then $S = 0$
- if height $(T) = 2$, then $S_i = \%$ of total number of feature points belonging to the quadrant i.
- if height $(T) > 2$, for $1 \leq i \leq 4$ let the subtree of T determined by the nodes having coordinate sequence i be called T_i. Also, each S_i is further divided into four equal size disjoint fields (i.e. 2 bits each for 32-bit, 4 bits each for 64-bit) $S_{(i,j)}, 1 \leq j \leq 4$ such that each field corresponds to one of the four quadrants in the recursive decomposition of T_i. Then for each T_i :
  1. if height $(T_i) = 1$, then $S_i = 0$
  2. if height $(T_i) = 2$, then $S_{(i,j)} = \%$ of total number of feature points belonging to the quadrant corresponding to j
  3. if height $(T_i) > 2$:
     - If two or more immediate descendants of T_i have occupancy greater than 0, then $S_{(i,j)} = \%$ of total number of feature points belonging to the quadrant corresponding to j.
     - In all other cases, $T_i = T_{(i,j)}$, such that root $\left(T_{(i,j)}\right) =$ only immediate descendant of root (T_i) with non-zero occupancy. Go to step 1.

In the original 32-bit scheme, % of total number of feature points is represented with the help of only two bits. In the updated 64-bit scheme, four bits are used to represent them. This is achieved by representing the % value (p) with the help of 4 ranges:

- $p < 25\%$

- $25\% \leq p < 50\%$

- $50\% \leq p < 75\%$

- $p > 75\%$

While both the 32-bit and 64-bit schemes generalize the information, the latter provides more flexibility during retrieval, as the block-wise values are retained despite increasing tolerance. This will be demonstrated by a practical example. Khan et. al observed better experimental results using a 32-bit scheme than Ahmad et al.'s original scheme. We expect our updated 64-bit spatial scheme to perform better than the 32-bit spatial scheme, which will then be used as an experimental baseline for our novel signature methodology.

Another important aspect of signature representation is the Tolerance Factor (Definition 4.3). Tolerance Factor is essential to accommodate the concept of similarity between a database and the query image [10] and affects the number of matched images retrieved. Therefore, it has an immediate effect on the number of retrieved signatures. The higher the value of the Tolerance Factor, the higher the number of retrieved signatures and possibly higher number of matched images. As the tolerance increases, every bit in the signature eventually tends to 1. For Khan et. al's representation, the modified definition of Tolerance Factor is given as:

Definition 4.3 [11] The Tolerance Factor (TF) is an addition of $\pm x$ to the % values of $S_{(i,j)}$, $1 \leq i, j \leq 4$ for each S_i of a database image signature S for similarity-based search and retrievals by providing a range-search capability [1].

TF $= 0$ is a special case and provides an exact match. The value of x in TF depends on application and extent of similarity. This allows

us to test a range of image signatures for a possible match against a given query image signature S_q, computed without any tolerance. In the second stage of filtering, only those images are taken into consideration for which the binary AND operation between the query image signature S_q and database image signature S results in the query image signature S_q i.e. $S_q \cap S \rightarrow S_q$.

Khan et. al's original 32-bit signature scheme example [11]

The original Khan et. al scheme assigned the following 2 bit binary codes to the 4 ranges:

- $p < 25\% \rightarrow 00$

- $25\% \leq p < 50\% \rightarrow 01$

- $50\% \leq p < 75\% \rightarrow 10$

- $p \geq 75\% \rightarrow 11$

Now, let us assume $S_{(1,1)}$ is 43% and accordingly the assigned binary code is 01. Let us also assume that the value of x for calculating TF is 10. Then we have two range values and consequent binary codes for $S_{(1,1)}$:

$$S'_{(1,1)} = S_{(1,1)} - x = 33 \rightarrow 01$$
$$S''_{(1,1)} = S_{(1,1)} + x = 53 \rightarrow 10$$

The final representation of $S_{(1,1)}$ is computed by taking binary representations of the three values and performing a binary OR operation. In other words:

$$S_{(1,1)} \rightarrow S_{(1,1)} \cup S'_{(1,1)} \cup S''_{(1,1)}$$
$$\Rightarrow S_{(1,1)} \rightarrow 01 \cup 01 \cup 10$$
$$\Rightarrow S_{(1,1)} \rightarrow 11$$

Similarly, let's assume $S_{(1,2)} \rightarrow 00, S_{(1,3)} \rightarrow 01$ and $S_{(1,4)} \rightarrow 10$. Then, by concatenating these binary codes we find S_1

$$S_1 \rightarrow S_{(1,1)} \left|S_{(1,2)}\right| S_{(1,3)} \mid S_{(1,4)}$$
$$\Rightarrow S_1 \rightarrow 11|00|01|10$$
$$\Rightarrow S_1 \rightarrow 11000110$$

Similarly, lets assume $S_2 \rightarrow 00111000, S_3 \rightarrow 00101001$ and $S_4 \rightarrow 00111100$. Finally, by concatenating these codes, we find the 32 -bit signature S for a database image:

$$S \rightarrow S_1 \left|S_2\right| S_3 \mid S_4$$
$$\Rightarrow S \rightarrow 11000110001110000010100100111100$$

Now, suppose the query image signature is

$$S_q \rightarrow 10000110001010000010000100110100.$$

Then, by performing a binary AND operation between S and S_q, we find:

$$\begin{array}{c} 11000110001110000010100100111100 \\ \cap \\ 10000110001010000010000100110100 \\ \hline 10000110001010000010000100110100 \end{array}$$

which is equal to the query image signature S_q. Hence, this database image will be accepted for the second stage.

Our 64-bit scheme addresses a fundamental flaw in the 32-bit scheme:

$25\% \leq p < 50\% \rightarrow 01$

$50\% \leq p < 75\% \rightarrow 10$

$\therefore 25\% \leq p < 75\% \rightarrow 11$

But, $p \geq 75\% \rightarrow 11$

We can see that 32-bits aren't enough to accommodate a wider tolerance value, and the signatures eventually start to lose their specificity. To

avoid this problem, we have proposed an extended spatial 64-bit signature scheme.

Extended 64-bit spatial signature scheme example

Our extended 64-bit signature scheme assigns the following 4 bit binary codes to the 4 ranges:

- $p < 25\% \rightarrow 1000$

- $25\% \leq p < 50\% \rightarrow 0100$

- $50\% \leq p < 75\% \rightarrow 0010$

- $p \geq 75\% \rightarrow 0001$

Now, let us assume $S_{(1,1)}$ is 43% and accordingly the assigned binary code is 0100. Let us also assume that the value of x for calculating TF is 10. Then we have two range values and consequent binary codes for $S_{(1,1)}$:

$$S'_{(1,1)} = S_{(1,1)} - x = 33 \rightarrow 0100$$
$$S''_{(1,1)} = S_{(1,1)} + x = 53 \rightarrow 0010$$

Our final representation of $S_{(1,1)}$ is computed by taking binary representations of the three values and performing a binary OR operation. In other words:

$$S_{(1,1)} \rightarrow S_{(1,1)} \cup S'_{(1,1)} \cup S''_{(1,1)}$$
$$\Rightarrow S_{(1,1)} \rightarrow 0100 \cup 0100 \cup 0010$$
$$\Rightarrow S_{(1,1)} \rightarrow 0110$$

Similarly, let's assume $S_{(1,2)} \rightarrow 0000, S_{(1,3)} \rightarrow 0100$ and $S_{(1,4)} \rightarrow 1000$. Then, by concatenating these binary codes we find S_1

$$S_1 \rightarrow S_{(1,1)} \left| S_{(1,2)} \right| S_{(1,3)} \mid S_{(1,4)}$$
$$\Rightarrow S_1 \rightarrow 0110|0000|0100|1000$$
$$\Rightarrow S_1 \rightarrow 011000001001000$$

Similarly, let's assume $S_2 \rightarrow 101001101000001$, $S_3 \rightarrow 001010001010010$ and $S_4 \rightarrow 001100001110010$. Finally, by concatenating these codes, we find the 64-bit signature S for a database image:

$$S \rightarrow S_1 \,|S_2|\, S_3 \mid S_4$$
$$\Rightarrow S \rightarrow 0110000010010001010011010110010010100010100100011000011100 10$$

Now, suppose our query image signature is

$$S_q \rightarrow 0100000010010000010011000000010010100010100100010000011000 00$$

Then, by performing a binary AND operation between S and S_q, we find:

$$\begin{array}{c}
0110000010010001010011010110010010100010100100011000011100 10 \\
\cap \\
\underline{0100000010010000010011000000010010100010100100010000011000 00} \\
0100000010010000010011000000010010100010100100010000011000 00
\end{array}$$

which is equal to the query image signature, S_q. Hence, this database image will be accepted for the second stage.

The advantage of using the updated 64-bit spatial signature scheme is that it allows a wide range of tolerance, and addresses the previously addressed flaw in the 32-bit scheme.

Spatial Signature from Quadtree Example:

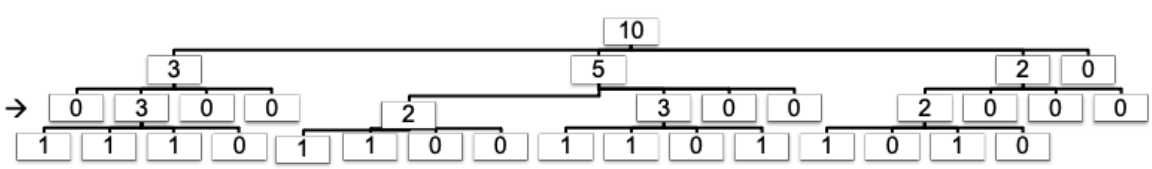


FIGURE 4.3: The signature will be calculated using 0300 2300 2000 0000

4.2.2 Proposed Method

A signature S of a quadtree T of an image is a 64-bit number. In the new scheme, the signature S consists of 16 4-bit disjoint fields $S_i, 1 \leq i \leq 4$. Each of these fields corresponds to one of the 16 quadrants in the recursive decomposition, representing an angle range for that quadrant.

- if height $(T) = 1$, then $S = 0$
- if height $(T) = 2$, then S_i = average angle of feature descriptors belonging to the quadrant i.
- if height $(T) > 2$, for $1 \leq i \leq 4$ let the subtree of T determined by the nodes having coordinate sequence i be called T_i. Also, each S_i is further divided into four equal size disjoint fields (i.e. 4 bits each) $S_{(i,j)}, 1 \leq j \leq 4$ such that each field corresponds to one of the four quadrants in the recursive decomposition of T_i. Then for each T_i :
  1. if height $(T_i) = 1$, then $S_i = 0$
  2. if height $(T_i) = 2$, then $S_{(i,j)}$ = average angle of feature descriptors belonging to the quadrant j
  3. if height $(T_i) > 2$:
     - If at least two of the four immediate descendants of T_i have occupancy greater than 0, then $S_{(i,j)}$ = the total average angle of feature descriptors belonging to the quadrant corresponding to j.
     - In all other cases, $T_i = T_{(i,j)}$, such that root $\left(T_{(i,j)}\right)$ = only immediate descendant of root (T_i) with non-zero occupancy. Go to step 1.

In this scheme, we need to represent the average angle of feature descriptors belonging to the quadrant with only four bits. This is achieved by representing the value (a) with the help of 4 ranges:

- $0^\circ \leq a < 90^\circ$

- $90^\circ \leq a < 180^\circ$

- $180^\circ \leq a < 270^\circ$

- $270^\circ \leq a < 360^\circ$

Although this may generalize the information to some extent, but the flexibility in the generated signatures outweighs this generalization. Further, the generated signatures provide significant improvement in the overall system performance, as can be observed from the experimental results.

In our representation, the modified definition of Tolerance Factor is given as:

Definition 4.4. The Tolerance Factor (TF) is an addition of $\pm x$ to the average angle values of $S_{(i,j)}$, $1 \leq i, j \leq 4$ for each S_i of a database image signature S for similarity-based search and retrievals by providing a range-search capability.

TF $= 0$ is a special case and provides an exact match. The value of x in TF depends on application and extent of similarity. This allows us to test a range of image signatures for a possible match against a given query image signature S_q, computed without any tolerance. In the second stage of filtering, only those images are taken into consideration for which the binary AND operation between the query image signature S_q and database image signature S results in the query image signature S_q i.e. $S_q \cap S \rightarrow S_q$ [11].

As an example, suppose we have assigned the following 4 bit binary codes to the 4 ranges:

- $0^\circ \leq a < 90^\circ \rightarrow 1000$

- $90^\circ \leq a < 180^\circ \rightarrow 0100$

- $180^\circ \leq a < 270^\circ \rightarrow 0010$

- $270^\circ \leq a < 360^\circ \rightarrow 0001$

Now, let us assume $S_{(1,1)}$ is 85° and accordingly the assigned binary code is 1000. Let us also assume that the value of x for calculating TF is 10. Then we have two range values and consequent binary codes for $S_{(1,1)}$:

$$S'_{(1,1)} = S_{(1,1)} - x = 75 \rightarrow 1000$$
$$S''_{(1,1)} = S_{(1,1)} + x = 95 \rightarrow 0100$$

Our final representation of $S_{(1,1)}$ is computed by taking binary representations of the three values and performing a binary OR operation. In other words:

$$\begin{aligned} &S_{(1,1)} \rightarrow S_{(1,1)} \cup S'_{(1,1)} \cup S''_{(1,1)} \\ &\Rightarrow S_{(1,1)} \rightarrow 1000 \cup 1000 \cup 0100 \\ &\Rightarrow S_{(1,1)} \rightarrow 1100 \end{aligned}$$

Similarly, lets assume $S_{(1,2)} \rightarrow 0100, S_{(1,3)} \rightarrow 0010$ and $S_{(1,4)} \rightarrow 0001$. Then, by concatenating these binary codes we find S_1

$$\begin{aligned} S_1 &\rightarrow S_{(1,1)} \left| S_{(1,2)} \right| S_{(1,3)} | S_{(1,4)} \\ \Rightarrow S_1 &\rightarrow 1100 \mid 0100 \mid 0010 \mid 0001 \\ \Rightarrow S_1 &\rightarrow 1100010000100001 \end{aligned}$$

Similarly, lets assume $S_2 \rightarrow 0010000000110001, S_3 \rightarrow 1000000111100001$ and $S_4 \rightarrow 0100010000000111$. Finally, by concatenating these codes, we find the 32 -bit signature S for a database image:

$$S \rightarrow S_1 \left| S_2 \right| S_3 \mid S_4$$
$$\Rightarrow S \rightarrow 11000100001000010010000000110001100000011110000101000100000001 11$$

Now, suppose our query image signature is

$$S_q \rightarrow 10000101001000010010000000110001100000011110000101000100100011 11$$

Then, by performing a binary AND operation between S and S_q, we find:

$$
\begin{array}{c}
1000010100100001001000000011000110000001111000010100010010001111 \\
\cap \\
1100010000100001001000000011000110000001111000010100010000000111 \\
\hline
1100010000100001001000000011000110000001111000010100010000000111
\end{array}
$$

which is equal to the query image signature S_q. Hence, this database image will be accepted for the second stage.

