University of Guadalajara

Centre of Economic and Administrative Sciences

CUCEA

A fast and low-cost method to detect near-duplicate Images in large dataset based on fingerprint extraction and Deep Learning

Student Fatemeh Nasri Shandiz

Doctor José Antonio Orizaga Trejo

A Thesis in the Field of Artificial intelligence and computer vision

For the Doctorate Degree in Information Technology

A picture is worth a thousand words

Abstract

Recognizing near-duplicate images from large datasets is a crucial task in image retrieval and content identification. Finding similar images in order to reduce redundancy is time-consuming in large datasets. Most of image representation targeting methods at conventional image retrieval issues for detecting duplicate are either computationally expensive to extract and match or have robustness limitations. In this work, we propose a fast method to detect near-duplicate images in a large dataset, which is computationally low cost and effective by using image fingerprints to determine similarity between a query image and near-duplicated images in a large dataset. We extract a series of fingerprints combining global and local features also using a deep learning model as a fingerprint for each image in the dataset and store them in a separate database. Then we apply successive filters to the query image, discarding non-similar images in the process until reaching a final set of near-duplicate images. we achieved to discarding most of the non-similar images in the early stages of the process and focuses on robustness in the latter stages, where the set of near-duplicate candidate images is significantly smaller. This allows to perform the query process on the fly. The proposed method and experimental results provide a right compromise between accuracy and speed in detecting near-duplicate images from a large dataset even via a low performance potential computer such has home use laptop or a workstation computer.

Dedication

Will be fill one the review process is done!

Acknowledgments

“Acknowledgment Will be fill one the review process is done!”

Table of Contents

Abstract 3

Dedication iv

Acknowledgments v

Table of Contents vi

List of Tables xi

List of Figures xiii

Chapter1. 16

Introduction 16

1.1 Problem Definition 20

1.2 Research Question 23

1.3 Hypothesis 23

1.4 Objective 24

1.4.1 General Objective 24

1.4.2 Specific Objective 24

1.5 Motivation 25

1.6 Contributions 26

1.7 Thesis Scope 27

1.8 Thesis Outline 28

Review of literature 30

2.1 Near-Duplicate Images (NDI) 30

2.1.1 Image Near-duplicate detection techniques 32
2.1.2 Image attacks 35
2.2 Image Retrieval 36
2.2.1 Text-Based Image Retrieval (TBIR) 37
2.2.1.1 Limitations of Text-Based image retrieval (TBIR) 37
2.2.2 Semantic-Based Image Retrieval (SBIR) 38
2.2.2.1 Limitations of Semantic-Based Image Retrieval (SBIR) ...39
2.2.3 Content-Based Image Retrieval (CBIR) 39
2.3 Fingerprint Method Content-Base Image Retrieval 44
2.3.1 Feature Extraction 44
2.3.2 Low -level Features 46
2.3.3 High-level Features 47
2.3.4 The combination of High-level and Low-level Features 48
2.4 Calculate Image Similarity 49
2.4.1 The Similarity Measurement 50
2.4.2 The Metric Distance 50
Chapter 3. 53
The Proposed Method 53
3.1 Phase-One 53
3.1.1 Hash Fingerprint 55
3.1.2 Color Histogram Fingerprint 59
3.1.3 Thumbnail Fingerprint 62
3.1.4 Deep Convolutional Neural Network Fingerprint 65

3.1.4.1 Very Deep Convolutional Networks VGG19 74
3.1.5 ORB (Oriented FAST and rotated BRIEF) Fingerprint 77
3.1.6 BOVW (Bag of Visual Words) Fingerprint 82
3.2 Phase-Two 85
3.2.1 Fuzzy search 87
3.2.1.1 BK-Tree Search 88
3.2.1.2 Query Image 89
3.2.2 Levenshtein Distance 89
3.2.3 Cosine Distance 91
3.2.4 Jaccard distance 92
3.3 Phase-Three 92
3.3.2 Similarity Calculation for Fingerprints 92
3.3.1 Matching system 93
3.3.2.1 Similarity measure of Histogram and Thumbnail 94
3.3.2.2 Similarity measure of ORB 95
3.3.2.3 Similarity measure of BOVW 96
3.3.2.4 Similarity measure of Vgg19 97
3.3.3 Voting Process 97
Chapter 4. 101
The Experimental Results 101
4.1 Data Collection 101
4.1.1 The Images Databases 101
4.1.1.1 Native Data Base 102

4.1.1.2 Query Images Data Base....102
4.1.1.3 Fingerprint database....103
4.2 Database settings....103
4.2.1 The attack images....104
4.3 Evaluation protocol....107
4.3.1 Experimental setup....107
4.3.2 Execution the algorithm....107
4.3.2.1 Performance Validation....109
4.3.3 Recall, Precision and F-Score....110
4.3.3.1 Parameters and Performance....112
4.3.3.2 Sequential vs OR method results....114
4.4 Results Analysis....115
4.4.1 Fuzzy Search maximum distances BK-Tree....115
4.4.2 Threshold values for the different fingerprints....118
4.4.3 Total Similarity Index....118
4.4.4 Evaluation of the execution speed....122
4.4.6 Improving the algorithm....131
4.4.7 Software and user interface....141
4.5 Comparison of the proposed method and pure-deep learning method....143
4.5.1 Resources consumption....145
4.5.2 Results from pure-deep learning method....147
4.5.3 Execution both methods....149
Chapter 5....151

5.1 Conclusions.....151

5.2 Future Work.....153

Appendix.....155

Symbols and abbreviations.....155

Appendix1.....157

Appendix2.....158

Appendix3.....160

List of Tables

Table. 4.1 Native Database specification 104

Table. 4.2 Query Database specification 104

Table. 4.3 Image attacks specification 106

Table. 4. 4 comparison of recall, precision and F1 score values for sequential and OR 114

Table. 4. 5 several combinations of Hamming with different threshold in sequential ... 119

Table. 4. 6 several combinations of Hamming distance with different threshold in OR 120

Table. 4. 7 several combinations of Cosine with different threshold in sequential 120

Table. 4. 8 several combinations of Cosine distance with different threshold in OR..... 121

Table. 4. 9 several combinations of Jaccard with different threshold in sequential 121

Table. 4. 10 several combinations of Jaccard distance with different threshold in OR.. 122

Table. 4. 11Execution time for Hamming distance 123

Table. 4. 12 Execution time for Cosine distance 123

Table. 4. 13 Execution time for Cosine distance 124

Table. 4. 14 The time of execution for both OR and Sequential method 125

Table. 4. 15 the best 3 combination of thresholds for Hamming distance..... 126

Table. 4. 16 The best 3 combination of thresholds for Jaccard distance 126

Table. 4. 17 The best 3 combination of thresholds for Cosine distance 126

Table. 4. 18 The accuracy and time of three metric distance 127

Table. 4. 19 The Histogram fingerprint threshold values and their accuracy..... 129

Table. 4. 20 The Thumbnail fingerprint threshold values and their accuracy 130

Table. 4. 21 The deep learning fingerprint threshold values and their accuracy 130

Table. 4. 22 The ORB fingerprint threshold values and their accuracy 130

Table. 4. 23 The BOWV fingerprint threshold values and their accuracy 130

Table. 4. 24 Algorithm execution with different random seeds 132

Table. 4. 25 average performance with different random seeds 132

Table. 4. 26 execution the algorithm with all six fingerprints 133

Table. 4. 27 execution the algorithm with all six fingerprints menus deep learning 133

Table. 4. 28 execution the algorithm with only deep learning fingerprints 133

Table. 4. 29 execution the algorithm with only deep learning fingerprints 134

Table. 4. 30 list of near duplicate images not identify by deep learning fingerprint 134

Table. 4. 31 average time with different number of query images 135

Table. 4. 32 List of False negative results show most of them had flipping attack 140

Table. 4. 33 results execution via sending 100 query images 149

Table. 4. 34 Comparing result of both methods 149

List of Figures

fig.1.1 Example of near-duplicate Image search my query text. 18

fig.1.2 Detect Near-duplicate image by query Image. 20

fig.1.3schematic database energy consumption 22

fig.1.4 Compromising between accuracy and speed to detect Near-duplicate image. 26

fig.1.5 The dissertation outline diagram 29

fig.2.1 Image Retrieval CBIR feature extraction methods. 36

fig.2.2 Content Base Image Retrieval CBIR feature extraction methods diagram. 41

fig.2.3 Extract Fingerprint from images by CBIR diagram 44

fig.2.4 Feature extraction methods for Classification by CBIR diagram. 46

fig.2.5 Different metric distance similarity for Content Base Image Retrieval CBIR. 51

fig. 3.1 schematic the proposed method..... 55

fig. 3.2 schematic hash fingerprint extraction. 58

fig. 3.3 Image with RGB color mode used as one of query images. 59

fig. 3.4 RGB channel of the image in fig.3.2 60

fig. 3.5 RGB color distribution of the image in fig.3.2..... 61

fig. 3.6 RGB color distribution of the image in fig.3.2..... 62

fig. 3.7 Thumbnail fingerprint calculation by down-sampling. 63

fig. 3.8 Thumbnail fingerprint process. 64

fig. 3.9 earliest Convolutional neural networks architecture. 67

fig. 3.10 Convolution layer (CONV) 68

fig. 3.11Pooling layer (Pool) 69

fig. 3.12 Fully connected (FC) 70

fig. 3.13 Parameter compatibility in convolution layer 72

fig. 3.14 Transfer Learning 73

fig. 3.15 (Simonyan, Very deep convolutional networks for large-scale 2014) 75

fig. 3.16 Vgg19 architecture (Chansung 2018) 76

fig. 3.17 Extracting features by Vgg19 pre-trained model 77

fig. 3.18 ORB creates pyramid to use in Fast algorithm 79

fig. 3.19 Keypoints obtained by Fast algorithm 80

fig. 3.20 Match Keypoints to recognize the similarity of image. 82

fig. 3.21 Extract features in Bag of Visual Words 83

fig. 3.22 Local Fingerprint the Bag of Visual Word by SIFT 84

fig. 3.23 Online stage for detect near duplicate image in proposed method 86

fig. 3.24 BK-Tree Fuzzy Search to find the near duplicate images 88

fig. 3.25 Matching process diagram with OR Method 94

fig. 3.26 Matching process diagram with Sequential Method 99

fig. 4.1 Serialized the reference list 110

fig. 4.2 Performance measurements parameters. 113

fig. 4. 3 The schematic image filtering with six fingerprints 136

fig. 4. 5 The True positive results of detecting near duplicate images 137

fig. 4. 6 The True negative results of detecting near duplicate images 138

fig. 4. 7 The False negative results of detecting near duplicate images..... 138

fig. 4. 8 The distribution of FP, FN, TP, TN results of detecting near duplicate images 139

fig. 4. 9 The Near duplicate detection system user interface 141

fig. 4.10 Choose the images from database and generate it hash fingerprint 142

fig. 4.11 Select query image 142

fig. 4.12 Select the detecting near duplicate method 143

fig. 4.13 table of results..... 143

fig. 4. 14 The amount of memory space during execution of our method..... 146

fig. 4. 15 The amount of memory space during execution of pure-deep learning..... 147

fig. 4. 16 The list of K-Nearest Neighbors to the candidate images..... 148

Chapter1.

Introduction

Vision is the ability to see; the ability to understand and anticipate (Oxford 2020). Computer vision is a research area dedicated to enabling computers to see, understand, and anticipate the same way as human visual system. Computer vision is closely linked with artificial intelligence, as the computer must interpret what it sees, and then perform appropriate analysis or act accordingly. The primary aim of computer vision is to enable computers to perform the same kind of tasks as humans with the same efficiency. There are many companies that are working in different areas of computer vision. Google is one such company. They recently worked on simulating the human brain's ability to understand and how to process visual data content (Brain 2020).

The explosive growth rate of technology has been very beneficial to society and at the same time there have been some challenges which are a major concern. With an ever-increasing number of social network users, an expanding number of high-tech devices, and ever-faster networking speeds, every second produces exponentially more information and data specifically visual data content (VDC) (Cisco 2017). A considerable portion of these data are duplicated or expired but remain on servers in data centers located around the world, known as dark data. (ZL UNIFIED ARCHIVE 2019) Maintaining massive duplicated or Near-duplicate data (NDD) is costly and against Green Computing (Jinsong Wu 2016)

since processing this amount of data can consume immense quantities of electrical energy in data centers. Besides the colossal expenditure of data storage, calculated based on the usage space, enterprise companies continue to drive the market in paying less for data storage and manage data for fast accessibility.

Scientist as well data centers researchers have been working on Near-duplication data to find an efficient and effective method for managing data in storage. Since images, image sequences, and internet of things (IOT) (i.e. cameras, sensors, and cloud services) are 80% of corporate and public unstructured data (Fritz Venter 2012) impacted by social networks, consequently they are data types that have more duplicity percentage on storage within the last decade. As result, researchers have provided several methods to detect and reduce Near-duplicate images in order to reduce data centers cost and boost data accessibility.

A Duplicate Image (DI) is exact copy of an image and near a Duplicate Image (NDI) is the altered version of an image, the altering could be a fact of capturing conditions or use of an application to make changes on the original image (DVMM LAB 2015). Detecting Near-duplicate images is not only to reduce redundancy, it has a vast usage range. Some applications of NDI are:

1. reducing redundancy
2. finding illegal and false version of images
3. copyrighted management
4. detection of forged images
5. automatic content recognition

6. content-aware advertising
7. business intelligence
8. categorizing image data base content

Figure 1 is an example of near-duplicate image search my query text.

fig.1.1 Example of near-duplicate Image search my query text.

Detecting Near-Duplicate images is a major tool used for effective data management in large repositories. Although there are several approaches used for Near-duplicated images, Content-Based Image Retrieval methods are some of the more practical approaches. Although many methods based on image retrieval content have been proposed during last decade, there is still a vast need to have an effective Near-duplicate image detection. Most of the research's focus is either to obtain more precise result or less computational workload.

For our research, a compromise between accuracy and efficiency by using both local and global image descriptors were investigated and presented in order to detect near-duplicate image in large image dataset.

To retrieve an image, the characteristic of the image should be extracted which is known as feature extracting process. These features are a small representation of the original image known as fingerprint. Since fingerprints are a subset of the original image, they are effective and speedy for image retrieval.

In this work we used efficient low-level features and accurate high-level features structures. We used a combination of six different fingerprints as a set of lightweight fingerprints including an image hash, thumbnail, color histogram used for global features, BOVW (Bag of visual word), ORB (Oriented FAST and rotated BRIF) and CNN (Convolutional Neural Network) used for local features. This composition provides a balance of accuracy (the use of local image fingerprints) and speed (the use of global fingerprints). Also, we used fuzzy search algorithm, BK-Three to speed up the near-duplicate detection process at the first stage and discriminate images which are suspect of being near-duplicate. Hence when the query image enters to the retrieval system after extracting its feature, they will be compared with features existing in fingerprint database to look for possible near-duplicate version of the image.

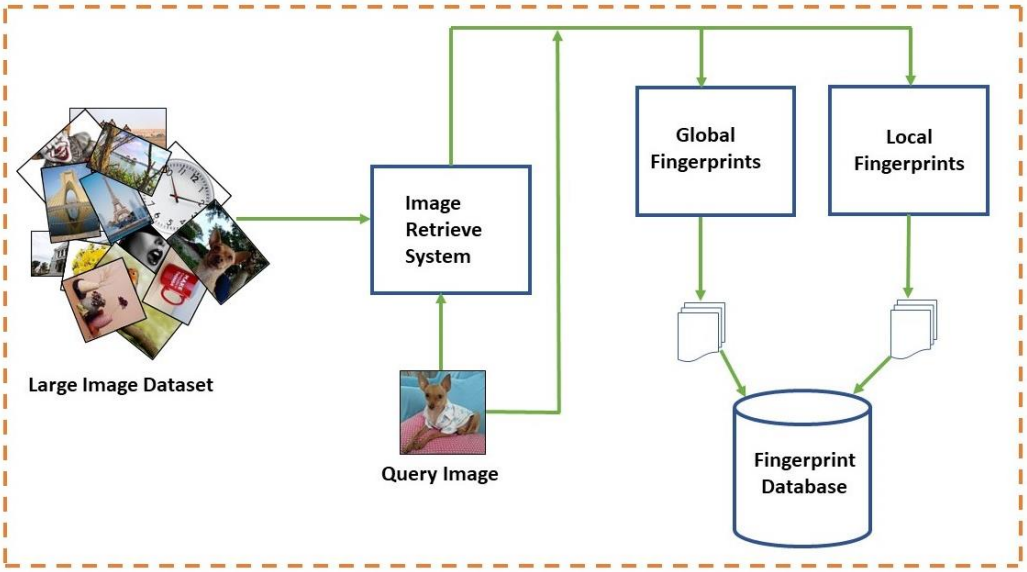


fig.1.2 Detect Near-duplicate image by query Image.

Then we applied a matching system by comparing the fingerprints of the query image to the fingerprints stored in the database, result is a set of near-duplicate images. Figure 2 is an illustration of this process. This method is better than alternative methods in that it has lower cost and faster image near-duplicate detection.

1.1 Problem Definition

Detecting and reducing Near-Duplicate images in large data set attack two different problem domains at the same time as following:

- Maintaining Near-duplicate images consume energy in data center.
- Most of image retrieval method to detect Near-duplicate images, either have computational complexity or doesn't meet an accurate result.

We will discuss both of mentioned in the order in which they are presented.

Maintaining Near-duplicate images consume energy in data center:

Reduce complexity and redundant data will reduce computer computational time, thus energy consumption in large scale in data centers will consequently decrease. High energy cost has become an important issue in data centers specifically in large data set computing. Most of the methods for detect and reduce Near-duplicate image subsequently provide electrical reduction in data centers.

The processing of images for a search is a factor of power consumption and processing time. By reducing the near-duplicate images, the searching time will decrease and consequently the reduction of energy computation. If we define power consumption as below:

$$Power\ consumption = time\ comsuming \begin{cases} computing \\ data\ access \end{cases} \longrightarrow \text{PW} = \text{Tc} \begin{cases} Fc \\ Fd \end{cases}$$

Now if we define power consumption as pw, t as computation time, d as data volume in data, and s as the amount of similarity, we can define power consumption as follows:

$$pw = t + ds$$

From this equation we can clearly see that computation time is a factor which impacts power consumption, hence long computational process time consumes more power.

$$\lim_{n\to\infty} \sum_{i=0}^{n} p(t)\ \Delta\ tn$$

Where P is power in watts,I is current and R is resistance.

Also based on ohm low we have:

$$P = I^2 R(W)$$

As following:

$$P_1 = \int_{b_1}^{a} I^2 \ R \ dt$$

Where P is energy, a, b are initial and final, dt is time differential, we can consider the formulate of consuming energy in a normal database which is including Near-duplicate images. Detect and reduce such data can provides the following electricity consumption:

$$P_2 = \int_{b_2}^{a} I^2 \ R \ dt$$

$$b_2 < b_1$$

$$P_2 < P_1$$

Where P is power,I is electricity current R is resistance, the P value always is positive.

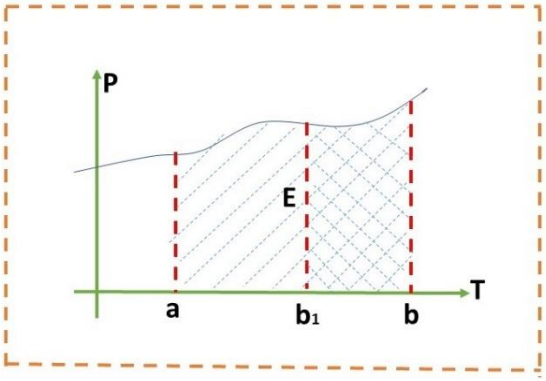


fig.1.3schematic database energy consumption

As can be shown from figure 3, computation process directly decreases the power consumption before and after image Near-duplicate reduction. This amount in large data

sets with massive image will provide a substantial saving energy rate. Less Near-duplicate images needs less electrical energy to compute unnecessary data in every search result.

Most of image retrieval method to detect Near-duplicate images, either have computational complexity or doesn't have accurate result:

Since fingerprints are small representation of an image, they are fast to compute. Although local features have more complexity, extracting the global features at first stage of the retrieval process is faster. What can help the retrieval further is to have a fingerprint database to compare the similarity between query image and the database image. At any times a new query image come to the database, the compression process will be between fingerprints rather than original. While the fingerprints representation has less dimensional, finding similarity between small vector is faster as compared to the same process in more dimensional matrix which is original image information.

1.2 Research Question

How to build a fingerprint algorithm which compromise accuracy and speed in large image dataset in order to detect Near-duplication images on the fly?

1.3 Hypothesis

It's possible to design a method for detect Near-duplicate images in large data set with balancing the precision and velocity using fingerprint image retrieval. To prove this hypothesis several image Near-duplication detection algorithms have been applied and examine in large data set. We also compare the result with the state-of-the-art algorithm which is proposed in this research.

1.4 Objective

1.4.1 General Objective

Define and analyze benchmark of image representation with a compact in light weight fingerprint that allow accurate and fast comparison of images to detect Near-duplicate images in a large dataset.

1.4.2 Specific Objective

- Explore and evaluate the state-of-the-art algorithms in computer vision for content-based image retrieval to better understand what compromises could be adopted to design various image fingerprint representation
- Propose an experimental set up for large image data set to analyze the different image retrieval representation that will be explored.
- Explore image retrieval algorithms by combining Local and global image features to reduce the retrieval computational complexity balance with accuracy in image retrieve process.
- Evaluate a technique using deep learning method as fingerprint representation benefit from its accuracy to retrieve images combined with ORB and BOVW as two other robust local features in order to link equilibrium between speed and precision.
- Query images, coming to the dataset could have either an exact copy or Near-duplicate version inside image database. If the database is a categorical image

dataset, such method allows the query image to classify in respective image category.

1.5 Motivation

Although there are several methods to detect Near-duplication images from large data sets, there is still a need to have methods which are more effective to retrieve and compare similarity between images in dataset and a query image. The amount of energy consumed, and load balance can be reduced by detecting and reducing Near-duplicate images stored in database.

Most existing methods and polices to detect Near-duplication images applied by cloud and data storage technicians do not give users permission to decide whether to keep or eliminate the near-duplicate version of images. They do not have authorization to define the threshold value for similarity measure between the query image and possible Near-duplicate image in dataset.

These methods also prevent cloud technicians from reducing duplications because they do not have permission to eliminate images in private accounts. Near-duplicate detection methods have heavy computational process which impact power. Because of the explosive growth of data, detecting images using Near-duplicate methods have become a leading research area in computer vision. The main goal of the research is to find an efficient and effective way for data storage. The main goal of our research is to find a compromise between detection accuracy and speed to detect Near-duplicate images. Near-duplicate image is a very important issue in data management which requires efficient and effective

methods for large scale data sets being used on the fly. Figure 4 is an illustration that shows the compromise between accuracy and speed to detect Near-duplicate images.

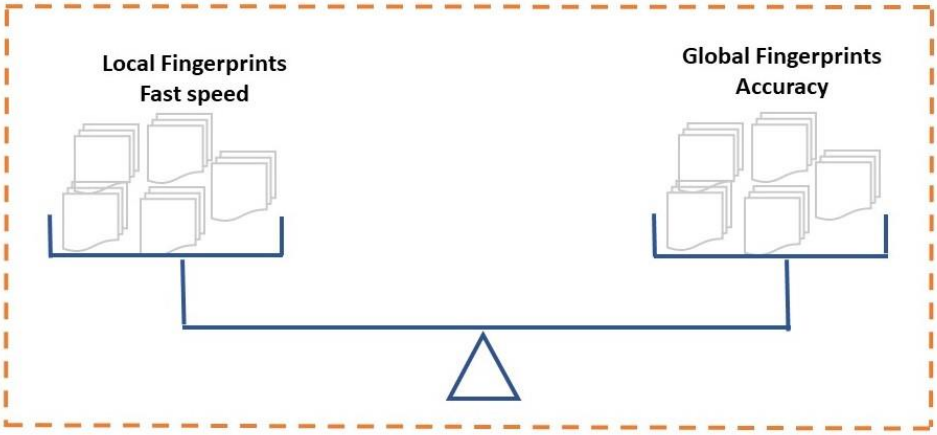


fig.1.4 Compromising between accuracy and speed to detect Near-duplicate image.

1.6 Contributions

This research follows three main contribution achievements as below:

- Provide an efficient fingerprint method to detect near-duplicate images in large dataset.
- A method in applicable in data management application for large scale data set, promote effectiveness and efficiency required in fast computation process.
- A fast and accrue method to retrieve image, ready to classify for image categorical dataset.

1.7 Thesis Scope

This research pursues an effective fingerprint extraction method using both low-level and high-level image feature to retrieve and detect Near-duplicate images, which are costly to maintain and cause slowness for image search in large image datasets.

The ideology of this research is creating a balance in image-based content retrieval via local and global fingerprints, in which local features have great accuracy and more complexity than global features which have less accuracy but faster to retrieve. Additionally, in this research we used deep learning method as one of the six fingerprints to create a local based fingerprint which provide us a great percentage of precision in high-level feature extraction. Applying BK-tree fuzzy search method also provides a fast search in finding Near-duplicate Hash fingerprint to discriminate images which are not close to being Near-duplicate compare with query images. In this work to design our algorithm and obtain the best results, we used three type of metric measure distance for different fingerprints with adjustable threshold value by users. Finally, we performed a matching process to detect near-duplicate image in our large dataset.

The fingerprints presented in this work have the responsibility to detect different image attacks, in order to find the similarity factors in Near-duplicate images system. These series of fingerprints, take the role of six different filters which sieve images by their characteristics, in order to find the similarity attribute such as color, texture, objects etc. Each fingerprint is sensitive to the different type of image attacks as following:

- **Hash fingerprint:** it's very compact, fast for look up search, and could be sued for building efficient search fuzzy structures, as we used in this work.
- **Color histogram fingerprint** is resistant to attacks where the colors don't change also is robust to scaling, rotation and flipping attacks.
- **Thumbnail:** Is resistant to the colorization attacks also it's a compact fingerprint.
- **Vgg19 fingerprint:** since in each layer different information discovers from image pixels this fingerprint is sensitive to the shape and objects as well other image characteristics. this fingerprint is not resistant to the adversarial attacks.
- **ORB fingerprint:** is resistant to image point of view changes, compression attacks, blurring, illumination and distortion attacks.
- **BOVW:** since it's just a small representation of local descriptor is scalable for large data base its resistant to geometric transformations such as rotating and scaling.

1.8 Thesis Outline

The remainder of this dissertation is structured as follows: in section 2 we present related work for near-duplicate image detection. In section 3, we explain the proposed A fast and low-cost method to detect near-duplicate images in large dataset based on fingerprint extraction and deep learning algorithm. Section 4 demonstrate evaluation methodology and the experimental results which show the effectiveness of the proposed approach. Conclusion of the paper is discussed section 5. Future work will be explained in chapter 6. Figure 5 shows this dissertation outline.

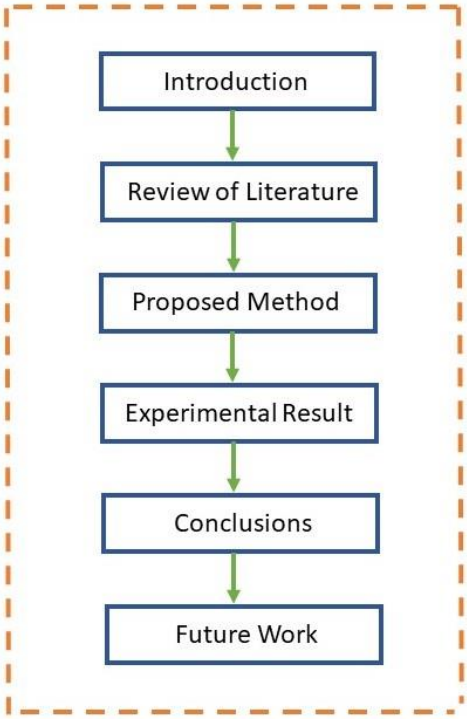


fig.1.5 The dissertation outline diagram

Symbols abbreviations, appendix and bibliography can be found and the end of the present document.

Chapter 2.

Review of literature

In this chapter we discuss previous works including various algorithms for near-duplicate image detection used in fingerprinting detection. Fingerprints have been used as one of the most popular light weight biometric authentication and verification measures because of their high acceptability, immutability and uniqueness. The effectiveness of fingerprinting algorithm is the ability to reduce the misclassification error. There are many fingerprinting algorithms, however in this dissertation we focus on those algorithms which provide a compromise between accuracy and speed to fulfill our objectives, the efficiency and effectiveness.

2.1 Near-Duplicate Images (NDI)

One of the most discussed topics in computer vison is image representation techniques to discover near-duplicate images. These techniques are crucial in copyright enforcement, forged image detection, image search engine progression, and storage optimization. Now a day we have more sensors and cameras than human papulation (Boren 2014) which are creating massive visual data every second. Cisco estimated that 85% of cyberspace are pixel data (V. N. Cisco 2016). Ubiquitous digital cameras, sensors used by internet of things, affordable digital storages, powerful graphics applications, fast networks and user-

friendly communication applications, incredibly increase the amount of multimedia data at the same time increments the possibility of duplicating or manipulating such data either deliberately or accidentally. Images are one of the most visual content for being duplicated or Near- duplicated since they are easier to create and faster to share. Images are a powerful tool which have a wide rank of usage including, education, commercial, medicine, biology, astronomy, geology, social life, etc. Images help attract audience attention. Studies shows (ADOBE 2014) that a presentation or post accompanied with an image 10 times more likely to receive feedback and being understand by users. Images have the potential to involve human brain with emotional reaction, (Xin Wang 2017)this characteristic helped social media and advertising expertise to use images for receiving emotional attention of users. The human brain has the propensity to keep scenes either in memory or register it somehow, this tendency has ushered in human being life the desire to draw, paint and invent camera. Since cerebral cortex tends to retain and share induced sentiment received, we mostly save or share the images which we connect with sentimentally. Hence the duplicate copy of images will be created and stored on the same storage or other databases that we have access. In the shared storages there is possibility that other users may save the same content with different names or manipulate the image in their own fondness. These manipulated images are not the exact duplicated copy of images, but a Near-duplicated of original images.

Near-duplicate image detection is the process to detect different versions of the same images. A near-duplicate image is an image which visually seems identical to the original image, but it has different binary form impacted by several editing procedure such as blurring, scaling, manipulating color mapping, etc. Also, images which are from the same

scene but with slight differences in editing operations, capturing conditions or rendering parameters, are identified as near-duplicate images. Also, photos from the same scene taken several times with different cameras or with different angles are known as Near-duplicate images. In this work, we consider all above definitions as Near-duplicate images. Near-duplicate images have the challenge of matching altered images copies to their originals termed as Near-duplicate image detection.

2.1.1 Image Near-duplicate detection techniques

There have been many techniques proposed to attack image Near-duplicated challenges during the last decade in parallel with fast growing sensor technology and distributing digital images all through local and online storages. Most of these techniques are faced with two major problems: the accuracy precision and computational cost. Besides these two issues, problems filtering the search engines results by eliminating duplicate images in order to provide fast result and consumes less storage space for users, have encouraged researchers and business technology providers to look for optimize solutions, which increases the relevancy of search results and decreases the computation search time. Most of the techniques to solve Near-duplicated challenges are based on computer vison and image processing techniques which are an attempt to automate the process as human vison system. These methods cover tasks such as acquiring, processing, analyzing, and understanding images. The Content-based image retrieval (CBIR) Image Retrieval (V. N. Gudivada and V. V. Raghavan 1995) is one of the most important method for detecting NDIs. This method understands and analyze images based on its contents and performs a result almost similar to human vision.

(Y. Hu 2009) proposed a coherent phrase model for image near-duplicate retrieval different from Bag of Visual Word method which represents local region by using multiple descriptors with enforces the coherency across multiple features for each local region. To represent feature and spatial coherency two type of visual phrase have been designed without increasing computational complexity.

(Yudong Cao 2013) proposed novel scheme for near duplicate image detection by modifying the similarity measure in order to improve duplicate image detection system performance. Using Affine-SIFT MSER and Hessian-Affine algorithms.

(Lingxi Xie. 2014) proposed novel solution for detecting near-duplicate Web images based on Local descriptors of BoVW(Bag of Visual Word) image retrieval algorithm. They present efficient data structure to capture images for ImageWeb by doping HITS algorithm (Hyperlink-Induced Topic Search) and context properties also used a query-dependent re-ranking to order ImageWeb. Their experiment results applied on large image search dataset shows accurate improvement performance while the method is highly scalable.

(Kim 2015) proposed a scalable approach to detect near-duplicate image in a dataset of billions of images. They applied local features for clustering in a large data set (one billion), generating cluster seeds via mini hashing to remove false positives while growing the seed according to a growing function. The result was examined quantitatively and qualitatively.

(A. K. Layek 2016) proposed a novel method to detect and group near-duplicate images in online social media. They proposed a hybrid image retrieval technique such as global moment-invariant feature which uses visual vocabulary (VV) and SURF algorithms as local features.

(Z. W. Zhou 2017) propose a fast and accurate method to reduce near-duplicate image for VSNs (Visual Sensor networks) in near-duplicate clustering stage, combining both global and local descriptors in order to obtain efficient NIGs (near-duplicate image groups) cluster. To select the seed images, they adopt PageRank algorithm. This algorithm can find the similarities between near-duplicate images based on their visual context in efficient way, so representative images with more similarities in each NIG will be reserved accurately. The experiments show that this approach resulted in desirable performance in efficiency and accuracy for VSNs.

(Zhang 2018) proposed a general straightforward similarity function from raw image pairs by exploring several convolutional neural network (CNN) models in order to find the best model for near duplicate images. Their model eliminates features extraction complex handcrafted.

(lia Morra 2019) proposed an approach to detect near-duplicate images to solve the problem with false alarms in dataset where these false alarms grow quadratically by growing the dataset size. They stablished the range performance in descriptors obtained via deep-learning (Convolutional Neural Network CNN) algorithms to detect unsupervised near-duplicate. The result was examined in both established and new benchmarks such as Mir-Flick Near-duplicate (MFND) dataset.

(Thyagharajan 2020) proposed a state-of-the-art approach review for detecting near-duplicate images. Their survey focus is based on feature extraction image retrieval methods.

2.1.2 Image attacks

Digital images are subjected to manipulation, tampering and modification which impacts different domains of digital images in such areas as copyrighting, forged images, forensics, spam images, etc.

Image tampering whether it is intentional or nonintentional is known as image attacks which causes Near-duplicate images (Jain 2018).

The most common attacks are as following:

- **Noising:** apply any noise on image pixels such as salt- paper noise.
- **Rotation:** geometrical attack changing the position of pixels.
- **Compression:** recoding the image pixels.
- **Flipping:** mirroring the image either horizontal or vertical.
- **Blurring:** decrees the sharpness of the image pixels.
- **Color Modifications:** any type of color tampering in image color channels space.
- **Cropping:** cut part of the image.
- **Scaling:** resizing the image with keeping the accept ration.
- **Translation:** shift the image along OX or OY axes.

2.2 Image Retrieval

Image retrieval system are techniques for finding, searching, recognizing and retrieving images in large datasets which are used in several purposes such as education, medical , healthcare, evidence-base, cybersecurity and reducing duplicate images (Hui Hui Wang 2010). Image retrieval techniques mostly divided into approaches as following:

- Text-Based Image Retrieval (TBIR)
- Semantic-Based Image Retrieval (SBIR)
- Content-Based Image Retrieval (CBIR)

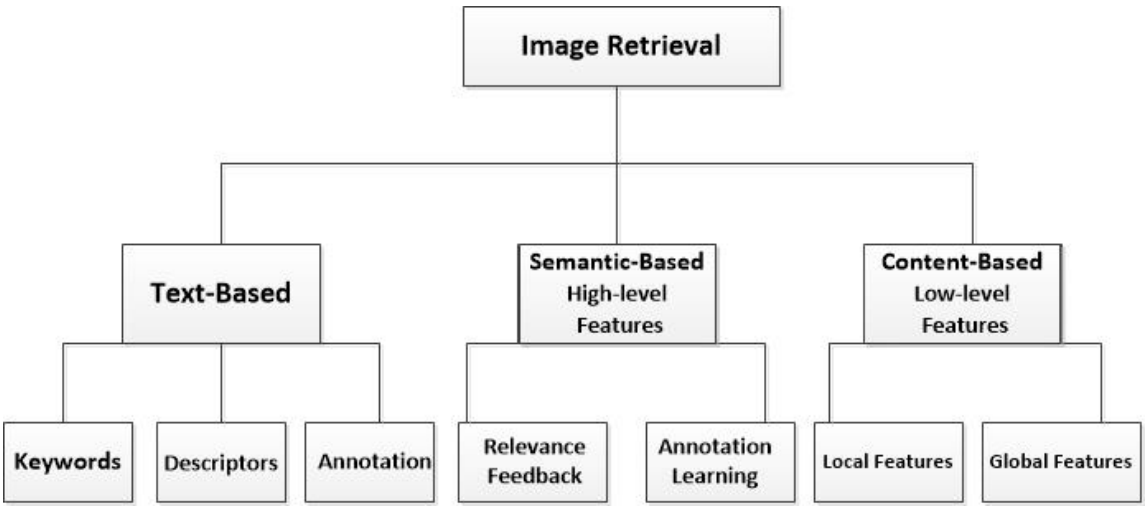


fig.2.1 Image Retrieval CBIR feature extraction methods.