Angle Signature from Quadtree Example:

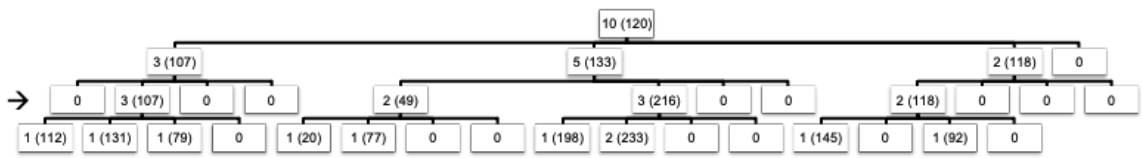


FIGURE 4.4: Signature Angles are encoded from Level 2

$$
\begin{array}{c}
\text{Signature in Figure 4.4: } (0\ 107\ 0\ 0)\ (49\ 216\ 0\ 0)\ (118\ 0\ 0\ 0)\ (0\ 0\ 0\ 0) \\
\Longrightarrow\ (1000\ 0100\ 1000\ 1000) \\
(1000\ 0010\ 1000\ 1000) \\
(0100\ 1000\ 1000\ 1000) \\
(1000\ 1000\ 1000\ 1000) \\
\Longrightarrow \\
(1000010010001000)(1000001010001000)(0100100010001000)(1000100010001000)
\end{array}
$$

4.3 K-Nearest Neighbour Algorithm

The basic version of the K-Nearest Neighbour algorithm assumes that all instances correspond to points in the n-dimensional space $\Re^n$ [46]. The nearest neighbours of an instance are often defined in terms of the standard Euclidean distance. More precisely, let an arbitrary instance be described by the vector $\langle x, f(x) \rangle$. f is the true concept function that gives the correct class value $f(x)$ for each instance x. The instance x can also be described by the feature vector

$$\langle a_1(x), a_2(x), \ldots a_n(x) \rangle$$

where $a_r(x)$ denotes the value of the rth attribute of instance x.

For our implementation, each feature vector represents a quadtree representation of the image. Therefore, K-NN algorithm helps us find nearest neighbour feature vectors, i.e., most similar images. The distance between two instances x_i and x_j is denoted as $d(x_i, x_j)$. The distance functions used in our experimentation are defined in Section 4.3.1.

Discrete-valued concept functions are of the form $f : \Re^n \rightarrow V$, where V is the finite set $\{v_1, \ldots v_s\}$ of class values [46]. The KNN algorithm is described in Algorithm 5.

Algorithm 5 k-Nearest Neighbour Algorithm [47]

Training algorithm:
For each training example $\langle x, f(x) \rangle$, add the example to the list *training_examples*

Classification algorithm:
Given a query instance x_q to be classified,
Let $x_1, \ldots x_k$ denote the k instances from *training_examples* that are nearest to x_q

$$\hat{f}(x_q) = \arg\max_{v \in V} \sum_{i=1}^{k} \delta(v, f(x_i))$$

where $\delta(a, b) = 1$ if $a = b$ and where $\delta(a, b) = 0$ otherwise.

The value $\hat{f}(x_q)$ returned by this algorithm as its estimate of $f(x_q)$ is just the mode (most common value) of the true concept function f among k training examples nearest to x_q. If $k = 1$, then the 1-NN algorithm assigns to $\hat{f}(x_q)$ the value $f(x_i)$ where x_i is the training instance nearest to x_q. For larger values of k, the algorithm assigns the mode among the k nearest training examples.

4.3.1 Distance Functions

Euclidean Distance

Euclidean distance [48] is the most commonly used metric when it comes to K-NN and other classification algorithms [49]. It is the distance between points in a straight line, using the Pythagoras theorem. The real world is a Euclidean space, hence Euclidean distance is the most natural to us. However, other distance measures outperform Euclidean in many scenarios, which may be unintuitive at first glance [50].

$$d_{Euclidean}\left(x_i, x_j\right) = \sqrt{\sum_{r=1}^{n} \left(a_r\left(x_i\right) - a_r\left(x_j\right)\right)^2}$$

Manhattan Distance

The Manhattan distance [51] was proposed by Hermann Minkowski in the late 19th century [52] and is defined as the sum of the absolute differences of the Cartesian coordinates. It is called the Manhattan distance because it is the distance a car would drive in a city (e.g., Manhattan) where the buildings are laid out in square blocks and the straight streets intersect at right angles. This is why it is also called City Block and Taxicab distance [53].

$$d_{Manhattan}(x, y) = \sum_{i=1}^{n} \left| \left(a_r\left(x_i\right) - a_r\left(x_j\right)\right| \right.$$

Braycurtis Distance

Braycurtis distance is a normalization method that views the space as a grid similar to Manhattan distance. An important property is that if all coordinates are positive, its value is between zero and one. The normalization is done using absolute difference divided by the summation [54].

$$d_{Braycurtis}(x, y) = \sum_{i=1}^{n} \frac{|a_r(x_i) - a_r(y_i)|}{(a_r(x_i) + a_r(y_i))}$$

Canberra Distance

The Canberra distance is a weighted version of the Manhattan distance initially proposed by Lance et al. [55] in the '60s. It measures the sum of absolute fractional differences between the features of a pair of data points and is very sensitive to a small change when both coordinates are nearest to zero [56].

$$d_{Canberra}(x, y) = \sum_{i=1}^{n} \frac{|a_r(x_i) - a_r(y_i)|}{|a_r(x_i)| + |a_r(y_i)|}$$

Chebyshev Distance

Chebyshev Distance measures distance between two points as the maximum difference over any of their axis values. Chebyshev distance is generally a useful distance measurement in games that allow unrestricted 8-way movement, where moving diagonally costs no more than moving in a cardinal direction [57].

$$d_{Chebyshev}(x, y) = \max_{i=1}^{n} |a_r(x_i) - a_r(x_j)|$$

4.4 Siamese Deep Neural Network

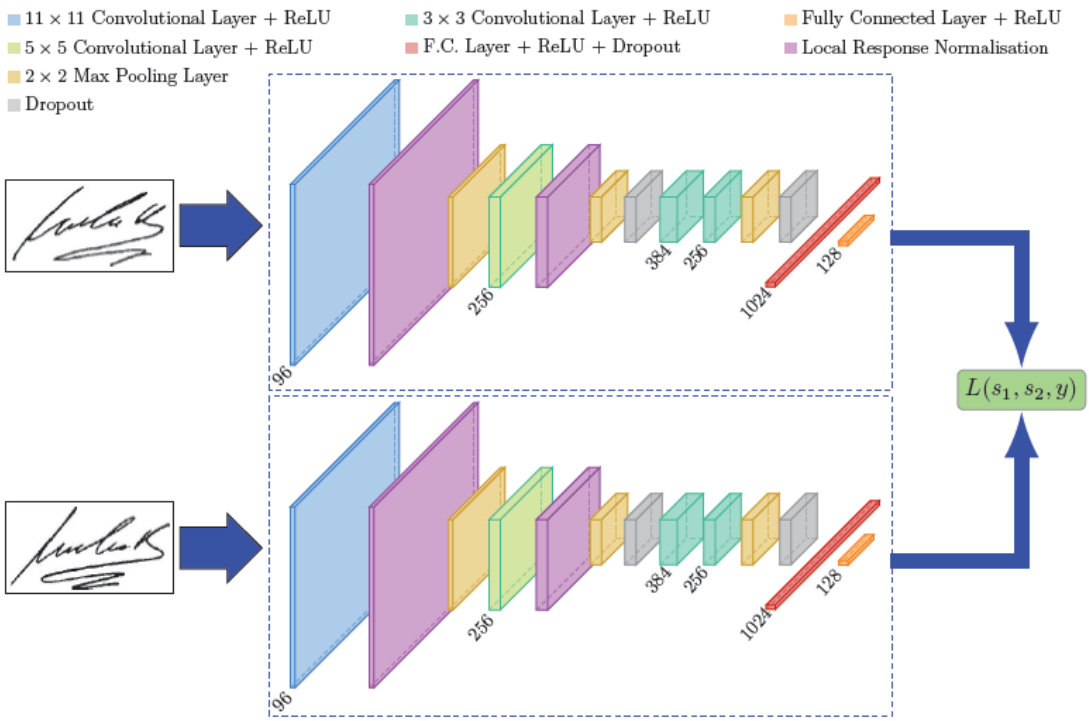


FIGURE 4.5: Siamese Network Example [58]

Our goal is to learn a general similarity function for image pairs. To encode such function, we propose a method based on a deep convolutional neural network [58].

In general, a pair of images goes through a network consisting of two branches during training. The outputs of these branches are fed to a loss layer. The loss layer tries to minimize squared Euclidean distance between the features of positive image pairs ($f(I_1)$ and $f(I_2)$) and maximize it for negative pairs.

We use a cost function which is capable to distinguish between pairs. More precisely, it encourages similar examples to be close, and dissimilar ones to have Euclidean Distance of at least margin m from each other.

To implement this, we use the margin-based contrastive loss function

proposed by Hadsell et al. [59] which is defined as follows:

$$\mathcal{L} = \frac{1}{2} l D^2 + \frac{1}{2}(1-l)\{\max(0, m-D)\}^2$$

where l is a binary label selecting whether the input pair consisting of image I_1 and I_2 is a positive $(l = 1)$ or negative $(l = 0), m > 0$ is the margin for dissimilar pairs and $D = \|f(I_1) - f(I_2)\|_2$ is the Euclidean Distance between feature vectors $f(I_1)$ and $f(I_2)$ of input images I_1 and I_2

Dissimilar pairs contribute to the loss function only if their distance is within the margin m. This loss function encourages matching pairs to be close together in feature space, while pushing non-matching pairs apart. Moreover, it can be clearly seen that negative pairs with a distance which is bigger than margin would not contribute to the loss.

The loss function penalizes positive pairs by the squared Euclidean distances and negative pairs by the squared difference between margin m and Euclidean distance for pairs which have a distance less than a margin m [58].

The hyperparameters and network structure can be adjusted depending on the kind of dataset, but the overall pattern of a neural network stays consistent. For our implementation, we will pass quadtree representations of the image, rather than the images directly, into the network.

Chapter 5

Implementation and Results

5.1 Image Preprocessing

Preprocessing of the image is done using the Python OpenCV library. The two essential steps while preprocessing the image are background removal and rotational correction. Our methodology relies on the background and rotation of the image to be handled before generation of keypoints.

The background removal step to bring the central object to the fore of the image is imperative so that the quadtree decomposition is evenly spread out among the quadrants, and the depth of the tree is likelier to be uniform for similar images. It is also necessary that images are rotated correctly because the generated quadtree is contingent on the spatial location of the points. For generated orientation values to be consistent across similar images, rotating the images uniformly helps. The user can customize the choice of preprocessing algorithms and parameters depending on the different characteristics of a dataset.

5.1.1 Background Removal

For Background Removal, we find the structuring element of the image. The structuring element is a crucial subject in morphological image processing, as the characteristics of the structuring element can affect the opening and closing operations. Opening is a technique for removing small objects from the front of a picture and placing them in the background [60]. In contrast, closing removes small foreground holes, transforming tiny islands of background into the foreground. The pixel value

1 is set on the structuring element as the foreground, while 0 is set as the background [61].

The morphological closing operation dilates the input image and will remove the background structure that is smaller than the structuring element. For background removal, we use morphological closing on each image [60].

For its implementation, we use advanced structural and morphological functions $cv.getStructuringElement()$ and $cv.getMorphologyEx()$ to extract the structuring element, perform the closing operation and then store the contour using $cv.findContours()$. The contour is then masked onto a new image, to extract a background-free image.

FIGURE 5.1: Before Background Removal

FIGURE 5.2: After Background Removal

5.1.2 Rotational Correction

After the background is removed and the central object (structural element) is maximized within the image, we derive a straight line along the "weight" of the image. The largest area contour is extracted, then we derive the starting and ending points of the line, which are calculated by minimizing the Euclidean distance between the line and all the points. These points are used to plot a line using $cv.line()$ on the image as shown in Figure 5.3. We then find the angle between the line and x-axis using their dot product and its cosine. If the derived angle is greater than $-45°$, we subtract 90°; if the derived angle is less than $-45°$, we add 90°. We do

this to fix a tilting problem because different images may tilt in different directions.

Once the angle of the line is calculated, we rotate the dominant contour of the image by that angle. To rotate an image, we first derive the rotation matrix of the contour with the calculated angle using $cv.getRotationMatrix2D()$. The rotation matrix is a 4x4 matrix of sin and cos values, and it is used to represent the orientation of an object. After some trigonometric operations with this rotation matrix on the contour coordinates, we use $cv.warpAffine()$ to derive the rotated contour. Rotation, translation or scaling are affine transformations of an image, as these operations preserve collinearity, parallelism and the ratio of distances between the points [62]. This contour is then placed on a blank image using masking, to get the final preprocessed image.