Figure 1 is a diagram of Image retrieval techniques and its divisions.

2.2.1 Text-Based Image Retrieval (TBIR)

In Text-Based image retrieval (TBIR), algorithm used metadata, annotation and keywords tag to each image and store them in database. In search time these metadata are the keys for finding required image. This method is time consuming due, to needs of human perception. (Salahuddin Unar 2019) Unar and Wang proposed a text-based image retrieval meth approach on embedded and scene text by detecting the candidate text regions via MSER algorithms hence they discriminated non-text regions by using geometric properties and SWT then the rest of connected components grouped via bounding boxes. To recognize the text, they faded detected and local text into the OCR. Then they used IN neural probabilistic to form the keywords and finally indexed the textual images based on keywords with four different measures (Essays 2018).

2.2.1.1 Limitations of Text-Based image retrieval (TBIR)

In Text-Based image retrieval (TBIR), image annotating keywords and tags are done manually, this annotating process are for both image content and inserting metadata such as image name, image format, dimensions, image size, etc. Hence to search specific image, user should formulate numeric or textual of the queries in order to retrieve the image which meets one of the annotation criteria. This process in large image dataset is so tedious and requires a great period of the time and in some cases metatags and annotations are incomplete based on human error precursors. Images have more details and attributes in content which sometimes due to limited discrimination criteria annotator could give different descriptions to two images with the same content. This method increases the risk of redundant or duplicated images. Language- dependent of textual annotation is another

drawback of Text-Based image retrieval method for detecting near duplicate images in large datasets.

2.2.2 Semantic-Based Image Retrieval (SBIR)

In Semantic-based image retrieval a specific region or object of interest are extracted from low-level features of the image based on similar characteristics of the visual feature. (Ying Liu 2007) The object or region of interest process are extracted in order to obtain image descriptors which are later stored in a database. To retrieve a query image, the search could be done on a high-level concept as well. The search query process is based on a set of textual words which go through the semantic features of the query images. A mapping process will find the most appropriate concept and will describe the clustered object or region based on the low features. The mapping process will be done via supervised and unsupervised learning to affiliate object concepts with low-level features then annotated with textual word through the image annotation process. The semantic content will be obtained by textual annotation or complex inference procedures based on visual content. The high level semantic based retrieval represents the important meanings of the objects relatively. Instead of extracting visual features such as color, texture, shape and spatial relation, it uses these attributes to obtain semantic meaning information of the image. Hence the conceptual semantic aspects will get closer to the user subjectivity (Wei Jiang 2005).

2.2.2.1 Limitations of Semantic-Based Image Retrieval (SBIR)

The major limitation of semantic based image retrieval is that it is computationally expensive to process. In this method there is dependency on image quality since the annotation process is manual. The process doesn't necessarily scale as the image segmentation become larger in subset size. Also, semantic base image retrieval depends on instance-level representation which should have objects of multiple interests in the image because this method depends on annotation which is usually identifying a single object or instance of interest. The retrieval process has limitation when there are multiple objects in complex scenes.

2.2.3 Content-Based Image Retrieval (CBIR)

Content Based Image Retrieval (CBIR) also known as Query By Image Content (QBIC), is a computer vision technique to retrieve images for specific image retrieval purpose such as facial recognition, similarity detection, etc. Detecting similar images is one of the problems solved by CBIR process which is done by extracting image visual features and by retrieving them to find similarity. In CBIR every image will be represented as a vector in a high dimensional feature space, where for a given query image its content will be retrieved to compare with similar content from other images in database. By using CBIR techniques the information will automatically be retrieved from large image repositories and datasets, such as medical image databases, news archives of news, digital archives in museums or education, online shop and stores websites, etc.

(Kato 1 April 1992) introduced the Content Based Image Retrieval by extracting shape and color features from digital images in order to retrieve them. At the National Science Foundation workshop for visual information management system a workshop was held to recognize new methods in image Database Management System.

The image contents in Content Based Image Retrieval (CBIR) system describe via low-level and high-level visual features which are mathematical representation of a digital image such as texture, color, shape size, etc. (Neelima Bagri 2015).

Content Based Image Retrieval (CBIR) system divided into the tow process:

- Feature Extraction
- Similarity Measurement

The feature vectors extracted from these set of features represent the image content that are stored in feature image database that are waiting for finding any similarities. Depending on the similarity measure between the target image and query image, the candidate list of the matched images will be found. The retrieval process starts when the user sends the query image to the system. The same feature extraction method applies to the query image in the system. A similarity measure will be used to calculate the distance between the feature vector of query and the target image feature vector which is already stored in feature database. Hence the system outputs whether the query image is similar to one of the images in the database according to the similarity measure. Figure 2 is a diagram of the Content Base Image Retrieval CBIR process for feature extraction.

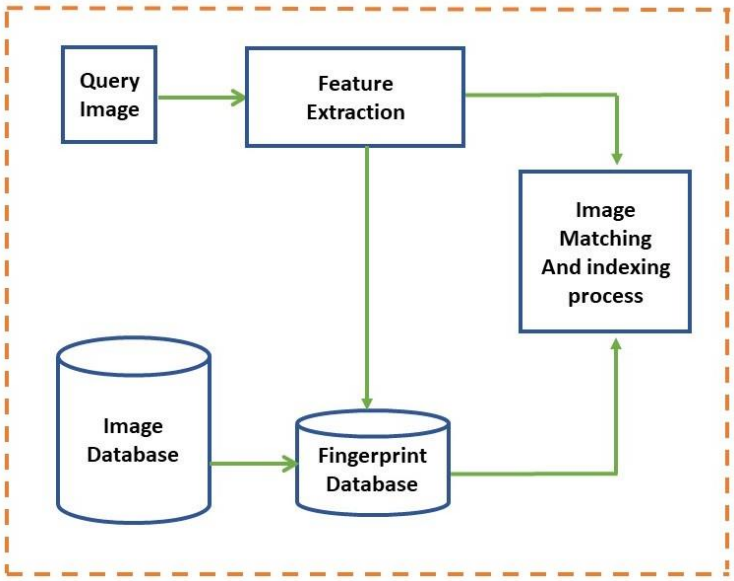


fig.2.2 Content Base Image Retrieval CBIR feature extraction methods diagram.

The effectiveness of CBIR Systems depends on either the performance of the features extracting algorithms and the similarity function optimum known as metric distance.

(Liu and Yang 2013) proposed a novel image feature representation method, used the color difference histogram (CDH). In this method the difference between two points counts from uniform color via different backgrounds colors and edges in L*a*b* color space while most of other techniques based on histogram count the number of pixel frequency. Results demonstrate an improvement comparing with some other feature descriptors content-based image retrieval methods such as MPEG-7 edge histogram descriptors, color autocorrelograms (serial correlation) and Multi-Texton Histograms algorithm.

(N. Puviarasan 2014) proposed a method by combining texture and shape feature extraction technique. They applied image segmentation via Fuzzy C-means clustering and compared

it with k-means to extract texture and shape features. They then use Euclidian distance to calculate the similarity.

(R. Ashraf 2015) proposed content base image retrieval technique by extracting features from a combination of color features and bandlet transformation methods. They used Artificial Neural Networks (ANN) with an inverted index to for efficient image retrieval and. They applied bandlet transformation-based representation in order to extract image salient objects. They examined their result in three data sets, Corel, Coil, and Caltech-101. They obtained better results in average precision and recall values compared to state of the art in standard content base image retrieval system

(Mehmood Z. 2016) proposed a novel image retrieval method base on CBIR combining both local and global histograms of visual word for each image. This allows them to capture the image semantic information by dividing image into local rectangular regions in order to generate a local histogram while a global histogram is used for visual information extracted from whole image.

(Rounak Kumar Singh 2017) explored the content base image retrieval techniques in detail. They discussed different methods with different combinations in order to obtain better image retrieval results. They used a combination of color, shape, texture and edges feature in order to obtain better results in image retrieval system.

(Jabeen S 2018) proposed three novel methods; visual words fusions, adaptive feature fusion, and simple feature fusion of SURF-FREAK feature descriptors which is based on the BOVW (Bag of Visual Word) methodology to reduce the semantic gap between low-level features and high-level semantic concepts that effect CBIR performance. They stated

that their method which is based on visual words fusion increases the CBIR performance as compared to the technique based on adaptive and simple features fusion of SURF-FREAK, standalone SURF, and standalone FREAK methods. The CBIR performance increment is a subsequent of word dictionary size compared to features fusion, standalone SURF, and standalone FREAK techniques. Since BVW and SURF are both local descriptors and have heavy computational process, they proposed different feature percentages per image in order to reduce the computational cost.

(Hamed Qazanfari 2019) proposed a new image retrieval method by combination of low-level features to reduce the semantic gap between the system perception and the human perception of an image. They extracted features via color histogram and CDH to obtain image meaningful information such as color and edge orientation information. Their experiments applied on three benchmark data bases demonstrated that this method has accuracy improvement as well efficiency compared to the methods such as MTH, MSD,

CDH, and BBC.

The below are the common phases in every Content Based Image Retrieval Systems CMIRS:

- Choosing adequate features to extract method
- Indexing dataset
- Defining the beneficial similarity metric according to the features
- Searching

2.3 Fingerprint Method Content-Base Image Retrieval

Image fingerprinting is analogous to physical human fingerprints which represents uniqueness identity. The fingerprint method is a process of extracting descriptors or features as unique image identification. This method was originally proposed for finding forged music and video for Digital Rights Management systems, but lately it has been used as a method in image retrieval in finding similar images, image forensics, image forging, (M. A. Gavrielides 2006), (J. Lee 2012), (Javier A.Montoya Zegarra 2009). The principal task in fingerprint method is extracting image features either from the whole image or a specific image region. These fingerprints are a small representation of image which can defines the image characteristics. Figure 3 is a diagram of the feature extraction process.

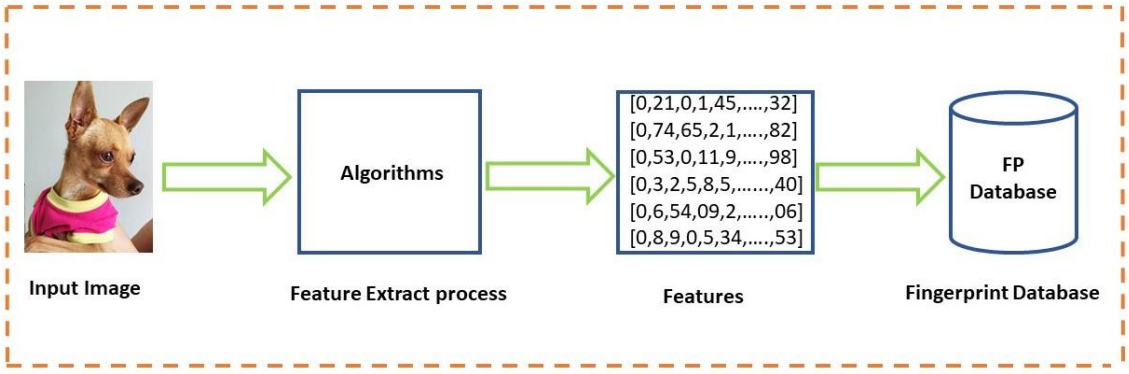


fig.2.3 Extract Fingerprint from images by CBIR diagram

2.3.1 Feature Extraction

Feature extraction (FE) refers to extracting the significant knowledge of images in mathematical representation. It uses the pixel values as features. The features are specific structure in images which all together represents the color, points, edges, textures and objects of the image. Sometimes the features are the result of collecting neighboring pixels.

Features are measurable properties of an image where this distinguishing characteristic represents vector information of the image.

Choosing the adequate feature to retrieve and the feature extraction algorithm are critical in application result performance. The most important benefit of feature extraction is that the amount of information which represent the image significantly decreases for comprehending the content of that image. Once the features are extracted, they are stored in dataset as feature database for further operations. According to the perception subjectivity and the complex composition of visual data, we cannot claim that there is best representation of visual feature which can only solve the image retrieve problem. Hence to obtain the foremost multiple approaches are used for visual features with different perspective (Umbaugh 2005).

Features selected based on retrieval application needs and requirements and the number of iterations require to extract and analyze features are the factors which determine the success and complexity of analyzing an image. The success result of a retrieval directly depends on features robustness (Vassilieva 2009).

Feature extraction is special method of dimensionality reduction where the input image data is large and difficult to process and sometimes the data is redundant. This large set of data is reduced into a set of features known as feature vector set. This feature vector is faster and more efficient than processing the full-size image data inputs. Figure 4 is a diagram of the feature extraction application.

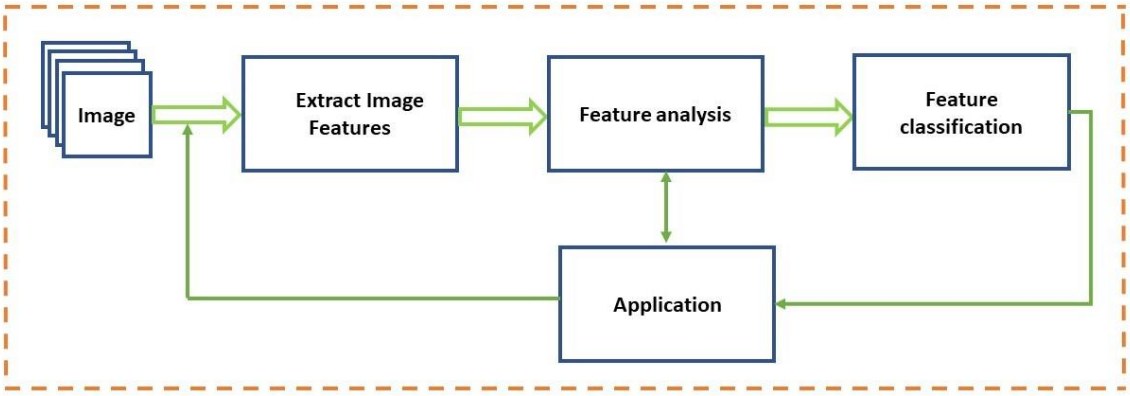


fig.2.4 Feature extraction methods for Classification by CBIR diagram.

Features divided in to the two categories as: Low-Level Features and High-Level Features which both provided phenomenal results to attack image retrieval content-based problems.

2.3.2 Low -level Features

Also known as image global descriptors, low -level features generate and obtain features from the whole image as a single entity. Low-level features extract from an image attributes such as color, texture, shape, and spatial layout. These features are extracted from an image without any object description process. Low-level features mostly are application independent where the pixel-level features are calculated at each pixel (K. Juneja 2015). The information extracted via low-level features does not provide any information on related objects, scene or location of scene, but it gives us a large independent descriptions information such as edge fragments, line fragments, spots, reflectance, etc. (Elnemr 2016) (Emine Gul Danaci 2016).

Low-level features are light weight and fast for data processing tasks. They have less computing complexity as compared to the High-level features. Global descriptors are more efficient and take less space for storage, however they are less robust to transformations such as cropping, occlusion, noising, etc. (Danaci 2016).

2.3.3 High-level Features

High-level features or local descriptors refer to distinctive pattern or structure in an image, such as a patch, points, edge, etc. Low-level features which are normally associated with image patch, differs from their immediate surroundings by color, intensity or textures. In local feature algorithm we are looking for pixels or a set of images which are different from their neighbor pixels in term of characteristics such as color and texture. These areas are known as interest points. Local descriptors describe key points of an object (Mezaris 2003).

High-level features generally capture human vision, subjective views, emotions, or cognition. It converts this information to a small vector representation for analysis and interpretation via a machine. The expectation from image algorithm specifically in large data sets, is to understand, recognize, detect and analyze massive amount of information faster than human vision with high accuracy.

Speeded-Up Robust Features (SURF) (Bay 2008) , Features from Accelerated Segment Test (FAST) (Rosten 2006), Scale Invariant Feature Transform (SIFT) (Lowe 2004), Binary Robust Independent Elementary Features(BRIEF) (Michael Calonder 2010) and Oriented FAST and Rotated BRIEF (ORB) (Rublee 2011) are the most renowned feature-description algorithms used in near-duplicate image detection. Studies have shown that low-level image representations or local-based descriptors generally have

superior results in accuracy terms. However, these features deal with higher complexity computing cost as compared to low-level features.

(Pourreza Alireza 2016) proposed SIFT matching algorithm for partial duplicate image retrieval by extracting dominant color as silent region. The SIFT algorithm detects key points of an image in two databases such as INSTRE and IPDID by combining color histogram and SIFT algorithm.

(Nian 2016) proposed efficient and compact image representation for online near-duplicate image detection based on collective information local descriptors. They extracted local features from local regions which are efficient and discriminative binary patterns. Also, they used global histogram generated by binary vector in order to obtain robust performance. Their experiments provided good representation result while considering both accuracy and efficiency on near-duplicate image detection and video keyframe detection tasks.

2.3.4 The combination of High-level and Low-level Features

Since visual data are subjective and complex, there is not a single best representation to extract all visual feature of an image. To achieve such goal multiple approaches are needed in order to obtain all image features from different perspectives. Each descriptor or feature represents a specific characterization of an image. For example, low level features cannot describe an object or a face in an image however, such problem can be solved by high-level semantics feature extraction algorithms (Zahid Mehmood 2016).

From a human perspective high-level features are those which can be seen in an image called visual features. From such features, the image can be interpreted in order to

recognize and distinguish one image from the other. In a simple world, a human can find the similarities between two images by their high-level features. But this task in machine mechanism is different. An automatic image retrieval system can extract image low-level features such as color, shape, texture, spatial region, color channels, etc. Hence, there is not a direct connection between high-level image features in a way that human can see and understand the image with low-level image features extracted by an algorithm.

Semantics describes the image interpretation from human's point of view. Since the low-level image features cannot satisfy human's perspective, there is a gap which is known as semantic gap. To cover such problem and deal with image retrieval from a human perspective, researchers proposed high-level semantic content base image retrieval which extract semantic features from an image which are visible (M. F. Sadique 2019).

Many research and studies have been done in last years to obtain the best result for content base image retrieval CBIR. These studies have shown that methods that are a combination of both high-level and low-level features give better result. (X. S. Zhou 2016).

In our proposed work, we used a combination of high-level and low-level features to obtain their information such as fingerprint. We compare them in order to find the similarity percentage between images and find near-duplicate images.

2.4 Calculate Image Similarity

As we already mentioned, the image features are represented as vectors which are the image characteristics or objects. To find how similar are two images, we need a method of comparing these characteristics and object from one image to another. There are several methods to compare image similarities measure. They are generally classified in two types:

- As similarity measure
- Metric distance metric

2.4.1 The Similarity Measurement

Similarity measurement also known as matching function, is used to compute the amount of content difference among images based on image features either high-level features or low-level features. Similarity measurement defines distance metric between image vectors which are generated by features extraction methods. This measurement is calculated among all images in database as well for query image.

The image retrieval system will recognize two images as near-duplicate images when the similarity measurement distance is minimum. This function is used for all features such as shape, text, color, texture, etc. Because of diversity in image features using the same similarity measurement distance method for all features may poor performance. Using different metric distance to train image retrieval system can lead to better retrieval system performance. So, choosing an adequate metric distance have great influence on the retrieval task and performance (Guo 2002).

2.4.2 The Metric Distance

The metric distance is used to find similarities between query image and database images. It is the difference between two vectors of images. A small difference between the two determines how similar are the two images. If two features have very close metric distance, it means they have large similarities. The difference between vectors measured by a

distance measurement has n-dimensional feature space. Figure 5 shows a diagram of the different metric distance similarity.

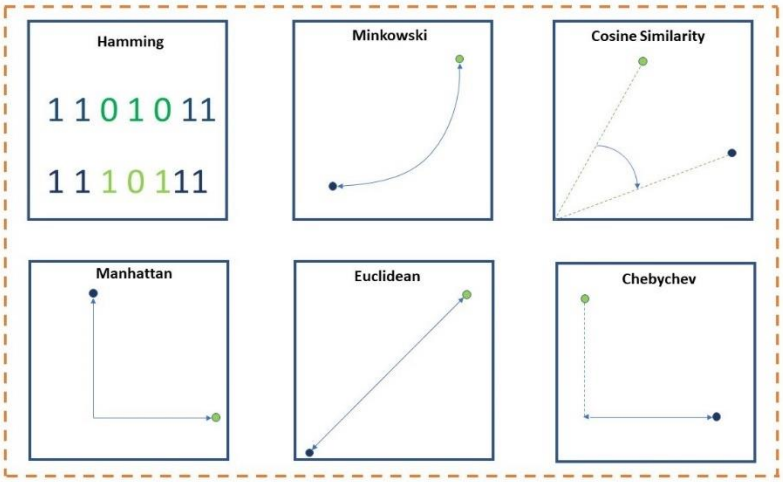


fig.2.5 Different metric distance similarity for Content Base Image Retrieval CBIR.

To find the similarity measurement. a value called threshold is used which could be defined by users. The value of threshold influences how close are two images. In every metric distance the value range is different.

$$Im_1 \approx Im_2$$

$$If$$

$$Sm(Im_1, Im_2) \geq \theta$$

There is different type of metric distance to calculate image similarity measurement which are used for different feature types: Manhattan, Chebyshev, Cosine Similarity, Chi-square, KullbackLeibler, Jeffrey, Minkowski Distances, Euclidean Distance, Mahalanobis distance, Jaccard distance, etc.

In our proposed method, we used the following three different metric distance for our Content-based Image retrieval system: Levenshtein distance (Levenshtein 1966),Jaccard distance (P. Jaccard 1901), and Cosine distance (H. Tang 2012), (Nguyen 2010).

Chapter 3.

The Proposed Method

We divide the proposed work into three phases:

- Phase-1: Offline stage to generate six lightweight fingerprints
- Phase-2: fuzzy search
- Phase-3: Matching system

3.1 Phase-One

Offline stage: In this phase we extracted six different features based on both local and global descriptors to generate a set of lightweight fingerprints with different characteristics which are providing robustness as well as speed in order to detect Near-duplicate images. These lightweight fingerprints are resistant to variety of image attacks such, rotation, flipping, illumination, color distortion, cropping, noise, etc.

We choose these six lightweight fingerprints after practice and examine other fingerprint algorithms from the state-of-art. We developed selected algorithms to unify a Near-duplicate detection system as an application and framework which is robust, fast and scalable. The fingerprints we used in this work are as following:

- Global fingerprints

We obtained three fingerprints based on low-level features including Hash, Color Histogram and Thumbnail.

- Local fingerprints

Three other fingerprints were generated bases on high-level features as ORB (Oriented FAST and rotated BRIEF), BOVW (Bag of Visual Words and finally, CNN (Convolutional Neural Network, using VGG-19 model trained for large datasets) (Simonyan, Very deep convolutional networks for large-scale image recognition. 2014).

These six types of fingerprints are extracted from images that we collected in a database called Native image dataset which contains 200,000 unsupervised images with different sizes, types, contents and color scales. The obtained fingerprints are stored in another database named, FPDB (fingerprint database).

As we mentioned in the-state-of-the-art extracting global features in order to obtain its fingerprints has the advantage of a fast image discrimination process to find images which are suspect of being Near-duplicate. Also using local features provides robustness to encounter image similarity characteristics.

These fingerprints are used as a set of inputs to the algorithm that calculates the similarity measurement compared to the query images.

Furthermore to engage with efficiency, obtain a fast and low cost result, for searching, we employed a metric tree fuzzy search known as Burkhard-Keller tree (BK-tree) data

structure which has a search speed of $O(N \log N)$, this search structure helps us to perform a fast search in the query phase of the method.

In this phase we also construct the fuzzy search tree by hash fingerprint data. For building the search tree there is need of a metric distance to compare the nodes, in this work we used three different type of metric distance including: Levenshtein(hamming), Jaccard and Cosine, in order to compare the performance of each metric distance. These three metric distances are used in search process to calculate the similarity measure of new image as query image and existing images in the database. It is important to mention that the phase one is only executed once per each image database.

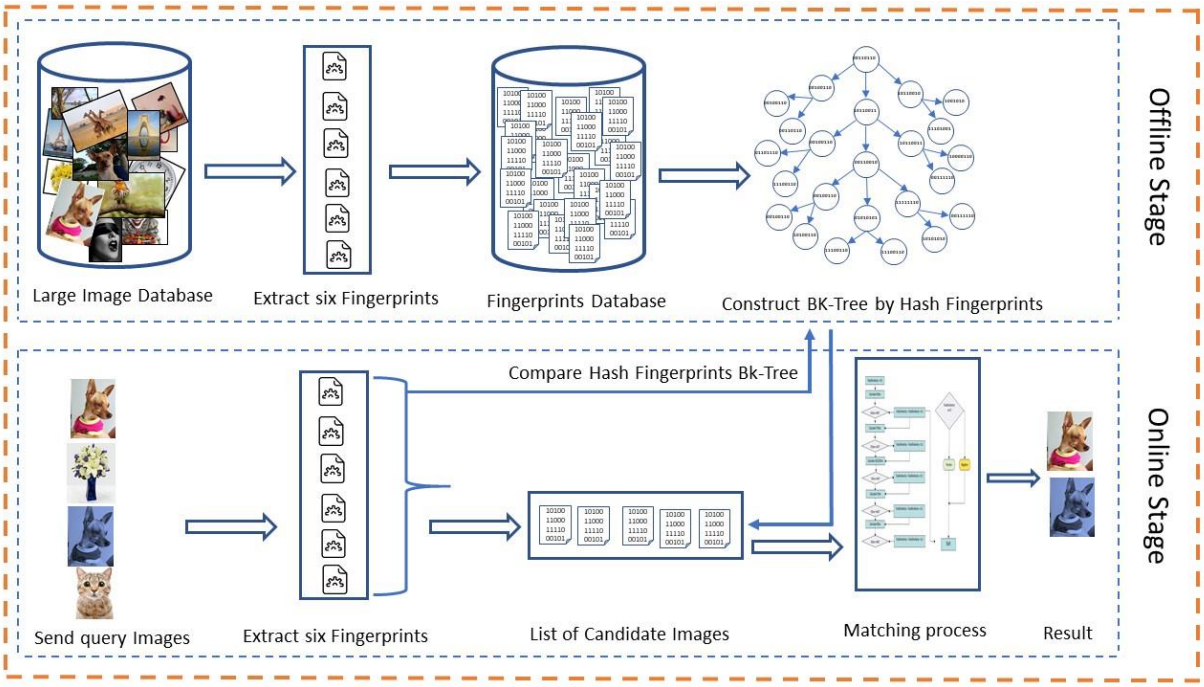


fig. 3.1 schematic the proposed method.

3.1.1 Hash Fingerprint

Since most of the *perceptual image hash algorithms* focus on robustness and for its structure, they are fast to process, in this work we used Block Hash algorithm to extract the

first fingerprint of our Near-duplicate detection system. Image Hash is one of the low-level feature extraction algorithms used in CBIR. There have been several approaches to the problem of Near-image duplicate detection. Until the 2000s, the most widely used method was Digital watermarking. This method requires the addition information about the identity of the image to the image itself. This approach has obvious disadvantages, such as the alteration of the image and the possibility of destroying the watermark using image processing techniques (M. ,. Srivastava 2019). Improving such problem (Fridrich 2000) proposed algorithm based on randomized image block projection, which novel and early ways of feature extraction.

In more recent years, Content-based copy detection techniques using hash algorithms have been gaining popularity. These techniques have the advantage that the detection system does not need to rely on any extra information added to the original image in order to detect copies as is was in previous methods.

Among the techniques that have gained popularity is the *Perceptual image Hash* technique: An image is provided to a processing function, and the result of the function is a Perceptual Hash (a string of bits) that represents the input image. An important property of this hash is that perceptually similar images will produce similar hash values. The hashing technology must be able to match two images as similar even in the presence of a certain degree of variation in one of them. This property is called Robustness. The perceptual hash also has the property that it is compact and allows for efficient storage in database structures specially designed for search efficiency. This property is called Scalability (Ton Kalker 2001).

The perceptual image Hash produced from an image can be represented internally as a number or a numerical vector. In order to compare a pair of hashes, several techniques can be used: Hamming distance, Normalized Hamming distance, L2 Norm/Euclidean distance, Correlation coefficient among others (Zauner 2010).

Near-duplicate detection methods based on perceptual image hashes have the advantage of being efficient and fast due to the small size of the hash (Song 2018).

Generally perceptual hash function which we used in proposed method should meet four properties as following:

- Robustness
- Discriminability
- Compactness
- Unpredictability

If we consider P as perceptual hash, H as hash function, L as length of the hash then Im as image Im_1 near-duplicate of the same image θ_0 and θ_1 as hash value

$\{0 < \theta_0 , \theta_1 < 1\}$ binary string of hash length L so we will have for each of properties as following:

Robustness for perceptually similar two images $Im\ and\ Im_1$:

$$P\big(H(Im) = H(Im_1)\big) \approx 1$$

Discriminability for perceptually similar two images $Im\ and\ Im_1$:

$$P\big(H(Im) = H(Im_1)\big) \approx 0$$

Compactness for perceptually similar two images $Im\ and\ Im_1$:

$$P(H(Im) = \theta_0 | H(Im_1) = \theta_1) \approx P(H(Im) = \theta_0), \forall \theta_0, \theta_1 \epsilon \{ 0 < \theta_0, \theta_1 < 1$$

Unpredictability for perceptually similar two images $Im\ and\ Im_1$:

$$P(H(Im) = \theta_0) \approx \frac{1}{2}, \forall \theta_0 \in \{0 < \theta_0, \theta_1 < 1$$

If $H(Im) \neq H(Im_1)$

So $Im\ and\ Im_1$ are not similar

If $H(Im) \approx H(Im_1)$

So $Im\ and\ Im_1$ are candidate of being Near-duplicated and they should be sent to next fingerprint to verify either they are duplicate or not. we consider the length of the hash as 32 bytes and the value of similarity for all images (its configurable user can change this value). The algorithm used in proposed work is compact, fast for look up search, and so convenient for building efficient search BK-tree fuzzy structure.

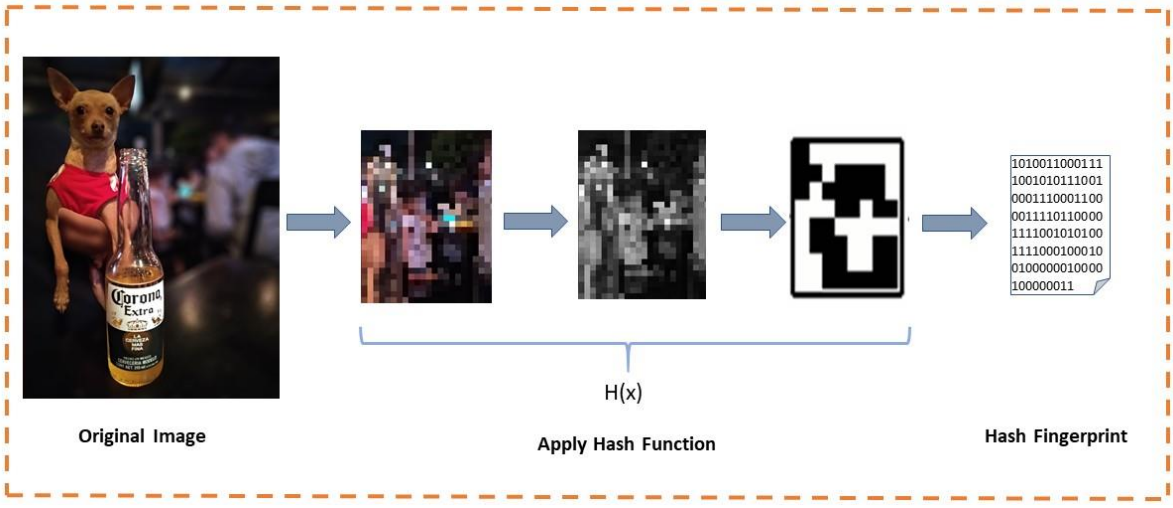


fig. 3.2 schematic hash fingerprint extraction.

Using the hash fingerprint at the first stage of our Near-duplicate detection systems benefits us to discard images that have close similarity measure and introduce them to the system as candidate images. These candidate images later will be filtered by other fingerprints in order to find final Near-duplicate images from a list of query images which were sent to the system on the fly.

3.1.2 Color Histogram Fingerprint

Image Color Histogram is one of the low-level feature extraction algorithms used in CBIR which extract information related to image colors as representation of the color distribution.

Histogram is the intensity values of a color channel versus pixels number of that value.

The color histogram gave the probability of the intensities for three color channels as RGB (Red, Green, Blue). Following image is one of the query images used with RGB channel in the proposed method.

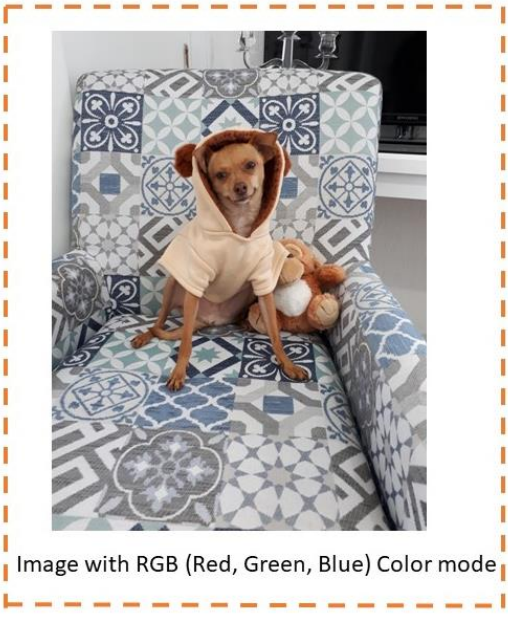


fig. 3.3 Image with RGB color mode used as one of query images.

Colors are one of most significant low-level visual features in fingerprint image retrieval which contains important information. In 1991 Swain and Ballard (Swain 1991) proposed a method know as indexing, which used color histogram indexing in order to identify the object in digital images. The color histogram is calculated as number of times that each color occurs in the image array as shown in the following image.

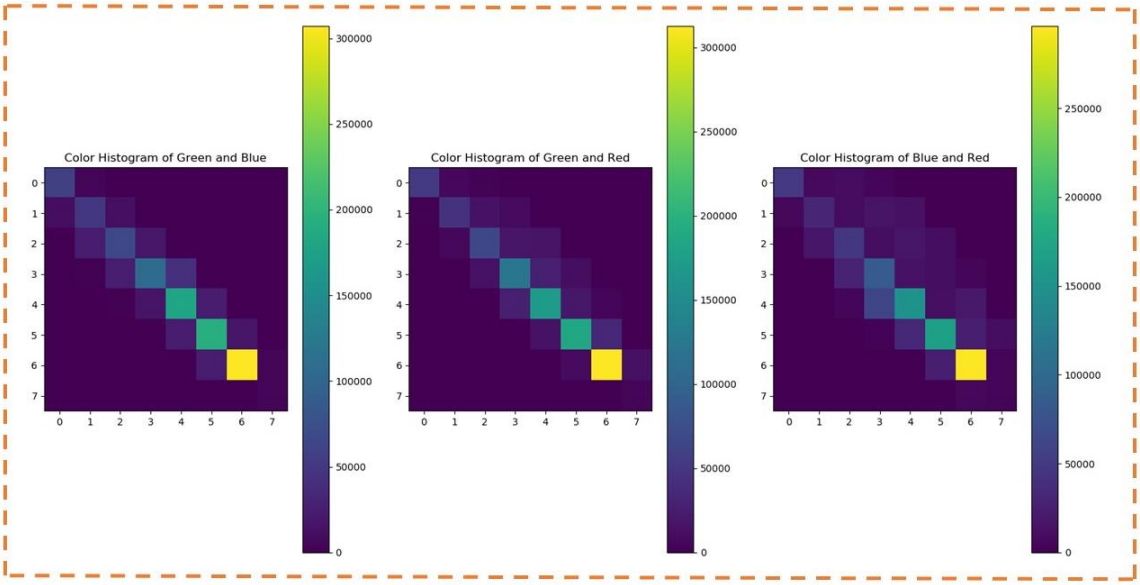


fig. 3.4 RGB channel of the image in fig.3.2

We used histogram as one of the fingerprints in out Near duplicate detection system because is invariant to image rotation as well translation image content (Murala, Gonde and Maheshwari 2009) beside histogram features comparison of is computationally fast and efficient.

This fingerprint is vulnerable to colorization attacks but its resistant to attacks which the original does not change image changes its original color, also is robust for other attacks such as scaling and flipping.

Color histogram fingerprint consider as H for a given image define as a vector:

$$H = \{H[0].H[1], \dots, H[i], \dots H[N]\}$$

Where i is color in histogram $H[i]$ is pixel color i number,N is number of bins used in histogram.

Comparing color histograms for a given image and a query images is used to calculate a similarity measure, since two near-duplicate images are expected to have almost similar color distribution unless the color distorting attack is high. In following image, the color distribution from the image in fig.3.2 is shown.

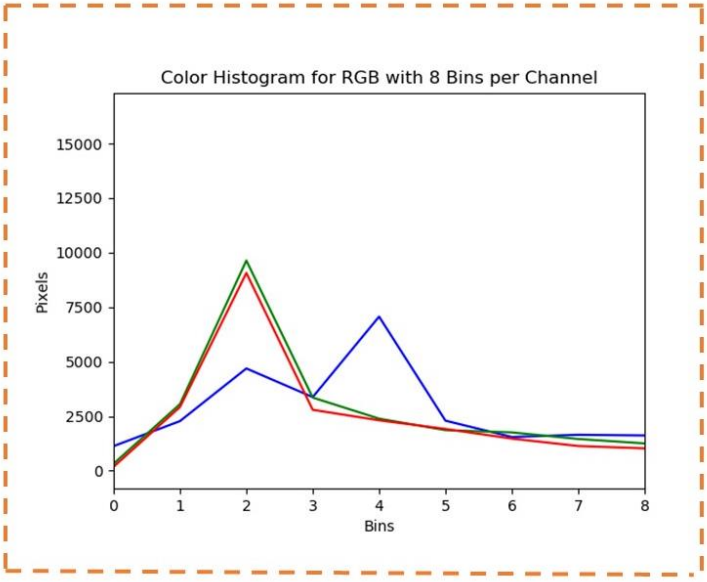


fig. 3.5 RGB color distribution of the image in fig.3.2

We calculated color histogram based on the RGB channels, using 8 bins per channel. With no mask and values from 0 to 256 per channel. The obtained histogram for each image is normalized so their values all fall into the same range. Our method uses the correlation

coefficient to obtain such measure (Rosebrock 2014).Following figure shows the Color Histogram fingerprint extraction process.

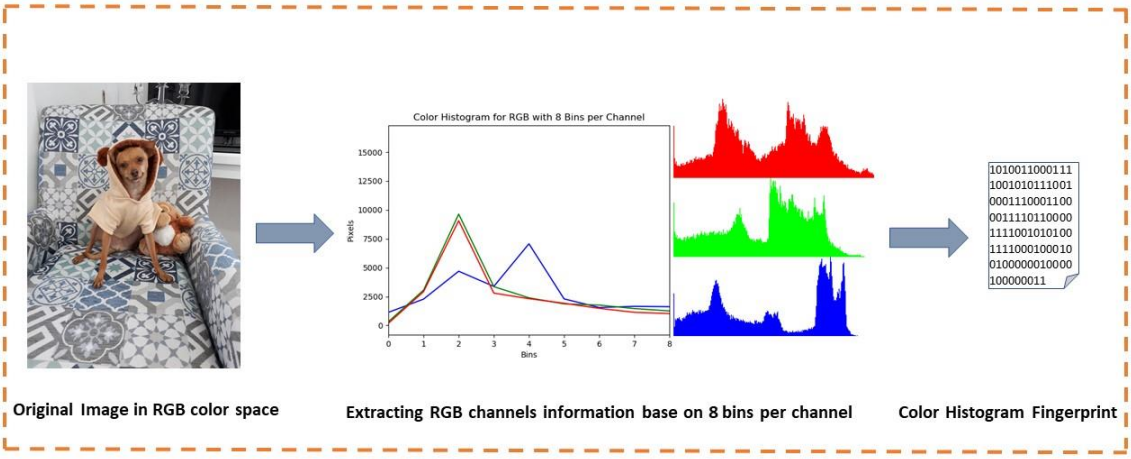


fig. 3.6 RGB color distribution of the image in fig.3.2

3.1.3 Thumbnail Fingerprint

The thumbnail of an image is a compact representation of the same image (fingerprint). Since it is a fingerprint of the visual characteristics of the image, it is also expected to render similar results for similar images. This fingerprint is used as a complement to the histogram: The histogram is based on the statistical color distribution of the image. The thumbnail fingerprint accomplishes the same for the spatial distribution of the image features. In order to ignore the color characteristics of the image, before generating the thumbnail fingerprint, each image is transformed to grayscale and resulting image is resized to a compact array of 30×30 elements. The grayscale levels of the array are normalized, and the resulting matrix is flattened. This flattened array is stored to our fingerprints database to be retrieved and used in subsequent calculations for finding the near-duplicate images (Muja 2009).

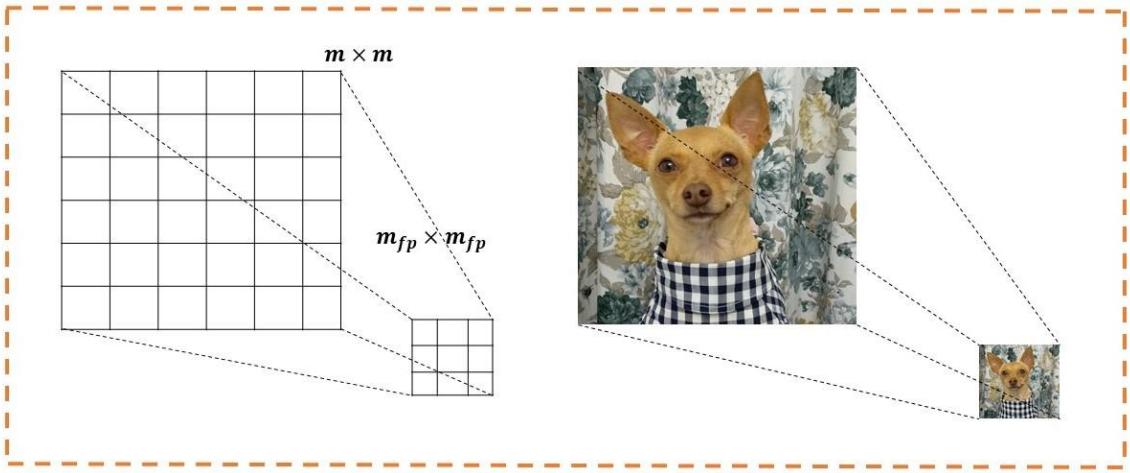


fig. 3.7 Thumbnail fingerprint calculation by down-sampling.

In the proposed word we used K-Nearest Neighbors (KNN) algorithm to down-sample the original images into the small representation thumbnail as a new fingerprint. In k-NN regression, the output value is the property value of the object which the is the average of the values of k nearest neighbors. K is a positive integer. To obtain the thumbnail fingerprint for each image the following process has been applied:

Divide the original image matrix into the number of required matrixes, consider the original image matrix as:

$$m \times m$$

The thumbnail fingerprint as

$$fp_m \times fp_m$$

for example, if the original image matrix was $Im = m \times m$ we should divide the matrix as:

$$\frac{m}{fp_m} = x$$

Then we divide original matrix row and column to X row and X columns as:

$$r = X, c = X$$

Then we will get the average of each divided Matrix to obtain the thumbnail fingerprint as:

$$Im_{pf} = [i \times r : r \times (i+1); j \times c : c \times (j+1)]$$

Where r defines the row, c defines the column, i is pixel in the row and j are the pixel in column.

Thumbnail fingerprint is resistant to flipping and colorization attacks also it's a compact and fast to retrieve.

To have this benefit of resistance in colorization attacks, we first convert the image to its Grayscale version then read the image in grayscale and applied the down-sampling and we did the normalization of grayscale. The normalization step is for invariance against illumination changes. The Thumbnail fingerprint is shown in below figure.

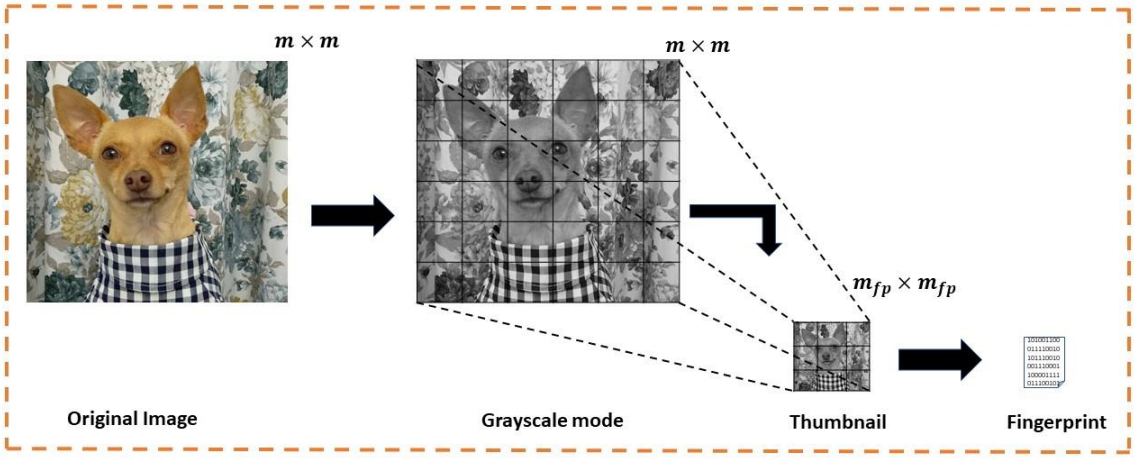


fig. 3.8 Thumbnail fingerprint process.

The Thumbnail fingerprint result is a vector archive with a very compact size:

$$Im_{pf} \geq 30$$

In our algorithm them size of Thumbnail fingerprint can be configure by user but as default we choose 30×30.

Local Fingerprints

3.1.4 Deep Convolutional Neural Network Fingerprint

Deep Learning is a hierarchical learning following by human brain data processing function which does decision making operation. A deep learning system learns from data representations. The architecture of a Deep Learning model use Convolutional Neural Networks extract importance of an image such as learnable weights and biases from different objects in the image as well differences between all objects in the image.

Nowadays, convolutional neural networks (CNNs) (Y. LeCun 1989) are one of the best representations image retrieval compact which proved outstanding performance in computer vision field.

The performance of any CBIR system either for detection or classification and recognition, strongly depends on the features employed in terms of obtaining an efficient and effective result. Therefore, finding the best image feature extraction method has been the high interest of researchers in computer vision field. The robustness to geometric transformations and effectiveness is one of the major properties of local fingerprints. Although local fingerprints provide accuracy to CBIR but due to their complexity

computational process of extracting process also storing and matching them they are not efficient as Global fingerprints are (Schmidhuber 2015).

Convolutional neural networks (CNNs) are one of the best representations for image retrieval, one of the earliest image retrieval approach based on CNN is ImageNet (Stanford Vision Lab 2016) which is a rich resource for researchers as well as educators.

CNN have been used in several domains such as image and video recognition, image classification, medical image analysis, natural language processing, financial applications recommendation systems etc. Convolutional neural network indicates a specific neural network which employs mathematical operations on two functions and produce a third function which presents how the shape of one function is modified by the other. The convolution refers to the result and process of computing functions.

The earliest Convolutional neural networks architecture composed as following layers:

- Input image
- Convolution layer (CONV)
- Pooling (POOL)
- Fully Connected (FC)

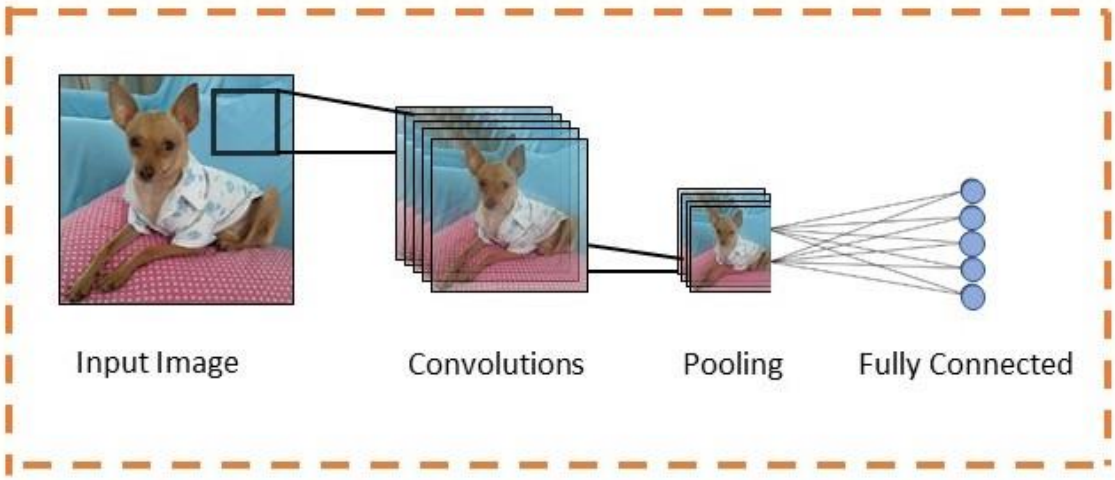


fig. 3.9 earliest Convolutional neural networks architecture.

Input image:

The input image is the image which CNN algorithm will apply on for retrieval process. The size of input image can have impact on length of process and the computational complexity.

Convolution layer (CONV):

Convolution is the core building block of a CNN; this layer uses learnable filters (kernels) which apply convolution operations by scanning the input image as(*I)* considering the image dimensions. Its hyperparameters contain both filter size (*F*) and stride(*S)*. This layer result output will be recognizing (*O)* as feature map or activation map.

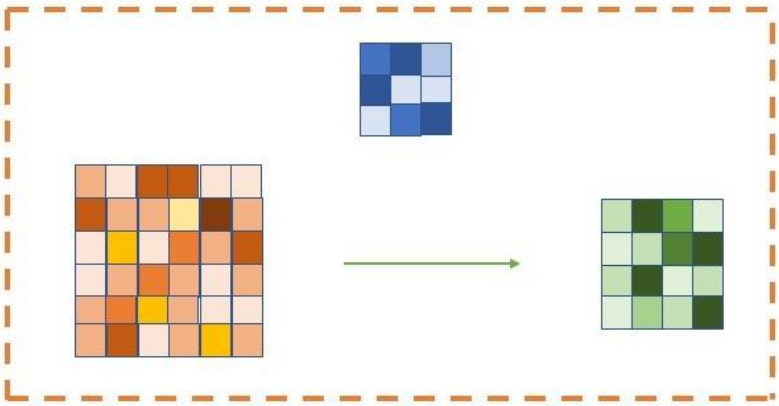


fig. 3.10 Convolution layer (CONV)

Pooling (POOL):

This layer is usually applied after convolution layer for spatial invariance operations. This layer does the down-sampling based on two, maximum and average mathematical operation. In Maximum operation a filter normal 2×2 dimensional applied into the input value, take the maximum number of each subregion that the filter convolves around it. In average operation the average number of each subregion will calculate as output value (Y. LeCun 2000-2018). The maximum pooling method is more common to use in Convolutional neural networks.

The important reason of using pooling layer is that once the specific feature in the original input volume where there is high activation value, was recognized, the exact location of the feature will not be as important as that relative location to the other features. This layer highly reduces the spatial dimension of the input volume. (length and width will reduce but depth will not change). Hence the amount of weights will reduce computation cost as well. Also, we will not face with overfitting in our model.

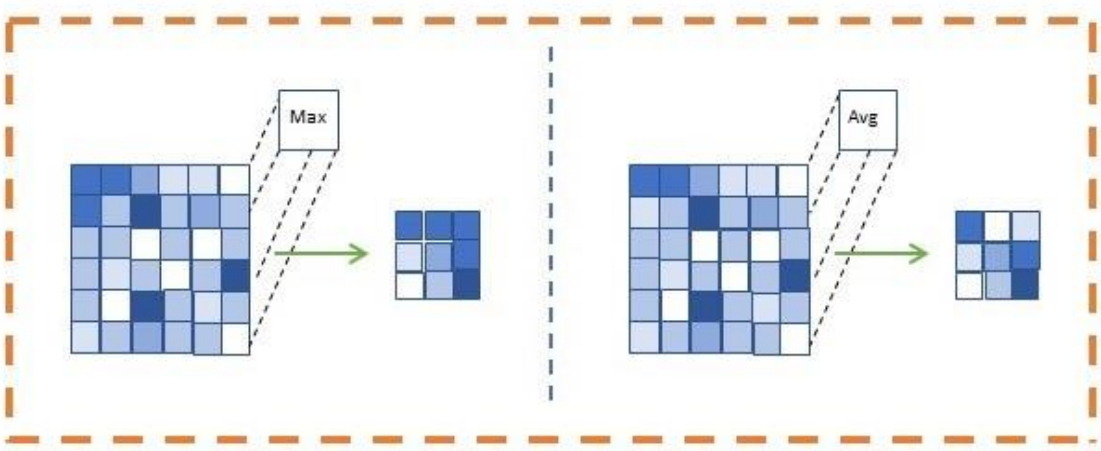


fig. 3.11Pooling layer (Pool)

Overfitting problem

Overfitting occurs when a model is very tuned to the training examples so cannot generalize well the validation and test sets. When a function is very closely fit a limited set of data the overfit will come up. Overfitted model contains more value of parameters which cannot be justified by the data.

However there have been several studies to prevent overfitting problem but still is one of the disadvantage of convolutional neural networks CNNs which is computationally expensive for image retrieval (N. H. Srivastava 2014).

Fully Connected (FC):

In this layer neurons are fully connected to all the activations in the previous layer, this operates on a flattened input. Usually Fully Connected layers are the last layer in Convolutional neural networks architectures. FC layers also are used for optimizing objectives such as class scores (Shervin and Afshin n.d.). The last Fully Connected layer is known as the “output layer” and in classification settings it represents the class scores.

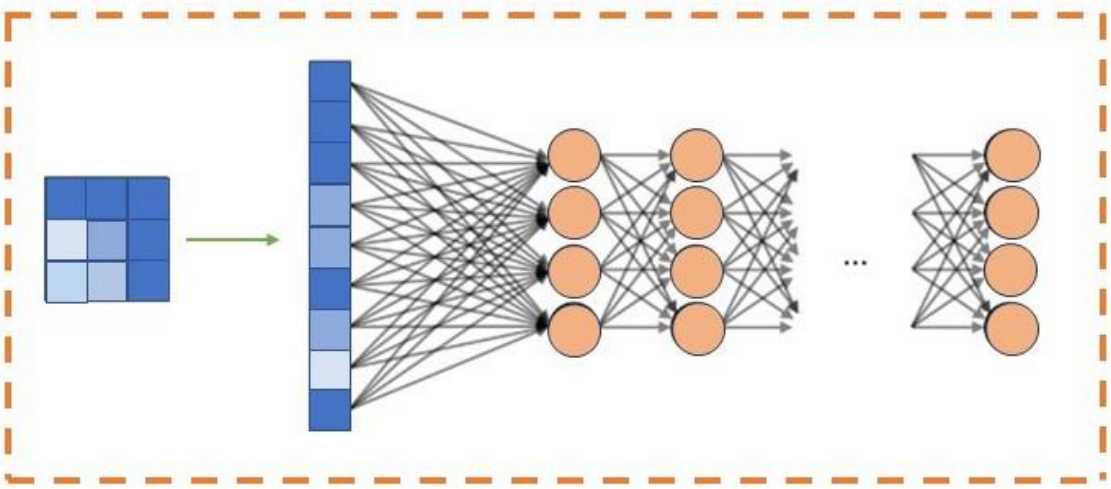


fig. 3.12 Fully connected (FC)

Stride and the Padding Zero

We already mentioned that convolution layer contains filters, after choosing filter size for each layer we need to justify two more concepts call Stride and padding.

- Stride does the inspection how the filter should convolve around the input volume shifting one unit at a time. The Stride value should be a number which provides an integer number for output volume. The stride ***S*** define the number of pixels by which the filter applies on each operation.
- Padding zero symmetrically add zeroes to the input matrix, it is used when dimensions of the input volume need to be preserved in the output volume. In this process ***P*** zeros added to each side of the input boundaries. The value could be either be justify automatically or manually based on three modes:
    - **Valid** $P = 0$ (1)
    - **Same** $P_{start} = [\frac{s\left\lceil\frac{1}{s}\right\rceil - I + F - S}{2}]$ (1)

$$P_{end} = \left[\frac{s\left[\frac{1}{s}\right] - I + F - S}{2}\right] \quad (2)$$

- **Full** $P_{start} \in [0, F-1]$ (1)

$$P_{end} \in [F-1] \quad (2)$$

Hyperparameters

There is not a standard to define and choose the number of layers applied in CNN model or the size of filters and the value of stride and padding. These variables are depending on the data type that we are dealing with. In images data can vary by complexity of scene, the image size, the image processing task and algorithm, etc. In deep learning a hyperparameter are setting and parameter which its value set before the learning process starts although the other parameters values of are derived by training process. Hyperparameters control the behavior of a deep learning algorithm. finding the right combination which provides abstractions of the image at a proper scale is the best way of choosing hyperparameters.

If we consider ***I*** as length of the input volume size, ***F*** as the filter length, ***P*** presents the amount of zero padding, ***S*** the stride and the output size ***O***, hence the feature map for such dimension will be calculated as following:

$$O = \frac{I - F + P_{start} + P_{end}}{S} + 1$$

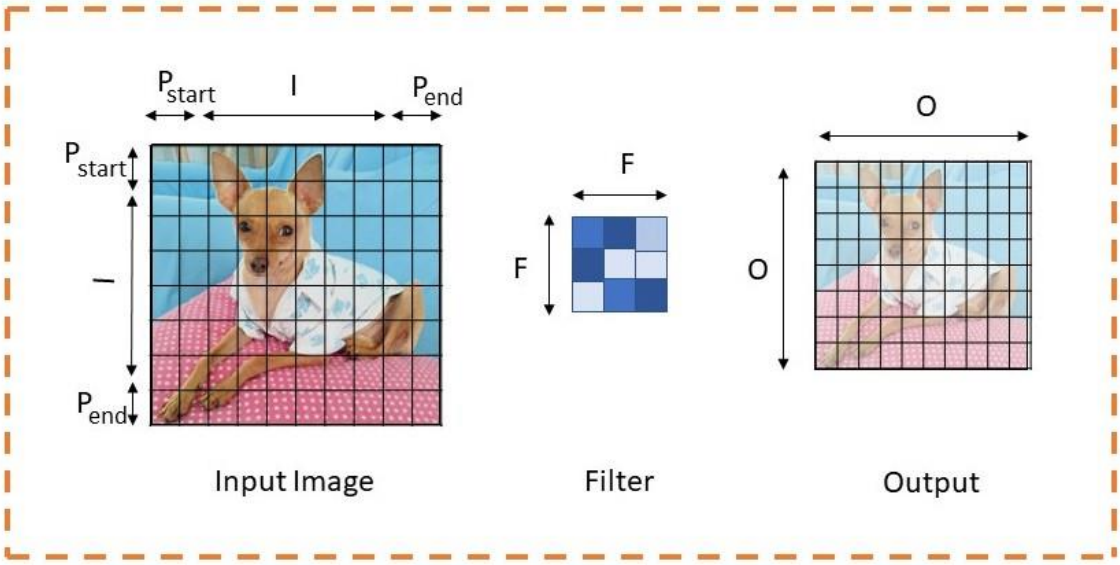


fig. 3.13 Parameter compatibility in convolution layer

Transfer Learning

Is a deep learning approach for improvement of learning by machine which is very practical in computer vision. In transfer learning the tasks first train in base network on all data from database known as leaned features, these learned features later will repurpose or transfer to the second network target for training task this process continues until the task process complete. All features that has been trained for a task are transferable to related task. Transfer learning are mostly used in predictive modeling and classification. Transfer learning is defined as domain and task where domain ***D*** two-element tuple consisting of feature space, ***X*** and a marginal probability distribution as ***P(X)***

Where $X = \{x_1, \dots x_n\} \in X$

And a domain represents as:$D = \{X, P(X)\}$,

tasks in transfer learning includes two components, a label space as; γ and predictive function as; μ which learned from feature vector and label$(x_i, y_i), x_i \in X\ , y_i \in \gamma$

for every feature vector from domain the predictive function corresponding label is as $\mu(x_i) = y_i$ so the task is define as: $T = \{\gamma, P(Y|X)\} = \{\gamma, \mu\}$ $\quad Y\{y_i, \dots y_n\}, y_i\ \epsilon\gamma$

By given source domain as; D_s and learning task as;T_s , target domain as; D_T and learning task as;T_T the transfer learning for improvement of the learning target predictive function μ in target domain D_T use the knowledge in source domain D_s and learning task T_s where the $D_s \neq D_T \quad or \quad T_S \neq T_T$

The following figure demonstrate the Transfer Learning diagram.

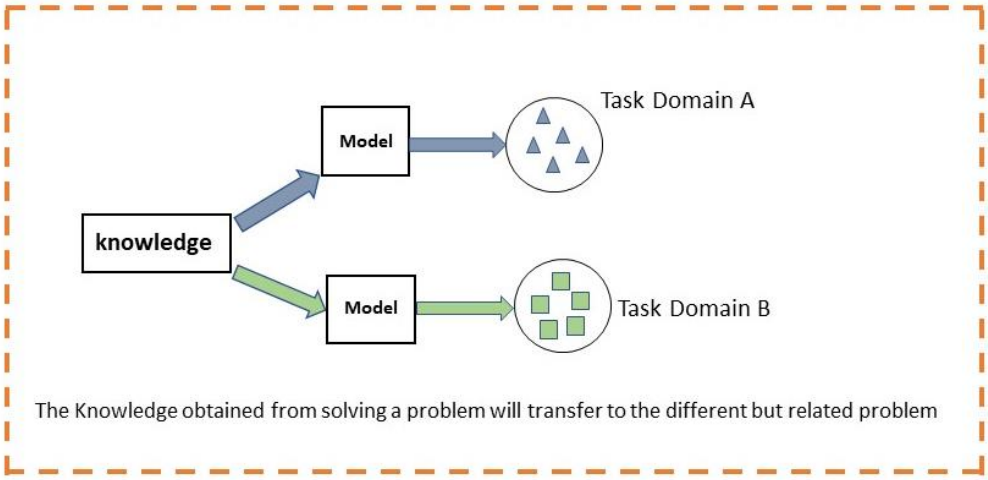


fig. 3.14 Transfer Learning

One of the most common approaches in transfer learning which are successfully is used in deep learning for image retrieval targets are Pre-trained Model.

Pre-trained Model

Deep learning Pre-trained models which are already implemented, developed and trained by other researcher or institutions in a large database to be used for solving other image retrieval problem approaches. These models are used as starting point sometimes we use all or just part of a pre-trained model, also we need to tune the model based on new task and needs of reuse. In the proposed work we used a pre-trained model called Vgg19 as one of our fingerprints.

3.1.4.1 Very Deep Convolutional Networks VGG19

VGG19 is a convolutional neural network trained model proposed by K. Simonyan and A. Zisserman from the University of Oxford (Simonyan, Very deep convolutional networks for large-scale 2014). The architecture of this convolutional neural network is characterized by its simple and efficient, this neural model uses only 3×3 convolutional layers stacked on top of each other in increasing depth. The vgg19 model uses Max pooling in order to reduce volume size. Also uses two fully connected layers, each with 4,096 nodes. Deep learning models are layered architectures that learn different features from different layers. This layered architecture helps us to use a pre-trained neural network. In vgg19 the number 19 stands for the number of weight layers in the neural network as seen in following figure.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

fig. 3.15 (Simonyan, Very deep convolutional networks for large-scale 2014)

The vgg19 architecture contains very small (3×3) convolution filters as well, which the first few layers of VGG are sensitive to the image style and last layers are sensitive to the image content. Some research shown that the effectiveness of using three 3×3 conv with stride 1 layers has same effective receptive field as one 7×7 conv layer has (Yeung 2017).

In this work we used pre- trained Vgg19 neural network as one of our fingerprints just to extract images features and latter compare them to find the similarity of the images. As we. Pre-trained models have several benefits, one of the most essential benefit is pre-trained models save the time of training a new model. While the model already has been trained

by computing the resources to learn many features. Learned features usually are transferable to different data. For example, if you train a model with large dataset of dog's images you will have learned features such as edges or horizontal lines which are transferable to any other dataset.

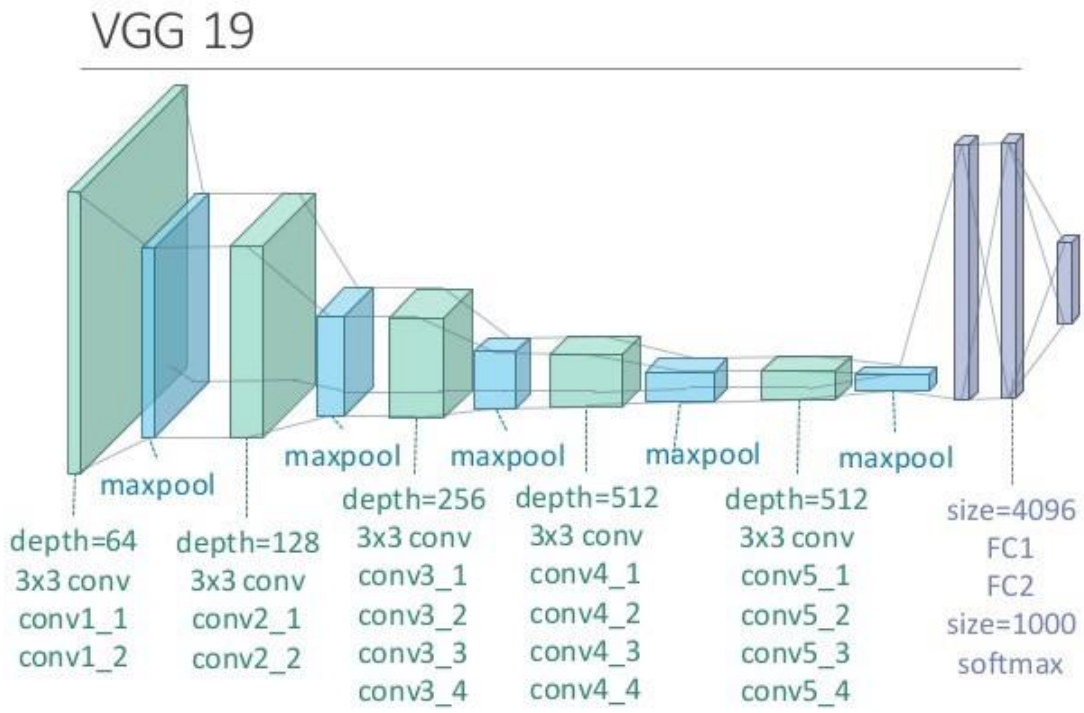


fig. 3.16 Vgg19 architecture (Chansung 2018)

A pre-trained model can be used as a feature extractor as we used in this work to extract features from all images earthier from large dataset and query images in order to have another fingerprint for compression process.

We cannot use the VGG19 model as-is. We need to remove the final output layer. The second last layer is fully connected and is a vector of 4096 integer numbers. This vector

can be used as input to a similarity measure. For this work we use the cosine similarity between the two feature vectors.

The last layer which is known as output layer is used for calcification and detection of objects, but what we need in this work was just an extractor of features. As shown in the following image.

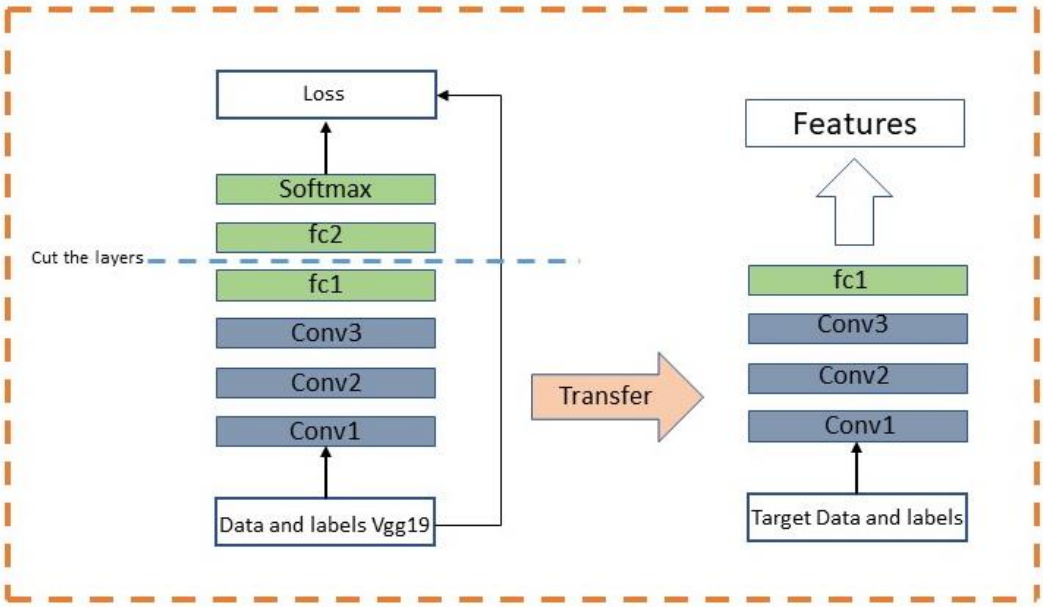


fig. 3.17 Extracting features by Vgg19 pre-trained model

3.1.5 ORB (Oriented FAST and rotated BRIEF) Fingerprint

Another approach commonly used in computer vision for Content Based Image Retrieval is the extraction of the local features and use of the features as an input to match algorithms. Several algorithms have been developed for this purpose such as SIFT, SURF, BRIEF, etc. SIFT and SURF have been demonstrated to give good performance and are widely used but unfortunately, they are patented algorithms. Hence to solve such problem [Rublee et al] proposed a new method for local features extraction that claimed to perform comparably to SIFT while using less computational power know as ORB (Oriented FAST and Rotated

BRIEF). In this work we used ORB as one of our fingerprints. To extract features by ORB we should process following steps:

- Find the keypoints position by FAST
- Select N best points
- Add direction the keypoints at Intensity Centroid
- Extracting Binary description by BRIEF
- Find how correlative pixel block
- Receive 256-bit feature

ORB takes advantage of 2 previously existing methods for the different phases of its execution: It uses the FAST algorithm for detection of keypoints and the BRIEF algorithm for the generation of the descriptors. These two algorithms perform well while not being very costly in computational terms. However, they have some limitations when compared to patented algorithms like SIFT. The BRIEF descriptors do not perform well under rotation operations and the FAST keypoints do not offer an orientation component.

The ORB method proposes additions to the limitations mentioned above and has the advantage of being free to use without license fees.

The ORB algorithm uses multiscale image pyramid which is representation of an image with different rotation position, each level of the pyramid is a down sample of the previous image level. When the pyramid is complete it uses the Fast algorithm (Features from Accelerated and Segments Test) in order to detect keypoints from the image in each level,

ORB effectively locate keypoints in different scale, hence is partial invariant. Creating level is shown in the following figure.

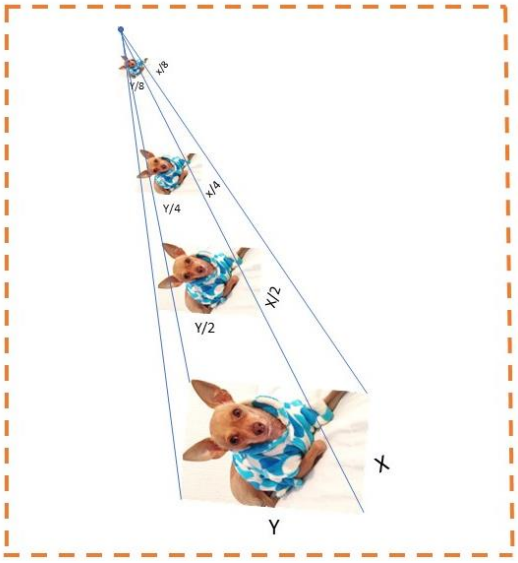


fig. 3.18 ORB creates pyramid to use in Fast algorithm

when locating keypoints is done ORB will assign an orientation to the keypoints depends on intensity around the keypoints (Tyagi 2019). To calculate corner orientation with centroid intensity, assuming, the corner intensity is unbalanced from the center we sued following:

$$C = \left(\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}}\right)$$

Where m_{ij} is the ORB descriptor-patch's moment define:

$$m_{pq} = \sum_{x.y} x^P y^q \, I(x, y)$$

To construct an ***OC*** vector from corner's center ***O*** the orientation will be as:

$$\theta = arctan\frac{m_{01}}{m_{10}}$$

One the orientation calculated for a patch them we can compute the descriptors and gain some rotation invariance. Following figure show the keypoint calcualted by Fast.