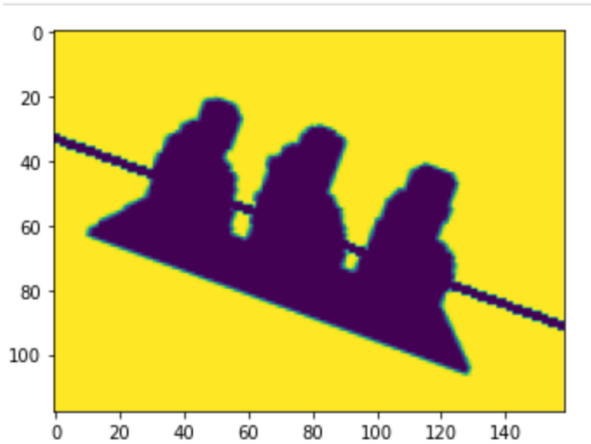


FIGURE 5.3: Rotated across this line for correction. This line aptly represents the orientation of the object in the image.

5.2 Keypoint Generation

Based on the extensive comparisons between different feature descriptors such as SIFT, SURF, KAZE, AKAZE and ORB, Tareen et al. [43] have concluded:

> "The overall accuracy of SIFT and BRISK is found to be highest for all types of geometric transformations, and SIFT is concluded as the most accurate algorithm."

Also, after generating keypoints of images using different feature descriptor algorithms, we determined heuristically that SIFT better encapsulated points of interest and their respective orientations, and was likelier to create more representative symbolic images. Therefore, we use SIFT in our implementation. The $cv.xfeatures2d.SIFT\_create()$ function creates a SIFT object, and then $SIFT.detect()$ generates the keypoints from that object. Each keypoint consists of critical information such as the (x,y) coordinates, size of the feature descriptor and strength of the keypoint. We use the first 500 keypoints based on their respective strength, to neglect inaccurate keypoints. Figure 5.4 shows the output of selected SIFT points on an image from the Moth dataset.

Figure 5.4: SIFT Points generated on Moth Dataset

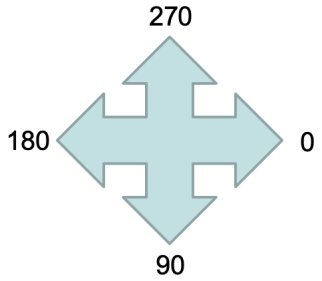


FIGURE 5.5: Angle Directions for Figure 5.6 and Figure 5.7

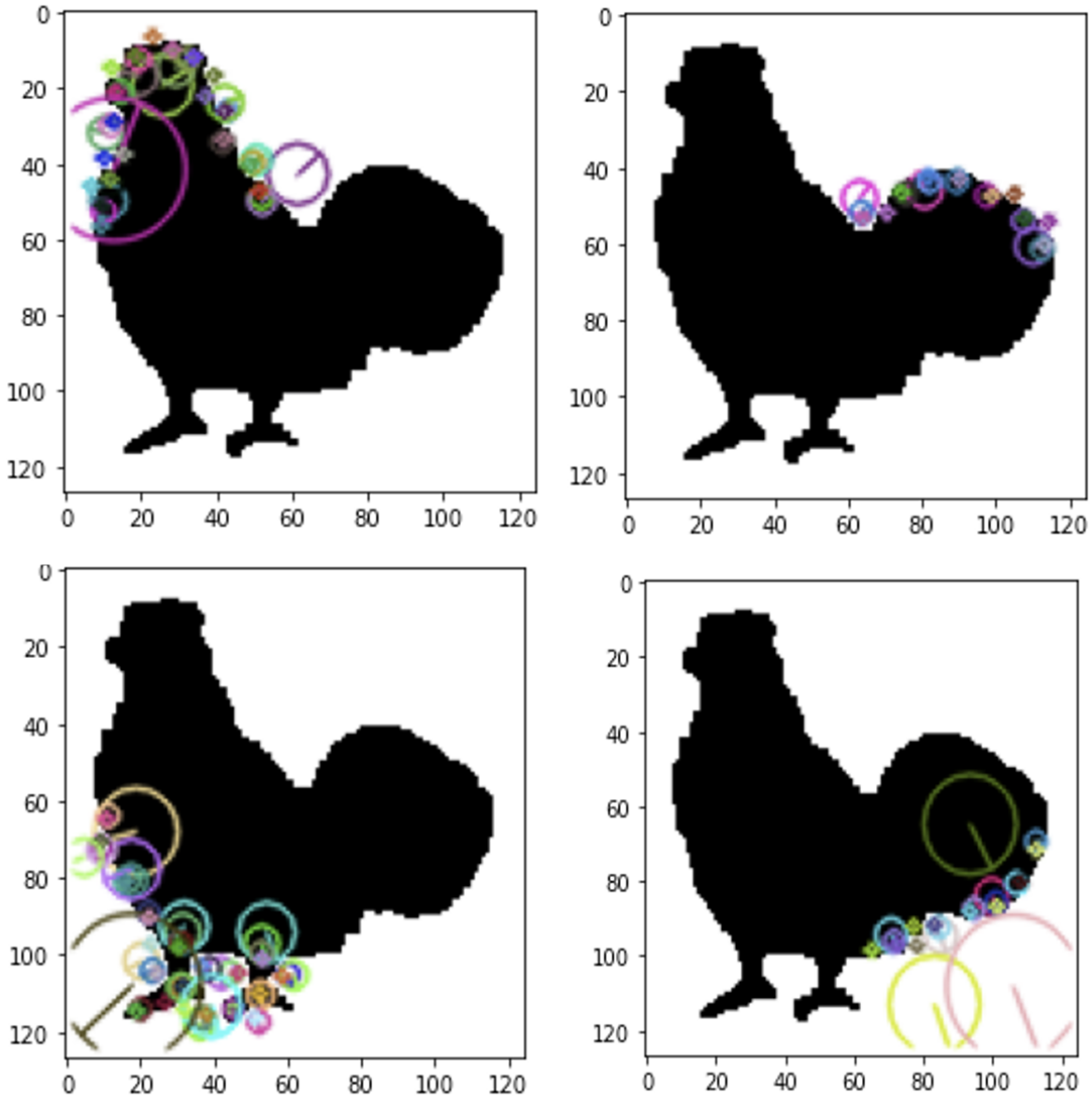


FIGURE 5.6: Level-1 Quadtree Average Angles - (top-left: 268°) (top-right: 291°) (bottom-left: 111°) (bottom-right: 60°)

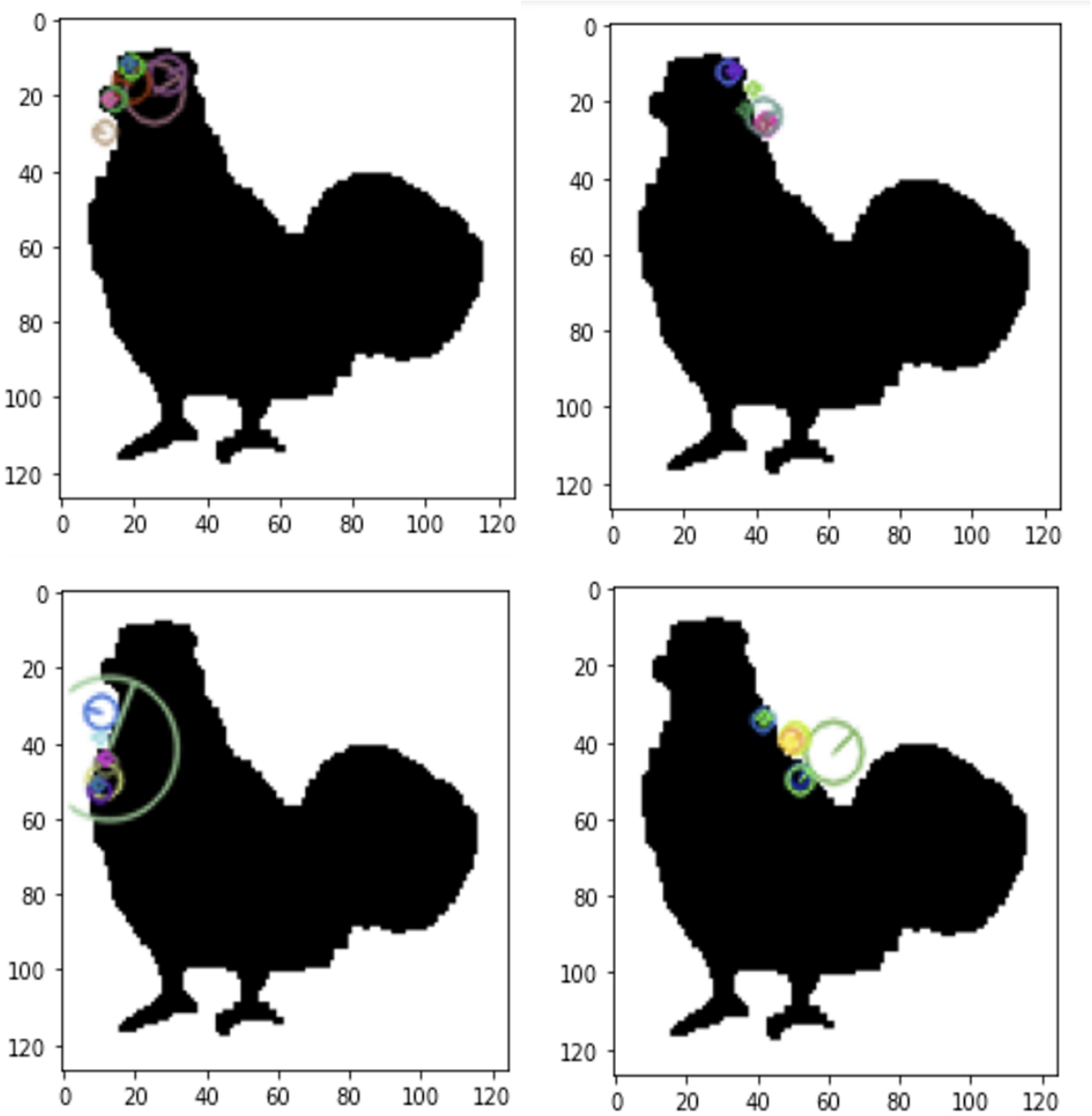


FIGURE 5.7: Level-2 Top-Left Quadtree Average Angles - (top-left: 233°) (top-right: 340°) (bottom-left: 202°) (bottom-right: 324°)

5.3 Datasets

For our experimental analysis, we have used three image datasets – SimDB, FaceDB and MothDB. The datasets vary from each other considerably in terms of their complexity and the number of images. The images from FaceDB and MothDB are much more intricate in terms of illumination and

intensity variation than the images from SimDB. As a result, it is more challenging to produce consistent symbolic images using older methods. Table 5.1 summarizes information about datasets used in our implementation.

5.3.1 SimDB

The SimDB database, formally known as MPEG7 CE-Shape-1 database [63], is a small shape database that consists of binary images of spatially similar objects collected from [64]. The original dataset contains a total of 1,400 silhouette binary images from 70 classes, where each class has 20 different shapes. We have collected images of 20 objects such that each object has 5 spatially similar images, making it a database of just 100 images. This database is essentially used for testing and preliminary analysis.

5.3.2 FaceDB

The second database we'll use is FaceDB. It refers to the popular Olivetti-Oracle Research Lab (ORL) face database [65]. FaceDB consists of 400 grayscale frontal face images; images of 40 individuals with 10 variations of each in terms of pose, illumination, facial expression (open/closed eyes, smiling/not smiling) and facial details (glasses/no glasses) [11].

5.3.3 MothDB

The dataset of Janzen and Hallwachs, derived from long-term sampling and caterpillar rearing, includes a broad range of moth and butterfly taxa sampled in north-western Costa Rica [66]. For simplicity, this dataset is called MothDB. We reduced the dataset to female individuals only and species with 6 images each. We've selected a total of 1590 images. As can be seen from the overview figure of the MothDB dataset in Figure 5.10, some categories have a very characteristic background, which we've removed during image preprocessing. The images that have different sizes are normalized by scaling to a fixed size for experimentation.