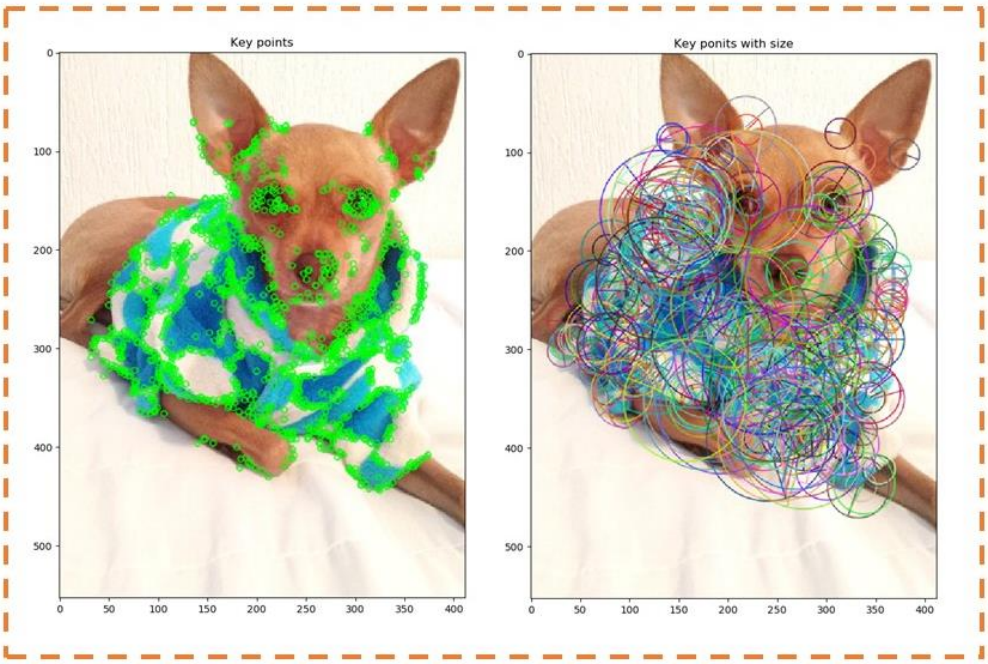


fig. 3.19 Keypoints obtained by Fast algorithm

Brief (Binary robust independent elementary features) will obtain all keypoints founded by Fast algorithm and convert them to the binary feature vectors, these vectors represents an object at the image. Vectors or features at Brief has 128-512 bits string. Brief select randomly pair of pixels by neighborhood, knows as patch. Patches calculated by square of pixel 2D dimension (width and height). ORB calculate the functionally considering the speed as following:

$$\tau(p; x.y) = \begin{cases} 1 : p(x) < p(y) \\ 0 : p(x) \geq p(y) \end{cases}$$

Where p is patch, τ is a binary test, p(x) is the intensity value at pixel x

For calculate feature vectors we have:

$$f(n) = \sum_{1<i<n} 2^{i-1}\, \tau(p; x_i, y_i)$$

Where p(x) is the intensity of p at a point x.

To perform matching keypoins ORB uses steer BRIEF related to the rotation of ketpoints so for each feature perform n binary test at (x_i, y_i) location by a matrix as following:

$$s = \begin{bmatrix} x_1, \dots . & x_n \\ y_1, \dots . & y_n \end{bmatrix}$$

$$s_\theta = R_\theta S$$

Where θ is the patch orientation R_θ is the corresponding rotation matrix and s_θ is to create a steered of s . Hence the steered BRIEF will be as:

$$g_n(p, \theta) = f_n(p) \mid (x_i, y_i) \in s_\theta$$

The angle will be increases to 12 degree to create a lookup table in order to precompute the BRIFE patterns. The set of points S_θ are used to compute features continually until all image feature are extract.

In following figure show an image with its transform apply ORB algorithm and similar keypints of an image are recognize.

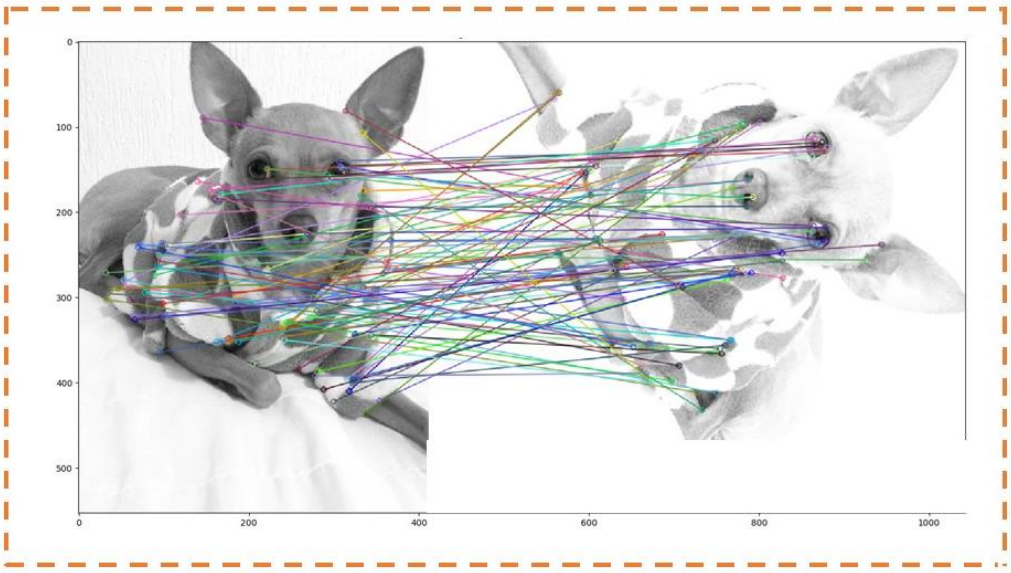


fig. 3.20 Match Keypoints to recognize the similarity of image.

However, the image had several image attacks, but the objects was match from the original to the transformed image.

3.1.6 BOVW (Bag of Visual Words) Fingerprint

The Bag of Visual Words (BOVW) approach has been use traditionally in language processing applications to create representations of text documents. The main idea of the method is to count occurrences of each word inside the text and keep a counting table of entries for them. This table with each word associated to its count number is known as the Bag of Words and it can be used as input to training algorithms for classifiers. For example: an article in a medical journal is expected to have a high frequency of medical and anatomical terms. Moreover, the frequency of the words used will be different for an article on neurology from an article on dermatology. The bag of word technique is efficient and

effective as well since any information regarding the order and locality of words will not be store.

This approach can also be used successfully in computer vision, but instead of using textual words, local image features are used. In Bag of Visual Word image patches and their associated feature vectors are treating as they are just a word, on a similar way to text documents, images that are similar can be expected to have a certain degree of overlap in the statistical counting of their local features. representing an image as a collection of unordered image patches;

fig. 3.21 Extract features in Bag of Visual Words

In order to use the BOVW method appropriately, the following steps are necessary:

- Extract features from images
- Define Vocabulary
- Map Codeword
- Construct Codebook

A vocabulary must be defined. This is usually done by grouping the feature vectors. Each group of vectors is mapped to a codeword and then the codewords are used to build a codebook.

Although the BOVW representation is effective image retrieval method, but it doesn't lend itself well to large scale databases. In (H. Y. Ling 2013), a method is described for using a Bag of Visual Words model to generate a compact image fingerprint.

The Bag of Visual Word used SIFT (Scale-invariant feature transform) for extracting local features from images and SIFT use Keypoints which are selected based on measures of their stability in the image they are extracted from the objects.

In this work we used BOVW the compact image fingerprint, obtained by transforming each element of the description vector into a single bit with its value set depending in its deviation from the vector mean. The result is a set of 32-bit words which are then represented as a histogram for the image under analysis. An intersection of the histograms for the corresponding images is used as a similarity measure for the images (Jabeen 2018).

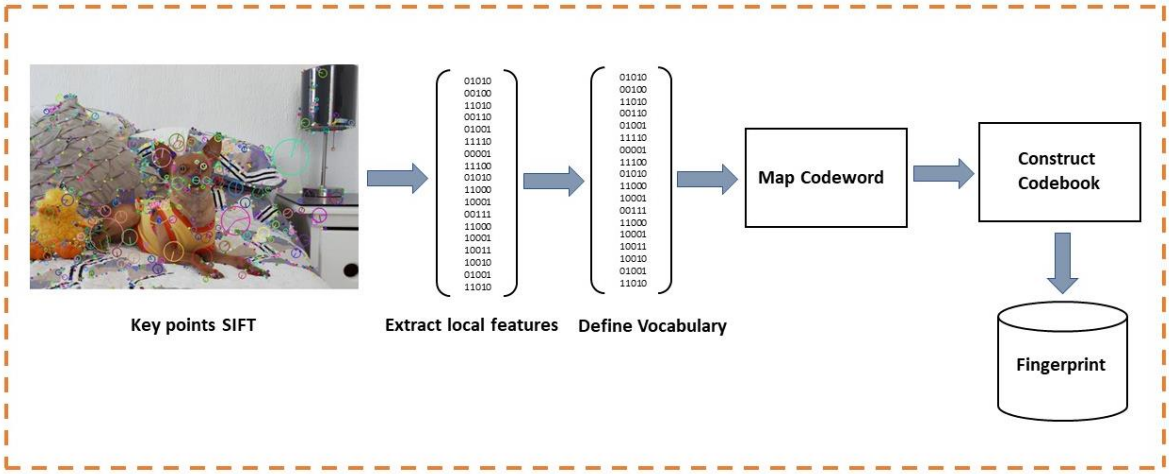


fig. 3.22 Local Fingerprint the Bag of Visual Word by SIFT

Using the Bag of Visual word and vocabulary construction for images as a fingerprint representation is efficient and scalable for large database. Since the vocabulary size in a large database grow the performance of finding near duplicate will increase as well so the process led to better efficiency rate. In this work we precalculated the vocabulary and features hence this speeded up the search process. The results will be show at chapter4.since the bag of visual word uses the SIFT algorithm hence, we can detect near duplicate images which had Scale, Rotation, Illumination and point of view image attacks

Viewpoint with a good result. Also is resistant to occlusions in object recognition, due to

Providing SIFT local descriptors extraction.

3.2 Phase-Two

Online stage: In this phase candidate images are discard, this step new images are presented to the system as a query images, immediately the first fingerprint (hash) of the query image will be calculated, then the binary string hash result will be searched in the fuzzy search BK-tree obtaining a list of candidate images similar to the query image using tree different metric distances sequentially for comparison purposes.

This Metric distance has the advantage of fast search between binary hashes and tolerance for long hashes as well as being configurable, so the user can manipulate it and specify a threshold for the metric distance's accuracy. By choosing smaller threshold the candidate images search will be more restrictive. In experimental results we tried several thresholds amounts in order to compare the results and finding a better balance between speed and accuracy. In following figure, the online stage process is shown.

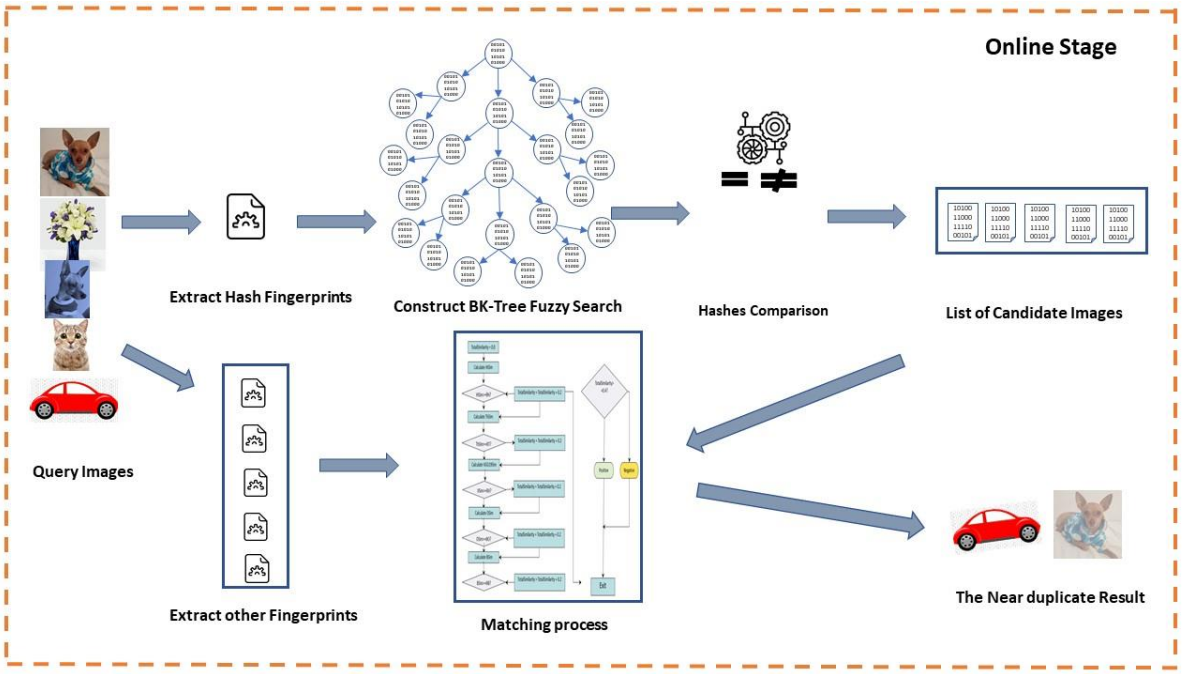


fig. 3.23 Online stage for detect near duplicate image in proposed method

The metric tree data structure determined to index data in metric space and search for similar elements to a given query, to find the similarity between data base hashes and query image hash, a distance function as ***d(a,b)*** is used for every pair of hashes ***(a,b)*** where ***a,b*** are objects of the metric space. which should satisfy the following properties:

Triangle inequality $d(a,b) \leq d(a,c) + d(c,b)$

Symmetry $d(a,b) = d(b,a)$

Non-Negativity $d(a,b) > 0\ if\ (a \neq b) and\ \ a(a,b) = 0\ if (a = b)$

In this work the algorithm provides the facility of storing the query image in another database, in case that user decide to add a new image to the database and scale up the size of dataset, first the algorithm obtains its six fingerprints and save them in FPDB (Fingerprints Data Base). Otherwise if the user intent is just detecting the Near-duplicate image in the dataset the query image fingerprints will not be stored.

3.2.1 Fuzzy search

The fuzzy search algorithms or similar search algorithms are used for search engines to find the similarity between two strings by calculating the metric distance of the strings. In this stage the image hashes that were previously generated are inserted into a BK-Tree, search tree. This data structure is used to reduce the search time for a specific query compared to a linear search. A query image (represented by a hash) will be presented to the tree and a search will be performed according to the BK-Tree algorithm.

In this work we used three different distance measures for the search tree to obtain the fast and more accurate result as following:

- Levenshtein Distance
- Cosine Distance
- Jaccard Distance

A maximum distance will be specified in the search function call. The result of the search will be a list of elements of the tree that have a distance less than or equal to the specified distance. These candidate images will be processed by the fingerprints in the next phase of our method. The usage of metric distance depend on case of searching algorithm any of

these metric distance can be used, Jaccard similarity distance (P. Jaccard 1901), Hamming distance (R. W. Hamming April 1950) and cosine distance.

3.2.1.1 BK-Tree Search

Burkhard-Keller tree is a metric tree algorithm which construct a balanced tree based on metrics triangle inequality abilities (W. A. Burkhard 1973).Bk-tree is a metric fuzzy search tree data structure which utilizes the triangle inequality, a property of the Levenshtein distance. BK-tree fuzzy search is a lookup method for quickly finding near-matches to a string in this work the strings are the hashes which we obtained from images. Two hashes will take by Bk-tree and a list of similar hashes will returns by using any metric distance, the threshold is configurable in all metric distance used for BK-tree. The hashes comparison will be between the hashes from images stored in dataset which we extract them in offline stage and the hashes of query images which calculate in online stage.

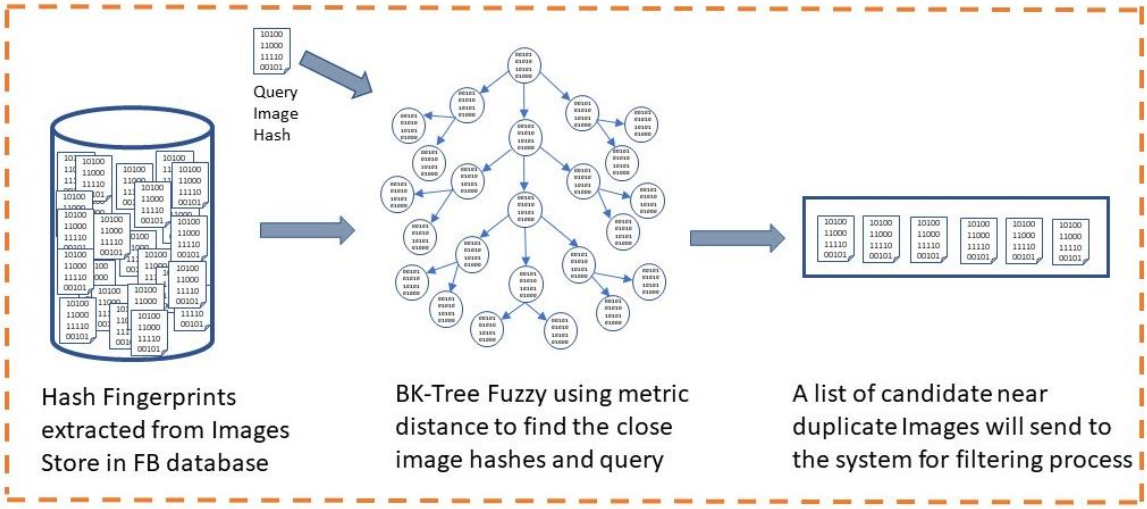


fig. 3.24 BK-Tree Fuzzy Search to find the near duplicate images

This part of the system is user-friendly and adjustable as well, the user can decide the distance similarity and set it up in the application. This is important to mention the number of thresholds play a great role in result precision. In this work we used three different metric distance. For Jaccard and Cosine the thresholds adjust between 0.0 to 1.0 and for Hamming is between 1 to 32. Choosing bigger number for threshold return a larger list of candidate images, we reported the result in experiment at chapter 4.

3.2.1.2 Query Image

Query Image is a request image from end user for retrieval process. The users enter and present the query image into the system retrieval then the system will extract its fingerprints and in online stage compare its relevant feature characteristics with other features extracted from dataset images. Our retrieval system has the capacity of receiving several query images at the same time. In the next chapter at 41.1.2 we explain about the query images sets we used for our experimental.

3.2.2 Levenshtein Distance

Most of the fuzzy search algorithms for calculating the metric distance commonly applied the Levenshtein distance which is a metric triangle inequality distance. (L. Yujian June 2007). The calculating process is through the number of operations which is needed to find the different distance of one string to other. which is representing the minimum number of insertions, deletions and replacements required to translate one hash into another (Gandhi 2017).

The fuzzy search used Levenshtein metric distance to find the measure of similarity between two strings as following:

$$d(i,j) \leq d(i.h) + d(h,j),$$

$$i,j,h \ \in X$$

Where $\boldsymbol{X}$ is a set of strings and $\boldsymbol{d}$ the metric distance between two pair of strings.

The Levenshtein distance between two has (strings) is represented as following:

$$Lev_{a,b(i,j)} = \begin{cases} \max(i,j) & if\ \min(i,j) = 0, \\ min\begin{cases} lev_{a,b}(i-1,j)+1 \\ leve_{a,b}(i,j-1)+1 \\ lev_{a,b}(i-1,j-1)+1(a_i,\neq b_j) \end{cases} & otherwise. \end{cases}$$

Where $\boldsymbol{a}$ and $\boldsymbol{b}$ are the length of the two string.

$\boldsymbol{i}$ is the terminal character position of string a

j is the terminal character position of string b

1*(ai≠bi)* is the indicator function equal to 0 when **ai≠bi** and equal to 1 otherwise, and leva, *b(i,j)* is the distance between the first *i* characters of *a* and the first *j* characters of *b*. To calculate the ratio of Levenshtein similarity base on Levenshtein we have:

$$\frac{(|a|+|b|) - leva,b(i,j)}{|a|+|b|}$$

The complexity of the algorithm is $\boldsymbol{o(mn)}$ and it consume the memory as $\boldsymbol{o(mn)}$, where $\boldsymbol{m}$ and $\boldsymbol{n}$ are the lengths of two compared strings. Hence Levenshtein algorithm is an efficient algorithm which is a good option in fuzzy search algorithms. (Hardesty 2015).

In this work the hash strings always have the same length.

3.2.3 Cosine Distance

Another similarity measure which we used to calculate the similarity distance between hashes is cosine similarity metric distance. As mentioned already, the image hash can be represented as a numeric vector (part of a vector space). A distance index can be derived from the angle between a pair of vectors in the space (the larger the angle, the larger the distance between both vectors). The cosine similarity between 2 vectors is defined as:

$$\text{sim}(\text{x}, \text{y}) = \frac{x.y}{||\text{x}||||\text{y}||'}$$

This formula represents the cosine of the angle between both vectors as a measure of similarity (Jiawei Han 2012). The cosine distance is defined as 1-cosine similarity (Singhal 2001), (Jhansi 2017). Two vectors with the same orientation have a cosine similarity of 1.

Given two vectors of $\boldsymbol{X}$ and $\boldsymbol{Y}$ we will have

$$similarity = \cos(\theta) = \frac{x \cdot y}{\|x\|\|y\|}$$

$$\frac{\sum_{i=1}^{n} x_i \, y_i}{\sqrt{\sum_{i=1}^{n} x_i^2} \; \sqrt{\sum_{i=1}^{n} y_i^2}}$$

Where $x_i \; and \; y_i$ are components of x and y vectors.

Cosine similarity mostly is used in positive space where the output is neatly bonded in [0,1]. As following:

$$D_c(x, y) = 1 - S_c(x, y) \tag{1}$$

Where D_c is the cosine distance and S_c is the cosine similarity.

3.2.4 Jaccard distance

Another metric distance we used to calculate the similarity measure between hashes is the Jaccard distance. Jaccard index J (A, B) is a measure of similarity between sets A and B (P. Jaccard 1901). Conversely, Jaccard distance is defined as 1 - J (A, B). This distance meets the conditions for a distance metric (Kosub 2019).

$$J(A,B) =_{def} \frac{|A \cap B|}{|A \cup B|}, \qquad J_\delta(A.B) =_{def} 1 - J(A,B) = 1 - \frac{|A \cap B|}{|A\ U\ B|} = \frac{|A\ \Delta\ B|}{|A \cup B|}$$

In purposes work, we take the digital representation of the image hash as a string of bits and count the bits that are turned on (set to 1) in each corresponding hash as the elements that exist in the respective sets, and then perform the calculation of the similarity index. The distance is calculated as 1 – similarity.

3.3 Phase-Three

3.3.2 Similarity Calculation for Fingerprints

Calculating the Similarity measurement is another challenge in CBIR where the query image similarity should be compared with images in dataset. This similarity metric calculates by finding the differences between query feature vector and the image feature vector. The small difference between two feature vectors demonstrate the large similarity. Hence the images with small amount of distance are the similar images to the query image.

Since we are using different fingerprints, so we used different similarity measure to calculate the similarity of query image with dataset images.

3.3.1 Matching system

In this phase we process each candidate image from the list which were discriminated by Hash from the second phase to discard either they are near-duplicate or not by comparing the similarity index of other five fingerprints one-by-one. As we organized in our method first, we do the comparison process by global fingerprints which are faster. Thus, the algorithm will load the five other fingerprints of each candidate image to match the similarity measure between them and the query image. Each fingerprint has its own similarity measure algorithm in order to detect the near-duplicate image. In experimental chapter we explain the order of applying each fingerprint comparison. So, if a candidate image passed all the filtering comparing process with each fingerprint a positive match will be reported to the detecting near-duplicate system it means that candidate image is a near-duplicate image which we have already a version of the image at our database.

Following figure demonstrate the matching process for detecting near-duplicate image.

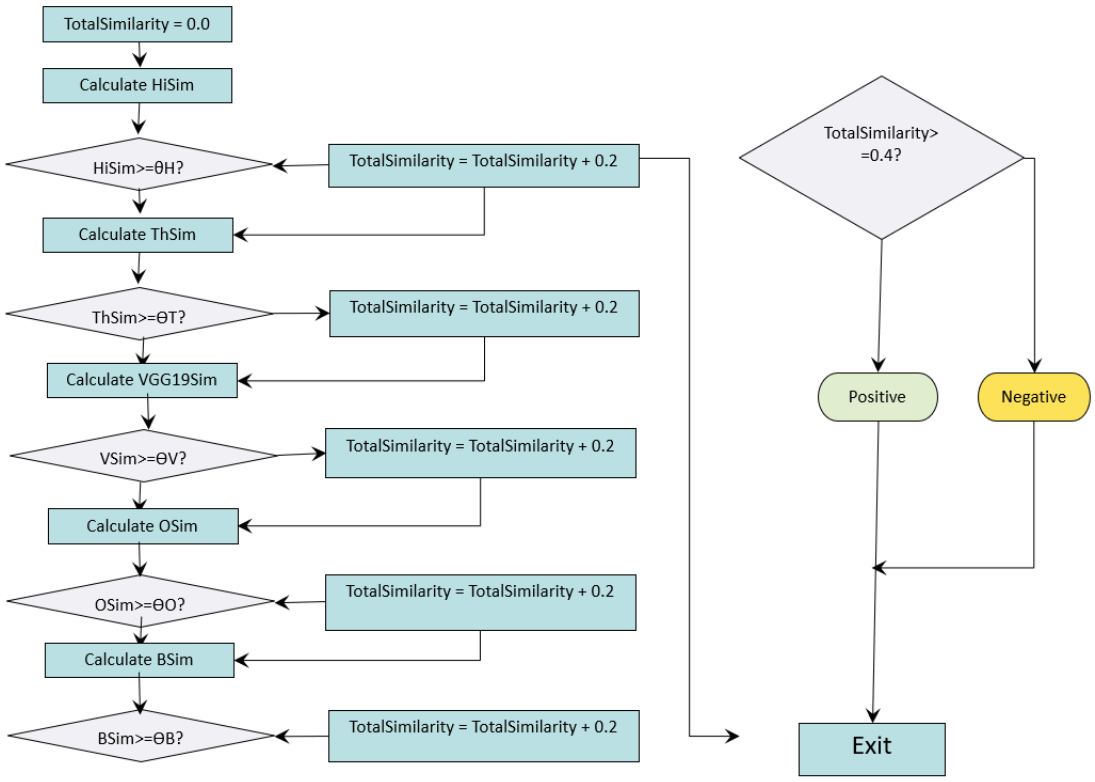


fig. 3.25 Matching process diagram with OR Method

3.3.2.1 Similarity measure of Histogram and Thumbnail

To calculate the similarity measure of these two fingerprints we used Correlation coefficient which is a numerical measure to define statistical relation between two variables, the variables could be from different datasets. The Correlation coefficients have a value range between -1 to +1 where ±1 indicates the strongest relationship and 0 is not relationship.

In proposed work we used Pearson Correlation coefficient for Histogram and Thumbnail as similarity measure to detect the near-duplicate features extract from images.

$$correlation_{xy} = \frac{\sum_{i=1}^{n}(x_{i-}\bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{(y_i - \bar{y})^2}}$$

Where ***n*** is total number of samples, ***x*** and ***y*** are sample points indexed by ***i.***

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

Is the sample mean. (Rummel 1976). To compare how similar is the relation between vector of histogram and thumbnail we used comp-correlation function of OpenCV library. (OpenCV.org 2020)

3.3.2.2 Similarity measure of ORB

In order to calculate a similarity measure between two images using their ORB features, we need to perform a feature matching process: The local features are calculated for each image and we apply an operation that reflects the number of matching features between both images. ORB can create a large number of features for a given image. For the purposes of efficiency, we will limit the number of features to the first 500 best features.

We will go through each of the 500 features in each image I_1, I_2 and will compare features one to one. For each pair of features the Hamming distance is calculated. Features calculated by ORB algorithm are integers and they can be represented in binary form. Every pair of features with a Hamming distance less than 16 will be added to a count. The similarity measure is the resulting count number.

Algorithm to calculate ORB similarity

Initialize

1: L1 = ORB_Features (I1)

2: L2 = ORB_Features (I2)

3: Count = 0

4: For F1 in L1

5: For F2 in L2

6: D = Hamming(F1, F2)

7: If (D < 16)

8: Count = Count + 1

9: Return Count

End

3.3.2.3 Similarity measure of BOVW

For an image I, the BOVW fingerprint of that image is defined as $H_w(I)$ where H_w is the histogram for the visual words contained in I.

Since the fingerprint is a Histogram, we select the Histogram intersection as similarity measure for a pair of images (I_1, I_2):

$$sim_h(I_1, I_2) = \frac{\sum_w min(H_w(I_1), H_w(I_2))}{\sum_w max\big(H_w(I_1), H_w(I_2)\big)}$$

For this work we used the method which was present by (H. Y. Ling, Fast image copy detection approach based on local fingerprint defined visual words 2013).

3.3.2.4 Similarity measure of Vgg19

This pre-trained model uses the Cosine similarity to calculate the similarity measure between the fingerprints. Using the following formula to calculate similarity.

$$similarity = \cos(\theta) = \frac{x \cdot y}{\|x\|\|y\|} =$$

Where ***X, Y*** are feature vectors for each image.

3.3.3 Voting Process

Sequential method and OR method

To match each query image against each candidate image we used to different approaches in order to find the result which is if the image is a near duplicate image exist in the database or not. later in experimental results we explain about which of these two approaches were more effective in of near duplicate detection system. We test and validate these two methods with different value of threshold for both Sequential and OR methods.

Sequential method: Compares the fingerprints of the query image with all fingerprints for the candidate and produces a negative match if the similarity index for one of the fingerprints is below the value of the threshold.

Algorithm Sequential Method

Initialize

1: if HistogramSimilarity < HistogramThreshold

2: Match = Negative

3: return Match

4: if ThumbnailMatch < Thumbnail Threshold

5: Match = Negative

6: return Match

7: if ORBMatch < ORBThreshold

8: Match = Negative

9: return Match

10: if BOWMatch < BOWThreshold

11: Match = Negative

12: return Match

13: if VGG19Match < VGG19Threshold

14: Match = Negative

15: return Match

15: Match = Positive

16: return Match

End

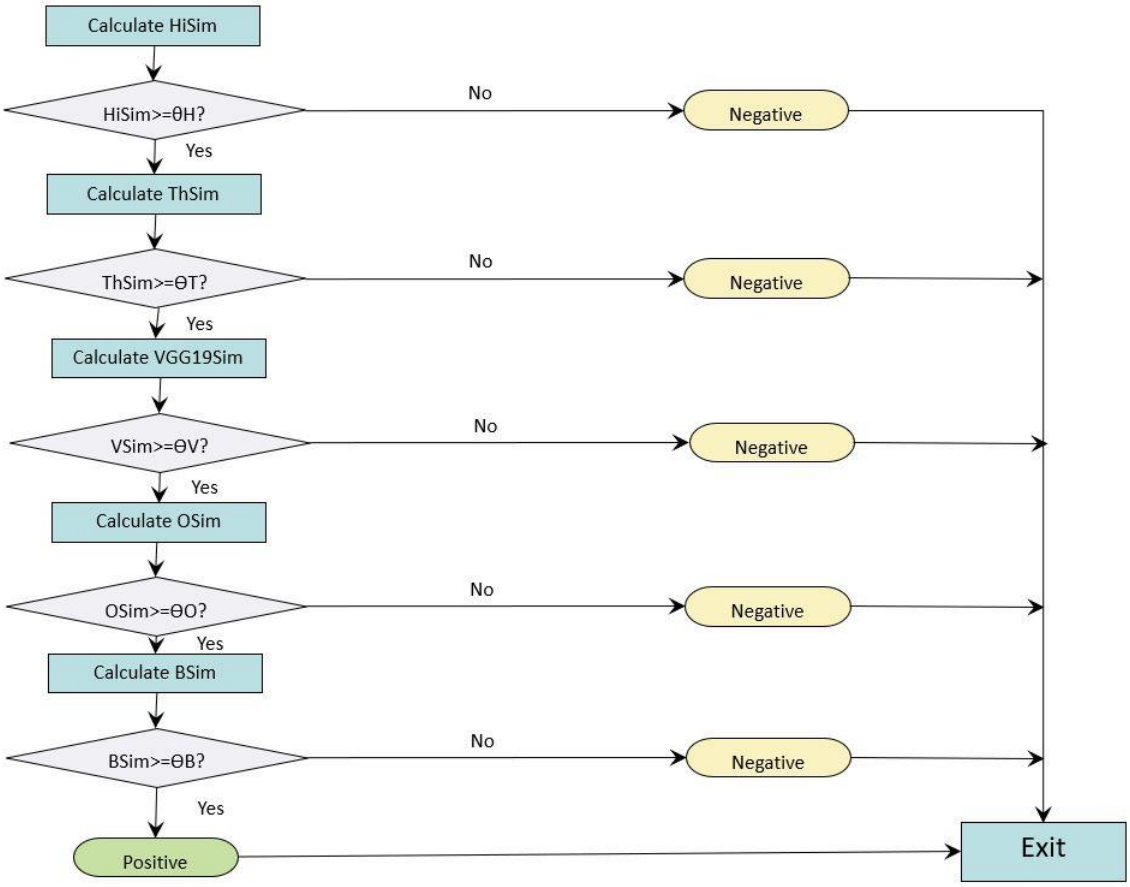


fig. 3.26 Matching process diagram with Sequential Method

OR method: Compares the fingerprints but will produce a positive match if at least 2 of the similarity indexes for the fingerprints are equal or above threshold.

Algorithm OR Method

Initialize

1: TotalSimilarity = 0

2: if HistogramSimilarity >= HistogramThreshold

3: TotalSimilarity = TotalSimilarity + 0.2

4: if ThumbnailMatch >= Thumbnail Threshold

5: TotalSimilarity = TotalSimilarity + 0.2

6: if VGG19Match >= VGG19Threshold

7: TotalSimilarity = TotalSimilarity + 0.2

8: if ORBMatch >= ORBThreshold

9: TotalSimilarity = TotalSimilarity + 0.2

10: if BOWMatch >= BOWThreshold

11: TotalSimilarity = TotalSimilarity + 0.2

12: if TotalSimilarityIndex >= 0.4

13: Match = Positive

14: else:

15: Match = Negative

End

Chapter 4.

The Experimental Results

In this chapter we first introduce the datasets used for proposed Near-duplicate detection system, applying our algorithms, making different experiment process in order to obtain results. Then we describe experimental settings, report results, their analysis and finally we compared our method to only deep learning approach and discussed about each method benefits.

4.1 Data Collection

4.1.1 The Images Databases

For proposed work we created an uncategorized dataset by collecting 200,000 images taking by authors called as Native Database including images with different resolution, size and formats, this database is scalable, user can add more image to this dataset. Since the focus of this work is on Content Based Image Retrieval to detect Near-duplicate images, for testing the process another image dataset is needed as well called as Query image dataset. Hence 3 image databases were created for the proposed work:

- Native Image Images Data
- Query Images Data Base
- Fingerprint Data Base

4.1.1.1 Native Data Base

This Data Base consists of 200,000 digital images in different image formats from personal collections. With the following composition:

- **Attack images:** Include 90,000 image with different image digital attacks, this concept will be discussed in the section below by details.
- **Non-attack images:** Include 110,000 images which are original and don't have any type of image attack but with different resolutions and qualities.

4.1.1.2 Query Images Data Base

To examine the functionally of proposed Near-duplicate image detection system and obtain results, we needed a predefine image dataset containing with 10,000 images, 50% of images from Native Database and 50% images which are not inside the Native Database. The query image will send to system to find the near-duplicate images, from the result we calculate precision and recall. The amount of query image for sending to the system is define by user and does not have limitation. The user can choose the query image manually or ask the system to send a specific amount of query images randomly. Query images sent to system on the fly.

This Data base consists following composition:

- **Internal images:** include 5000 images which are guaranteed to have existing near duplicates in the Native Data Base, and they are images with we already applied image attacks on them.

- **Reference list:** For each internal image a data base entry is stored on disk. This entry contains a list of near duplicate images for those images which are already stored in Native Database as near duplicate image and we the list of them to identify later. We used this reference list later in validation process to find false positive, false negative, true positive and true negative. To create the reference list, we applied the Pickle Module which is python library for Serialization.

- **External images:** include 5000 images which are guaranteed not to have near duplicates in the Native Database.

4.1.1.3 Fingerprint database

The fingerprint database call as FDB, contain 1,200,000 fingerprints obtain from the Native Image Database. We setup this stage as offline and its one-time process, if we need to add more Images to the Native Database, we just need extract new images six fingerprints and add to the FDB dataset. In this database also we used serialization method to obtain an effective data store.

4.2 Database settings

The aim of proposed work is detecting Near-duplicate images to have the precise result and find either our system can detect near-duplicate images or not we need to create a large set of near-duplicated images from the original images in Native Database. Hence, we performed an algorithm to apply different image attacks including rotation, nosing, color

modification, compression, flipping, blurring, cropping, etc. The images with these attacks are 45% of our Native Database images.

4.2.1 The attack images

In order to exercise the algorithms, we must guarantee that a significant amount of our query images will have corresponding near-duplicate images inside the Native Data Base. Hence 5000 images were taken from the Query Images Data Base, and through image processing algorithms for applying different type of image attacks on these 5000 images. A set of 18 near-duplicate image attack were applied for each image so we reached to 90,000 attack images from original images and added to the Native Data Base.