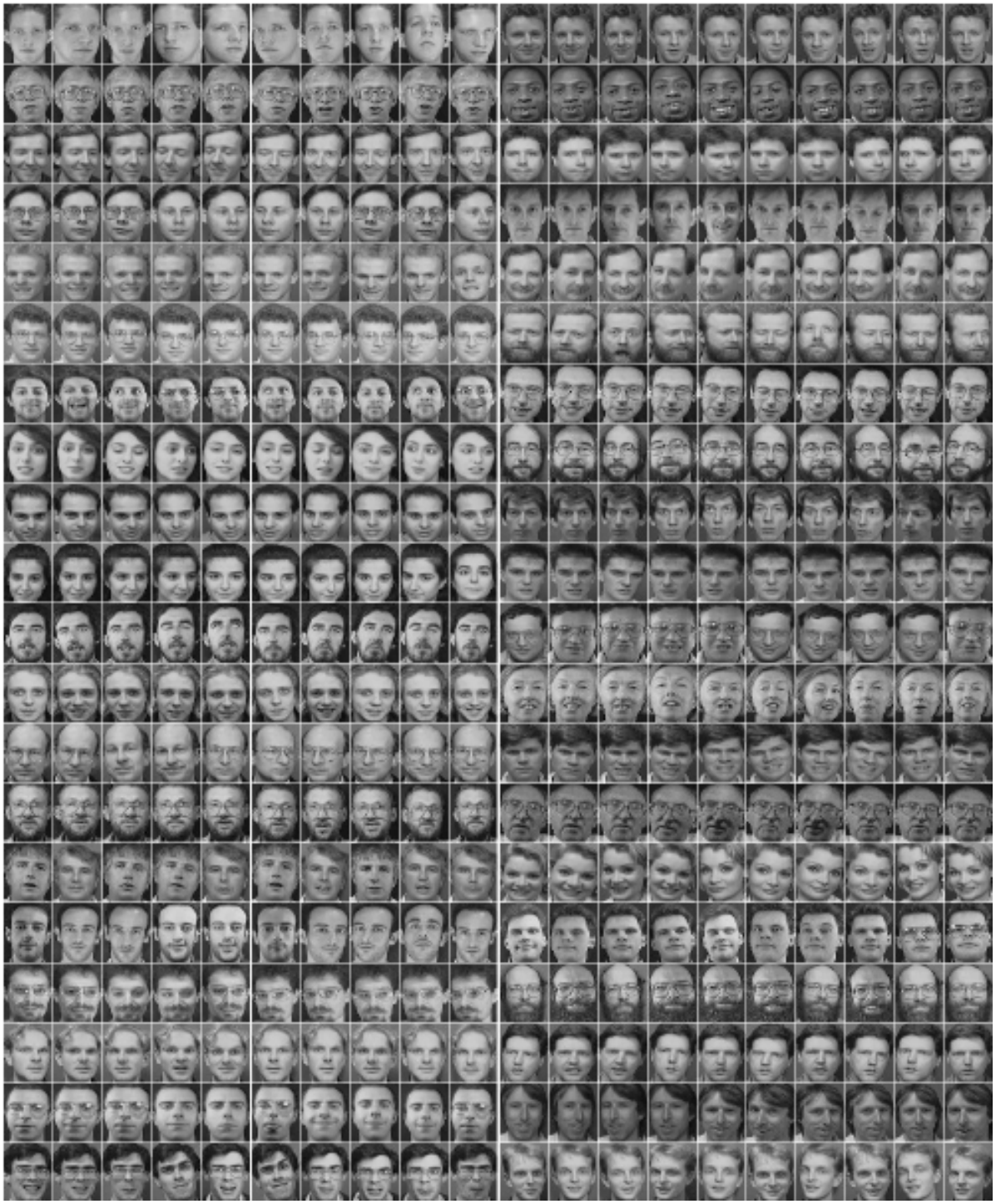


FIGURE 5.8: FaceDB - ORL Database [67]

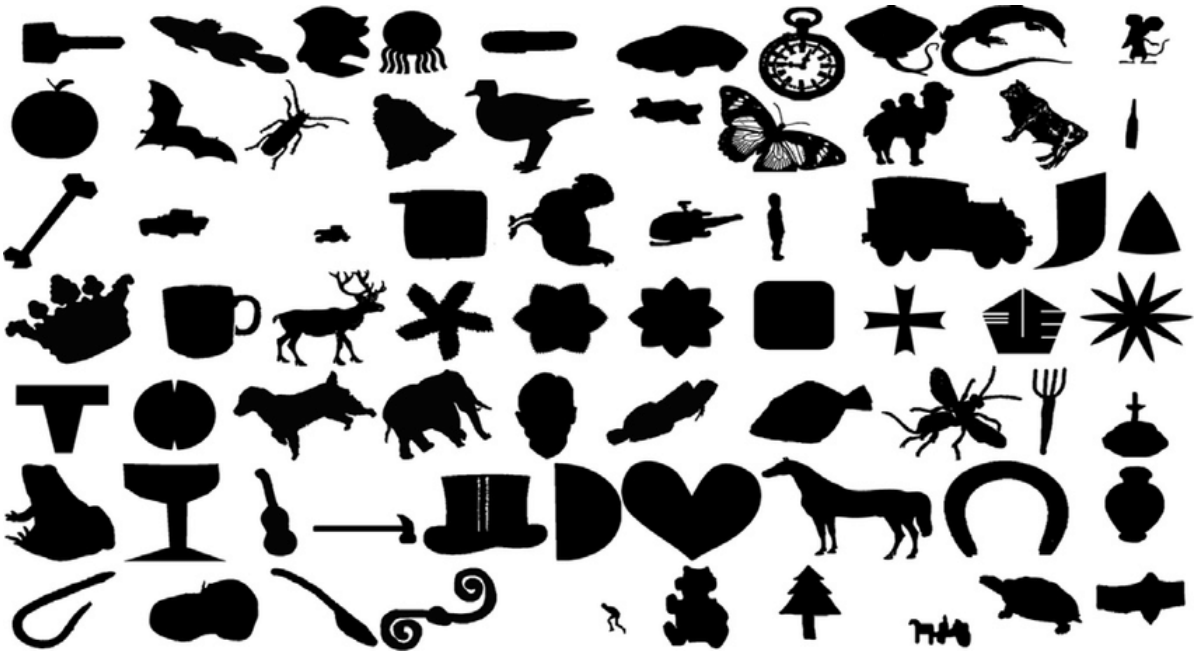


FIGURE 5.9: SimDB - MPEG7 Shape Database [68]

FIGURE 5.10: MothDB Database [69]

Dataset	No. of Images	Images / Class	Characteristics
SimDB [64]	100	5	2D Objects Binary images For testing Variation in rotation, scaling, skewing of similar objects
FaceDB [65]	400	10	Human faces Grayscale Variation in pose, illumination, facial expression of the same human
MothDB [66]	1590	6	Moths of different species Coloured Complex patterns Variations are different specimens of the same species

Table 5.1: Summary of datasets and their characteristics

5.4 Experimental Setup

The experimental results for existing and proposed systems are all collected on the same hardware configurations. We've used a personal computer with a 3.3 GHz Dual-Core Intel Core i7 processor, 16 GB RAM and MacOS 11.4.

The image preprocessing and feature description generation stages are programmed using Python on a Jupyter Notebook. OpenCV Python library is used extensively throughout the experimentation. Not only was it used in the background removal and rotation invariant stages, but also while creating an experimental multi-object image retrieval system using image segmentation (Appendix A). Numpy and Pandas are also used for complex image calculations and transformations. For the machine learning stage, we've used scikit-learn, and the deep learning Siamese networks are created using Tensorflow. For Tensorflow, we've used a Google Collab notebook for additional computation.

In an object-oriented pattern, the primary quadtree generation, signature indexing, quadtree comparison, feature representation and metric & analysis stages for both existing and proposed methodologies are programmed with Java 8. No additional Java libraries are used in our implementation.

5.5 Signature Indexing Retrieval Results

During signature indexing retrieval, while it is acceptable to retrieve false positives, true positives mustn't be discarded [27]. To show image retrievals across all images, we have used a precision-recall heatmap. Each point on the heatmap indicates one image's best possible retrieval. There are 100 such points for the SimDB graphs, as there are 100 images. Similarly, there are 400 points for FaceDB and 1690 points for MothDB. We use a heatmap because it is easy to illustrate a concentration of points. As we can see in Figure 5.11, dark red indicates the highest concentration of images with a certain precision and recall value, while dark blue indicates the least or none.

To arrive at the precision and recall values for each image in the heatmap, we have used Algorithm 6. It iterates through all images in a dataset and calculates the best possible precision-recall value for each image signature retrieval. This algorithm ensures that for each image, we plot the best possible retrieval result. It makes our comparison uniform across different parameters of $toleranceVal$ and $ignoreBlocksVal$, and reduces the chance of parameter favouring. During application, the user would tune the tolerance and ignoreBlock ranges manually.

$toleranceVal$ is the same as the tolerance value defined in the previous sections. $ignoreBlocksVal$ is an arbitrary parameter that we've introduced to further increase the flexibility of the signature. If set to 0, all 16 bit blocks (each bit block corresponds to a level 2 quadrant) in the two signatures need to be a match. As the $ignoreBlocksVal$ increases, the comparison loosens up and allows us to ignore $16 - ignoreBlockVal$ signature bit blocks. For example, if $ignoreBlocksVal = 1$, then if two signatures have 15 bit blocks in common, they are still a match. So, while the $toleranceVal$ increases the tolerance of the signature by value, $ignoreBlocksVal$ increases the tolerance by block.

Algorithm 6 Signature Indexing Precision-Recall Heatmap

```
tolSet ← (0, 5, 10, 15, 20)                    ▷ If Spatial Signature Indexing
tolSet ← (0°, 10°, 20°, 30°, 40°)              ▷ If Angle Signature Indexing
ignoreSet ← (0, 1, 2, 3, 4, 5)
for imageId i in dataset do
    metricsForImageList ← {}
    for toleranceVal t in tolSet do
        for ignoreBlocksVal g in ignoreSet do
            imagesList ← findSimilarImages(signatureFor(i, t, g))
            precision, recall, fScore ← calculateMetrics(imagesList)
            metricsForImageList.add(precision, recall, fScore)
        end for
    end for
    bestPrecision, bestRecall ← takeMaxFScore(metricsForImageList)
    plotToHeatmap(bestPrecision, bestRecall)
end for
```

Along with the heatmap, we have also listed the average values of

precision, recall, fScore, tolerance value and ignoreBlock value for each image in the heatmap. Values like this can be generated to aggregate potential parameters for a dataset using a subset of images.

We list the results for SimDB, FaceDB and MothDB in the subsequent sections.

FIGURE 5.11: Colour Bar from Blue (low) to Red (high)

5.5.1 SimDB (100 Images)

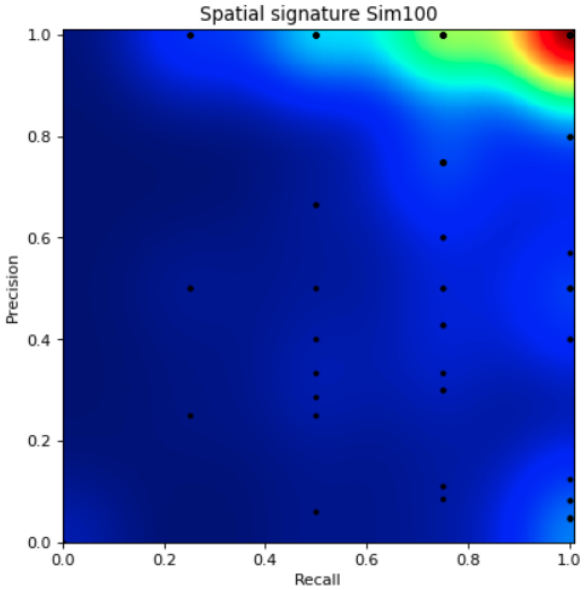


FIGURE 5.12: SimDB Spatial Signature Retrieval

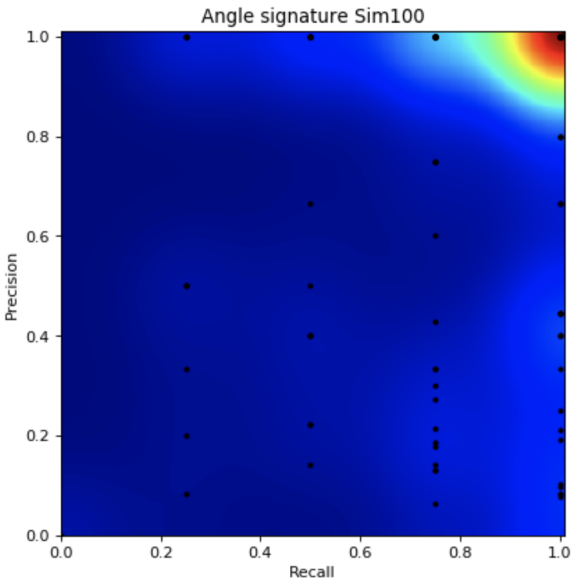


FIGURE 5.13: SimDB Angle Signature Retrieval

	Spatial SimDB	Angle SimDB
Precision Average	0.738	0.672
Recall Average	0.745	0.78
FScore Average	0.674	0.628
Tolerance Average	7.57	22.4
Ignore Blocks Average	1.92	1.32

TABLE 5.2: SimDB Heatmaps Value Analysis

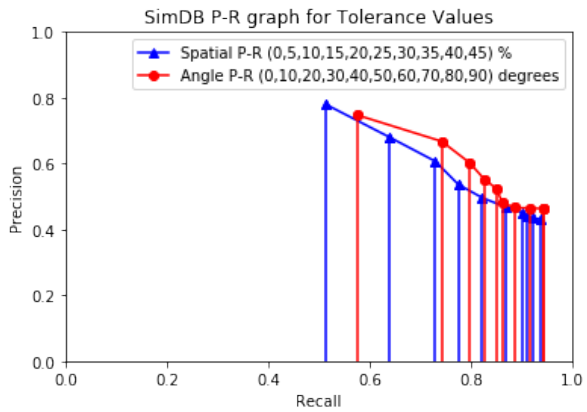


FIGURE 5.14: SimDB P-R graph for Tolerance Values

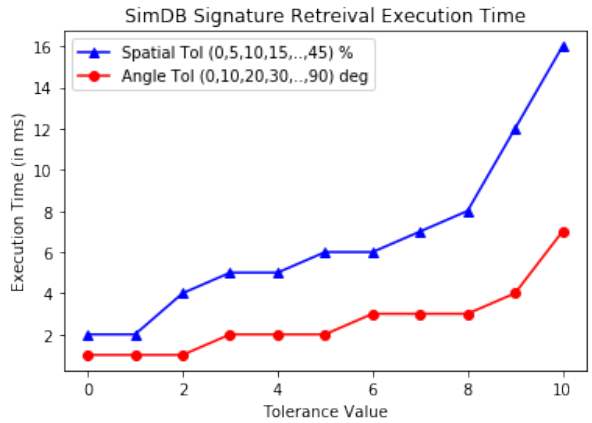


FIGURE 5.15: SimDB Signature Retrieval Execution Time

For SimDB, we can see that both the existing spatial signature retrieval and the angle signature retrieval methods perform similarly well. Both heatmaps (Figure 5.12 and 5.13) have most precision-recall values close to 1-1. It is important to note that, on average across all tolerance values, the spatial scheme gets more precise results than the angle scheme, albeit with a slightly lower recall. This is because of the higher average F-Score value for spatial. However, Figure 5.14 indicates that at a recall of 0.8 and more, our scheme has slightly higher precision. This means that as the flexibility of both signatures increases, we get lesser false positives. We hypothesize that our angle signature scheme does not perform better overall because SimDB has black and white pictures with only structural quality. It, therefore, makes much more sense to use region spatial points than angle points, as they don't add much value. We can conclude that our signature scheme does not contribute much if the images in a dataset aren't feature-rich enough. Nevertheless, the execution of angle signature retrieval is significantly faster, especially for higher tolerance values.

5.5.2 FaceDB (400 Images)

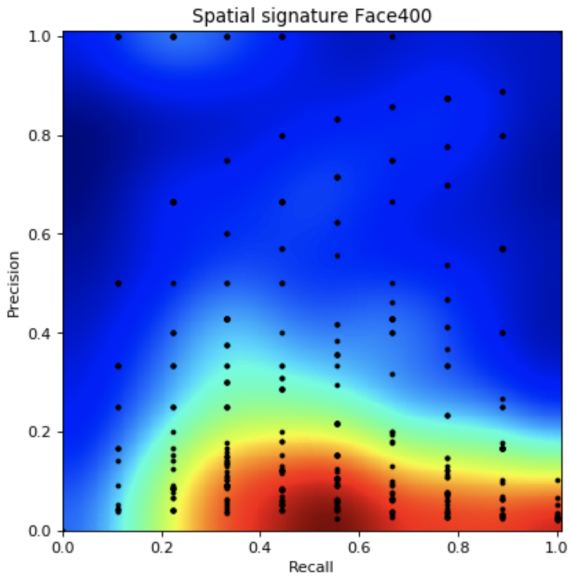


FIGURE 5.16: FaceDB Spatial Signature Retrieval

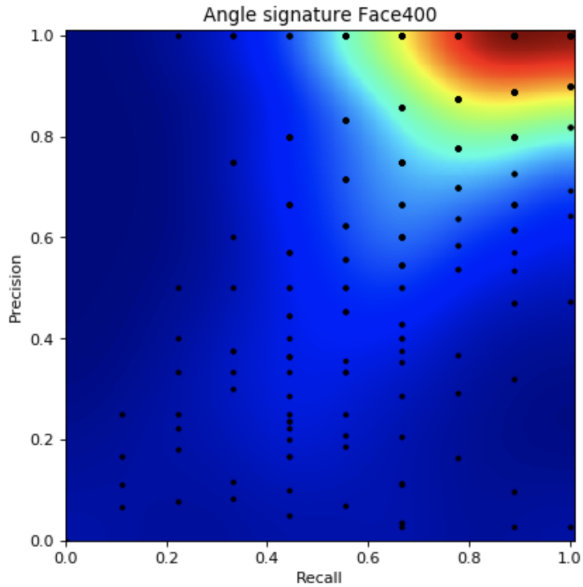


FIGURE 5.17: FaceDB Angle Signature Retrieval

	Spatial FaceDB	Angle FaceDB
Precision Average	0.297	0.749
Recall Average	0.528	0.702
FScore Average	0.288	0.70
Tolerance Average	6.175	28.35
Ignore Blocks	0.632	2.97

TABLE 5.3: FaceDB Heatmaps Value Analysis

For FaceDB, our angle signature scheme does have precision-recall values closer to 1-1 (Figure 5.17), though the results aren't as accurate as SimDB. The spatial results, though, are wildly inaccurate (Figure 5.16), with an average precision of merely 0.3. Here, the advantages of our proposed angle signature scheme become much more apparent. FaceDB, though grayscale, has very complex features and gradient distributions around them. Every face has nearly the same structural features – two eyes, one nose, two ears, e.t.c, which is why the gradient around feature points becomes crucial to take into account. This is where the orientations of illumination provided by our generated feature descriptors become much more helpful.

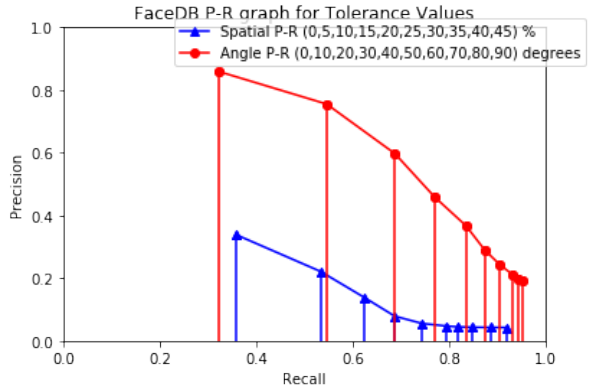


FIGURE 5.18: FaceDB P-R graph for Tolerance Values

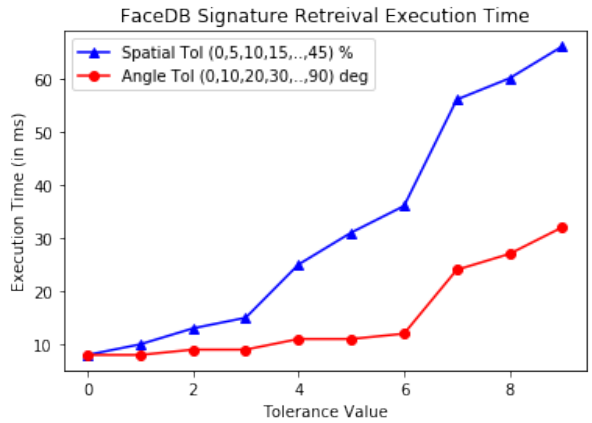


FIGURE 5.19: FaceDB Signature Retrieval Execution Time

5.5.3 MothDB (1590 Images)

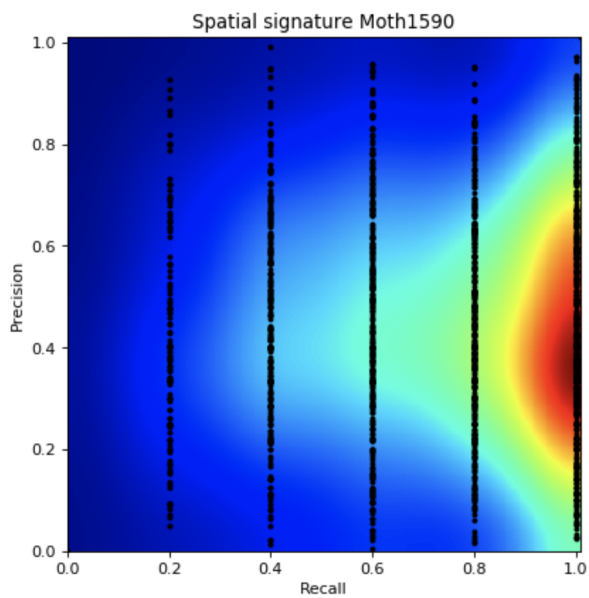


FIGURE 5.20: MothDB Spatial Signature Retrieval

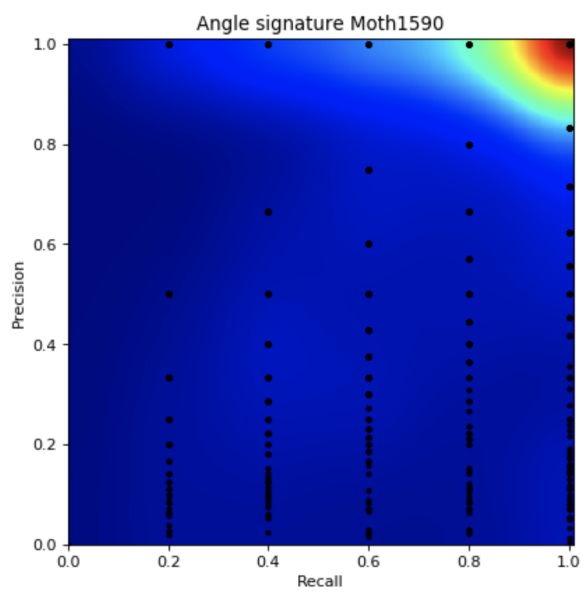


FIGURE 5.21: MothDB Angle Signature Retrieval

	Spatial MothDB	Angle MothDB
Precision Average	0.445	0.744
Recall Average	0.746	0.709
FScore Average	0.567	0.677
Tolerance Average	5.325	12.43
Ignore Blocks	2.675	1.06

TABLE 5.4: MothDB Heatmaps Value Analysis

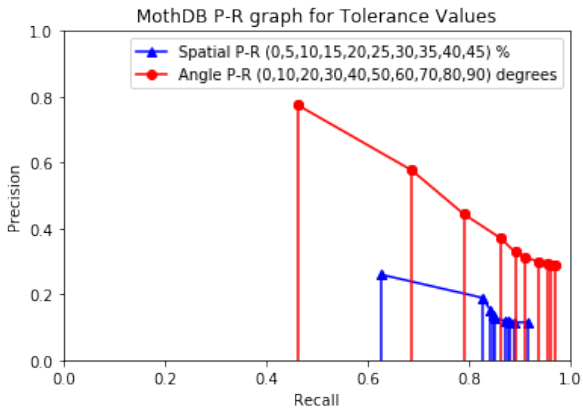


FIGURE 5.22: MothDB P-R graph for Tolerance Values

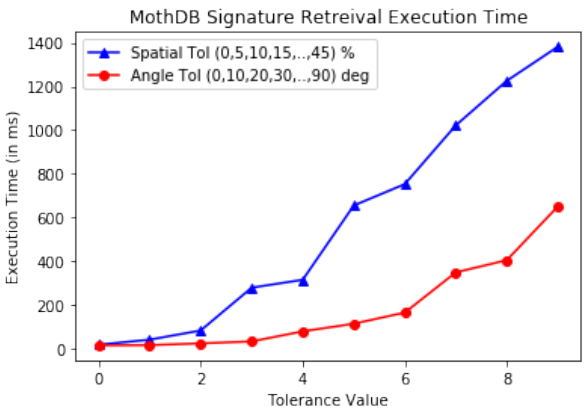


FIGURE 5.23: MothDB Signature Retrieval Execution Time

In the MothDB database, the spatial signature scheme (Figure 5.20) performs better than it did for FaceDB, because each moth has structural features that the existing spatial scheme can exploit. But our angle signature scheme (Figure 5.21) still outperforms the spatial scheme by around 0.3 precision points on average. Each moth has patterns on its body that our feature angle quadtree can meaningfully extract information from. This dataset is an ideal representation of the kind of images our orientation-based methodology is useful for.

5.6 Quadtree Comparison Results

As established already, the signature scheme filters images for more evolved quadtree comparisons. This section evaluates how well quadtree comparisons work for spatial and angle quadtrees on all datasets. We have used Mean Average Precision (mAP) to evaluate the quality of our results, as

it rewards the metric if true positives are retrieved first (in order) and penalizes results in which non-similar images are placed in the beginning. This is appropriate for us, as we get a similarity value between 0.0 and 1.0 for each quadtree comparison between two images, rather than absolute true or false values like in signature retrieval.

The results in Figure 5.24, 5.25 and 5.26 compare our quadtree comparison algorithms with Ahmad et al.'s. We do this by retrieving similar images for each image in the dataset, taking the mAP score, and then averaging the mAP for all images. We do this across varying depths of the quadtree. Our results show that for SimDB, the existing spatial quadtree comparison algorithm performs slightly better, although our scheme is much better for FaceDB and MothDB. This is, again, because of the lower orientational complexity of the features of SimDB, which are either completely black or completely white. The existing scheme is a better choice for purely structural image shapes, while our scheme can also consider the underlying feature complexities, illuminations, and gradients.

Another observation from these graphs is that in SimDB, the results become nearly consistent after depth 4. This is due to the availability of lesser feature points in SimDB, so the feature representation quadtrees of each image in SimDB rarely exceed a depth of 4. FaceDB similarly plateaus at around depth 6, and MothDB at depth 8. The more complex the feature representation of an image is, the more depth the quadtree has.

We also postulate that the mAP falls after depth 4 for FaceDB and MothDB for spatial quadtree comparisons because as quadtrees take into account more and more complex features (and quadtree depth), the results become worse. After all, these datasets aren't purely structural like SimDB. Too much emphasis on only structural and spatial representation penalizes the quadtree comparisons. On the other hand, our angle quadtree comparison becomes progressively better as, contrarily, the increased depth helps improve feature representation.

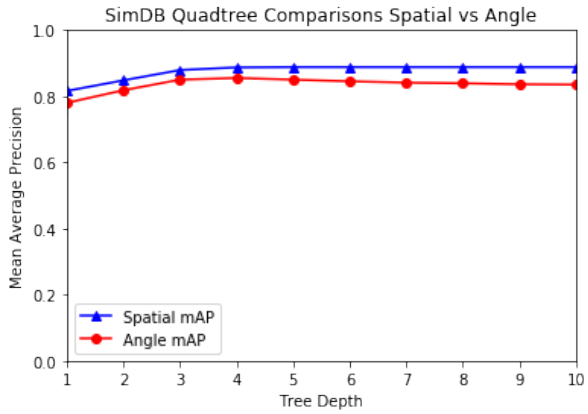


FIGURE 5.24: SimDB Spatial vs Angle

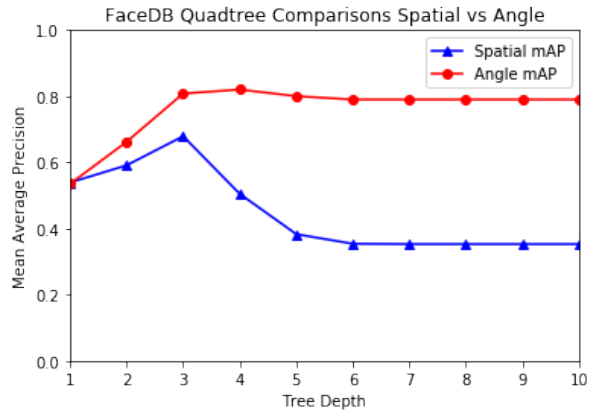


FIGURE 5.25: FaceDB Spatial vs Angle

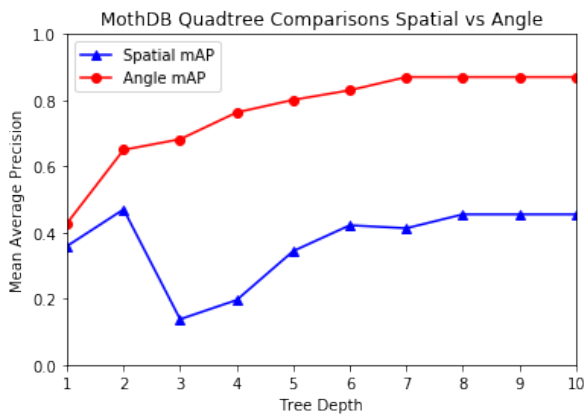


FIGURE 5.26: MothDB Spatial vs Angle

The angle quadtree comparison algorithm has no efficiency improvement, w.r.t the existing spatial quadtree comparison algorithm. Each quadtree comparison takes between 0 and 1 milliseconds for both the comparison algorithms, with minuscule and predictable variations with changing depth.