The following table shown image distribution on Native image database:

Amount	Database
200000	Native Database
90000	attack images
110000	Non-attack images

Table. 4.1 Native Database specification

The following table shown image distribution on Query image database:

Amount	Database
10000	Query image Database
5000	Internal images
5000	External images

Table. 4.2 Query Database specification

The attack images were generated using the following transformations methods for Native Data base:

Type of attack	Description	Percentage
Sanity Test	An exact copy of the original image	Santy check
Compression attack 1	Drop the JPEG quality of the original image	30%
Compression attack 2	Drop the JPEG quality of the original image	40%
Compression attack 3	Drop the JPEG quality of the original image	50%
Blurring attack 1	Blur the original image using a Gaussian filter	%10
Blurring attack 2	Original image through a smooth filter	45%
Cropping attack 1	Cropping the original image	3%
Cropping attack 2	Cropping the original image	5%
Cropping attack 3	Cropping of the original image	10%
Rescaling attack 1	Rescaling the original image size	90%
Rescaling attack 2	Rescaling the original image size	95%
Colorization attack 1	Adjust the color balance of the original image	90%
Colorization attack 2	Adjust the color balance of the original image	95%
Colorization attack 3	Posterize (reduce 1 bit from each color channel) the original image.	-
Rotation attack 1	Rotate the original image 1 degree clockwise	-
Rotation attack 2	Rotate the original image 1-degree counter-clockwise	-
Flipping attack 1	Flip the original image horizontally	%100
Flipping attack 2	Flip the original image vertically.	%100

Table. 4.3 Image attacks specification

As shown in below figure, attacks apply on images can do very small changes which sometimes is not possible to distinguish them to the unaided eye. These types of attacks mostly are used in forge images hence finding the original image will be very difficult for users.

fig. 4.3 Image with different attacks which we used for proposed work

To examine our algorithm results we created some exact copy of images randomly in Native database as well in query image database.

4.3 Evaluation protocol

4.3.1 Experimental setup

- **Hardware**

Experiments of the proposed work have been performed in:

PC Intel i7 processor at 1.8 GHz and 16 GB Memory RAM.

Hard disk 2 T, for collecting image databases and fingerprint database.

OS, Windows 10 Home Single Language

- **Software**

For writing and employing our algorithm we used:

Python 3.75 (64bit)

Open Source Computer Vision Library (OpenCV) 3.4.2.17

NumPy Scientific Computing Library 1.18.1

TensorFlow Open Source Machine Learning Platform 2.1.0

Keras Python Deep Learning Library 2.3.1

Scipy Scientific Computing Library 1.4.1

Pillow Image Processing L8ibrary 7.0.0

Pandas Python Data Analysis Library 0.25.3

4.3.2 Execution the algorithm

Our algorithm starts with generating first fingerprint, the Hash then a BK-Tree is built as a search data structure. Each node of the Tree is formed by 2 components: The Hash and the Image Id.

For the BK-Tree to be valid, the distance measure between its components must be a metric distance. Three different metric distances have been evaluated separately:

- Hamming distance
- Cosine distance
- Jaccard distance

On each experiment, a set of random images from the Query Images Database was selected between 1 to 100 query images. Each query image hash is provided as input (along with a maximum distance) to the BK-Tree, and the search produces a list of candidate images, with a hash distance less than or equal to the maximum distance specified by user.

After generation list of candidate images which are suspected to be near duplicate, the list is traversed sequentially for each candidate image, all fingerprint algorithms will be applied one by one in order to discard or verify the near duplicate image from the candidate image list. The early image candidate hash list store in the memory as well as all the comparison process with other fingerprints on the fly.

The algorithm requires a threshold called as ***Th.*** The threshold defined for each type of fingerprint with different number of thresholds. This amount determines the similarity between the original image and the query image.

We execute the algorithm by each fingerprint separately with different value for threshold of three metric distance used in this work, this process was to get the best metric distance and the best threshold value. The execution was running in both Sequential and OR method.

Then we execute the algorithm with different combination of fingerprints and record the result in order to get the best combination. Later we ran the algorithm with and without deep learning method, record and time of execution and the accuracy. Finally, when we got the best results, we scaled-up the number of query images to observe the algorithm behavior in larger searches.

Since deep learning techniques are methods which are becoming more and more widely used to solve Image retrieval problems so, we implement another approach based purely on deep learning method for detecting near duplicate images which is a pre-trained model call as ResNet (He 2016) with KNN (nearest neighbor) (Keller 1985) combination on this mode to detect near duplicate images in our large database. Experiments and results are shown in the 4.4 Result and Analysis section.

4.3.2.1 Performance Validation

To test the performance of our algorithm we created a references list which contains a list of the names of different attack images which are store in Query database. we Serialized the list of names of image attacks in order to compare them with the result and confirm if the output result is a correct response or not. This process is a qualification and validation process which provides us the four decision elements:

False positive, False Negative, True positive, True Negative.

Figure x shows the serialized list for each query image

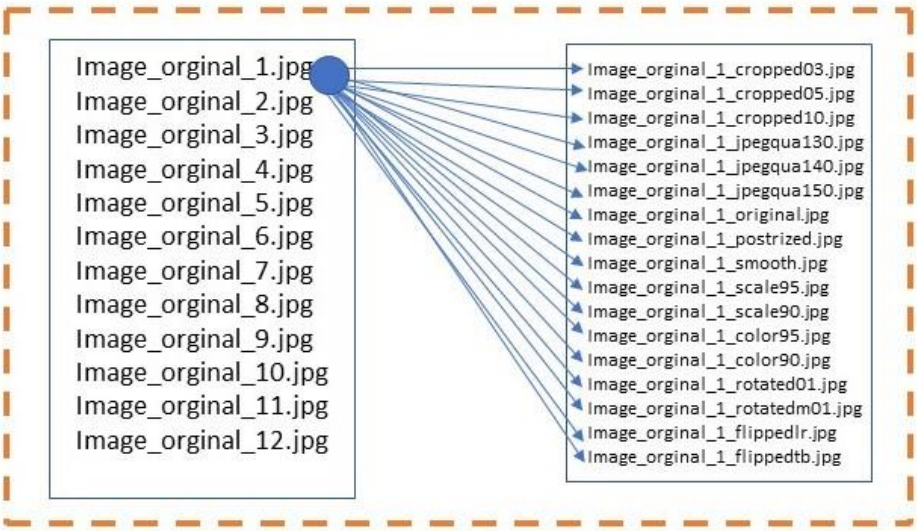


fig. 4.1 Serialized the reference list

4.3.3 Recall, Precision and F-Score

To calculate these measures, we first define the following concepts:

- **True positive:** The images that algorithm detects as Near-duplicate and its real near-duplicate images
- **False positive:** The images that algorithm detects as Near-duplicate but it's Not near-duplicate images
- **True negative:** The images that algorithm detects as Non-Near-duplicate and it's Not near-duplicate images
- **False negative:** The images that algorithm detects as Non-Near-duplicate but it's near-duplicate images

Precision

The precision is measurement of the ratio between the true positive to the sum of true positive plus false positive as following:

$$precision = \frac{TP}{TP + FP}$$

Recall

Recall is the measurement of the ratio between true negative to the sum of true negative plus false positive as following:

$$recall = \frac{TP}{TP + FN}$$

For example, in one of our experiments, for a query image we retrieve totally 10 images with 8 relevant images out of totally 30 relevant images in database. So, the precision was 8/10 = 80% and recall was 8/30 = 27%. Thus, this shows that only recall cannot measure the effectiveness of the CBIR system, precision must also be computed as well to obtain the correct result.

F-Score

The precision and recall measure the accuracy of near-duplicate images with relevancy to the query image and Native Database images and these two values are computed to show the effectiveness of Near-duplicate image detecting system. But these to measure cannot be consider and complete accuracy for the system effectiveness. So, to calculate the final

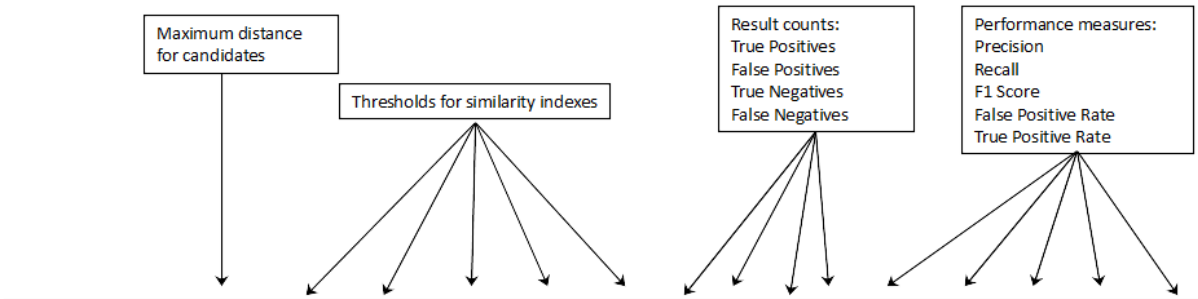
accuracy of our algorithm we combined these two values and obtained a final accuracy measure which is called as F-Score defined as following:

$$F - Score = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

The F-score result is a single value that indicates the overall effectiveness of the our Near-duplicate image detection system. We executed and examined our algorithm with a different threshold via 3 different metric distance (Hamming, Jaccard and Cosine) to find the best result accuracy comparing to the time factor. All these results we stored and in excel database using Panadas library for analyzing data. The data from algorithm converted to a data frame useful for analyzing via panda and return an excel output. We also calculate the Accuracy value after detecting the best result from three metric distance and compared this value with the time of execution.

4.3.3.1 Parameters and Performance

To analyze the performance of the method, the experiment was carried out with different parameter values and the positive predictive value (precision) and the sensitivity (recall) were calculated after each execution. Following Figure show as example of how to read the algorithm results:



Method	Metric	Max.Dist	Hi.Thr	Th.Thr	V.Th	ORB.Thr	BOW.Thr	TP	FP	TN	FN	PR	RE	F1	FPR	TPR
OR	Cosine	0.025	0.85	0.85	0.4	150	0.1	53	0	21	19	1	0.73611	0.59551	0	0.73611
OR	Hamming	16	0.85	0.85	0.4	150	0.1	63	0	18	9	1	0.875	0.63636	0	0.875
OR	Jaccard	0.15	0.85	0.85	0.4	150	0.1	64	0	4	8	1	0.88889	0.64	0	0.88889
Sequential	Cosine	0.025	0.85	0.85	0.4	150	0.1	38	0	21	34	1	0.52778	0.51351	0	0.52778
Sequential	Hamming	16	0.85	0.85	0.4	150	0.1	44	0	18	28	1	0.61111	0.55	0	0.61111
Sequential	Jaccard	0.15	0.85	0.85	0.4	150	0.1	44	0	4	28	1	0.61111	0.55	0	0.61111

fig. 4.2 Performance measurements parameters.

As shown in this image we used two different type of methods for algorithm execution as Sequential method and OR method in following section will explain each in detail.

Maximum distance and BK-Tree

Each metric distance can return different number depend on metric distance algorithm for example the return number of Hamming distances is an integer number between 0 to n where the n is the number of hash bits.

- Metric Stands for the three metric distance Hamming, Jaccard, Cosine.
- Hi.Thr, Th.Thr, V.Thr ,ORB.Thr, BOW.Thr are the threshold value of fingerprints.
- TP(true positive),FP(False Positive), TN(True Negative) and TP is for True positive.
- PR is precision value
- RE is for recall value

- F1 is F-score value
- FPR is false positive rate
- TPR is true positive rate

4.3.3.2 Sequential vs OR method results

Sequential method: Compares the fingerprints of the query image with all fingerprints for the candidate and produces a negative match if the similarity index for one of the fingerprints is below the value of the threshold.

OR method: Compares the fingerprints but will produce a positive match if at least 2 of the similarity indexes for the fingerprints are equal or above threshold.

The sequential method only produces a positive match if all the fingerprints produce a similarity index equal or above threshold. This makes it much more restrictive, so in order to see if it presents a significant difference from the OR method, we execute the algorithm with some initial values for the thresholds and analyze the recall measurement of the results: A more restrictive algorithm would produce lower recall values. As shown in the below table:

Method	Metric	Max.Dist	Hi.Thr	Th.Thr	V.Th	ORB.Thr	BOW.Thr	TP	FP	TN	FN	PR	RE	F1	FPR	TPR
OR	Hamming	16	0.8	0.8	0.6	150	0.1	570	1	1710	78	0.99824869	0.8796296	0.63758	0.00058	0.87963
Sequential	Hamming	16	0.8	0.8	0.4	150	0.1	371	0	1711	277	1	0.5725309	0.53381	0	0.57253
OR	Cosine	0.025	0.72	0.73	0.6	150	0.1	520	1	2340	128	0.99808061	0.8024691	0.61611	0.00043	0.80247
Sequential	Cosine	0.025	0.72	0.72	0.4	150	0.1	365	0	2341	283	1	0.5632716	0.52975	0	0.56327
OR	Jaccard	0.12	0.79	0.79	0.4	150	0.1	565	10	1178	83	0.9826087	0.8719136	0.63555	0.00842	0.87191
Sequential	Jaccard	0.12	0.79	0.79	0.4	150	0.1	371	0	1188	277	1	0.5725309	0.53381	0	0.57253

Table. 4. 4 comparison of recall, precision and F1 score values for sequential and OR

Hence by this experimental result we got to conclusion that its preferable to use OR method using Hamming metric distance this combination in general produces better Recall values. The details will be explained in the next section.

4.4 Results Analysis

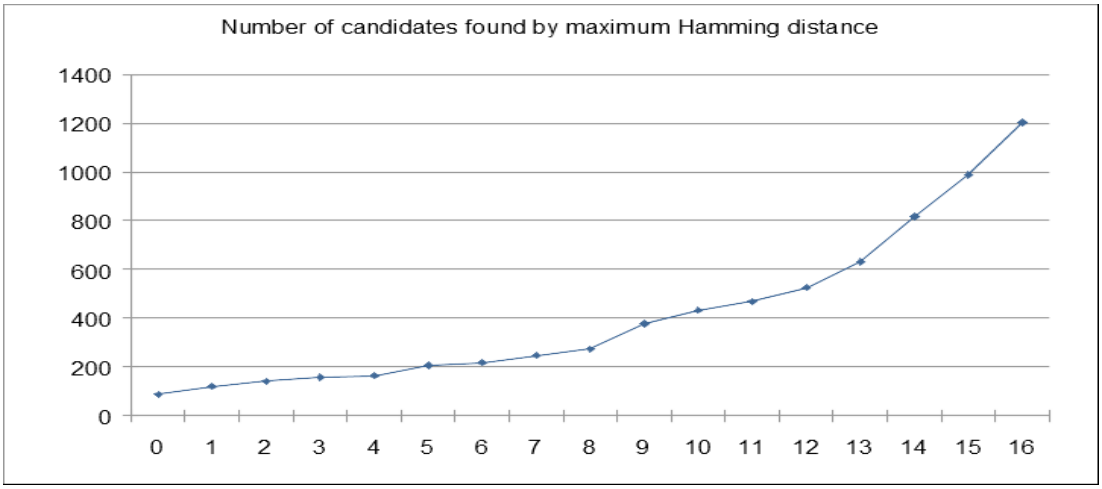
4.4.1 Fuzzy Search maximum distances BK-Tree

The search algorithm for the BK-Tree receives 2 parameters as input: A query hash and a distance. The distance parameter affects the number of candidates returned by BK-Tree. A small distance will produce less candidates and is more restrictive. The number of candidates increases as the maximum distance value is larger.

The distance parameter for fuzzy search must be selected in a way that provide a wide range of near-duplicate images, but also not too permissive, since this would increase the number of compare operations while doing the search and would slow down the algorithm. Hence finding a suitable distance for fuzzy search tree in crucial in algorithm performance. In our experimental we ran algorithm with many different distance to find the best value which provided an acceptable result of a candidate image list. Also, we performed the selecting distance for all three-distance metrics under study: Hamming, Cosine and Jaccard.

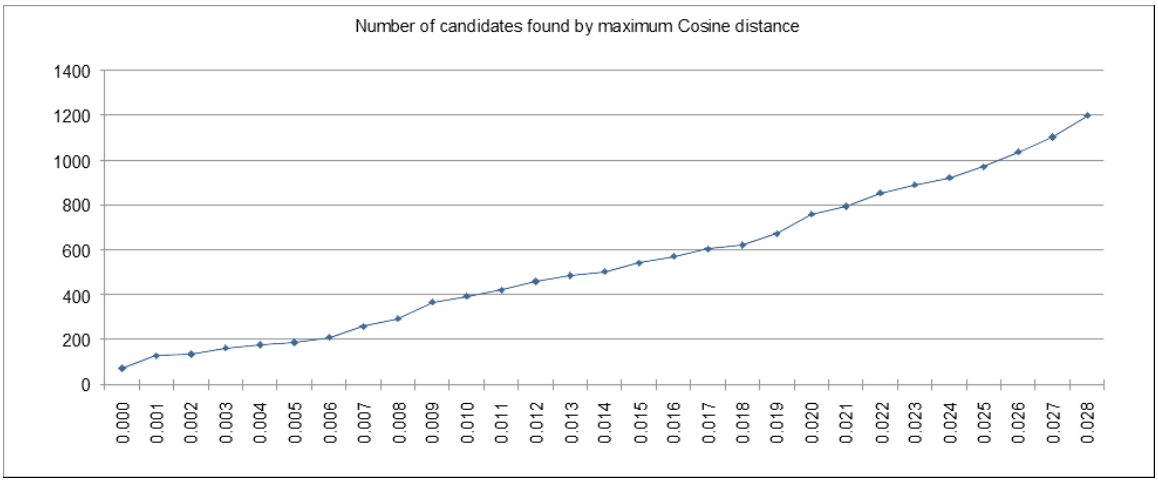
Considering first the case of Hamming distance, we will see how the total number of candidate images increases as the distance gets larger. The following runs were performed

for a set of 1000 random images from the Query Images DB. For Hamming distance, the number of candidates retrieved by maximum value is shown below graph:



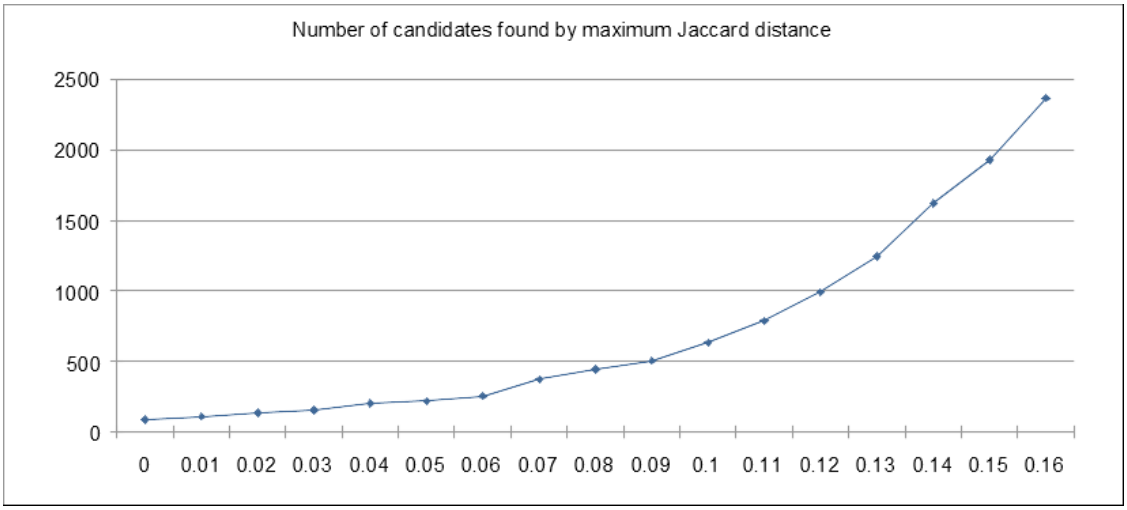
Graph.4.1 shows an approximation of number of the candidate images that are return by the fuzzy search for each hamming distance

The graph shows that the number of candidates starts increasing more rapidly around a Hamming distance of 14-16 bits. So, in this work we will consider a maximum Hamming distance of 16 bits in order not to slow down the search process. The same measures are taken for the Cosine distance:



Graph.4.2 shows an approximation of number of the candidate images that are return by the fuzzy search for each Cosine distance

The number of candidates found grows more slowly compared to the Hamming distance. In order to obtain a number of candidates comparable to Hamming distance, we will set the maximum Cosine distance to 0.028. For the Jaccard distance, the number of candidates is shown below graph:



Graph.4.3 shows an approximation of number of the candidate images that are return by the fuzzy search for each Jaccard distance

The number of candidates starts growing more rapidly compared to Hamming and Jaccard. We will set the maximum Jaccard distance to 0.12.

Hence in this way we could find the best values for each metric distance which returned approximately the same number of candidate images. When we chose large values for each metica distance the candidate number growth exponentially.

4.4.2 Threshold values for the different fingerprints

To provide robustness to the method, the similarity level for a candidate image is calculated based on a set of different fingerprints. The main reason for using several fingerprints at once is that each one of them is resistant to certain image attacks while being vulnerable to other attacks. For example: The histogram fingerprint is resistant to attacks that leave the color distribution of the image untouched, but at the same time is vulnerable to colorization attacks. The thumbnail fingerprint is resistant to colorization attacks, but it can be vulnerable to rotation attacks.

Since the final classification of the algorithm (a positive or negative match) is based on a combination of the match result for all the different fingerprints, adequate thresholds must be selected for all of them.

The match algorithm is described as follows. Histogram Similarity is the similarity index of the histogram for the query image and the histogram of the candidate image. The rest of the fingerprints also have their similarity indexes calculated. Total Similarity is the calculated result for the combinations of similarities for all fingerprints. We are using a threshold of 0.40 (at least two of the fingerprints result in a positive match) to report the final match result (Positive or Negative).

4.4.3 Total Similarity Index

The Total Similarity Index is composed by all the individual similarity values for each fingerprint, we chose 0.2 as value of contribution which determines if the fingerprint has a positive result, assuming all five fingerprints get the positive result so number 1 should be return as verifying that they query image has maximum rank of similarity with the chosen

image. Our tests showed that the best results are obtained when the Total Similarity Threshold in matching system is set at 0.4. This means that two or more fingerprints have produced positive matches. Since the OR method provided the best result, we selected above values just for OR method as final value for the system. The sequential method was just for experimental and comparison process to choose either one of them.

We experimented with several combinations of values. Since the combinations space for the thresholds of the fingerprints is a 5-dimensional space, we will not include all the tested combinations in this document in appendix there are the most significant table of experimental results. Following table shows several combinations of similarity measure and different threshold values for Sequential and OR Methods.

Method	Metric	Max.Dist	Hi.Thr	Th.Thr	V.Th	ORB.Thr	BOW.Thr	TP	FP	TN	FN	PR	RE	F1	FPR	TPR
Sequential	**Hamming**	**16**	**0.8**	**0.8**	**0.4**	**150**	**0.1**	**371**	**0**	**1711**	**277**	**1**	**0.572531**	**0.533813**	**0**	**0.572531**
Sequential	**Hamming**	**16**	**0.8**	**0.8**	**0.5**	**150**	**0.1**	**371**	**0**	**1711**	**277**	**1**	**0.572531**	**0.533813**	**0**	**0.572531**
Sequential	**Hamming**	**16**	**0.8**	**0.8**	**0.6**	**150**	**0.1**	**371**	**0**	**1711**	**277**	**1**	**0.572531**	**0.533813**	**0**	**0.572531**
Sequential	Hamming	16	0.8	0.81	0.4	150	0.1	369	0	1711	279	1	0.569444	0.532468	0	0.569444
Sequential	Hamming	16	0.8	0.81	0.5	150	0.1	369	0	1711	279	1	0.569444	0.532468	0	0.569444
Sequential	Hamming	16	0.8	0.81	0.6	150	0.1	369	0	1711	279	1	0.569444	0.532468	0	0.569444
Sequential	**Hamming**	**16**	**0.81**	**0.8**	**0.4**	**150**	**0.1**	**371**	**0**	**1711**	**277**	**1**	**0.572531**	**0.533813**	**0**	**0.572531**
Sequential	**Hamming**	**16**	**0.81**	**0.8**	**0.5**	**150**	**0.1**	**371**	**0**	**1711**	**277**	**1**	**0.572531**	**0.533813**	**0**	**0.572531**
Sequential	**Hamming**	**16**	**0.81**	**0.8**	**0.6**	**150**	**0.1**	**371**	**0**	**1711**	**277**	**1**	**0.572531**	**0.533813**	**0**	**0.572531**
Sequential	Hamming	16	0.81	0.81	0.4	150	0.1	369	0	1711	279	1	0.569444	0.532468	0	0.569444
Sequential	Hamming	16	0.81	0.81	0.5	150	0.1	369	0	1711	279	1	0.569444	0.532468	0	0.569444
Sequential	Hamming	16	0.81	0.81	0.6	150	0.1	369	0	1711	279	1	0.569444	0.532468	0	0.569444

Table. 4. 5 several combinations of Hamming with different threshold in sequential

Method	Metric	Max.Dist	Hi.Thr	Th.Thr	V.Th	ORB.Thr	BOW.Thr	TP	FP	TN	FN	PR	RE	F1	FPR	TPR
OR	Hamming	16	0.8	0.8	0.4	150	0.1	570	15	1696	78	0.97	0.87963	0.637584	0.01	0.87963
OR	Hamming	16	0.8	0.8	0.5	150	0.1	570	4	1707	78	0.99	0.87963	0.637584	0	0.87963
OR	**Hamming**	**16**	**0.8**	**0.8**	**0.6**	**150**	**0.1**	**570**	**1**	**1710**	**78**	**1**	**0.87963**	**0.637584**	**0**	**0.87963**
OR	Hamming	16	0.8	0.81	0.4	150	0.1	570	10	1701	78	0.98	0.87963	0.637584	0.01	0.87963
OR	Hamming	16	0.8	0.81	0.5	150	0.1	570	2	1709	78	1	0.87963	0.637584	0	0.87963
OR	Hamming	16	0.8	0.81	0.6	150	0.1	570	0	1711	78	1	0.87963	0.637584	0	0.87963
OR	Hamming	16	0.81	0.8	0.4	150	0.1	570	15	1696	78	0.97	0.87963	0.637584	0.01	0.87963
OR	Hamming	16	0.81	0.8	0.5	150	0.1	570	4	1707	78	0.99	0.87963	0.637584	0	0.87963
OR	**Hamming**	**16**	**0.81**	**0.8**	**0.6**	**150**	**0.1**	**570**	**1**	**1710**	**78**	**1**	**0.87963**	**0.637584**	**0**	**0.87963**
OR	Hamming	16	0.81	0.81	0.4	150	0.1	570	10	1701	78	0.98	0.87963	0.637584	0.01	0.87963
OR	Hamming	16	0.81	0.81	0.5	150	0.1	570	2	1709	78	1	0.87963	0.637584	0	0.87963
OR	Hamming	16	0.81	0.81	0.6	150	0.1	570	0	1711	78	1	0.87963	0.637584	0	0.87963

Table. 4. 6 several combinations of Hamming distance with different threshold in OR

Method	Metric	Max.Dist	Hi.Thr	Th.Thr	V.Th	ORB.Thr	BOW.Thr	TP	FP	TN	FN	PR	RE	F1	FPR	TPR
Sequential	**Cosine**	**0.025**	**0.72**	**0.72**	**0.4**	**150**	**0.1**	**365**	**0**	**2341**	**283**	**1.000000**	**0.563272**	**0.529753**	**0.000000**	**0.563272**
Sequential	**Cosine**	**0.025**	**0.72**	**0.72**	**0.5**	**150**	**0.1**	**365**	**0**	**2341**	**283**	**1.000000**	**0.563272**	**0.529753**	**0.000000**	**0.563272**
Sequential	**Cosine**	**0.025**	**0.72**	**0.72**	**0.6**	**150**	**0.1**	**365**	**0**	**2341**	**283**	**1.000000**	**0.563272**	**0.529753**	**0.000000**	**0.563272**
Sequential	**Cosine**	**0.025**	**0.72**	**0.73**	**0.4**	**150**	**0.1**	**365**	**0**	**2341**	**283**	**1.000000**	**0.563272**	**0.529753**	**0.000000**	**0.563272**
Sequential	Cosine	0.025	0.72	0.73	0.5	150	0.1	365	0	2341	283	1.000000	0.563272	0.529753	0.000000	0.563272
Sequential	Cosine	0.025	0.72	0.73	0.6	150	0.1	365	0	2341	283	1.000000	0.563272	0.529753	0.000000	0.563272
Sequential	Cosine	0.025	0.72	0.74	0.4	150	0.1	361	0	2341	287	1.000000	0.557099	0.527007	0.000000	0.557099
Sequential	Cosine	0.025	0.72	0.74	0.5	150	0.1	361	0	2341	287	1.000000	0.557099	0.527007	0.000000	0.557099
Sequential	Cosine	0.025	0.72	0.74	0.6	150	0.1	361	0	2341	287	1.000000	0.557099	0.527007	0.000000	0.557099
Sequential	Cosine	0.025	0.73	0.72	0.4	150	0.1	365	0	2341	283	1.000000	0.563272	0.529753	0.000000	0.563272
Sequential	Cosine	0.025	0.73	0.72	0.5	150	0.1	365	0	2341	283	1.000000	0.563272	0.529753	0.000000	0.563272
Sequential	Cosine	0.025	0.73	0.72	0.6	150	0.1	365	0	2341	283	1.000000	0.563272	0.529753	0.000000	0.563272
Sequential	Cosine	0.025	0.73	0.73	0.4	150	0.1	365	0	2341	283	1.000000	0.563272	0.529753	0.000000	0.563272
Sequential	Cosine	0.025	0.73	0.73	0.5	150	0.1	365	0	2341	283	1.000000	0.563272	0.529753	0.000000	0.563272
Sequential	Cosine	0.025	0.73	0.73	0.6	150	0.1	365	0	2341	283	1.000000	0.563272	0.529753	0.000000	0.563272
Sequential	Cosine	0.025	0.73	0.74	0.4	150	0.1	361	0	2341	287	1.000000	0.557099	0.527007	0.000000	0.557099
Sequential	Cosine	0.025	0.73	0.74	0.5	150	0.1	361	0	2341	287	1.000000	0.557099	0.527007	0.000000	0.557099
Sequential	Cosine	0.025	0.73	0.74	0.6	150	0.1	361	0	2341	287	1.000000	0.557099	0.527007	0.000000	0.557099
Sequential	Cosine	0.025	0.74	0.72	0.4	150	0.1	365	0	2341	283	1.000000	0.563272	0.529753	0.000000	0.563272
Sequential	Cosine	0.025	0.74	0.72	0.5	150	0.1	365	0	2341	283	1.000000	0.563272	0.529753	0.000000	0.563272
Sequential	Cosine	0.025	0.74	0.72	0.6	150	0.1	365	0	2341	283	1.000000	0.563272	0.529753	0.000000	0.563272
Sequential	Cosine	0.025	0.74	0.73	0.4	150	0.1	365	0	2341	283	1.000000	0.563272	0.529753	0.000000	0.563272
Sequential	Cosine	0.025	0.74	0.73	0.5	150	0.1	365	0	2341	283	1.000000	0.563272	0.529753	0.000000	0.563272
Sequential	Cosine	0.025	0.74	0.73	0.6	150	0.1	365	0	2341	283	1.000000	0.563272	0.529753	0.000000	0.563272
Sequential	Cosine	0.025	0.74	0.74	0.4	150	0.1	361	0	2341	287	1.000000	0.557099	0.527007	0.000000	0.557099
Sequential	Cosine	0.025	0.74	0.74	0.5	150	0.1	361	0	2341	287	1.000000	0.557099	0.527007	0.000000	0.557099
Sequential	Cosine	0.025	0.74	0.74	0.6	150	0.1	361	0	2341	287	1.000000	0.557099	0.527007	0.000000	0.557099

Table. 4. 7 several combinations of Cosine with different threshold in sequential

Method	Metric	Max.Dist	Hi.Thr	Th.Thr	V.Th	ORB.Thr	BOW.Thr	TP	FP	TN	FN	PR	RE	F1	FPR	TPR
OR	Cosine	0.025	0.72	0.72	0.4	150	0.1	520	55	2286	128	0.904348	0.802469	0.616114	0.023494	0.802469
OR	Cosine	0.025	0.72	0.72	0.5	150	0.1	520	19	2322	128	0.964750	0.802469	0.616114	0.008116	0.802469
OR	Cosine	0.025	0.72	0.72	0.6	150	0.1	520	2	2339	128	0.996169	0.802469	0.616114	0.000854	0.802469
OR	Cosine	0.025	0.72	0.73	0.4	150	0.1	520	53	2288	128	0.907504	0.802469	0.616114	0.022640	0.802469
OR	Cosine	0.025	0.72	0.73	0.5	150	0.1	520	18	2323	128	0.966543	0.802469	0.616114	0.007689	0.802469
OR	**Cosine**	**0.025**	**0.72**	**0.73**	**0.6**	**150**	**0.1**	**520**	**1**	**2340**	**128**	**0.998081**	**0.802469**	**0.616114**	**0.000427**	**0.802469**
OR	Cosine	0.025	0.72	0.74	0.4	150	0.1	520	52	2289	128	0.909091	0.802469	0.616114	0.022213	0.802469
OR	Cosine	0.025	0.72	0.74	0.5	150	0.1	520	18	2323	128	0.966543	0.802469	0.616114	0.007689	0.802469
OR	Cosine	0.025	0.72	0.74	0.6	150	0.1	520	1	2340	128	0.998081	0.802469	0.616114	0.000427	0.802469
OR	Cosine	0.025	0.73	0.72	0.4	150	0.1	520	49	2292	128	0.913884	0.802469	0.616114	0.020931	0.802469
OR	Cosine	0.025	0.73	0.72	0.5	150	0.1	520	19	2322	128	0.964750	0.802469	0.616114	0.008116	0.802469
OR	Cosine	0.025	0.73	0.72	0.6	150	0.1	520	2	2339	128	0.996169	0.802469	0.616114	0.000854	0.802469
OR	Cosine	0.025	0.73	0.73	0.4	150	0.1	520	47	2294	128	0.917108	0.802469	0.616114	0.020077	0.802469
OR	Cosine	0.025	0.73	0.73	0.5	150	0.1	520	18	2323	128	0.966543	0.802469	0.616114	0.007689	0.802469
OR	Cosine	0.025	0.73	0.73	0.6	150	0.1	520	1	2340	128	0.998081	0.802469	0.616114	0.000427	0.802469
OR	Cosine	0.025	0.73	0.74	0.4	150	0.1	520	46	2295	128	0.918728	0.802469	0.616114	0.019650	0.802469
OR	Cosine	0.025	0.73	0.74	0.5	150	0.1	520	18	2323	128	0.966543	0.802469	0.616114	0.007689	0.802469
OR	Cosine	0.025	0.73	0.74	0.6	150	0.1	520	1	2340	128	0.998081	0.802469	0.616114	0.000427	0.802469
OR	Cosine	0.025	0.74	0.72	0.4	150	0.1	520	45	2296	128	0.920354	0.802469	0.616114	0.019223	0.802469
OR	Cosine	0.025	0.74	0.72	0.5	150	0.1	520	19	2322	128	0.964750	0.802469	0.616114	0.008116	0.802469
OR	Cosine	0.025	0.74	0.72	0.6	150	0.1	520	2	2339	128	0.996169	0.802469	0.616114	0.000854	0.802469
OR	Cosine	0.025	0.74	0.73	0.4	150	0.1	520	43	2298	128	0.923623	0.802469	0.616114	0.018368	0.802469
OR	Cosine	0.025	0.74	0.73	0.5	150	0.1	520	18	2323	128	0.966543	0.802469	0.616114	0.007689	0.802469
OR	Cosine	0.025	0.74	0.73	0.6	150	0.1	520	1	2340	128	0.998081	0.802469	0.616114	0.000427	0.802469
OR	Cosine	0.025	0.74	0.74	0.4	150	0.1	520	42	2299	128	0.925267	0.802469	0.616114	0.017941	0.802469
OR	Cosine	0.025	0.74	0.74	0.5	150	0.1	520	18	2323	128	0.966543	0.802469	0.616114	0.007689	0.802469
OR	Cosine	0.025	0.74	0.74	0.6	150	0.1	520	1	2340	128	0.998081	0.802469	0.616114	0.000427	0.802469

Table. 4. 8 several combinations of Cosine distance with different threshold in OR

Method	Metric	Max.Dist	Hi.Thr	Th.Thr	V.Th	ORB.Thr	BOW.Thr	TP	FP	TN	FN	PR	RE	F1	FPR	TPR
Sequential	**Jaccard**	**0.12**	**0.79**	**0.79**	**0.4**	**150**	**0.1**	**371**	**0**	**1188**	**277**	**1**	**0.572531**	**0.533813**	**0**	**0.572531**
Sequential	**Jaccard**	**0.12**	**0.79**	**0.79**	**0.5**	**150**	**0.1**	**371**	**0**	**1188**	**277**	**1**	**0.572531**	**0.533813**	**0**	**0.572531**
Sequential	**Jaccard**	**0.12**	**0.79**	**0.79**	**0.6**	**150**	**0.1**	**371**	**0**	**1188**	**277**	**1**	**0.572531**	**0.533813**	**0**	**0.572531**
Sequential	Jaccard	0.12	0.79	0.8	0.4	150	0.1	368	0	1188	280	1	0.567901	0.531792	0	0.567901
Sequential	Jaccard	0.12	0.79	0.8	0.5	150	0.1	368	0	1188	280	1	0.567901	0.531792	0	0.567901
Sequential	Jaccard	0.12	0.79	0.8	0.6	150	0.1	368	0	1188	280	1	0.567901	0.531792	0	0.567901
Sequential	Jaccard	0.12	0.79	0.81	0.4	150	0.1	366	0	1188	282	1	0.564815	0.530435	0	0.564815
Sequential	Jaccard	0.12	0.79	0.81	0.5	150	0.1	366	0	1188	282	1	0.564815	0.530435	0	0.564815
Sequential	Jaccard	0.12	0.79	0.81	0.6	150	0.1	366	0	1188	282	1	0.564815	0.530435	0	0.564815
Sequential	Jaccard	0.12	0.8	0.79	0.4	150	0.1	371	0	1188	277	1	0.572531	0.533813	0	0.572531
Sequential	Jaccard	0.12	0.8	0.79	0.5	150	0.1	371	0	1188	277	1	0.572531	0.533813	0	0.572531
Sequential	Jaccard	0.12	0.8	0.79	0.6	150	0.1	371	0	1188	277	1	0.572531	0.533813	0	0.572531
Sequential	Jaccard	0.12	0.8	0.8	0.4	150	0.1	368	0	1188	280	1	0.567901	0.531792	0	0.567901
Sequential	Jaccard	0.12	0.8	0.8	0.5	150	0.1	368	0	1188	280	1	0.567901	0.531792	0	0.567901
Sequential	Jaccard	0.12	0.8	0.8	0.6	150	0.1	368	0	1188	280	1	0.567901	0.531792	0	0.567901
Sequential	Jaccard	0.12	0.8	0.81	0.4	150	0.1	366	0	1188	282	1	0.564815	0.530435	0	0.564815
Sequential	Jaccard	0.12	0.8	0.81	0.5	150	0.1	366	0	1188	282	1	0.564815	0.530435	0	0.564815
Sequential	Jaccard	0.12	0.8	0.81	0.6	150	0.1	366	0	1188	282	1	0.564815	0.530435	0	0.564815
Sequential	Jaccard	0.12	0.81	0.79	0.4	150	0.1	371	0	1188	277	1	0.572531	0.533813	0	0.572531
Sequential	Jaccard	0.12	0.81	0.79	0.5	150	0.1	371	0	1188	277	1	0.572531	0.533813	0	0.572531
Sequential	Jaccard	0.12	0.81	0.79	0.6	150	0.1	371	0	1188	277	1	0.572531	0.533813	0	0.572531
Sequential	Jaccard	0.12	0.81	0.8	0.4	150	0.1	368	0	1188	280	1	0.567901	0.531792	0	0.567901
Sequential	Jaccard	0.12	0.81	0.8	0.5	150	0.1	368	0	1188	280	1	0.567901	0.531792	0	0.567901
Sequential	Jaccard	0.12	0.81	0.8	0.6	150	0.1	368	0	1188	280	1	0.567901	0.531792	0	0.567901
Sequential	Jaccard	0.12	0.81	0.81	0.4	150	0.1	366	0	1188	282	1	0.564815	0.530435	0	0.564815
Sequential	Jaccard	0.12	0.81	0.81	0.5	150	0.1	366	0	1188	282	1	0.564815	0.530435	0	0.564815
Sequential	Jaccard	0.12	0.81	0.81	0.6	150	0.1	366	0	1188	282	1	0.564815	0.530435	0	0.564815

Table. 4. 9 several combinations of Jaccard with different threshold in sequential

Method	Metric	Max.Dist	Hi.Thr	Th.Thr	V.Th	ORB.Thr	BOW.Thr	TP	FP	TN	FN	PR	RE	F1	FPR	TPR
OR	Jaccard	0.12	0.79	0.79	0.4	150	0.1	565	10	1178	83	1	0.871914	0.635546	0	0.871914
OR	Jaccard	0.12	0.79	0.79	0.5	150	0.1	565	2	1186	83	1	0.871914	0.635546	0	0.871914
OR	**Jaccard**	**0.12**	**0.79**	**0.79**	**0.6**	**150**	**0.1**	**565**	**1**	**1187**	**83**	**1**	**0.871914**	**0.635546**	**0**	**0.871914**
OR	Jaccard	0.12	0.79	0.8	0.4	150	0.1	565	9	1179	83	1	0.871914	0.635546	0	0.871914
OR	Jaccard	0.12	0.79	0.8	0.5	150	0.1	565	1	1187	83	1	0.871914	0.635546	0	0.871914
OR	Jaccard	0.12	0.79	0.8	0.6	150	0.1	565	1	1187	83	1	0.871914	0.635546	0	0.871914
OR	Jaccard	0.12	0.79	0.81	0.4	150	0.1	565	7	1181	83	1	0.871914	0.635546	0	0.871914
OR	Jaccard	0.12	0.79	0.81	0.5	150	0.1	565	0	1188	83	1	0.871914	0.635546	0	0.871914
OR	Jaccard	0.12	0.79	0.81	0.6	150	0.1	565	0	1188	83	1	0.871914	0.635546	0	0.871914
OR	Jaccard	0.12	0.8	0.79	0.4	150	0.1	565	10	1178	83	1	0.871914	0.635546	0	0.871914
OR	Jaccard	0.12	0.8	0.79	0.5	150	0.1	565	2	1186	83	1	0.871914	0.635546	0	0.871914
OR	Jaccard	0.12	0.8	0.79	0.6	150	0.1	565	1	1187	83	1	0.871914	0.635546	0	0.871914
OR	Jaccard	0.12	0.8	0.8	0.4	150	0.1	565	9	1179	83	1	0.871914	0.635546	0	0.871914
OR	Jaccard	0.12	0.8	0.8	0.5	150	0.1	565	1	1187	83	1	0.871914	0.635546	0	0.871914
OR	Jaccard	0.12	0.8	0.8	0.6	150	0.1	565	1	1187	83	1	0.871914	0.635546	0	0.871914
OR	Jaccard	0.12	0.8	0.81	0.4	150	0.1	565	7	1181	83	1	0.871914	0.635546	0	0.871914
OR	Jaccard	0.12	0.8	0.81	0.5	150	0.1	565	0	1188	83	1	0.871914	0.635546	0	0.871914
OR	Jaccard	0.12	0.8	0.81	0.6	150	0.1	565	0	1188	83	1	0.871914	0.635546	0	0.871914
OR	Jaccard	0.12	0.81	0.79	0.4	150	0.1	565	10	1178	83	1	0.871914	0.635546	0	0.871914
OR	Jaccard	0.12	0.81	0.79	0.5	150	0.1	565	2	1186	83	1	0.871914	0.635546	0	0.871914
OR	Jaccard	0.12	0.81	0.79	0.6	150	0.1	565	1	1187	83	1	0.871914	0.635546	0	0.871914
OR	Jaccard	0.12	0.81	0.8	0.4	150	0.1	565	9	1179	83	1	0.871914	0.635546	0	0.871914
OR	Jaccard	0.12	0.81	0.8	0.5	150	0.1	565	1	1187	83	1	0.871914	0.635546	0	0.871914
OR	Jaccard	0.12	0.81	0.8	0.6	150	0.1	565	1	1187	83	1	0.871914	0.635546	0	0.871914
OR	Jaccard	0.12	0.81	0.81	0.4	150	0.1	565	7	1181	83	1	0.871914	0.635546	0	0.871914
OR	Jaccard	0.12	0.81	0.81	0.5	150	0.1	565	0	1188	83	1	0.871914	0.635546	0	0.871914

Table. 4. 10 several combinations of Jaccard distance with different threshold in OR

The best performing combination for each metric distance is marked in bold. In general, the combination that produces maximum values for Precision and Recall measurements was selected as the final result.

4.4.4 Evaluation of the execution speed

The value of time is determined by calculating the execution time for three metric distance used in purposed work for the same query images and the same number of query images. We execute 100 query images at the same time to find their near duplicate imaged in Native database. Assuming that, several executions was done with different threshold for our metric distances in order to obtain less false positive in result, it means to find images that are really duplicate, or they are near duplicate to the query images that have been sent to the system. The time value calculate from the query images was received by system until

that near duplicate images are return to the system, shows in a the list of near duplicate images table.

Following tables show the average times for 100 query images to get a near-duplicate match in three different metric distance (Hamming, Jaccard, Cosine):

Method	Metric	Max.Dist	Hi.Thr	Th.Thr	V.Th	ORB.Thr	BOW.Thr	Time
OR	Hamming	16	0.8	0.8	0.4	150	0.1	0.497462
					0.5	150	0.1	0.505284
					0.6	150	0.1	0.552626
				0.81	0.4	150	0.1	0.570995
					0.5	150	0.1	0.574184
					0.6	150	0.1	0.563192
			0.81	0.8	0.4	150	0.1	0.581746
					0.5	150	0.1	0.554539
					0.6	150	0.1	0.576367
				0.81	0.4	150	0.1	0.575672
					0.5	150	0.1	0.57926
					0.6	150	0.1	0.568254

Table. 4. 11Execution time for Hamming distance

Method	Metric	Max.Dist	Hi.Thr	Th.Thr	V.Th	ORB.Thr	BOW.Thr	Time
OR	Cosine	0.025	0.72	0.72	0.4	150	0.1	1.494521
					0.5	150	0.1	1.645425
					0.6	150	0.1	1.472919
				0.73	0.4	150	0.1	1.503704
					0.5	150	0.1	1.654399
					0.6	150	0.1	1.597683
				0.74	0.4	150	0.1	1.422042
					0.5	150	0.1	1.648096
					0.6	150	0.1	1.432569
			0.73	0.72	0.4	150	0.1	1.619124
					0.5	150	0.1	1.573023
					0.6	150	0.1	1.46163
				0.73	0.4	150	0.1	1.517022
					0.5	150	0.1	1.574942
					0.6	150	0.1	1.655458
				0.74	0.4	150	0.1	1.712913
					0.5	150	0.1	1.504661
					0.6	150	0.1	1.485354
			0.74	0.72	0.4	150	0.1	1.539035
					0.5	150	0.1	1.584519
					0.6	150	0.1	1.604612
				0.73	0.4	150	0.1	1.479852
					0.5	150	0.1	1.423961
					0.6	150	0.1	1.497149
				0.74	0.4	150	0.1	1.528126
					0.5	150	0.1	1.407009
					0.6	150	0.1	1.606196

Table. 4. 12 Execution time for Cosine distance

Method	Metric	Max.Dist	Hi.Thr	Th.Thr	V.Th	ORB.Thr	BOW.Thr	Time
OR	Jaccard	0.12	0.79	0.79	0.4	150	0.1	0.639918
					0.5	150	0.1	0.66766
					0.6	150	0.1	0.723979
				0.8	0.4	150	0.1	0.700044
					0.5	150	0.1	0.637367
					0.6	150	0.1	0.756909
				0.81	0.4	150	0.1	0.698482
					0.5	150	0.1	0.660884
					0.6	150	0.1	0.617174
			0.8	0.79	0.4	150	0.1	0.697819
					0.5	150	0.1	0.703552
					0.6	150	0.1	0.758895
				0.8	0.4	150	0.1	0.750693
					0.5	150	0.1	0.750458
					0.6	150	0.1	0.774359
				0.81	0.4	150	0.1	0.736287
					0.5	150	0.1	0.695659
					0.6	150	0.1	0.71468
			0.81	0.79	0.4	150	0.1	0.760385
					0.5	150	0.1	0.725508
					0.6	150	0.1	0.769325
				0.8	0.4	150	0.1	0.729964
					0.5	150	0.1	0.779494
					0.6	150	0.1	0.607913
				0.81	0.4	150	0.1	0.725932
					0.5	150	0.1	0.620154
					0.6	150	0.1	0.699851

Table. 4. 13 Execution time for Cosine distance

The best times for each metric distance are marked in color. This average time includes the elapsed time to create the fingerprints for the query image, find the candidate images in the BK-Tree, calculate the similarity index for each fingerprint and the matching process to obtain the final decision and result.

The difference between the execution times can be explained by the comparing functions for the search tree. A search operation inside a BK-Tree requires performing comparing operations between the query image hash and each nodes of the tree.

The Hamming distance function for 2 vectors is defined as the count of bits that are different. This operation can be easily implemented by performing an XOR operation between both hashes and then counting the resulting bits with a value of 1.

The Jaccard distance, in a similar way, can be performed mostly with bitwise operations. It is defined as 1 minus the intersection between the vectors divided by the union.

Cosine distance is a much more complicated operation, requiring calculations of dot products and magnitude of vectors. These operations are much more computationally expensive, producing slower results. Since the goal of this work is to obtain a balance between speed and accuracy, the Hamming distance is the first choice for metric distances, because its execution time is much faster than both Cosine and Jaccard.

This is important to mention that we already created fingerprints of all images at Native database in offline stage, hence when we send a query image to the system the matching process to find near duplicate are applied on FB databased located on hard disk.

following example table shows the average time for searching 100 query images executed by both OR and Sequential method in Hamming, Jaccard and Cosine metric distance on images on Native database.

Method	Metric	Max.Dist	Hi.Thr	Th.Thr	V.Thr	ORB.Thr	BOW	Time
OR	Hamming	16	0.8	0.8	0.4	150	0.1	**0.497462**
	Cosine	0.025	0.74	0.74	0.5	150	01	**1.407009**
	Jaccard	0.12	0.81	0.8	0.6	150	01	**0.607913**
Sequential	Hamming	16	0.8	0.8	0.5	150	0.1	**0.12445**
	Cosine	0.025	0.72	0.74	0.4	150	0.1	**1.044546**
	Jaccard	0.12	0.81	0.8	0.6	150	0.1	**0.304215**

Table. 4. 14 The time of execution for both OR and Sequential method

By applying several experimental and increase the amount of query images from 1 to 100 images, we obtained better results for each metric distance, in OR method which we choose as final method for our near duplicate detection system.

Bellow tables demonstrate the Precision, Recall, F1 score and accuracy values for hamming distance execution on 100 query images with three different combination of thresholds for fingerprints.

Method	Metric	Max.Dist	Hi.Thr	Th.Thr	V.Thr	ORB.Thr	BOW	Pre	Recall	F1	AC
OR	Hamming	16	0.8	0.81	0.6	150	0.1	**1.000000**	0.879630	**0.637584**	**0.966935**
OR	Hamming	16	0.8	0.8	0.6	150	0.1	**0.998249**	0.879630	0.637584	0.966511
OR	Hamming	16	0.8	0.81	0.5	150	0.1	**0.996503**	0.879630	0.637584	0.966087

Table. 4. 15 the best 3 combination of thresholds for Hamming distance

The following table shows Precision, Recall, F1 score and accuracy values for Jaccard.

Method	Metric	Max.Dist	Hi.Thr	Th.Thr	V.Thr	ORB.Thr	BOW	Pre	Recall	F1	AC
OR	Jaccard	0.12	0.79	0.81	0.5	150	0.1	**1.000000**	0.871914	**0.635546**	**0.954793**
OR	Jaccard	0.12	0.79	0.79	0.6	150	0.1	**0.998233**	0.871914	0.635546	**0.954248**
OR	Jaccard	0.12	0.79	0.79	0.5	150	0.1	**0.996473**	0.871914	0.635546	**0.953704**

Table. 4. 16 The best 3 combination of thresholds for Jaccard distance

The following table Precision, Recall, F1 score and accuracy values for Cosine distance:

Method	Metric	Max.Dist	Hi.Thr	Th.Thr	V.Thr	ORB.Thr	BOW	Pre	Recall	F1	AC
OR	Cosine	0.025	0.72	0.73	0.6	150	0.1	**0.998081**	0.802469	**0.616114**	0.956842
OR	Cosine	0.025	0.72	0.72	0.6	150	0.1	**0.996169**	0.802469	0.616114	0.956507
OR	Cosine	0.025	0.72	0.73	0.5	150	0.1	**0.966543**	0.802469	0.616114	0.951154

Table. 4. 17 The best 3 combination of thresholds for Cosine distance

The accuracy measures

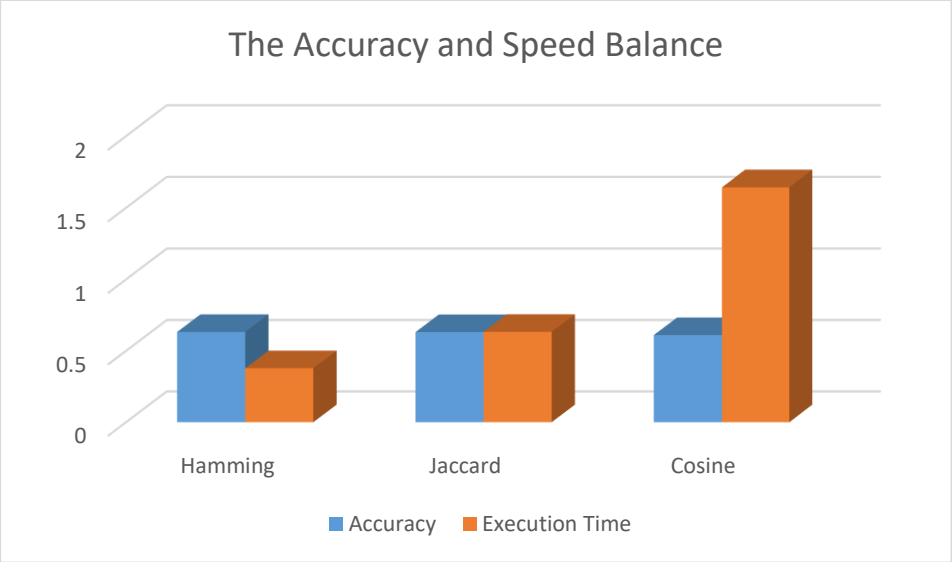
We calculate the accuracy measure for each method by following formula:

$$Acc = \frac{TP + TN}{TP + TN + FP + FN}$$

The accuracy measure shows how well our method which based on binary classification test, correctly identify positive or negative results. In this case positive results are the image which are near duplicate images and negative results are the images which are not near duplicate. Following table show the accuracy and time for three metric distance (Hamming, Jaccard, Cosine) in OR method:

Metric Distance	Accuracy	Time
Hamming	**0.966935**	**0.563192**
Jaccard	**0.954793**	**0.660884**
Cosine	**0.956842**	**1.597683**

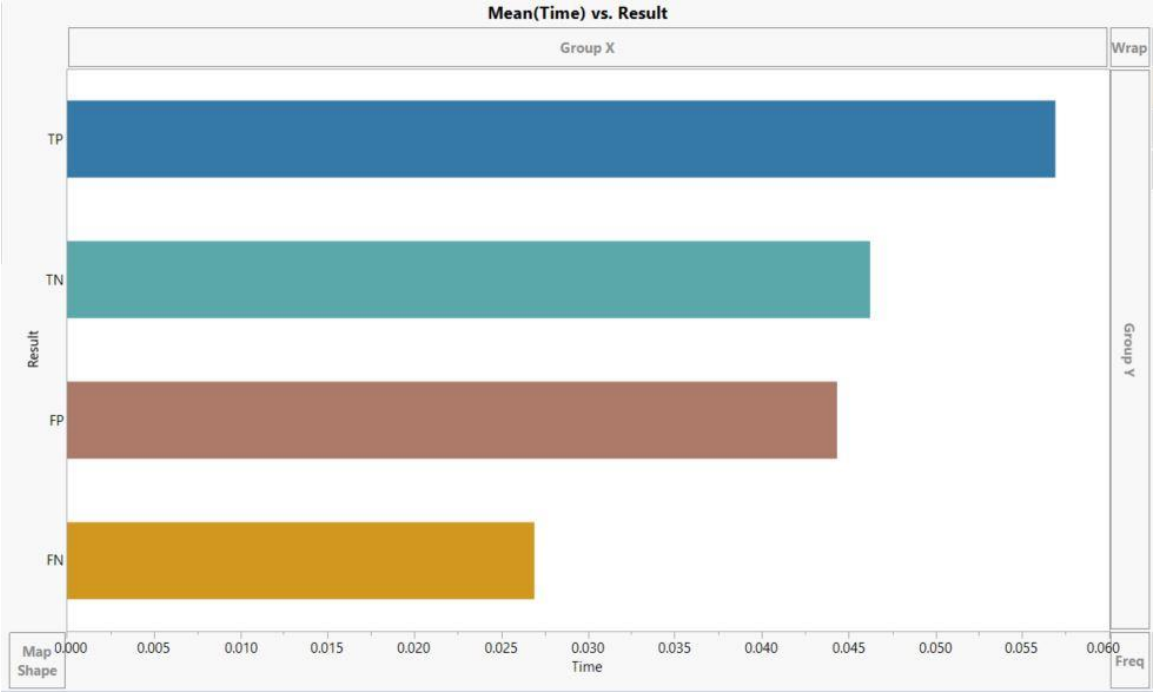
*Table. 4. 18 The accuracy and time of three metric distance*In the following graph we can see clearly the performance of three similarity distance (Hamming, Jaccard, Cosine) versus the time of execution to find near duplicate images with the same number of query images. As is demonstrated in the following graph the Hamming distance provides better results and performance in execution time and accuracy.



Graph.4.4 The Accuracy and Speed Balance for three metric distance

This is important to mention the best value of accuracy from execution table results, running with different threshold doesn't have the best time value. This is obvious that having the best accuracy will take more time in execution. In this work we could find the value of threshold for each fingerprint which lead us to a very convenience accuracy and great execution time. Table of results shown that there is not a big difference between various threshold and their execution time versus accuracy.

Also results shown that the number of True positive is significantly high which is a prof that algorithm was able to find near duplicate images. The following graph shows the statues of false positive, false negative, true positive and true negative versus the time of finding them by execution of algorithm.



Graph.4.5 The execution time versus,FP,FN,TP,TN in Hamming distance

Experimental with different threshold for each fingerprint

We executed each fingerprint separately with different threshold values to obtain the best performance for each fingerprint just on hamming distance, we already discover the hamming distance in our method had the best performance results. Following tables demonstrate each fingerprint performance with best threshold values.

Method	Metric	Max.Dist	Hi.Thr	TP	FP	TN	FN	PR	RE	F1	Accuracy
OR	Hamming	16	0.84	7573	64	12355	1139	0.991620	0.869261	0.634839	0.943069
OR	Hamming	16	0.85	7573	58	12361	1139	0.992399	0.869261	0.634839	0.943353
OR	Hamming	16	0.86	7572	49	12370	1140	0.993570	0.869146	0.634809	0.943732
OR	Hamming	16	0.87	7569	36	12383	1143	0.995266	0.868802	0.634717	0.944205
OR	Hamming	16	0.88	7567	25	12394	1145	0.996707	0.868572	0.634656	0.944631
OR	Hamming	16	0.89	7565	4	12415	1147	0.999472	0.868343	0.634594	0.945530
OR	Hamming	16	0.9	7562	0	12419	1150	1.000000	0.867998	0.634502	0.945578
OR	Hamming	16	0.91	7553	0	12419	1159	1.000000	0.866965	0.634226	0.945152
OR	Hamming	16	0.92	7542	0	12419	1170	1.000000	0.865702	0.633888	0.944631
OR	Hamming	16	0.93	7534	0	12419	1178	1.000000	0.864784	0.633642	0.944253

Table. 4. 19 The Histogram fingerprint threshold values and their accuracy

Method	Metric	Max.Dist	Th.Thr	TP	FP	TN	FN	PR	RE	F1	Accuracy
OR	Hamming	16	0.9	5940	69	12350	2772	0.988517	0.681818	0.576923	0.865553
OR	Hamming	16	0.91	5827	45	12374	2885	0.992337	0.668848	0.572228	0.861341
OR	Hamming	16	0.92	5728	16	12403	2984	0.997214	0.657484	0.568029	0.858028
OR	Hamming	16	0.93	5605	15	12404	3107	0.997331	0.643365	0.562695	0.852255
OR	Hamming	16	0.94	5477	3	12416	3235	0.999453	0.628673	0.557002	0.846765
OR	Hamming	16	0.95	5342	0	12419	3370	1.000000	0.613177	0.550835	0.840519
OR	Hamming	16	0.96	5146	0	12419	3566	1.000000	0.590680	0.541570	0.831243

Table. 4. 20 The Thumbnail fingerprint threshold values and their accuracy

Method	Metric	Max.Dist	V.Th	TP	FP	TN	FN	PR	RE	F1	Accuracy
OR	Hamming	16	0.4	7576	3815	8604	1136	0.665086	0.869605	0.634931	0.765700
OR	Hamming	16	0.45	7576	2324	10095	1136	0.765253	0.869605	0.634931	0.836260
OR	Hamming	16	0.5	7576	1380	11039	1136	0.845913	0.869605	0.634931	0.880933
OR	Hamming	16	0.55	7576	632	11787	1136	0.923002	0.869605	0.634931	0.916331
OR	Hamming	16	0.6	7576	246	12173	1136	0.968550	0.869605	0.634931	0.934598
OR	Hamming	16	0.65	7576	144	12275	1136	0.981347	0.869605	0.634931	0.939425
OR	Hamming	16	0.7	7576	44	12375	1136	0.994226	0.869605	0.634931	0.944158
OR	Hamming	16	0.75	7574	13	12406	1138	0.998287	0.869376	0.634870	0.945530
OR	Hamming	16	0.8	7561	3	12416	1151	0.999603	0.867883	0.634472	0.945388
OR	Hamming	16	0.85	7530	0	12419	1182	1.000000	0.864325	0.633518	0.944063

Table. 4. 21 The deep learning fingerprint threshold values and their accuracy

Method	Metric	Max.Dist	ORB.Thr	TP	FP	TN	FN	PR	RE	F1	Accuracy
OR	Hamming	16	14	7570	5	12414	1142	0.999340	0.868916	0.634748	0.945720
OR	Hamming	16	15	7570	5	12414	1142	0.999340	0.868916	0.634748	0.945720
OR	Hamming	16	16	7569	4	12415	1143	0.999472	0.868802	0.634717	0.945720
OR	Hamming	16	17	7569	3	12416	1143	0.999604	0.868802	0.634717	0.945767
OR	Hamming	16	18	7569	3	12416	1143	0.999604	0.868802	0.634717	0.945767
OR	Hamming	16	19	7568	2	12417	1144	0.999736	0.868687	0.634686	0.945767
OR	Hamming	16	20	7568	2	12417	1144	0.999736	0.868687	0.634686	0.945767
OR	Hamming	16	21	7567	2	12417	1145	0.999736	0.868572	0.634656	0.945720
OR	Hamming	16	22	7565	1	12418	1147	0.999868	0.868343	0.634594	0.945672
OR	Hamming	16	23	7563	1	12418	1149	0.999868	0.868113	0.634533	0.945578
OR	Hamming	16	24	7562	1	12418	1150	0.999868	0.867998	0.634502	0.945530
OR	Hamming	16	25	7562	0	12419	1150	1.000000	0.867998	0.634502	0.945578

Table. 4. 22 The ORB fingerprint threshold values and their accuracy

Method	Metric	Max.Dist	BOW.Thr	TP	FP	TN	FN	PR	RE	F1	Accuracy
OR	Hamming	16	0.02	7557	36	12383	1155	0.995259	0.867424	0.634349	0.943637
OR	Hamming	16	0.04	7481	14	12405	1231	0.998132	0.858701	0.632001	0.941082
OR	Hamming	16	0.06	7267	6	12413	1445	0.999175	0.834137	0.625226	0.931333
OR	Hamming	16	0.08	6825	0	12419	1887	1.000000	0.783402	0.610411	0.910700
OR	Hamming	16	0.1	6357	0	12419	2355	1.000000	0.729683	0.593391	0.888552
OR	Hamming	16	0.12	5907	0	12419	2805	1.000000	0.678030	0.575563	0.867257

Table. 4. 23 The BOWV fingerprint threshold values and their accuracy

From above experimental we choose the best value of threshold for each fingerprint to execute the algorithm by these new values.

4.4.6 Improving the algorithm

To acknowledge the certainty of results we examine our method with several preventive measures and test validation. We tested various value for thresholds until we reached to the accurate value for each fingerprint. All the fingerprints separately were evaluated by different series of query images to find their accuracy against several image attacks. Following are some points that we applied for improving our near duplicate detection algorithm:

Refactoring code

The algorithm source code refactoring was improved in the testing and validation process by calculating the time of execution and resource consumption. For example, using Pickle for save the reference list and BK-tree helped to reduce execution time. since the process is automated for finding False negative, True negative, False positive and True positive, so we reduced the manual tagging errors as well.

Using random seed in evaluation process

Random seed is an initial value for the pseudorandom number generator. During algorithm validation process we used query image from our Query Image database since all images either inside Native database and Query database are a collection of images with different size, format, category of objects and scenes we had the risk of getting inaccurate result if each time of testing process the algorithm choose the different set of query images. Hence

to avoid such issue, we used at the first a unique random seed to send query images to our near duplicate image detection system for several validation execution, then we tried different random seed for different executions as well finally we calculate the average of the results. This process helped us to assure we are covering a variety renege of query images sending to the system and the algorithm produced results correctly. This is obvious that retrieve an image with complicated scenes and a large size computationally is heavier than an image with white background, single object and a small size, knowing above we automated the algorithm to choose randomly a series of query images by random seed.

Seed	Method	Metric	Max.Dist	Hi.Thr	Th.Thr	V.Th	ORB.Thr	BOW.Thr	TP	FP	TN	FN	PR	RE	F1	Accuracy
2	OR	Hamming	16	0.89	0.9	0.75	17	0.02	7576	0	12419	1136	1.000000	0.869605	0.634931	0.946240
3	OR	Hamming	16	0.89	0.9	0.75	17	0.02	7565	16	8585	1147	0.997889	0.868343	0.634594	0.932825
4	OR	Hamming	16	0.89	0.9	0.75	17	0.02	8057	4	10100	1249	0.999504	0.865786	0.633910	0.935446
5	OR	Hamming	16	0.89	0.9	0.75	17	0.02	7832	16	13078	1186	0.997961	0.868485	0.634633	0.945640
6	OR	Hamming	16	0.89	0.9	0.75	17	0.02	8058	5	12711	1230	0.999380	0.867571	0.634388	0.943874
7	OR	Hamming	16	0.89	0.9	0.75	17	0.02	8058	5	12711	1230	0.999380	0.867571	0.634388	0.943874
Average									7857.666667	7.666666667	11600.66667	1196.333333	0.999019	0.8678934	0.6344742	0.9413165

Table. 4. 24 Algorithm execution with different random seeds

In following table, we can see the average performance with different seeds in various execution where we choose the best threshold for each fingerprint.

TP	FP	TN	FN	PR	RE	F1	Accuracy
7576	0	12419	1136	1.000000	0.869605	0.634931	0.946240
7565	16	8585	1147	0.997889	0.868343	0.634594	0.932825
8057	4	10100	1249	0.999504	0.865786	0.633910	0.935446
7832	16	13078	1186	0.997961	0.868485	0.634633	0.945640
8058	5	12711	1230	0.999380	0.867571	0.634388	0.943874
8058	5	12711	1230	0.999380	0.867571	0.634388	0.943874
7857.666667	**7.666666667**	**11600.66667**	**1196.333333**	**0.999019**	**0.8678934**	**0.6344742**	**0.9413165**

Table. 4. 25 average performance with different random seeds

Execution with and without deep learning fingerprint

Another experiment which provide us an interesting result was running the algorithm with and without deep learning fingerprint. In each of executing of this step experiment set the

best value of threshold, and different random seed. This experiment was to indicate our method performance. Following steps was implemented and the results was compared.

- **Step one:** ran the algorithm with all of six fingerprints
- **Step two:** ran the algorithm with all fingerprints minus deep learning fingerprint
- **Step three:** ran the algorithm with only deep learning fingerprint using Bk-Tree

Following tables show each step experiment results:

FP	TN	FN	PR	RE	F1	Accuracy
0	12419	1136	1.000000	0.869605	0.634931	0.946240
16	8585	1147	0.997889	0.868343	0.634594	0.932825
4	10100	1249	0.999504	0.865786	0.633910	0.935446
16	13078	1186	0.997961	0.868485	0.634633	0.945640
5	12711	1230	0.999380	0.867571	0.634388	0.943874
5	12711	1230	0.999380	0.867571	0.634388	0.943874
7.666667	**11600.666667**	**1196.333333**	**0.999019**	**0.867893**	**0.634474**	**0.941316**

Table. 4. 26 execution the algorithm with all six fingerprints

FP	TN	FN	PR	RE	F1	Accuracy
0	12419	1143	1.000000	0.868802	0.634717	0.945909
9	8592	1154	0.998811	0.867539	0.634380	0.932825
4	10100	1253	0.999504	0.865356	0.633795	0.935240
9	13085	1194	0.998851	0.867598	0.634396	0.945595
5	12711	1237	0.999379	0.866817	0.634187	0.943556
8	12460	1205	0.999005	0.869504	0.634904	0.944107
5.833333	11561.166667	1197.666667	0.999258	0.867603	0.634396	**0.941205**

Table. 4. 27 execution the algorithm with all six fingerprints menus deep learning

FP	TN	FN	PR	RE	F1	Accuracy
13	12406	1138	0.998287	0.869376	0.634870	0.945530
19	8582	1154	0.997492	0.867539	0.634380	0.932247
1	10103	1253	0.999876	0.865356	0.633795	0.935394
18	13076	1188	0.997706	0.868263	0.634573	0.945459
26	12690	1234	0.996782	0.867140	0.634273	0.942738
4	12464	1203	0.999502	0.869721	0.634962	0.944383
13.500000	11553.500000	1195.000000	0.998274	0.867899	0.634476	**0.940959**

Table. 4. 28 execution the algorithm with only deep learning fingerprints

After obtaining above results we compare them to get final decision to use either just deep learning fingerprint or a combination of all fingerprints in order to choose the best filtering method for detecting near duplicate images. Results show in following table:

Fingerprint combanition	Accuracy
All fingerprints	0.941316
All fingerprints minus deep learning	0.941205
Deep learning fingerprint	0.940959

Table. 4. 29 execution the algorithm with only deep learning fingerprints

As it is shown in the above table by implementing and executing these combinations, we got to the conclusion that our method including six fingerprints lead us to the best result accuracy measure. In comparison the step two returned less accuracy and the third step was returned the lowest accuracy. Getting to the list of False positive demonstrated that deep learning (vgg19) is vulnerable to the blur, rotation and crop image attacks. Following table demonstrate the list of near duplicate images with image attacks that third step was not able to identify them.