5.7 K-Nearest Neighbours Results

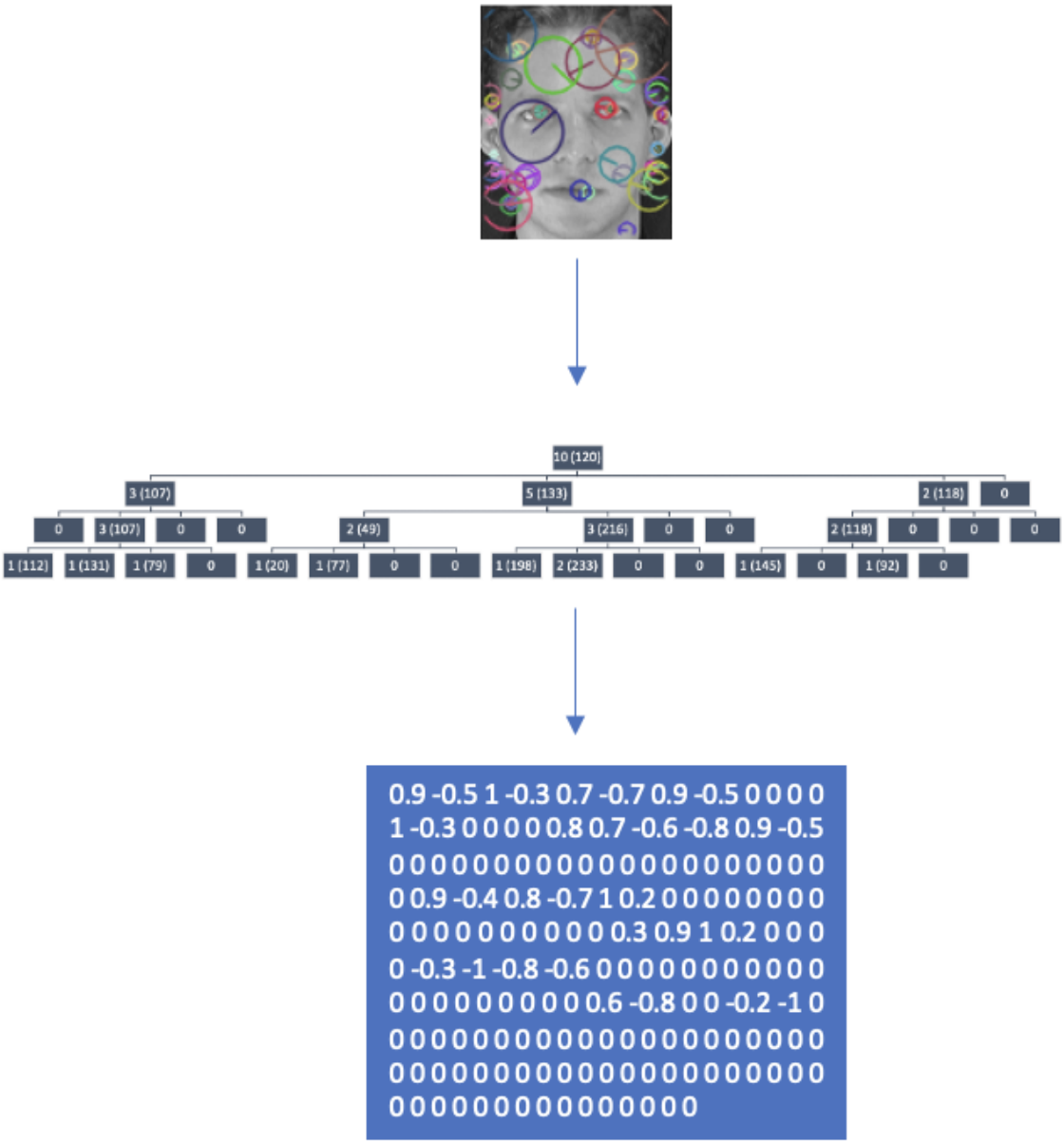


FIGURE 5.27: Example of feature vector which is used in KNN

For using K-NN on quadtree representations, we first derive a cyclical sin-cos representation of the orientation-valued quadtree as described in Section 3 and shown in Figure 5.27. Essentially, we convert the computed orientation-valued quadtree in each image to a simple feature vector with angle (sin-cos) values, by depth-first traversal. This process helps us

transform our generated angle quadtree into a feature vector suited for machine learning and deep learning models training. As the depth of the quadtree increases, the size of the feature vectors for each image also increases.

After these feature vectors are derived, it is only a matter of running KNN on each feature vector one by one. KNN helps in finding all feature vectors "closest" to the input feature vector. In other words, it acts as a similar image detection system because each feature vector represents an image.

We run KNN on each image in the dataset and calculate the mAP score for each image. We can calculate the mAP because every "neighbour" or image result has a closeness score, which determines the output result's place in the $similarImageList$, a critical metric needed to calculate the mAP score.

We use various distance functions to calculate these distances – Bray-Curtis, Canberra, Manhattan, Euclidean and Chebyshev. These distance functions are detailed in Section 4.3.1. The calculated mAP scores for each depth and distance function are enlisted in Table 5.5. Based on our results, we can see that for FaceDB and MothDB we get nearly 80% mAP score with a quadtree depth of 4. The mAP of our proposed representation is slightly lesser for SimDB due to the database's lack of feature complexity. We believe that more complex images with multiple distinct features would provide better results, as the feature vectors would become progressively unique.

TABLE 5.5: K-NN Results for Quadtree of varying depths

	Brycts.	Cnbra.	Mnhtn.	Eucldn.	Chbshv.
SimDB - 100 Images					
Depth 2, 21 Nodes	0.53	0.52	0.54	0.50	0.39
Depth 3, 85 Nodes	0.63	0.62	0.63	0.60	0.35
Depth 4, 341 Nodes	0.65	0.64	0.65	0.61	0.29
FaceDB - 400 Images					
Depth 2, 21 Nodes	0.72	0.706	0.72	0.70	0.66
Depth 3, 85 Nodes	0.77	0.75	0.77	0.73	0.66
Depth 4, 341 Nodes	0.78	0.76	0.78	0.75	0.67
MothDB - 1590 Images					
Depth 2, 21 Nodes	0.57	0.52	0.57	0.55	0.48
Depth 3, 85 Nodes	0.71	0.64	0.71	0.70	0.50
Depth 4, 341 Nodes	0.80	0.71	0.79	0.81	0.49

5.8 Siamese Network Results

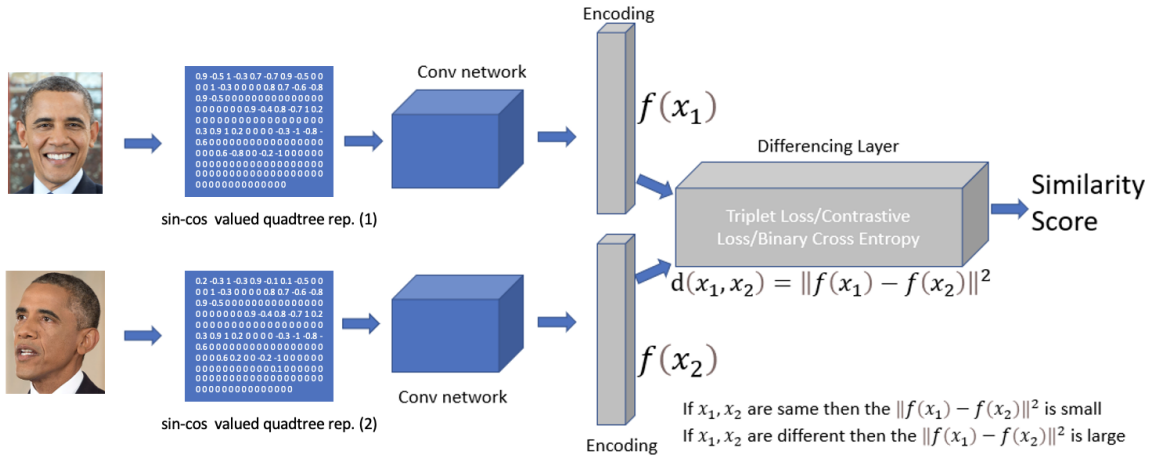


FIGURE 5.28: Abstracted overview of our Siamese Network implementation [70]

To train our angle quadtree representation scheme using Siamese Deep Neural Networks, we split the dataset into 70% training data and 30% test data. Then, we generate random similar and dissimilar pairs from

the training dataset. These pairs are used to train the neural network. The accuracy and loss are calculated on the unseen test dataset.

Contrastive loss takes the network's output for a positive example, calculates its distance to an image of the same class, and contrasts it with a negative image. Essentially, the contrastive loss evaluates how well the Siamese network is distinguishing between the image pairs.

Our results, in Figure 5.29 and 5.30, indicate an accuracy of about 80% for MothDB and FaceDB. This accuracy indicates that, given any two random pairs in the dataset, the algorithm can tell with 80% accuracy if the two images are similar or not.

We believe that the network's accuracy can improve further if larger datasets can be trained on more complex networks. Currently, our technique is meant to be added to existing neural networks through ensemble or boosting methods. Our methodology could help add more valuable training data (the derived quadtree structure representations) to the network. The derived feature vector of an image is extremely lightweight and agglomerative compared to taking all pixel values of the image, so the implementation cost would be significantly lesser.

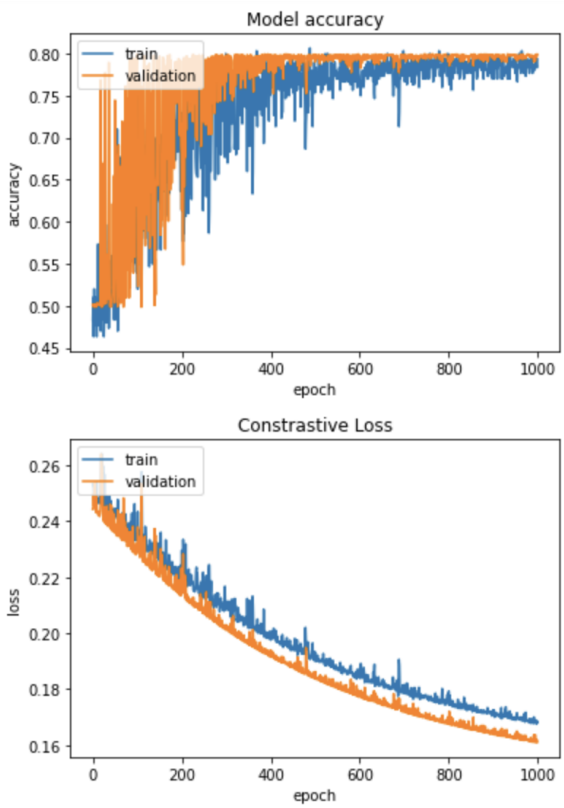


FIGURE 5.29: Siamese Face - 400 (341sincos)

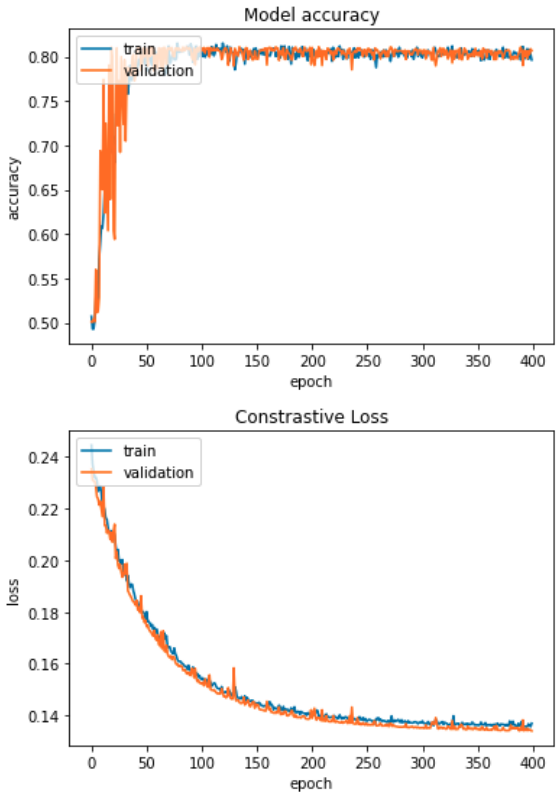


FIGURE 5.30: Siamese Moth - 1590 (341sincos)

Chapter 6

Conclusion & Future Work

Our results indicate that our quadtree generation scheme, based on the orientation of feature descriptors, works significantly better than using spatial feature corners, especially on images with more complex features. We have demonstrated that our proposed framework can be used with various image retrieval methods, such as binary signatures, machine learning and deep learning. The derived quadtree representation scheme can also be used separately as a dimensionality-reduction method by representing each image as a simple feature vector of numeric values. This feature vector representation can be used within various existing networks and can provide orientational value to the network, as Gorkani et al. [13] had stipulated. For example, feature vectors of images could be integrated into larger deep neural networks using feature embedding.

One drawback of our proposed scheme is that the object within the image needs to be identified during preprocessing and rotated (at least partially) to its natural orientation. We have used relatively primitive computer vision techniques for background removal and rotational correction. Combined with advanced preprocessing and object detection methods, researchers can use our quadtree representation scheme to index or better train images with little additional overhead.

In the future, researchers could work on multi-object image retrieval. They could create quadtree representations not only for the image at a global level, but also for various objects within the image at a local level. We have implemented a scheme based on this idea, in which images with multiple objects can be indexed in a dataset for CBIR. It is detailed in

Appendix A. However, this is a preliminary implementation and works only for images that are trivial to segment, such as SimDB's images. Researchers could work on segmenting complex images with background noise.

Another future research direction could be to harness the histogram values derived from each feature descriptor. In our proposed technique, we have considered only the feature point orientation derived from this descriptor. We believe that future quadtree-based feature representations of the image could be constructed using the 128 histogram values around the keypoint region. Either separate quadtrees could be built using each feature descriptor, or a global quadtree could incorporate all histograms.

Appendix A

Multi-object Similar Image Retrieval

To address the problem of multi-object similar image retrieval, objects within the image can be indexed as quadtrees and signatures of their own, instead of entire images. We preliminarily implemented and outlined a framework for combining image segmentation techniques with our methodology to work for multiple objects within an image.

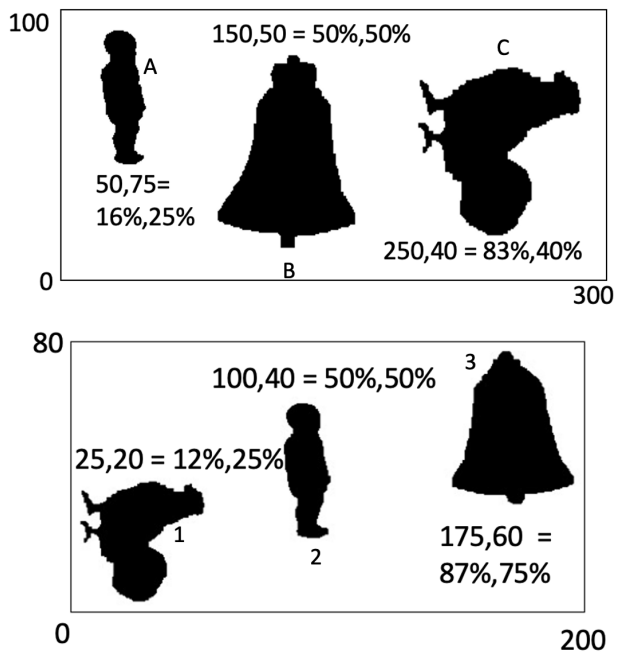
Pre-existing image segmentation techniques are used for object identification. For our implementation, a simple binary contour-based thresholding image segmentation algorithm is used. The equation is stated below, and it simply classifies blobs of pixel values over a certain threshold as one image. It is ideal for our SimDB shape dataset because it has completely black images. The process of segmentation and feature generation is illustrated with an example in Figure A.1.

$$\mathrm{dst}(x, y) = \begin{cases} \mathrm{maxval} & \text{if } \mathrm{src}(x, y) > \text{ thresh} \\ 0 & \text{otherwise} \end{cases}$$

Our current implementation can generate synthetic datasets with multiple objects from a single-object image database; run the existing quadtree and signature retrieval process with image segmentation; and calculate various useful metrics.

We've defined metrics to determine how similar two multi-object images are – **average accuracy, discard object penalty (DOP) and**

positional distance of objects within the image. The mean accuracy is the average of all object similarities calculated using the previously defined quadtree distance metric. DOP is a custom metric that indicates the ratio of objects discarded from the result to the total objects involved in the comparison. It is a negative metric, which means the lower it is, the more similar the two images are. We use DOP because we need to discard objects which are not similar to any other objects in the compared image. The positional distance of objects within the image is normalized by percentage with respect to the values of image width and height. In essence, for two images to be considered similar, we need to minimize the DOP and positional distance; and maximize the accuracy. An example of the calculations of these metrics is given in Figure A.1.



Accuracy			
	Boy (A)	Bell (B)	Chicken (C)
Chicken (1)	0.05	0.2	**1**
Boy (2)	**0.9**	0.15	0.2
Bell (3)	0.1	**0.8**	0.3

Boy (A2): 0.9 **[50%-16%,50%-25%] = 34%,25%**
Bell (B3): 0.8 **[87%-50%,75%-50%] = 37%,25%**
Chicken (C1): 1.0 **[12%-83%,25%-40%] = 71%,15%**

Mean accuracy =
= (0.9+0.8+1)/3
= 2.7/3
= 90%

Total distance between objects = **47%,21%**

Discard object penalty = 0/6

Figure A.2: Example of Multi-object image similarity metrics

For the following implementation outputs, DOP is the discard object penalty as defined; Accuracy is from 0 to 1, where 0 is the most similar and 1 is the most different; PosDistance is the absolute sum of distances between every image (we didn't use % values because all images are the same size); AccPos is the accuracy and positional distance of each object in the first image from left to right.

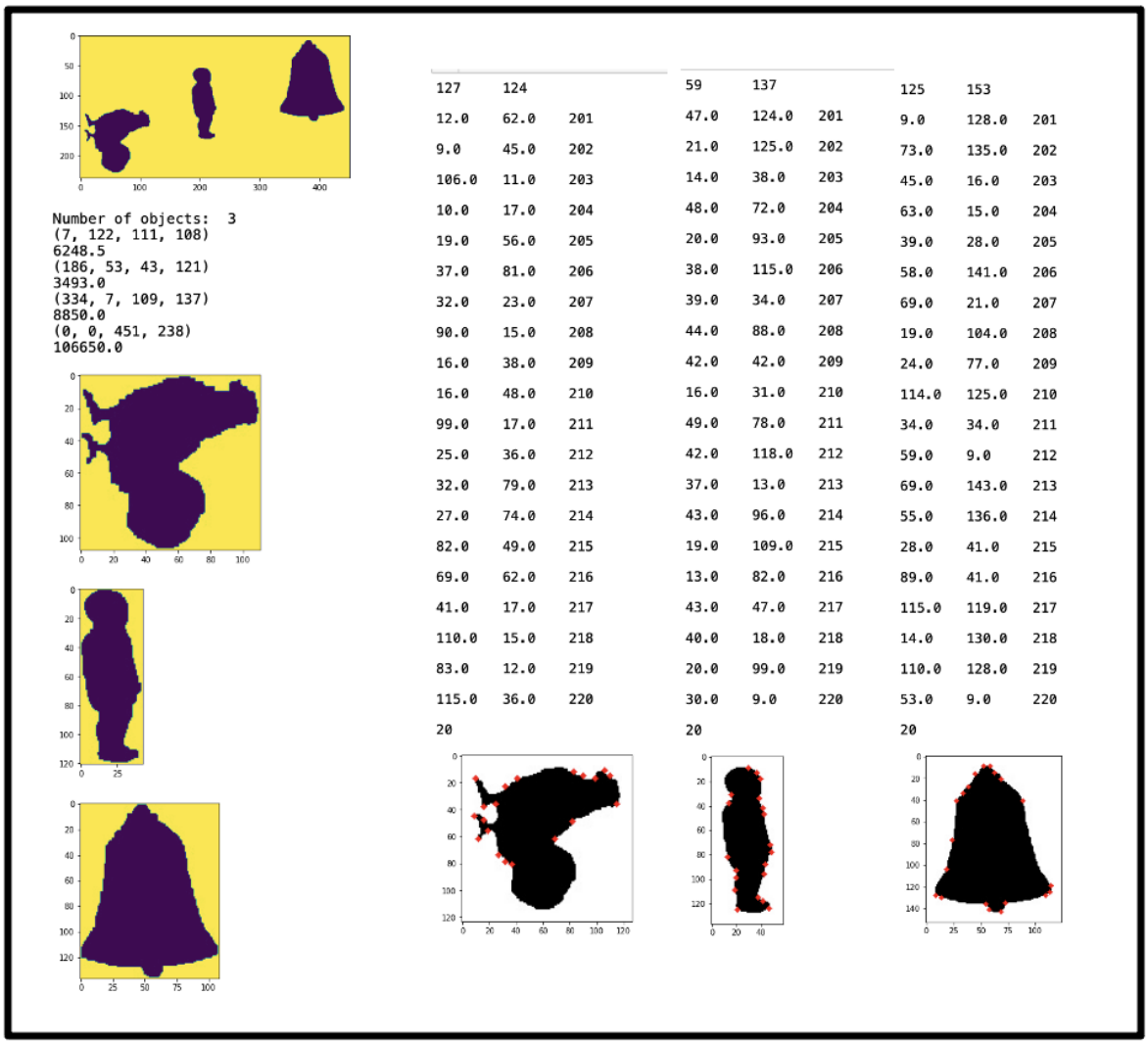


FIGURE A.1: Breakdown of Image Segmentation – Segmentation and Feature Points Generation of each object in the image

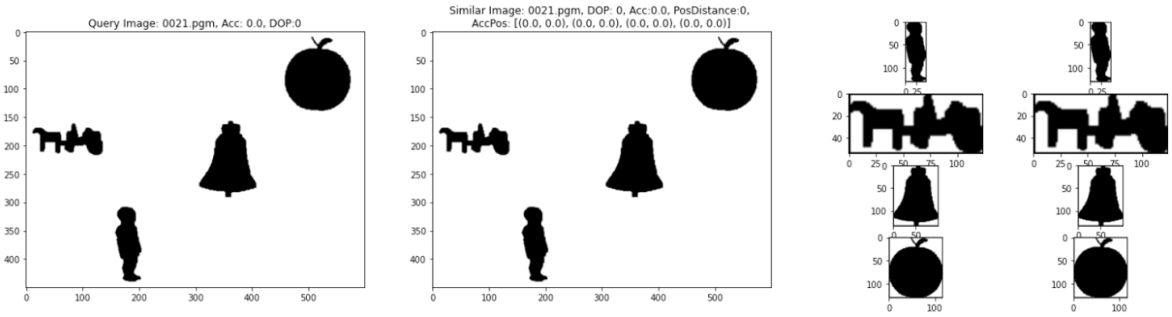


FIGURE A.3: Example of Multi-object image similarity implementation and metrics #1

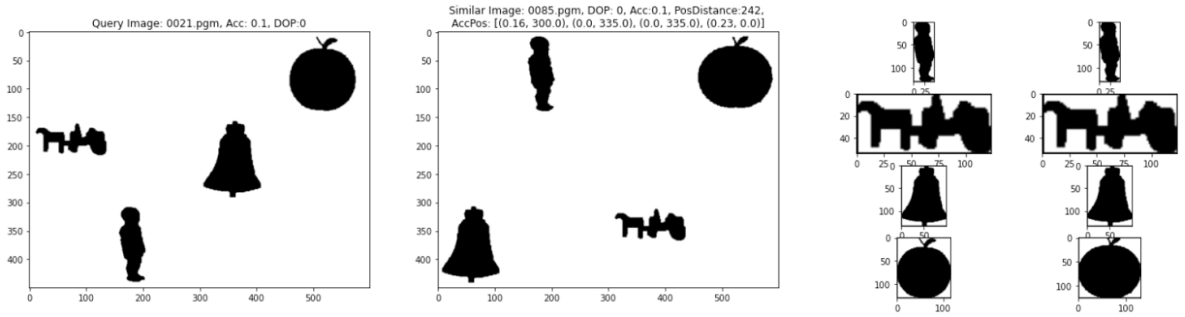


FIGURE A.4: Example of Multi-object image similarity implementation and metrics #2

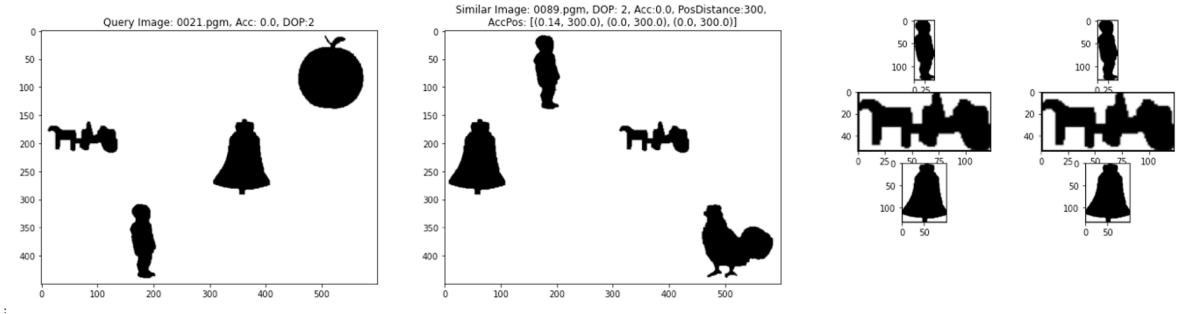


FIGURE A.5: Example of Multi-object image similarity implementation and metrics #3

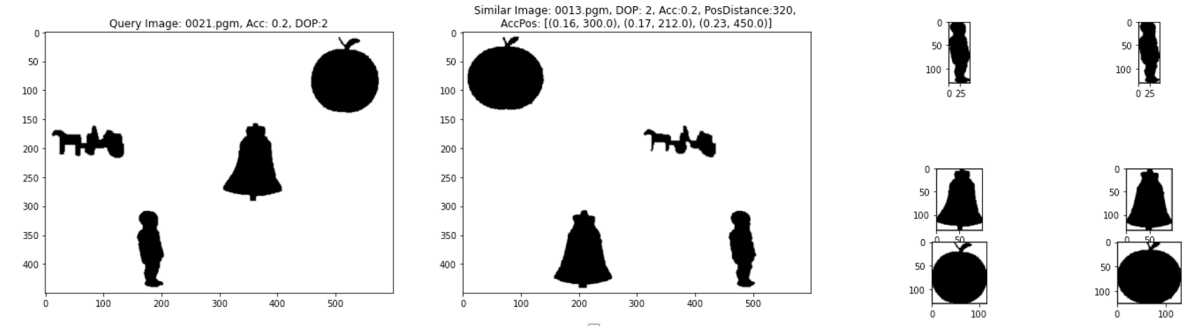


FIGURE A.6: Example of Multi-object image similarity implementation and metrics #4

Bibliography

[1] A Nanda Gopal Reddy and Roheet Bhatnagar. Data mining techniques for logical analysis of data in content based image retrieval system. 2013.