Seed	Metric	Max.Dist	Method	Query Image	Candidate	Source	Distance	V.Th	V.Sim	Match	Result	Time
2	Hamming	16	OR	000000177935_original.cc.in.jpg	000000177935_blur.jpg	SelectedBy	2	0.75	0.7456168	N	FN	0.0189079
2	Hamming	16	OR	000000125257_original.cc.in.jpg	000000125257_blur.jpg	SelectedBy	2	0.75	0.7302111	N	FN	0.0216871
3	Hamming	16	OR	000000534664_original.cc.in.jpg	000000534664_blur.jpg	SelectedBy	2	0.75	0.6991352	N	FN	0.0166227
3	Hamming	16	OR	000000459809_original.cc.in.jpg	000000459809_rotatedm01.jpg	SelectedBy	15	0.75	0.7341818	N	FN	0.0170119
3	Hamming	16	OR	000000433204_original.cc.in.jpg	000000433204_rotatedm01.jpg	SelectedBy	2	0.75	0.7448753	N	FN	0.0335811
3	Hamming	16	OR	000000088951_original.cc.in.jpg	000000088951_cropped03.jpg	SelectedBy	8	0.75	0.7250248	N	FN	0.0118461
3	Hamming	16	OR	000000088951_original.cc.in.jpg	000000088951_cropped05.jpg	SelectedBy	11	0.75	0.7320546	N	FN	0.0118459
3	Hamming	16	OR	000000304545_original.cc.in.jpg	000000304545_blur.jpg	SelectedBy	3	0.75	0.6443304	N	FN	0.019126
3	Hamming	16	OR	000000076261_original.cc.in.jpg	000000076261_rotated01.jpg	SelectedBy	13	0.75	0.7334553	N	FN	0.0166823
4	Hamming	16	OR	000000217948_original.cc.in.jpg	000000217948_blur.jpg	SelectedBy	0	0.75	0.7103388	N	FN	0.0156123
4	Hamming	16	OR	000000433204_original.cc.in.jpg	000000433204_rotatedm01.jpg	SelectedBy	2	0.75	0.7448753	N	FN	0.0145618
4	Hamming	16	OR	000000511760_original.cc.in.jpg	000000511760_rotated01.jpg	SelectedBy	7	0.75	0.7458634	N	FN	0.0169739
4	Hamming	16	OR	000000125257_original.cc.in.jpg	000000125257_blur.jpg	SelectedBy	2	0.75	0.7302111	N	FN	0.0222854
5	Hamming	16	OR	000000099242_original.cc.in.jpg	000000099242_flippedtb.jpg	SelectedBy	16	0.75	0.650165	N	FN	0.0125967
5	Hamming	16	OR	000000545129_original.cc.in.jpg	000000545129_blur.jpg	SelectedBy	2	0.75	0.7061688	N	FN	0.031197
5	Hamming	16	OR	000000069106_original.cc.in.jpg	000000069106_blur.jpg	SelectedBy	2	0.75	0.6249214	N	FN	0.021768
6	Hamming	16	OR	000000531134_original.cc.in.jpg	000000531134_blur.jpg	SelectedBy	0	0.75	0.7167462	N	FN	0.0195004
6	Hamming	16	OR	000000304545_original.cc.in.jpg	000000304545_blur.jpg	SelectedBy	3	0.75	0.6443304	N	FN	0.0190721
6	Hamming	16	OR	000000574520_original.cc.in.jpg	000000574520_blur.jpg	SelectedBy	2	0.75	0.7273574	N	FN	0.0116667
6	Hamming	16	OR	000000562561_original.cc.in.jpg	000000562561_blur.jpg	SelectedBy	0	0.75	0.7419263	N	FN	0.0517802
7	Hamming	16	OR	000000172935_original.cc.in.jpg	000000172935_cropped05.jpg	SelectedBy	16	0.75	0.5620505	N	FN	0.0227001
7	Hamming	16	OR	000000517832_original.cc.in.jpg	000000517832_blur.jpg	SelectedBy	0	0.75	0.7421222	N	FN	0.0198638
7	Hamming	16	OR	000000511760_original.cc.in.jpg	000000511760_rotated01.jpg	SelectedBy	7	0.75	0.7458634	N	FN	0.0386807
7	Hamming	16	OR	000000574520_original.cc.in.jpg	000000574520_blur.jpg	SelectedBy	2	0.75	0.7273574	N	FN	0.0170958
7	Hamming	16	OR	000000492878_original.cc.in.jpg	000000492878_cropped10.jpg	SelectedBy	15	0.75	0.7272006	N	FN	0.0382001
7	Hamming	16	OR	000000185292_original.cc.in.jpg	000000185292_rotated01.jpg	SelectedBy	3	0.75	0.7072335	N	FN	0.047129
7	Hamming	16	OR	000000465718_original.cc.in.jpg	000000465718_blur.jpg	SelectedBy	0	0.75	0.6699065	N	FN	0.0542409
7	Hamming	16	OR	000000465718_original.cc.in.jpg	000000465718_smooth.jpg	SelectedBy	0	0.75	0.7277796	N	FN	0.0542113

Table. 4. 30 list of near duplicate images not identify by deep learning fingerprint

Scale-up number of the query images

We scaled up the number of query image and checked the performance of our near-duplicate image detection system comparing with previous results.

More restrictive setting for our method produce less false positive in the result even in large series of query images. Following table demonstrate the value of accuracy and time for different series of executions. The time value is average time from candidate selection to the finding final match for one image.

Proposed method	Accuracy	Average Time
1000 query images	0.946240	0.463161 sec
1500 query images	0.945047	0.473756 sec
2000 query images	0.942408	0.461858 sec

Table. 4. 31 average time with different number of query images

This is very important to mention that our large database is uncategorical database, which has different images size with different scene complexity hence, by increasing the number of query images the distribution of choosing image and its variety will increase as well. Above table result shows however we increase the number of query image but the average time of detecting near duplicate images did not increase.

The reason of using different fingerprints

As we already explained in the previous sections, each fingerprint plays the role of a filter against different type of image attacks. We separately executed the six fingerprints presented in this work to analyze their performance of detecting near duplicate images. Each of them had vulnerability to different image attacks which lead the result to a small

list of False negatives were the images have been detect by algorithm as Non-Near-duplicate but truly these images were near-duplicate. The following image is an illustration of image filtering via our six fingerprints which receive a large amount of query images and detect their near duplicate version through the database on the fly. These filters work as a sieving machine which sieves valuable minerals from stones. This sieving process provides robustness to our near duplicate detection system.

fig. 4. 3 The schematic image filtering with six fingerprints

The combination of these six fingerprints reduce significantly the number of False negative results. In our method when the algorithm returns False positive images mostly it's because there was remarkable similarity between query images and images detected as False positive which sometimes is difficult even to recognize with an unaided eye.

Our experiments and results show that the proposed method has the ability to detect near duplicate images with 95% of accuracy in a large database based on uncategorical images which every image has different image characteristics in a wide distribution. Following images shows the outcome of detecting near duplicate images by our method

demonstrating images as False positive, True positive, False negative and True negative values result in 2000 query image execution.

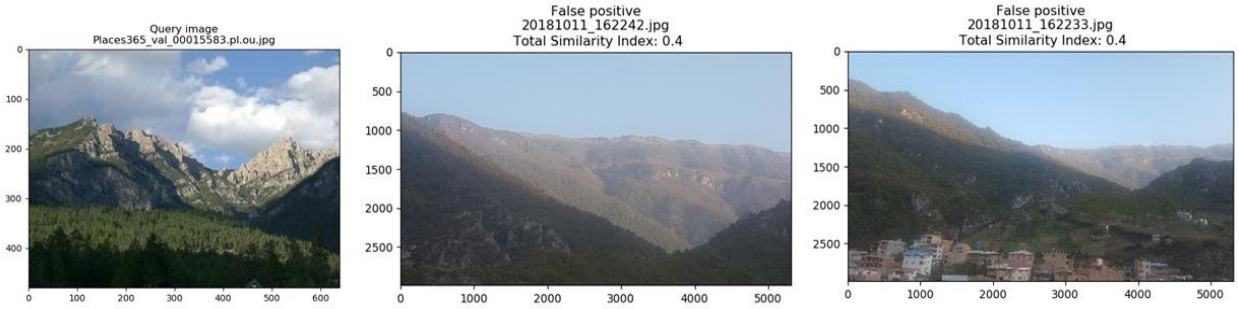


fig. 4. 4 The False positive results of detecting near duplicate images

Above figures represent the False positive results returned from the algorithm, as it shown the query image has a notable similarity with the other images. since there were several similar characteristics between them, the algorithm confused and presented them as near duplicate images however they are not near duplicate.

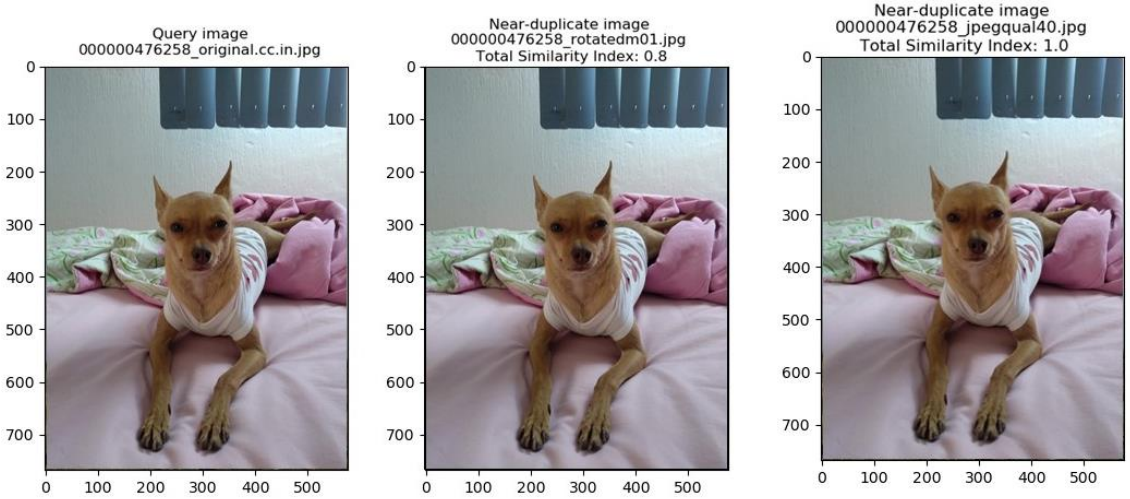


fig. 4. 4 The True positive results of detecting near duplicate images

above figures show the True positive results returned form the algorithm both images detected as near duplicate to the query are truly a version of query image but manipulated

with image attacks, one a slightly rotation and the other a compression format. So, the algorithm truly detects them as near duplicate images.

//

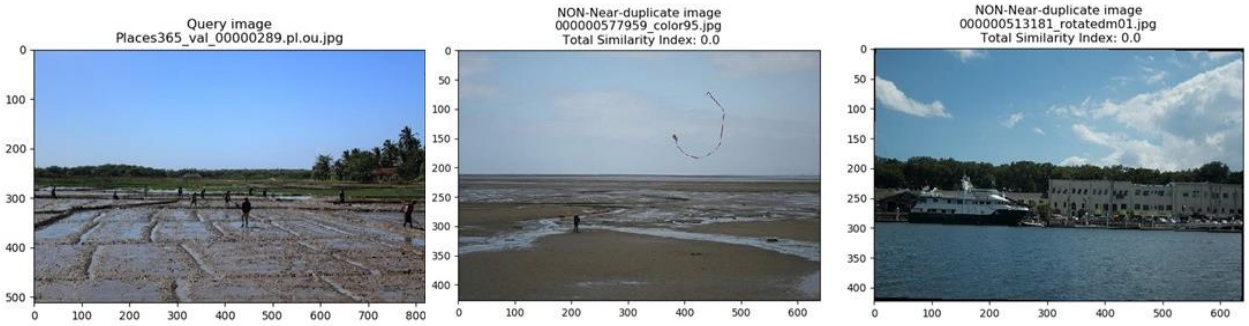


fig. 4. 5 The True negative results of detecting near duplicate images

Above figures demonstrate the True negative results returned by our algorithm. However, the images have a narrow similarity to the query image, but they are not duplicate images and the algorithm detect them as not near duplicate images.

fig. 4. 6 The False negative results of detecting near duplicate images

Above figures show the False negative results return by our algorithm. Since a flipped image attack is consider as near duplicate so the BK-tree and hash rejected these images

with flipped attack and did not present them as candidate image of being near duplicate, but we could find them via reference list. Flipped attacks are the limitation of our method.

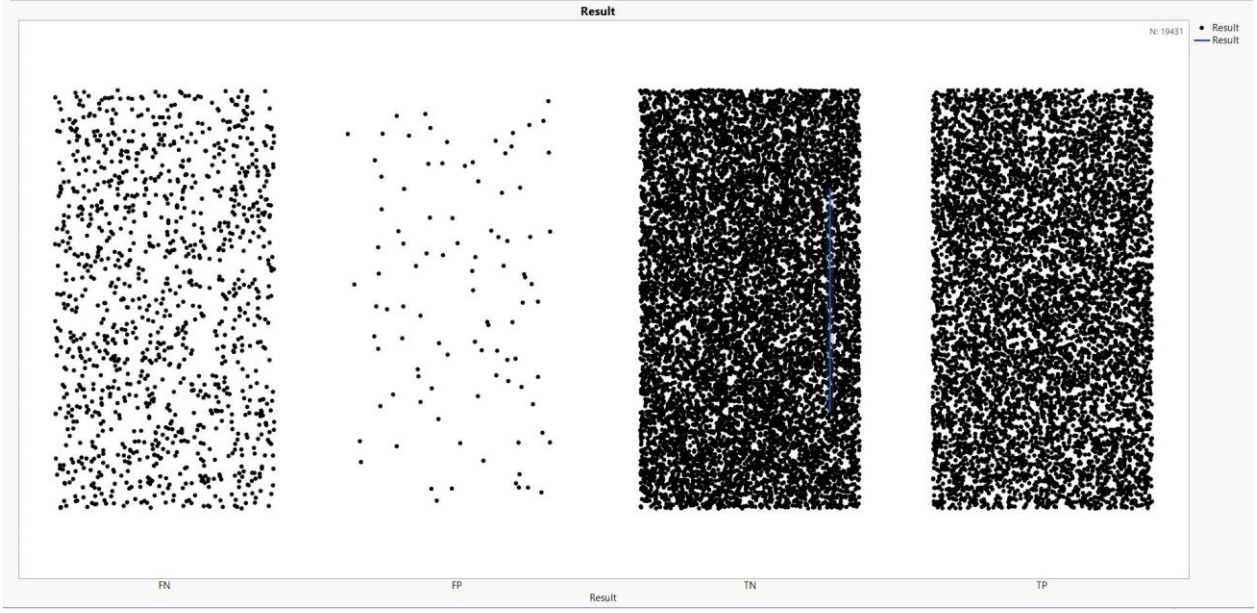


fig. 4. 7 The distribution of FP, FN, TP, TN results of detecting near duplicate images

The above scatter plot shows the distribution of finding False positive, True positive, False negative and True negative images by executing our algorithm with 2000 query images. This visualization demonstrate that the number of True positive and True negative is significantly higher than number of False Negative and False positive which verify the accuracy of the algorithm. Most of the False negatives returned from the algorithm, are flipping attacks.

The following table shows the result of an execution with 2000 query images where the greatest number of False negatives are Flipped attacks that were rejected by the has

algorithm so never passed through another fingerprints filtering. As it is demonstrated in the table the reference list returned the image names with the type of attacks so we could easily track the reason of the False negative results.

Query Image	Candidate	Source	Resul	Time
000000109827_original.cc.in.jpg	000000109827_flippedlr.jpg	RejectedByTree	FN	0.016685
000000109827_original.cc.in.jpg	000000109827_flippedtb.jpg	RejectedByTree	FN	0.016685
000000173183_original.cc.in.jpg	000000173183_flippedlr.jpg	RejectedByTree	FN	0.017894
000000173183_original.cc.in.jpg	000000173183_flippedtb.jpg	RejectedByTree	FN	0.017894
000000162732_original.cc.in.jpg	000000162732_flippedlr.jpg	RejectedByTree	FN	0.024295
000000162732_original.cc.in.jpg	000000162732_flippedtb.jpg	RejectedByTree	FN	0.024295
000000321333_original.cc.in.jpg	000000321333_cropped10.jpg	RejectedByTree	FN	0.020244
000000321333_original.cc.in.jpg	000000321333_flippedlr.jpg	RejectedByTree	FN	0.020244
000000321333_original.cc.in.jpg	000000321333_flippedtb.jpg	RejectedByTree	FN	0.020244
000000476258_original.cc.in.jpg	000000476258_flippedlr.jpg	RejectedByTree	FN	0.032352
000000476258_original.cc.in.jpg	000000476258_flippedtb.jpg	RejectedByTree	FN	0.032352
000000403353_original.cc.in.jpg	000000403353_flippedlr.jpg	RejectedByTree	FN	0.014432
000000403353_original.cc.in.jpg	000000403353_flippedtb.jpg	RejectedByTree	FN	0.014432
000000066817_original.cc.in.jpg	000000066817_cropped10.jpg	RejectedByTree	FN	0.027235
000000066817_original.cc.in.jpg	000000066817_flippedlr.jpg	RejectedByTree	FN	0.027235
000000066817_original.cc.in.jpg	000000066817_flippedtb.jpg	RejectedByTree	FN	0.027235
000000299720_original.cc.in.jpg	000000299720_flippedlr.jpg	RejectedByTree	FN	0.017077
000000299720_original.cc.in.jpg	000000299720_flippedtb.jpg	RejectedByTree	FN	0.017077
000000509699_original.cc.in.jpg	000000509699_flippedlr.jpg	RejectedByTree	FN	0.02625
000000509699_original.cc.in.jpg	000000509699_flippedtb.jpg	RejectedByTree	FN	0.02625
000000067180_original.cc.in.jpg	000000067180_flippedlr.jpg	RejectedByTree	FN	0.019976
000000067180_original.cc.in.jpg	000000067180_flippedtb.jpg	RejectedByTree	FN	0.019976
000000050896_original.cc.in.jpg	000000050896_flippedlr.jpg	RejectedByTree	FN	0.0186
000000050896_original.cc.in.jpg	000000050896_flippedtb.jpg	RejectedByTree	FN	0.0186
000000311909_original.cc.in.jpg	000000311909_flippedlr.jpg	RejectedByTree	FN	0.034166
000000311909_original.cc.in.jpg	000000311909_flippedtb.jpg	RejectedByTree	FN	0.034166
000000338325_original.cc.in.jpg	000000338325_flippedlr.jpg	RejectedByTree	FN	0.025642
000000338325_original.cc.in.jpg	000000338325_flippedtb.jpg	RejectedByTree	FN	0.025642
000000450758_original.cc.in.jpg	000000450758_flippedtb.jpg	RejectedByTree	FN	0.018043
000000440336_original.cc.in.jpg	000000440336_flippedlr.jpg	RejectedByTree	FN	0.02279
000000440336_original.cc.in.jpg	000000440336_flippedtb.jpg	RejectedByTree	FN	0.02279
000000043816_original.cc.in.jpg	000000043816_flippedlr.jpg	RejectedByTree	FN	0.018689
000000043816_original.cc.in.jpg	000000043816_flippedtb.jpg	RejectedByTree	FN	0.018689
000000336309_original.cc.in.jpg	000000336309_flippedlr.jpg	RejectedByTree	FN	0.012019
000000336309_original.cc.in.jpg	000000336309_flippedtb.jpg	RejectedByTree	FN	0.012019
000000330369_original.cc.in.jpg	000000330369_flippedlr.jpg	RejectedByTree	FN	0.025621
000000330369_original.cc.in.jpg	000000330369_flippedtb.jpg	RejectedByTree	FN	0.025621

Table. 4. 32 List of False negative results show most of them had flipping attack

Data science tools and libraries

Since the output of our algorithm are several large series of number and in validation process this amounts and data series grows, to control and compare results we used some data since tools such as Pandas to generate table of results automatically in form of Excel

document. Also, to analyzing results we used different libraries install on python to map the data, sort them and visualize the results.

4.4.7 Software and user interface

To provide ease of use our near-duplicate detection method, we created a graphic user interface which help users choose the different value for threshold or entering the amount of query images to test the algorithm. As shown in following figures.

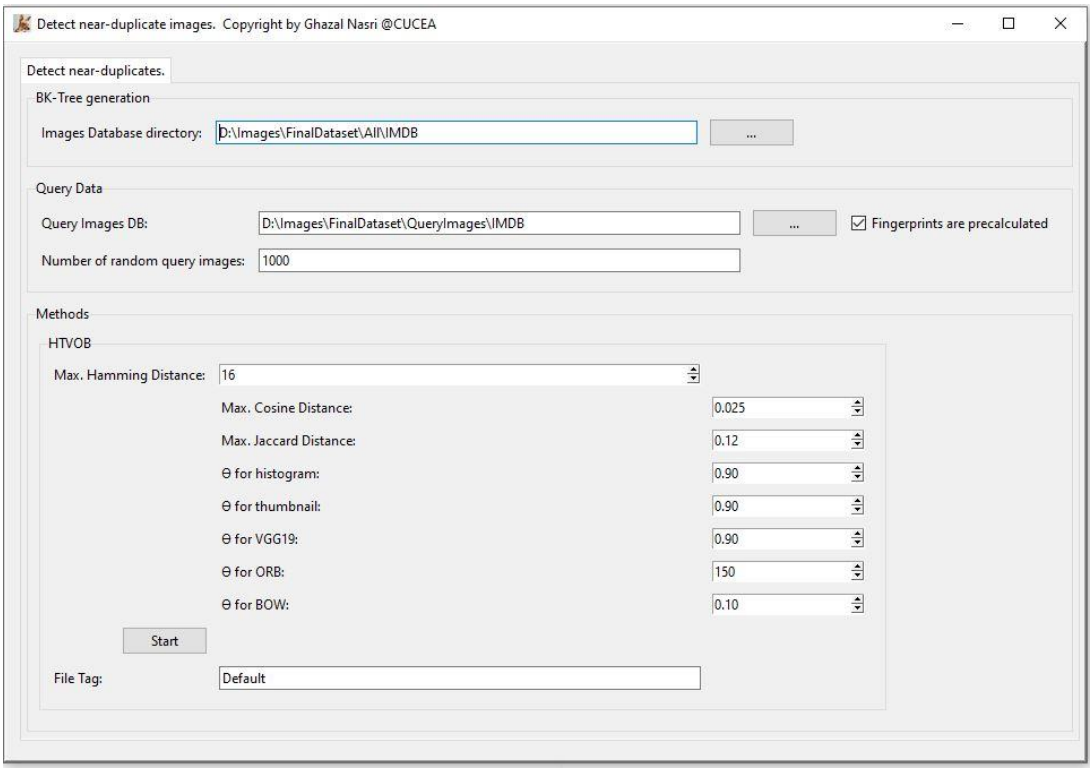


fig. 4. 8 The Near duplicate detection system user interface

The user can choose images from the database then generate its hash fingerprints and store the fingerprint in a BK-Tree as shown in following figure:

fig. 4.9 Choose the images from database and generate it hash fingerprint

in the next step they can select the query images from any other places located at hard disk or database in order to detect a near duplicate version of image with query figure as following:

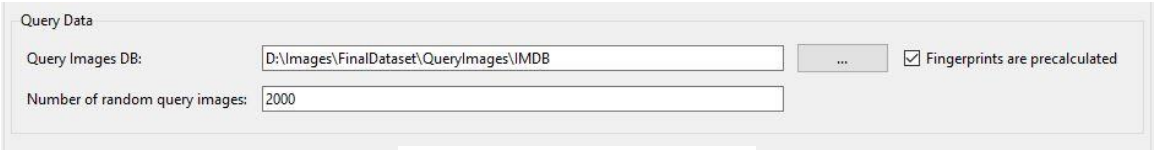


fig. 4.10 Select query image

Also, the user can select the desire threshold for each metric distance also can set up the threshold of each fingerprint similarity in order to detect the near duplicate images from the database. Following figure show these setting via the interface.

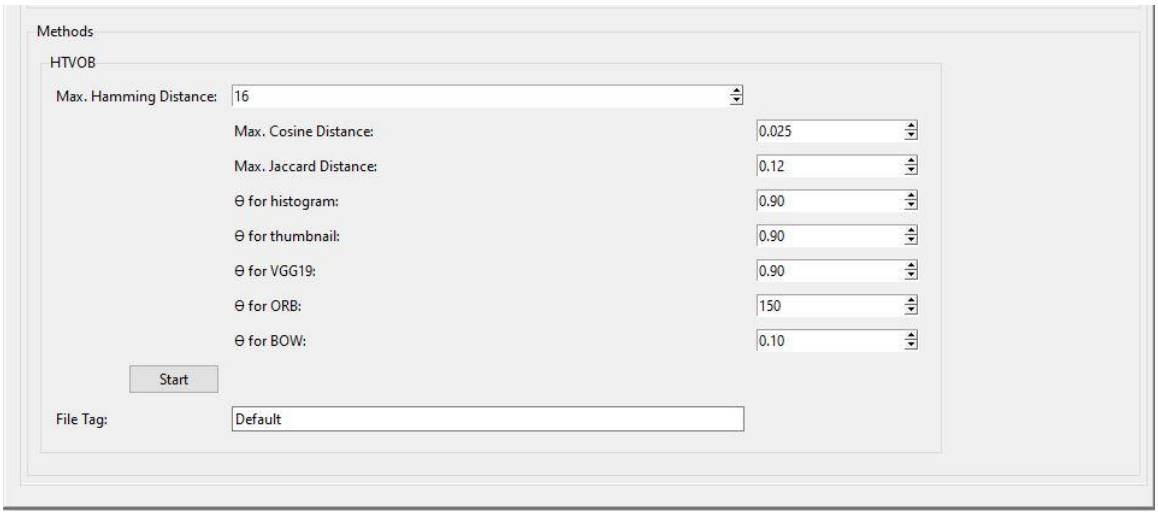
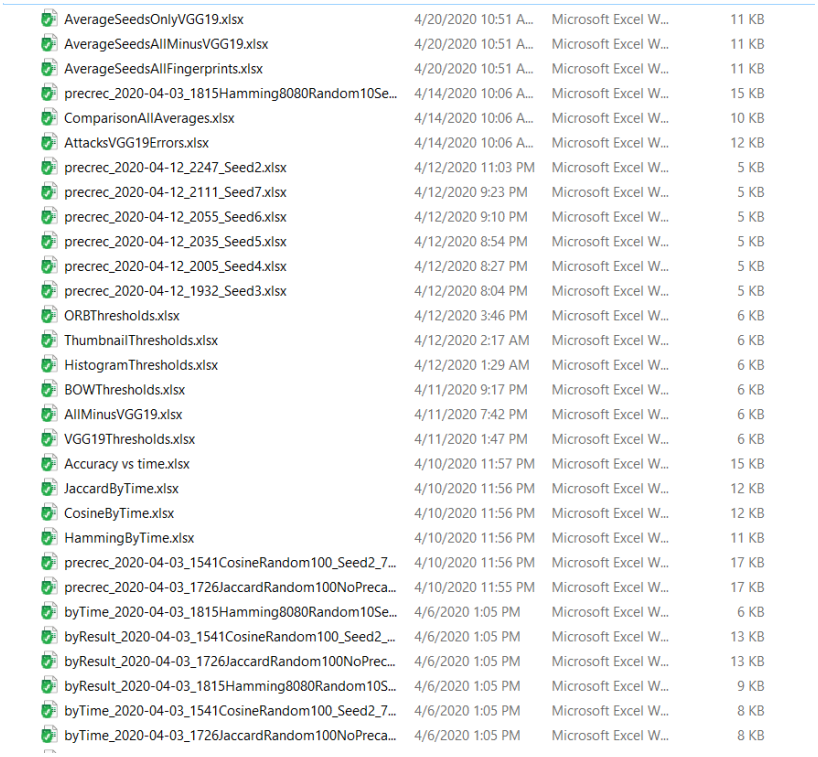


fig. 4.11 Select the detecting near duplicate method

after selecting an appropriate threshold value for each fingerprint, we could choose the location to store table of results and run the algorithm by start button. The following figure shows a location where some of our table result was store in form of Excel documents.



AverageSeedsOnlyVGG19.xlsx	4/20/2020 10:51 A...	Microsoft Excel W...	11 KB
AverageSeedsAllMinusVGG19.xlsx	4/20/2020 10:51 A...	Microsoft Excel W...	11 KB
AverageSeedsAllFingerprints.xlsx	4/20/2020 10:51 A...	Microsoft Excel W...	11 KB
precrec_2020-04-03_1815Hamming8080Random10Se...	4/14/2020 10:06 A...	Microsoft Excel W...	15 KB
ComparisonAllAverages.xlsx	4/14/2020 10:06 A...	Microsoft Excel W...	10 KB
AttacksVGG19Errors.xlsx	4/14/2020 10:06 A...	Microsoft Excel W...	12 KB
precrec_2020-04-12_2247_Seed2.xlsx	4/12/2020 11:03 PM	Microsoft Excel W...	5 KB
precrec_2020-04-12_2111_Seed7.xlsx	4/12/2020 9:23 PM	Microsoft Excel W...	5 KB
precrec_2020-04-12_2055_Seed6.xlsx	4/12/2020 9:10 PM	Microsoft Excel W...	5 KB
precrec_2020-04-12_2035_Seed5.xlsx	4/12/2020 8:54 PM	Microsoft Excel W...	5 KB
precrec_2020-04-12_2005_Seed4.xlsx	4/12/2020 8:27 PM	Microsoft Excel W...	5 KB
precrec_2020-04-12_1932_Seed3.xlsx	4/12/2020 8:04 PM	Microsoft Excel W...	5 KB
ORBThresholds.xlsx	4/12/2020 3:46 PM	Microsoft Excel W...	6 KB
ThumbnailThresholds.xlsx	4/12/2020 2:17 AM	Microsoft Excel W...	6 KB
HistogramThresholds.xlsx	4/12/2020 1:29 AM	Microsoft Excel W...	6 KB
BOWThresholds.xlsx	4/11/2020 9:17 PM	Microsoft Excel W...	6 KB
AllMinusVGG19.xlsx	4/11/2020 7:42 PM	Microsoft Excel W...	6 KB
VGG19Thresholds.xlsx	4/11/2020 1:47 PM	Microsoft Excel W...	6 KB
Accuracy vs time.xlsx	4/10/2020 11:57 PM	Microsoft Excel W...	15 KB
JaccardByTime.xlsx	4/10/2020 11:56 PM	Microsoft Excel W...	12 KB
CosineByTime.xlsx	4/10/2020 11:56 PM	Microsoft Excel W...	12 KB
HammingByTime.xlsx	4/10/2020 11:56 PM	Microsoft Excel W...	11 KB
precrec_2020-04-03_1541CosineRandom100_Seed2_7...	4/10/2020 11:56 PM	Microsoft Excel W...	17 KB
precrec_2020-04-03_1726JaccardRandom100NoPreca...	4/10/2020 11:55 PM	Microsoft Excel W...	17 KB
byTime_2020-04-03_1815Hamming8080Random10Se...	4/6/2020 1:05 PM	Microsoft Excel W...	6 KB
byResult_2020-04-03_1541CosineRandom100_Seed2_...	4/6/2020 1:05 PM	Microsoft Excel W...	13 KB
byResult_2020-04-03_1726JaccardRandom100NoPrec...	4/6/2020 1:05 PM	Microsoft Excel W...	13 KB
byResult_2020-04-03_1815Hamming8080Random10S...	4/6/2020 1:05 PM	Microsoft Excel W...	9 KB
byTime_2020-04-03_1541CosineRandom100_Seed2_7...	4/6/2020 1:05 PM	Microsoft Excel W...	8 KB
byTime_2020-04-03_1726JaccardRandom100NoPreca...	4/6/2020 1:05 PM	Microsoft Excel W...	8 KB

fig. 4.12 table of results

Having these tables significantly increased the tracking process of algorithm performance. We specified each execution with their parameters automatically this ability can speed-up the controlling process for users whom will use our algorithm to detect near duplicate images in their images database.

4.5 Comparison of the proposed method and pure-deep learning method

To compare the present work performance with another method in state of the art, we used a pre- trained deep learning algorithm on the same database to detect near duplicate images. Then we compared both method advantages and disadvantages. The main motivation for this comparison is to demonstrate that, even though pure-deep learning methods are well known and modern approaches for content based image retrieval CBIR with many applications to solve different computer vision problems, our method can still compare favorably to pure-deep learning for the specific problem that we want to solve: Detection of near-duplicate images.

The proposed method has already been discussed in detail, so in this section we will give a quick overview of the pure-deep learning method that is used to detect near-duplicate images as well.

We used ResNet50 (He 2016) which is a pre-trained deep learning model combine with the KNN (K-Nearest-Neighbors) algorithm to detect near duplicate images. To prived the same situation for both method (our method and pure-deep learning method) the same hardware configuration and execute the algorithm based on our Native database and Query image database.

The following steps was done to detect near duplicate images by our pure-deep learning method using ResNet50.

1. Uses a KNN (K-Nearest Neighbors) algorithm for selection of the candidate images.
2. The initial candidates image list is performed by specifying two parameters:

a. A query image (represented by a features vector). The vector is obtained by providing the image as input to a CNN (Convolutional Neural Network) that has been previously trained for image classification. As mentioned in previous chapters, in order to use the model as a feature extractor, the top classification layers must be removed from the model before processing the image.
b. Second parameter is a number k of the Nearest Neighbors that we want

3. The search will produce a list of the k Nearest Neighbors. The important characteristic of this approach is that there is no limit for the distance from the query image to the neighbor. In practice, this may produce candidates that are far from the query image and are not truly near duplicate images.

4.5.1 Resources consumption

The search phase of our proposed method uses a 32-byte hash to represent an image. This is stored as an integer value, thus is very efficient in terms of memory consumption. The candidate's evaluation phase uses 5 fingerprints. Some of them take up more memory space such as ORB fingerprint, but only a limited number of fingerprints are loaded into memory.

Since our code is a Python application, the amount of memory occupied by the corresponding process reflects to the amount of memory occupied by the BK-Tree. The following figure show the total memory consumption of a BK-Tree with 200,000 fingerprints in memory.

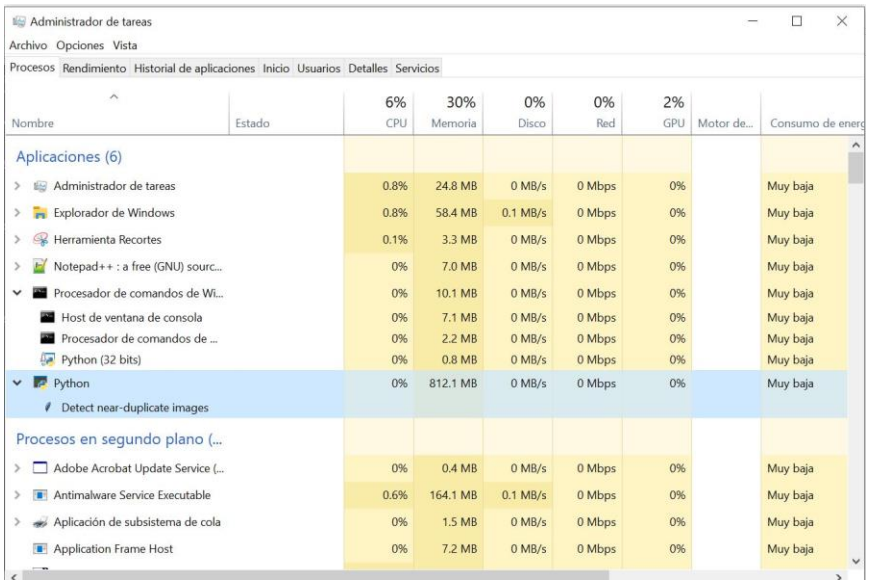


fig. 4. 13 The amount of memory space during execution of our method

as it is demonstrated in the above figure the complete set of 200,000 hashes (representing images) takes 812.1 Mb in memory space which is very reasonable amount for memory consumption.

For the pure-deep learning method, each features vector contains 2048 floating point values. Each floating-point value can occupy 32 or 64 bits depending on the architecture. Taking the best-case scenario, a features vector for a single image in pure-deep learning method would occupy: $2048 \times 4 = 8192$ bytes (8.2 Kb)

The following figure shows the memory consumption for a more limited dataset of 18,000 images. The reason we had to use a limited size dataset will be apparent immediately:

The amount of memory taken up by the Python process for 18,000 features in memory, is considerably bigger than memory consumption executing our method.

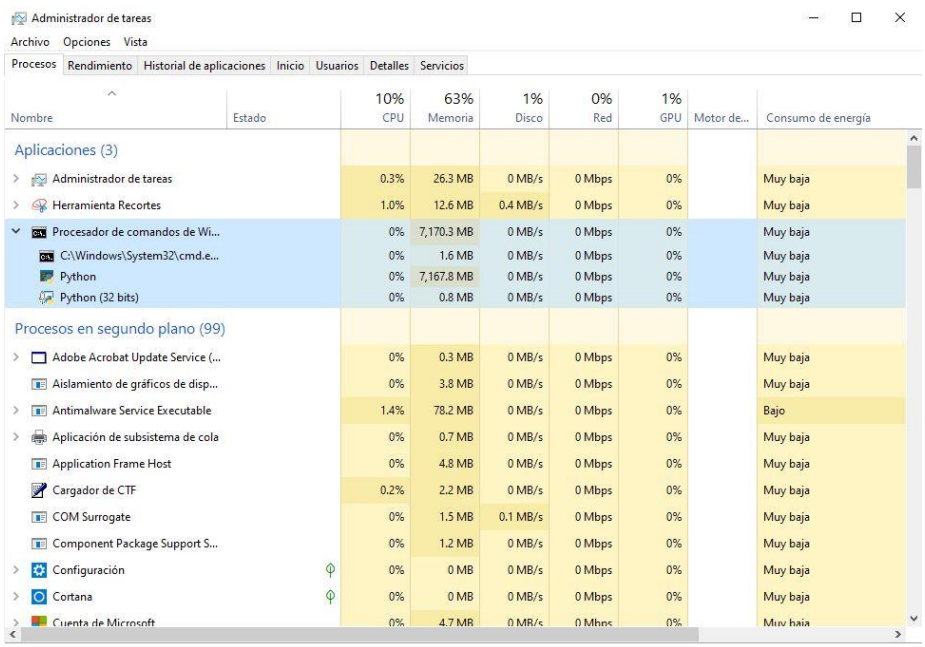


fig. 4. 14 The amount of memory space during execution of pure-deep learning

as it shown in the figure the memory space for executing pure-deep learning algorithm is over 7 Gb for the process and the features.