[2] John P Eakins and Margaret E Graham. Content-based image retrieval, a report to the jisc technology applications programme, 1999.

[3] Ishwar K Sethi, Ioana L Coman, and Daniela Stan. Mining association rules between low-level image features and high-level concepts. In *Data mining and knowledge discovery: theory, tools, and technology III*, volume 4384, pages 279–290. International Society for Optics and Photonics, 2001.

[4] Jun Yue, Zhenbo Li, Lu Liu, and Zetian Fu. Content-based image retrieval using color and texture fused features. *Mathematical and Computer Modelling*, 54(3-4):1121–1127, 2011.

[5] Ritendra Datta, Dhiraj Joshi, Jia Li, and James Z Wang. Image retrieval: Ideas, influences, and trends of the new age. *ACM Computing Surveys (Csur)*, 40(2):1–60, 2008.

[6] Shi-Kuo Chang and Shao-Hung Liu. Picture indexing and abstraction techniques for pictorial databases. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (4):475–484, 1984.

[7] Ying Liu, Dengsheng Zhang, Guojun Lu, and Wei-Ying Ma. A survey of content-based image retrieval with high-level semantics. *Pattern recognition*, 40(1):262–282, 2007.

[8] William I Grosky and Rajiv Mehrotra. Image database management. In *Advances in Computers*, volume 34, pages 237–291. Elsevier, 1992.

[9] John Y Chiang and Yen-Ren Huang. A spatial similarity ranking framework for symbolic pictures retrieval. In *2008 Tenth IEEE International Symposium on Multimedia*, pages 286–293. IEEE, 2008.

[10] Imran Ahmad and William I Grosky. Indexing and retrieval of images by spatial constraints. *Journal of Visual Communication and Image Representation*, 14(3):291–320, 2003.

[11] Naimul Mefraz Khan and Imran Ahmad. An efficient signature representation for retrieval of spatially similar images. *Signal, Image and Video Processing*, 6:55–70, 3 2012. doi: 10.1007/s11760-010-0179-3.

[12] Pornchai Mongkolnam, Thanee Dechsakulthorn, and Chakarida Nukoolkit. Extracted structural features for image comparison. In *Innovations and Advanced Techniques in Computer and Information Sciences and Engineering*, pages 13–17. Springer, 2007.

[13] Monika M Gorkani and Rosalind W Picard. Texture orientation for sorting photos" at a glance". In *Proceedings of 12th International Conference on Pattern Recognition*, volume 1, pages 459–464. IEEE, 1994.

[14] Liang Zheng, Yi Yang, and Qi Tian. Sift meets cnn: A decade survey of instance retrieval. *IEEE transactions on pattern analysis and machine intelligence*, 40(5):1224–1244, 2017.

[15] Daniela Stan and Ishwar K Sethi. Mapping low-level image features to semantic concepts. In *Storage and Retrieval for Media Databases 2001*, volume 4315, pages 172–179. International Society for Optics and Photonics, 2001.

[16] Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. Siamese neural networks for one-shot image recognition. In *ICML deep learning workshop*, volume 2. Lille, 2015.

[17] Haoran Wu, Zhiyong Xu, Jianlin Zhang, Wei Yan, and Xiao Ma. Face recognition based on convolution siamese networks. In *2017 10th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*, pages 1–5. IEEE, 2017.

[18] Kelly L Wiggers, Alceu S Britto, Laurent Heutte, Alessandro L Koerich, and Luiz S Oliveira. Image retrieval and pattern spotting using siamese neural network. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2019.

[19] Hsin-Hui Chen, Jian-Jiun Ding, and Hsin-Teng Sheu. Image retrieval based on quadtree classified vector quantization. *Multimedia tools and applications*, 72(2):1961–1984, 2014.

[20] Genevieve Jomier, Maude Manouvrier, Vincent Oria, and Marta Rukoz. Multi-level index for global and partial content-based image retrieval, 2005.

[21] Charles W Emerson and Sivagurunathan Chinniah. A region quadtree approach to content based image retrieval. In *Proceedings of ASPRS*, pages 1–11, 2006.

[22] Cheng-I Chen and Po-Whei Huang. A new method for image retrieval based on shape decomposition. volume 2, pages 439–444, 2008. ISBN 9780769531199. doi: 10.1109/CISP.2008.649.

[23] TN Vikram, DS Guru, and Shalini R Urs. Face indexing and retrieval by spatial similarity. volume 1, pages 543–547, 2008. ISBN 9780769531199. doi: 10.1109/CISP.2008.740.

[24] Ebrahim Karami, Siva Prasad, and Mohamed Shehata. Image matching using sift, surf, brief and orb: Performance comparison for distorted images, 2016.

[25] Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories, 2008.

[26] Antonio Torralba, Kevin P Murphy, William T Freeman, and Mark A Rubin. Context-based vision system for place and object recognition, 2008.

[27] Eyas El-Qawasmeh. A quadtree-based representation technique for indexing and retrieval of image databases. *Journal of Visual Communication and Image Representation*, 14:340–357, 2003. doi: 10.1016/S1047-3203(03)00034-8.

[28] Shu-Ming Hsieh and Chiun-Chieh Hsu. Retrieval of images by spatial and object similarities. *Information Processing and Management*, 44: 1214–1233, 5 2008. doi: 10.1016/j.ipm.2007.09.008.

[29] Abduljawad A Amory, Rachid Sammouda, Hassan Mathkour, and Rami Mohammad Jomaa. A content based image retrieval using k-means algorithm. In *Seventh International Conference on Digital Information Management (ICDIM 2012)*, pages 221–225. IEEE, 2012.

[30] Tinne Tuytelaars and Krystian Mikolajczyk. *Local invariant feature detectors: a survey*. Now Publishers Inc, 2008.

[31] Sanaa Khudayer Jadwaa. Feature extraction for hand gesture recognition: A review. *International Journal of Scientific Engineering Research*, 6(7), 2015.

[32] Konstantinos G Derpanis. The harris corner detector. *York University*, 2, 2004.

[33] Deepak Geetha Viswanathan. Features from accelerated segment test (fast). In *Proceedings of the 10th workshop on Image Analysis for Multimedia Interactive Services, London, UK*, pages 6–8, 2009.

[34] Jianbo Shi and Tomasi. Good features to track. In *1994 Proceedings of IEEE conference on computer vision and pattern recognition*, pages 593–600. IEEE, 1994.

[35] Ramadhan J. Mstafa, Younis Mohammed Younis, Haval Ismael Hussein, and Muhsin Atto. A new video steganography scheme based on shi-tomasi corner detector. *IEEE Access*, 8:161825–161837, 2020. doi: 10.1109/ACCESS.2020.3021356.

[36] Rubayat Ahmed Khan, Samiul Islam, and Rubel Biswas. Automatic detection of defective rail anchors. In *17th international IEEE conference on intelligent transportation systems (ITSC)*, pages 1583–1588. IEEE, 2014.

[37] Song Wu, Ard Oerlemans, Erwin M Bakker, and Michael S Lew. A comprehensive evaluation of local detectors and descriptors. *Signal Processing: Image Communication*, 59:150–167, 2017.

[38] Mohanad Babiker, Othman O Khalifa, Kyaw Kyaw Htike, Aisha Hassan, and Muhamed Zaharadeen. Harris corner detector and blob analysis featuers in human activty recognetion. In *2017 IEEE 4th International Conference on Smart Instrumentation, Measurement and Application (ICSIMA)*, pages 1–5. IEEE, 2017.

[39] Lubas Juranek, Jiri Stastny, and Vladislav Skorpil. Effect of low-pass filters as a shi-tomasi corner detector's window functions. In *2018 41st International Conference on Telecommunications and Signal Processing (TSP)*, pages 1–5. IEEE, 2018.

[40] Dae-Min Cho, Panagiotis Tsiotras, Guangcong Zhang, and Marcus Holzinger. Robust feature detection, acquisition and tracking for relative navigation in space with a known target. In *AIAA Guidance, Navigation, and Control (GNC) Conference*, page 5197, 2013.

[41] Bronislav Přibyl, Alan Chalmers, and Pavel Zemčík. Feature point detection under extreme lighting conditions. In *Proceedings of the 28th Spring Conference on Computer Graphics*, pages 143–150, 2012.

[42] Ahmed A Abd EL-Latif, Bassem Abd-El-Atty, and Salvador E Venegas-Andraca. A novel image steganography technique based on quantum substitution boxes. *Optics & Laser Technology*, 116:92–102, 2019.

[43] Shaharyar Tareen, Khan Ahmed, and Zahra Saleem. A comparative analysis of sift, surf, kaze, akaze, orb, and brisk. In *2018 International conference on computing, mathematics and engineering technologies (iCoMET)*, pages 1–10. IEEE, 2018.

[44] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[45] Zack Akil. Machine learning tip - using rotational data, 2018. URL `https://towardsdatascience.com/machine-learning-tip-using-rotational-data-b67ded0a33ad`. Accessed on 23.07.2021.

[46] GEAPA Batista, Diego Furtado Silva, et al. How k-nearest neighbor parameters affect its performance. In *Argentine symposium on artificial intelligence*, pages 1–12. Citeseer, 2009.

[47] Tom M Mitchell. Artificial neural networks. *Machine learning*, 45: 81–127, 1997.

[48] Liwei Wang, Yan Zhang, and Jufu Feng. On the euclidean distance of images. *IEEE transactions on pattern analysis and machine intelligence*, 27(8):1334–1339, 2005.

[49] Kilian Q Weinberger and Lawrence K Saul. Distance metric learning for large margin nearest neighbor classification. *Journal of machine learning research*, 10(2), 2009.

[50] AURA Conci and CS Kubrusly. Distance between sets-a survey. *arXiv preprint arXiv:1808.02574*, 2018.

[51] Érick O Rodrigues. Combining minkowski and cheyshev: New distance proposal and survey of distance metrics using k-nearest neighbours classifier. *Pattern Recognition Letters*, 110:66–71, 2018.

[52] Sung-Hyuk Cha. Comprehensive survey on distance/similarity measures between probability density functions. *City*, 1(2):1, 2007.

[53] Claude Sammut and Geoffrey I Webb. *Encyclopedia of machine learning*. Springer Science & Business Media, 2011.

[54] J Roger Bray and John T Curtis. An ordination of the upland forest communities of southern wisconsin. *Ecological monographs*, 27(4): 326–349, 1957.

[55] Godfrey N Lance and William T Williams. Computer programs for hierarchical polythetic classification ("similarity analyses"). *The Computer Journal*, 9(1):60–64, 1966.

[56] D. R. Chatterjee. Log book - guide to distance measuring approaches for k- means clustering. medium., 2020. URL `https://www.towardsdatascience.com/log-book-guide-to-distance-measuring-approaches-for-k_-means-clustering-f137807e8e21`. Accessed on 03.10.2021.

[57] Measuring distance - github pages, 2020. URL https://chris3606.github.io/GoRogue/articles/grid_components/measuring-distance.html. Accessed on 03.10.2021.

[58] Iaroslav Melekhov, Juho Kannala, and Esa Rahtu. Siamese network features for image matching. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 378–383. IEEE, 2016.

[59] Raia Hadsell, Sumit Chopra, and Yann LeCun. Dimensionality reduction by learning an invariant mapping. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pages 1735–1742. IEEE, 2006.

[60] Khairul Anuar Mat Said, Asral Bahari Jambek, and Nasri Sulaiman. A study of image processing using morphological opening and closing processes. *International Journal of Control Theory and Applications*, 9(31):15–21, 2016.

[61] Opencv - morphological transformations, 2016. URL https://opencv24-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_morphological_ops/py_morphological_ops.html. Accessed on 26.09.2021.

[62] Affine transformation, 2020. URL https://theailearner.com/tag/cv2-warpaffine/. Accessed on 26.09.2021.

[63] Xingwei Yang, Lakshman Prasad, and Longin Jan Latecki. Affinity learning with diffusion on tensor product graph. *IEEE transactions on pattern analysis and machine intelligence*, 35(1):28–38, 2012.

[64] Longin Jan Latecki and Rolf Lakamper. Shape similarity measure based on correspondence of visual parts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(10):1185–1190, 2000.

[65] Hua Yu and Jie Yang. A direct lda algorithm for high-dimensional data — with application to face recognition. *Pattern Recognition*, 34 (10):2067–2070, 2001.

[66] DH Janzen and W Hallwachs. Philosophy, navigation and use of a dynamic database (acg caterpillars srnp) for an inventory of the

macrocaterpillar fauna, and its food plants and parasitoids, of the area de conservacion guanacaste (acg), northwestern costa rica, 2005.

[67] Saeed Meshgini, Ali Aghagolzadeh, and Hadi Seyedarabi. Face recognition using gabor filter bank, kernel principle component analysis and support vector machine. *International Journal of Computer Theory and Engineering*, 4(5):767, 2012.

[68] Chengzhuan Yang, Hui Wei, and Qian Yu. A novel method for 2d nonrigid partial shape matching. *Neurocomputing*, 275:1160–1176, 2018.

[69] Erik Rodner, Marcel Simon, Gunnar Brehm, Stephanie Pietsch, J Wolfgang Wägele, and Joachim Denzler. Fine-grained recognition datasets for biodiversity analysis. *arXiv preprint arXiv:1507.00913*, 2015.

[70] One-shot learning with siamese network, 2021. URL `https://medium.com/swlh/one-shot-learning-with-siamese-network-1c7404c35fda`. Accessed on 11.10.2021.

Vita Auctoris

NAME:	Eisa Adil
PLACE OF BIRTH:	Agra, India
YEAR OF BIRTH:	1997
EDUCATION:	Delhi Public School, Riyadh, K.S.A. High School, CBSE, 2013-15
	Manipal University, Dubai, U.A.E. Bachelor of Technology, Computer Science, 2015-19
	University of Windsor, Windsor ON, Canada Master of Science, Computer Science, 2019-21