4.5.2 Results from pure-deep learning method

The K-Nearest Neighbors algorithm receives a vector of features as input (representing a query image) and a number K: which defines the number of neighbors. So, the output will be a list of those K neighbors, along with their distances:

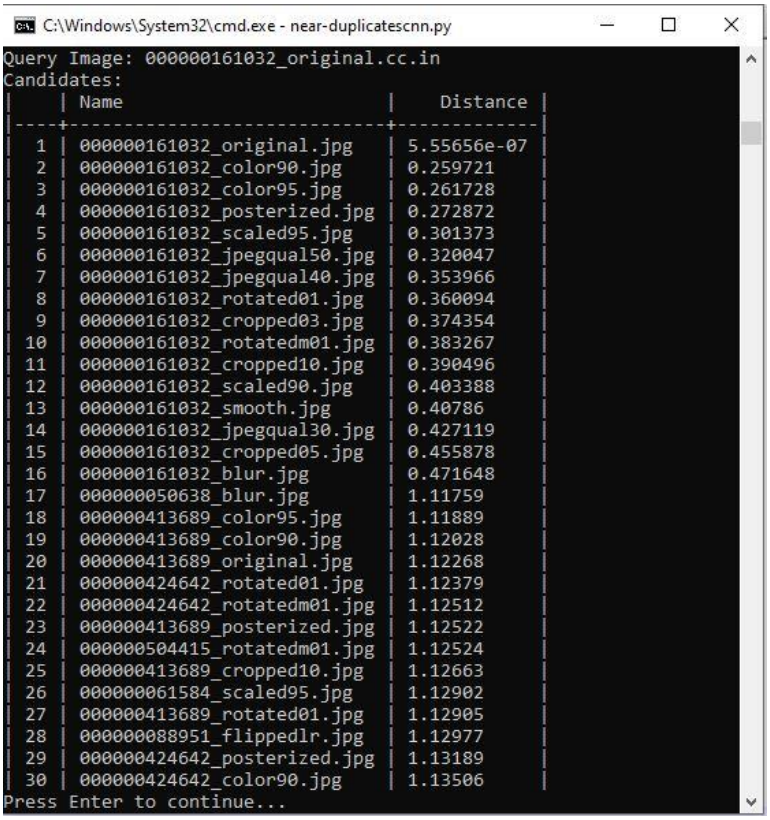


fig. 4. 15 The list of K-Nearest Neighbors to the candidate images

The K-Nearest Neighbors algorithm will return a list of K candidates regardless of their distance from each query images. This means that some of these candidates have a large distance, so they cannot be considered as near duplicates. observing manually the images from above list we found that number 17 to 30 are not near duplicate images. Hence just some of the images from the K list are near duplicate.

We need to establish a threshold value, in the same way that we did in our method for finding the similarity measure between the fingerprints. So, we defined a threshold value as *Th*. The distance value should be less or equal to *Th* value in order to return a positive result which is a near duplicate image to the query, otherwise the result is a negative match.

In this implementation we also tried several threshold values and random seed to get the best and optimize value for each of them which provide better Precision, Recall and accuracy measures. The following table shows results execution via sending 100 query images to the algorithm:

Thr	Seed	TP	FP	TN	FN	PR	RE	F1	Acc	Averge Match Time
1.1	2	613	132	1743	35	0.822819	0.945988	0.654216	0.933809	17.618152

Table. 4. 33 results execution via sending 100 query images

4.5.3 Execution both methods

We execute both methods, the pure-deep learning and our method presented in this work, under the same hardware configuration on one database with the same image attacks to compare the results. Following table shows the comparing result:

Method	Metric	Max.Dist	Hi.Thr	Th.Thr	V.Th	ORB.Thr	BOW.Thr	TP	FP	TN	FN	PR	RE	F1	Accuracy	Average Match Time	Our method
OR	Hamming	16	0.89	0.9	0.75	17	0.02	570	0	1711	78	1.000000	0.879630	0.637584	0.966935	0.406900	
							Thr	TP	FP	TN	FN	PR	RE	F1	Acc	Average Match Time	Pure_deep Learning
							1.1	613	132	1743	35	0.822819	0.945988	0.654216	0.933809	17.618152	

Table. 4. 34 Comparing result of both methods

As is shown in the result table the average time of finding a match for each image in our method is significantly lower than pure-deep learning method (more than 40 times). Also, the value of Accuracy and Precision in our method is higher than the pure-deep learning method, but pure-deep learning method has better Recall value.

Hence, based on obtained result we can conclude that our method has a balance between accuracy and time, comparing to the pure-deep learning method its faster and provide comparable result to detect near duplicate images.

In terms of memory consumption our method is much more efficient than pure-deep learning method because KNN requires storing all the features extract form image in the memory at the same time, since the features extracting buy pure deep learning has a large size storing them in memory for use of KNN requires either having more memory or reduce the number of images that we retrieving for each execution.

In pure- deep learning method the size of features is often larger than the size of image however in our method the size of fingerprints is very lightweight and fast to retrieve.

One of the advantages of our method comparing to the pure-deep learning method is that we once in offline stage calculate all fingerprints and store them in FP database so each time new query images come to the system first their fingerprints will be obtained then they will be compared with the fingerprints existing in FP database. This process decreases significantly the time execution and resource consumption. However, in pure-deep learning method if the model is not already trained.

Our method doesn't have any dependency to the size of data for retrieve nor to the hardware any home used system can be used to execute and detect near duplicate images from a large image dataset. however, the pure-deep learning method requires high performing hardware which increases cost to the users to deal with image data.

Also, pure-deep learning algorithms has a great result on categorical database, but our method can retrieve data images from uncategorical data with the same result and performance.

Chapter 5.

5.1 Conclusions

In this dissertation we present a fast and low-cost method to detect near duplicate images in a large dataset by generating six different fingerprints which are a combination of filtering different image attacks. Each fingerprint is an extraction of image features and store in fingerprint database This method provided us a great balance between execution time of detecting near duplicate image and the accuracy. We used both local and global features and generated six different type of fingerprints, we also used deep learning method to generate one of our fingerprints, later we used a matching system to compare the fingerprints results with the fingerprint of query images in order to find the perfect match of near duplicate image to the query images. However recently in Content Based Image Retrieval CBIR, deep learning algorithms are widely used as unique method but our experimental results compared to the state-of-the-art shows that deep learning algorithms by themselves need a lot of data training and have high false positives in detection of near duplicate images, this rates grows especially when images are very close to each other or have a vast similarity. The deep learning process need to retrain data with a large number of examples, specially of near duplicate types so they can learn to detect duplicates. However, in the proposed method we don't need to do retraining or even need large amount of data to make the algorithm understand about the images because we narrow down images by combination of BK-tree and hashing and start filter images via other

fingerprints. Each of our fingerprints has robustness against different attack images this helps our algorithms in further stages enhancing accuracy faster time of detection.

Our method is based on lightweight fingerprint which are fast to retrieve, by adding several fingerprints we add more accuracy to the algorithm for detecting near duplicate images.

In the experimental work we tried a pure-deep learning method using a pre-trained algorithm for detect near duplicate images in the same database and we compared the results. Experimental shows however pure-deep learning algorithm returned a competitive accuracy to our method, but it was so slow in execution time also it needs very high hardware potential. Thus, pure-deep learning method to detect near duplicate images has longer time for execution and it is computationally expensive due to their natural of having several training processes. Such methods should be complemented with other approaches such as using global features as hashing, histogram, thumbnail and using a fuzzy search method to discard not- near duplicate images at the first stage before consuming the computer or server resource for training process. In this research by conjunction the deep learning as a fingerprint with other robust image retrieval algorithm; we obtained not only the accuracy increases, but also the retrieval time without the need of train the algorithm and spend time as well resources for that. Hence, we don’t need a large amount of data to retrain or train either.

Our approach improves the accuracy of near duplicate detection images with a great compromise with the time execution.

Another advantage of our method is that can be use in any image database for experimental results we used an uncategorized database (Native database) which is more complicated to

retrieve, including a vast variety of image type, size from 201×251 to 4000×3800, resolutions, format and scene complexity with a balanced image type distribution.

Our experimental results show the proposed method can detect near duplicate images by the error rate of 5% in uncategorical data images by sending 2000 query images at the same time and return the near duplicate images on the fly. This error rate can be changed according to type of the data and the quantity of data which are both factors of result analyzing. The present work is not only used for finding near duplicate images but also can benefit to detect forged images which are difficult to discover with unaided eye, specific in massive number of images in large database.

Our algorithm and method don't have hardware dependency even to retrieve images from a large dataset. Whole algorithm in large database was executed on a laptop Intel i7 processor at 1.8 GHz and 16 GB Memory RAM. In the other hand, we were not able to execute and retrieve the same amount of data via pure-deep learning algorithm alone which requires high performance hardware.

5.2 Future Work

We applied our experimental in a large image dataset with 200000 images which had different size, format and resolutions. In future work we will scale up the number of images to one million in an online database with a public access for researchers.

We also explore the image classification and store fingerprints in FP dataset in specific category. This can help researchers to have a large data categorical fingerprint database. Adding more image attacks to cover copyright regulation is considered as well.

Compensate for the weakness of our first fingerprint (Hash) to reduce the number of False negative, look for another alternative in order to provide more robustness to the near duplicate detecting method.

In the present work we implemented our algorithm based on RGB (Red, Green, Blue) color space, adding more color space and image format could be part of future work. We had hardware limitation to implement and exercise our algorithm, execute and improve the algorithm via specialized hardware can increase performance and results.

Appendix

Symbols and abbreviations

Δ(q, d) Hamming Distance

KNN (K-Nearest Neighbors)

Max.Dist

Hi.Thr Histogram threshold

Th.Thr Thumbnail threshold

V.Thr Vgg19 threshold

ORB.Thr Oriented FAST and rotated BRIEF threshold

BOW.Thr The Bag of Visual word threshold

P Precision

R Recall

-R Real negative

PR is precision value

RE is for recall value

F1 is F-score value

FPR is false positive rate

TPR is true positive rate

NDI: Near duplicate Image

IOT: Internet of things

BOF Bag of Features

FP False Positive

FPA False Positive Average

FPR False Positive Rate

FN False Negative

FNA False Negative Average

FV Fisher Vector

k-d tree k-dimensional tree

K-NN K-Nearest Neighbor

NDID Near-Duplicate Image Detection

NN Nearest Neighbor

ORB Oriented FAST and Rotated BRIEF fast robust feature detector

OSS Open Source Software

PCA Principal Component Analysis

px A pixel

PR Precision-Recall

RGB red, green, blue color space

ROC Receiver Operating Characteristic

SDC Symmetric Distance Computation

SIFT Scale Invariant Feature Transform

SSR Signed Square Root normalization

SURF Sped Up Robust Features

tf-idf term frequency - inverse document frequency

TP True Positive

TNR True Negative Rate

TPR True Positive Rate

TN True Negative

UA User Agent facilitates end user interaction with Web content (Browser)

UI User Interface

URI Uniform Resource Identifier aka URL

Appendix1

Method	Metric	Max.Dist	Hi.Thr	Th.Thr	V.Th	ORB.Thr	BOW.Thr	Time
OR	Cosine	0.025	0.83	0.83	0.4	150	0.1	1.88662
				0.84	0.4	150	0.1	2.17751
				0.85	0.4	150	0.1	2.26077
			0.84	0.83	0.4	150	0.1	2.86903
				0.84	0.4	150	0.1	1.81721
				0.85	0.4	150	0.1	1.93953
			0.85	0.83	0.4	150	0.1	1.69307
				0.84	0.4	150	0.1	1.64257
				0.85	0.4	150	0.1	1.69446
	Hamming	16	0.83	0.83	0.4	150	0.1	0.41881
				0.84	0.4	150	0.1	0.37775
				0.85	0.4	150	0.1	0.39309
			0.84	0.83	0.4	150	0.1	0.48095
				0.84	0.4	150	0.1	0.53343
				0.85	0.4	150	0.1	0.40287
			0.85	0.83	0.4	150	0.1	0.39302
				0.84	0.4	150	0.1	0.60534
				0.85	0.4	150	0.1	0.42482
	Jaccard	0.12	0.83	0.83	0.4	150	0.1	0.83002
				0.84	0.4	150	0.1	0.70284
				0.85	0.4	150	0.1	0.63305
			0.84	0.83	0.4	150	0.1	0.67264
				0.84	0.4	150	0.1	0.86697
				0.85	0.4	150	0.1	0.88821
			0.85	0.83	0.4	150	0.1	0.93514
				0.84	0.4	150	0.1	0.8708
				0.85	0.4	150	0.1	0.7309

Appendix2

Method	Metric	Max.Dist	Hi.Thr	Th.Thr	V.Th	ORB.Thr	BOW.Thr	TP	FP	TN	FN	PR	RE	F1	FPR	TPR
OR	Cosine	0.025	0.83	0.83	0.4	150	0.1	183	1	786	51	0.99457	0.78205	0.61	0.00127	0.78205
OR	Cosine	0.025	0.83	0.84	0.4	150	0.1	366	2	1572	102	0.99457	0.78205	0.61	0.00127	0.78205
OR	Cosine	0.025	0.83	0.85	0.4	150	0.1	549	3	2358	153	0.99457	0.78205	0.61	0.00127	0.78205
OR	Cosine	0.025	0.84	0.83	0.4	150	0.1	732	4	3144	204	0.99457	0.78205	0.61	0.00127	0.78205
OR	Cosine	0.025	0.84	0.84	0.4	150	0.1	915	5	3930	255	0.99457	0.78205	0.61	0.00127	0.78205
OR	Cosine	0.025	0.84	0.85	0.4	150	0.1	1098	6	4716	306	0.99457	0.78205	0.61	0.00127	0.78205
OR	Cosine	0.025	0.85	0.83	0.4	150	0.1	1281	7	5502	357	0.99457	0.78205	0.61	0.00127	0.78205
OR	Cosine	0.025	0.85	0.84	0.4	150	0.1	1464	8	6288	408	0.99457	0.78205	0.61	0.00127	0.78205
OR	Cosine	0.025	0.85	0.85	0.4	150	0.1	1647	9	7074	459	0.99457	0.78205	0.61	0.00127	0.78205
OR	Hamming	16	0.83	0.83	0.4	150	0.1	201	2	1000	33	0.99015	0.85897	0.63208	0.002	0.85897
OR	Hamming	16	0.83	0.84	0.4	150	0.1	402	4	2000	66	0.99015	0.85897	0.63208	0.002	0.85897
OR	Hamming	16	0.83	0.85	0.4	150	0.1	603	6	3000	99	0.99015	0.85897	0.63208	0.002	0.85897
OR	Hamming	16	0.84	0.83	0.4	150	0.1	804	8	4000	132	0.99015	0.85897	0.63208	0.002	0.85897
OR	Hamming	16	0.84	0.84	0.4	150	0.1	1005	10	5000	165	0.99015	0.85897	0.63208	0.002	0.85897
OR	Hamming	16	0.84	0.85	0.4	150	0.1	1206	12	6000	198	0.99015	0.85897	0.63208	0.002	0.85897
OR	Hamming	16	0.85	0.83	0.4	150	0.1	1407	13	7001	231	0.99085	0.85897	0.63208	0.00185	0.85897
OR	Hamming	16	0.85	0.84	0.4	150	0.1	1608	14	8002	264	0.99137	0.85897	0.63208	0.00175	0.85897
OR	Hamming	16	0.85	0.85	0.4	150	0.1	1809	15	9003	297	0.99178	0.85897	0.63208	0.00166	0.85897
OR	Jaccard	0.12	0.83	0.83	0.4	150	0.1	200	1	791	34	0.99502	0.8547	0.63091	0.00126	0.8547
OR	Jaccard	0.12	0.83	0.84	0.4	150	0.1	400	2	1582	68	0.99502	0.8547	0.63091	0.00126	0.8547
OR	Jaccard	0.12	0.83	0.85	0.4	150	0.1	600	3	2373	102	0.99502	0.8547	0.63091	0.00126	0.8547
OR	Jaccard	0.12	0.84	0.83	0.4	150	0.1	800	4	3164	136	0.99502	0.8547	0.63091	0.00126	0.8547
OR	Jaccard	0.12	0.84	0.84	0.4	150	0.1	1000	5	3955	170	0.99502	0.8547	0.63091	0.00126	0.8547
OR	Jaccard	0.12	0.84	0.85	0.4	150	0.1	1200	6	4746	204	0.99502	0.8547	0.63091	0.00126	0.8547
OR	Jaccard	0.12	0.85	0.83	0.4	150	0.1	1400	6	5538	238	0.99573	0.8547	0.63091	0.00108	0.8547
OR	Jaccard	0.12	0.85	0.84	0.4	150	0.1	1600	6	6330	272	0.99626	0.8547	0.63091	0.00095	0.8547
OR	Jaccard	0.12	0.85	0.85	0.4	150	0.1	1800	6	7122	306	0.99668	0.8547	0.63091	0.00084	0.8547

Appendix3

Method	Metric	Max.Dist	Hi.Thr	Th.Thr	V.Th	ORB.Thr	BOW.Thr	Match	Result	Candidates Evaluated
OR	Cosine	0.025	0.83	0.83	0.4	150	0.1	N	FN	118
									TN	787
								P	TP	116
				0.84	0.4	150	0.1	N	FN	119
									TN	787
								P	TP	115
				0.85	0.4	150	0.1	N	FN	119
									TN	787
								P	TP	115
			0.84	0.83	0.4	150	0.1	N	FN	118
									TN	787
								P	TP	116
				0.84	0.4	150	0.1	N	FN	119
									TN	787
								P	TP	115
				0.85	0.4	150	0.1	N	FN	119
									TN	787
								P	TP	115
			0.85	0.83	0.4	150	0.1	N	FN	118
									TN	787
								P	TP	116
				0.84	0.4	150	0.1	N	FN	119
									TN	787
								P	TP	115
				0.85	0.4	150	0.1	N	FN	119
									TN	787
								P	TP	115
	Hamming	16	0.83	0.83	0.4	150	0.1	N	FN	109
									TN	1002
								P	TP	125
				0.84	0.4	150	0.1	N	FN	110
									TN	1002
								P	TP	124
				0.85	0.4	150	0.1	N	FN	110
									TN	1002
								P	TP	124
			0.84	0.83	0.4	150	0.1	N	FN	109
									TN	1002
								P	TP	125

Bibliography

A. K. Layek, A. Gupta, S. Ghosh and S. Mandal. 2016. "Fast near-duplicate detection from image streams on online social media during disaster events." *IEEE Annual India Conference (INDICON)* 1-6.

ADOBE, DIGITAL INDEX. 2014. *Social Intelligence Report.* ADOBE.

Bay, A. Ess, T. Tuytelaars, L. Van Gool,. 2008. "SURF:Speeded Up Robust Features." *Computer Vision and Image Understanding (CVIU)* 346-359.

Boren, Zachary Davies. 2014. *THERE ARE OFFICIALLY MORE MOBILE DEVICES THAN PEOPLE IN THE WORLD.* https://www.independent.co.uk.

Brain, Google. 2020. *Google Brain.* Accessed January 19, 2020. https://research.google/teams/brain/.

Chansung, Park. 2018. Jun 6. Accessed March 4, 2020. https://towardsdatascience.com/transfer-learning-in-tensorflow-9e4f7eae3bb4.

Cisco. 2017. *Cisco Visual Networking Index: Forecast and Trends,2017-2020.* White paper, Cisco Public.

Cisco, Visual Networking. 2016. *The zettabyte era–trends and analysis.* Cisco.

Danaci, Emine Gul,Ikizler-Cinbis, Nazli. 2016. "Low-level features for visual attribute recognition: An evaluation." *Pattern Recognition Letters* 185-191.

DVMM LAB, Columbia University. 2015. *Detecting Image Near-Duplicate for Linking Multimedia Content.* December 21. Accessed January 19, 2020. http://www.ee.columbia.edu/ln/dvmm/researchProjects/FeatureExtraction/NearDuplicateByParts/INDDetection.html.

Elnemr, H. A., Zayed, N. M., & Fakhreldein, M. A. 2016. "Feature Extraction Techniques: Fundamental Concepts and Survey." In *Handbook of Research on Emerging Perspectives in Intelligent Pattern Recognition, Analysis, and Image Processing*, by Heba Ahmed, Nourhan Mohamed Zayed and Mahmoud Abdelmoneim Fakhreldein Elnemr, 264-294. IGI Global.

Emine Gul Danaci, Nazli Ikizler-Cinbis. 2016. "Low-level features for visual attribute recognition: An evaluation." *Pattern Recognition Letters,Elsevier* 185-191.

Essays. 2018. *Limitations Of Text Based Image Retrieval Psychology Essay.* November. https://www.ukessays.com/essays/psychology/limitations-of-text-based-image-retrieval-psychology-essay.php?vref=1.

Fridrich, J. and Goljan, M.,. 2000. "Robust hash functions for digital watermarking." *Proceedings International Conference on Information Technology: Coding and Computing,IEEE* 178-183.

Fritz Venter, Andrew Stein. 2012. *Analytics-Magazine.* December. Accessed January 19, 2020. http://analytics-magazine.org/images-a-videos-really-big-data/.

Gandhi, T. Patel and S. 2017. "A survey on context based similarity techniques for image retrieval." *International Conference on Innovative Mechanisms for Industry Applications (ICIMIA)* 219-223.

Guo, G.D., Jain, A.K., Ma, W.Y. and Zhang, H.J. 2002. "Learning similarity measure for natural image retrieval with relevance feedback." *IEEE Transactions on Neural Networks* 811-820.

H. Tang, S. Chu, M. Hasegawa-Johnson and T. Huang. 2012. "Partially Supervised Speaker Clustering." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 959-971.

Hamed Qazanfari, Hamid Hassanpour, Kazem Qazanfari. 2019. "Content-Based Image Retrieval Using HSV Color Space Features." *International Journal of Computer and Information Engineering* 537-545.

Hardesty, Larry. 2015. "Proof that a 40-year-old algorithm is the best possible will come as a relief to computer scientists." *MIT News Office* http://news.mit.edu/2015/algorithm-genome-best-possible-0610.

He, K., Zhang, X., Ren, S., & Sun, J. 2016. "Deep Residual Learning for Image Recognition." *Proceedings of the IEEE conference on computer vision and pattern recognition* pp. 770-778.

Hui Hui Wang, Dzulkifli Mohamad, N.A. Ismail. 2010. "Approaches, Challenges and Future Direction of Image Retrieval." *JOURNAL OF COMPUTING* 193-199.

J. Lee, r. jin, A. Jain and W. Tong. 2012. "Image Retrieval in Forensics: Tattoo Image Database Application." *IEEE MultiMedia* 40-49.

Jabeen S, Mehmood Z, Mahmood T, Saba T, Rehman A, Mahmood MT. 2018. "An effective content-based image retrieval technique for image visuals representation based on the bag-of-visual-words model." *PLoS ONE* .

Jabeen, S., Mehmood, Z., Mahmood, T., Saba, T., Rehman, A., & Mahmood, M. T. 2018. "An effective content-based image retrieval technique for image visuals representation based on the bag-of-visual-words model." *PloS one* 13(4). Jabeen, S., Mehmood, Z., Mahmood, T., Saba, T., Rehman, A., & Mahmood, M. T. (2018). An effective content-based image retrieval technique for image visuals representation based on the bag-of-visual-words model. PloS one, 13(4).

Jaccard, P. 1901. "Distribution de la flore alpine dans le bassin des Dranses et dans quelques régions voisines." *Bulletin de la Société Vaudoise des Sciences Naturelles* 241-272.

Jaccard, P. 1901. "Étude comparative de la distribution florale dans une portion des alpes et du jura." *Bull. Société Vaudoise des Sciences Naturelles* 547-579.

Jain, Margarita N. FavorskayaLakhmi C. 2018. *Computer Vision in Control Systems-4.* Springer International Publishing AG.

Javier A.Montoya Zegarra, Neucimar J.Leite,Ricardoda Silva Torresb. 2009. "Wavelet-based fingerprint image retrieval." *Journal of Computational and Applied Mathematics* 294-307.

Jhansi, Y., & Sreenivasa Reddy, E. 2017. "Sketch Based Image Retrieval with Cosine Similarity." *International Journal of Advanced Research in Computer Science* 8(3).

Jiawei Han, Micheline Kamber, Jian Pei. 2012. "Getting to Know Your Data." *Morgan Kaufmann* 39-82.

Jinsong Wu, Song Guo,Jie Li,Deze Zeng. 2016. "Big Data Meet Green Challenges: Greening Big Data." *IEEE SYSTEMS JOURNAL* 873-887.

K. Juneja, A. Verma, S. Goel and S. Goel. 2015. "A Survey on Recent Image Indexing and Retrieval Techniques for Low-Level Feature Extraction in CBIR Systems." *IEEE International Conference on Computational Intelligence & Communication Technology* 67-72.

Kato, Toshikazu. 1 April 1992. "Database architecture for content-based image retrieval." *Image Storage and Retrieval Systems,https://doi.org/10.1117/12.58497* Proc. SPIE 1662,.

Keller, J. M., Gray, M. R., & Givens, J. A. (). 1985. "A fuzzy k-nearest neighbor algorithm." *IEEE transactions on systems, man, and cybernetics* 580-585.

Kim, S., Wang, X. J., Zhang, L., & Choi, S. 2015. "Near duplicate image discovery on one billion images. ." *IEEE Winter Conference on Applications of Computer Vision* 943-950.

Kosub, Sven. 2019. "A note on the triangle inequality for the Jaccard distance." *Pattern Recognition Letters* 36-38.

L. Yujian, L. Bo. June 2007. "A Normalized Levenshtein Distance Metric." *IEEE Transactions on Pattern Analysis and Machine Intelligence* vol. 29, no. 6, pp. 1091-1095.

LeCun, Y. 1989. "Generalization and network design strategies." *Connectionism in perspective* 143-155.

LeCun, Yann. 2000-2018 . *LeNet.* Accessed March 4, 2020. http://yann.lecun.com/exdb/lenet/.

Levenshtein, V. I. 1966. "Binary Codes Capable of Correcting Deletions, Insertions and Reversals." *Soviet Physics Doklady* 707.

lia Morra, Fabrizio Lamberti. 2019. "Benchmarking unsupervised near-duplicate image detection." *Expert Systems with Applications* 313-326.

Ling, H., Yan, L., Zou, F., Liu, C. and Feng, H. 2013. "Fast image copy detection approach based on local fingerprint defined visual words." *Signal Processing*, 2328-2338.

Ling, H., Yan, L., Zou, F., Liu, C., & Feng, H. 2013. " Fast image copy detection approach based on local fingerprint defined visual words." *Signal Processing* 2328-2338.

Ling, H., Yan, L., Zou, F., Liu, C., & Feng, H. 2013. "Fast image copy detection approach based on local fingerprint defined visual words." *Signal Processing* 2328-2338.

Lingxi Xie., Tian, Q., Zhou, W. and Zhang, B. 2014. "Fast and accurate near-duplicate image search with affinity propagation on the imageweb." *Computer Vision and Image Understanding* 31-41.

Liu, G.H., and J.Y. Yang. 2013. "Content-based image retrieval using color difference histogram." *Pattern* 188–198.

Lowe, D. G. 2004. "Distinctive image features from scale-invariant keypoints." *International journal of computer vision* 91-110.

M. A. Gavrielides, E. Sikudova and I. Pitas. 2006. "Color-Based Descriptors for Image Fingerprinting." *IEEE Transactions on Multimedia* 740-748.

M. F. Sadique, B. Kumar Biswas and S. M. Rafizul Haque. 2019. "Unsupervised Content-Based Image Retrieval Technique Using Global and Local Features." *International Conference on Advances in Science, Engineering and Robotics Technology* 1-6.

Mehmood Z., Anwar S.M., Ali N., Habib H.A., and Rashid M. 2016. "A Novel Image Retrieval Based on a Combination of Local and Global Histograms of Visual Words." *Mathematical Problems in Engineering* 1–12.

Mezaris, V., Kompatsiaris, I., & Strintzis, M. G. . In Proceedings. 2003. "An ontology approach to object-based image retrieval." *International Conference on Image Processing IEEE* II-511.

Michael Calonder, Vincent Lepetit,Christoph StrechaPascal Fua. 2010. "Brief: Binary robust independent elementary features." *European Conference on Computer Vision* 778-792.

Muja, M. and Lowe, D.G.,. 2009. "Fast approximate nearest neighbors with automatic algorithm configuration." *VISAPP* 331-340.

Murala, Subrahmanyam, Anil Balaji Gonde, and R. P. Maheshwari. 2009. "Color and Texture Features for Image Indexing and Retrieval." *Advance Computing Conference, IACC, IEEE International.*

N. Puviarasan, Dr. R. Bhavani,A. Vasnthi. 2014. "Image Retrieval Using Combination of Texture and Shape Features." *International Journal of Advanced Research in Computer and Communication Engineering* 5873-5877.

Neelima Bagri, Punit Kumar Johari. 2015. "A Comparative Study on Feature Extraction using Texture and Shape for Content Based Image Retrieval." *International Journal of Advanced Science and Technology* (International Journal of Advanced Science and Technology) 41-52.

Nguyen, H.V. and Bai, L. 2010. "Cosine similarity metric learning for face verification." *In Asian conference on computer vision* 709-720.

Nian, F., Li, T., Wu, X., Gao, Q. and Li, F. 2016. "Efficient near-duplicate image detection with a local-based binary representation." *Multimedia Tools and Applications* 2435-2452.

OpenCV.org. 2020. *Histogram Comparison.* March. Accessed March 4, 2020. https://docs.opencv.org/3.4/d8/dc8/tutorial_histogram_comparison.html.

Oxford, Press Univrsity. 2020. *OED.* Accessed 2020. https://www.oed.com/.

Pourreza Alireza, Kourosh Kiani. 2016. "A partial-duplicate image retrieval method using color-based SIFT." *Electrical Engineering* 1410-1415.

R. Ashraf, K. Bashir, A. Irtaza, and M. T. Mahmood,. 2015. "Content base image retrieval using embedded neural network with bandletizes regions." *Entropy* 3552-3580.

R. W. Hamming. April 1950. "Error detecting and error correcting codes." *in The Bell System Technical Journal* pp. 147-160.

Rosebrock, Adrian. 2014. *pyimagesearch.com.* January 22. Accessed March 4, 2020. https://www.pyimagesearch.com/2014/01/22/clever-girl-a-guide-to-utilizing-color-histograms-for-computer-vision-and-image-search-engines/.

Rosten, E., & Drummond, T. 2006. "Machine learning for high-speed corner detection." *European conference on computer vision Springer* 430-443.

Rounak Kumar Singh, Kailash Patidar,Rishi Kushwah,Sudeesh Chouhan. 2017. "Content based image retrieval system techniques: a review and analysis." *ACCENTS Transactions on Image Processing and Computer Vision.*

Rublee, E., Rabaud, V., Konolige, K. and Bradski, G.R. 2011. "ORB: An efficient alternative to SIFT or SURF." *IEEE International Conference on Computer Vision* 2564-2571.

Rummel, R.J. 1976. *Understanding Correlation,Honolulu: Department of Political Science.* Accessed March 4, 2020. http://www.hawaii.edu/powerkills/UC.HTM.

Salahuddin Unar, Xingyuan Wang. 2019. "Detected text-based image retrieval approach." *IET, The Institution of Engineering and Technology* 515-521.

Schmidhuber, J. 2015. "Deep learning in neural networks: An overview." *Neural networks 61* 85-117.

Shervin, Amidi, and Amidi Afshin. n.d. *Convolutional Neural Networks cheatsheet.* Accessed March 4, 2020. https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks.

Simonyan, K., & Zisserman, A. 2014. "Very deep convolutional networks for large-scale." *arXiv preprint arXiv* 1409-1556.

Simonyan, K., & Zisserman, A. 2014. "Very deep convolutional networks for large-scale image recognition." *arXiv preprint arXiv* 1409.1556. Accessed January 21, 2020. http://www.image-net.org.

Singhal, A. 2001. "Modern information retrieval A brief overview." *IEEE Data Engineering Bulletin* 35-43.

Song, Jingkuan, et al. 2018. "Quantization-based hashing: a general framework for scalable image and video retrieval." *Pattern Recognition* 175-187.

Srivastava, Mayank ,Siddiqui, Jamshed ,Ali, Mohammad. 2019. "A Review of Hashing based Image Copy Detection Techniques." *Cybernetics and Information Technologies* 3-27.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. 2014. "A simple way to prevent neural networks from overfitting." *The journal of machine learning research*, 1929-1958.

Stanford Vision Lab, Stanford University. 2016. *ImageNet.* Accessed March 4, 2020. http://www.image-net.org/.

Swain, M.J., Ballard D.H. 1991. "Color indexing." *Int J Comput Vision* 11-32.

Thyagharajan, K.K., Kalaiarasi, G. 2020. "A Review on Near-Duplicate Detection of Images using Computer Vision Techniques." *Archives of Computational Methods in Engineering*, 1-20.

Ton Kalker, Jaap Haitsma, Job C. Oostveen,. 2001. "Issues with digital watermarking and perceptual hashing." *Multimedia Systems and Applications* 189-197.

Tyagi, Deepanshu. 2019. *Introduction to ORB (Oriented FAST and Rotated BRIEF).* January 1. Accessed March 4, 2020. https://medium.com/analytics-vidhya/introduction-to-orb-oriented-fast-and-rotated-brief-4220e8ec40cf.

Umbaugh, Scott E. 2005. *Computer Imaging Digital Image Analysis and Processing.* CRC Press.

V. N. Gudivada and V. V. Raghavan. 1995. "Content based image retrieval systems." *in Computer* 18-22.

Vassilieva, N. S. 2009. "Content-based Image Retrieval Methods." *Programming and Computer Software* 158–180.

W. A. Burkhard, R. M. Keller. 1973. "Some Approaches to Best-match File Searching." *ACM* 230--236.

Wei Jiang, Kap luk chan, Mingjing Li. 2005. "Mapping low level features to high level semantic concept in region based image retrieval." *IEEE.*

Xin Wang, Jingna Jin,Zhipeng LiuTao Yin. 2017. "Study on Differences of Early-Mid ERPs Induced by Emotional Face and Scene Images." *springer.*

Y. Hu, X. Cheng, L. Chia, X. Xie, D. Rajan and A. Tan. 2009. "Coherent Phrase Model for Efficient Image Near-Duplicate Retrieval." *IEEE Transactions on Multimedia* 1434-1445.

Yeung, Fei-Fei Li & Justin Johnson & Serena. 2017. "Lecture 9,CNN Architectures." *stanford.edu.* May 2. Accessed March 4, 2020. http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture9.pdf.

Ying Liu, Dengsheng Zhang, Guojun Lu, Wei-Ying M. 2007. "A survey of content-based image retrieval with high-level semantics." *Pattern Recognition* 262 – 282.

Yudong Cao, H. Zhang, Yanyan Gao and Jun Guo. 2013. "An efficient duplicate image detection method based on Affine-SIFT feature." *3rd IEEE International Conference on Broadband Network and Multimedia Technology* 794-797.

Zahid Mehmood, Syed Muhammad Anwar, Nouman Ali,Hafiz Adnan Habib,. 2016. "A Novel Image Retrieval Based on a Combination of Local and Global Histograms of Visual Words." *Mathematical Problems in Engineering.*

Zauner, Christoph. 2010. *Implementation and Benchmarking of Perceptual Image Hash Functions.* Master Thesis, Hagenberg : Austria University of Applied Sciences, Hagenberg Campus.

Zhang, Y., Zhang, Y., Sun, J., Li, H. and Zhu, Y. 2018. "Learning near duplicate image pairs using convolutional neural networks." *International Journal of Performability Engineering* 168.

Zhou, Xiang Sean, and Huang, Thomas S. 2016. "CBIR: From Low-Level Features to High-Level Semantics." *Beckman Institute for Advanced Science and Technology University of Illinois at Urbana Champaign* 433-444.

Zhou, Z., Wu, Q. J., Huang, F., & Sun, X. 2017. "Fast and accurate near-duplicate image elimination for visual sensor networks." *nternational Journal of Distributed Sensor Networks.*

ZL UNIFIED ARCHIVE. 2019. *What is Dark Data? The Risks of ROT.* White Paper, ZL UNIFIED ARCHIVE.

L. Yujian and L. Bo, "A Normalized Levenshtein Distance Metric," in *IEEE* Transactions *on Pattern Analysis and Machine Intelligence*, vol. 29, no. 6, pp.

1091-1095, June 2007.doi: 10.1109/TPAMI.2007.1078
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4160958&isnumber=4160941

P. JaccardÉtude comparative de la distribution florale dans une portion des alpes et du jura Bull. Société Vaudoise des Sciences Naturelles, 37 (1901), pp. 547-579

W. A. Burkhard and R. M. Keller. 1973. Some approaches to best-match file searching. Commun. ACM 16, 4 (April 1973), 230-236. DOI=http://dx.doi.org/10.1145/362003.362025

T. Patel and S. Gandhi, "A survey on context based similarity techniques for image retrieval," 2017 International Conference on Innovative Mechanisms for Industry Applications (ICIMIA), Bangalore, 2017, pp. 219-223.
doi: 10.1109/ICIMIA.2017.7975606
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7975606&isnumber=7975508

Larry Hardesty," Longstanding problem put to rest, Proof that a 40-year-old algorithm is the best possible will come as a relief to computer scientists, MIT News Office, June 10, 2015, http://news.mit.edu/2015/algorithm-genome-best-possible-0610

W. Jiang et al., "Similarity-based online feature selection in content-based image retrieval", *Image Processing IEEE Transactions on*, vol. 15, no. 3, pp. 702-712, 2006.

Boren, Zachary Davies. There are officially more Mobile devices than people in the world. City: https://www.independent.co.uk. Tuesday 7 October.

R. W. Hamming, "Error detecting and error correcting codes," in The Bell System Technical Journal, vol. 29, no. 2, pp. 147-160, April 1950.
doi: 10.1002/j.1538-7305.1950.tb00463.x
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6772729&isnumber=6772728