

# **Image Recognition: Detection of nearly duplicate images**

A Thesis Presented to  
The Faculty of the Computer Science Department  
California State University Channel Islands

In (Partial) Fulfillment  
of the Requirements for the Degree  
Masters of Science in Computer Science

by  
Deepa Suryawanshi  
Advisor: Dr. Michael Soltys

May 2018

© 2018  
Deepa Suryawanshi  
ALL RIGHTS RESERVED

APPROVED FOR THE COMPUTER SCIENCE  
PROGRAM



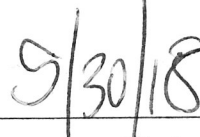
May 30, 2018

Advisor: Dr. Michael Soltys

Date



Dr. Brian Thoms



Date



5/30/2018

Dr. Pawel Pilarczyk

Date

APPROVED FOR THE UNIVERSITY

Dr. Joseph Shapiro

Date

# Image Recognition

Deepa Suryawanshi

May 31, 2018

## **Abstract**

Image recognition is used in many applications to detect images with same or different image content. This paper proposes image similarity measures. Usual method of serving photographs as evidence has been carried out for over a century. As there is digital information revolution, these methods are required to improve with same pace of time. Digital images are used and recognized in law enforcement as important tools in criminal investigations. The technique that is used in this paper is designed to identify if the image found on crime scenes is present in criminal database handled by forensic departments. Two important features of this thesis include recovering deleted images from any disk and detection of nearly duplicate images. Technique used for detection of nearly duplicate images is called as image fingerprinting, also known as image hashing. During criminal investigations, fingerprint evidence plays an important role. Image fingerprinting is defined as its literal meaning. As one's fingerprints are unique and they represent particular human being, similarly images have unique image fingerprints. Image fingerprints can be used to identify particular image in case of crime. Perceptual Hash gives the result to find image fingerprints. The techniques used in this paper uses difference hash for given images. Hamming distance is used to check the images which are almost similar with slight modification. Fingerprints of images don't change if an image is re-sized, compressed or expanded. Difference hash is used to identify such images. But if image is cropped or if image is taken from different angle then hamming distance is required. The paper also talks about different tools required to recover deleted images.

# Contents

<b>1</b>	<b>Background</b>	<b>1</b>
1.1	Digital Images . . . . .	1
1.1.1	Pixel . . . . .	1
1.1.2	Categories . . . . .	2
1.2	Digital Images types . . . . .	4
1.2.1	Binary Images . . . . .	5
1.2.2	Grayscale images . . . . .	6
1.2.3	Color images . . . . .	7
1.2.4	Multispectral Images . . . . .	9
1.3	Image storage in computer . . . . .	9
1.4	How computer reads images . . . . .	10
<b>2</b>	<b>Introduction</b>	<b>12</b>
2.1	Motivation . . . . .	12
2.1.1	Content-based image retrieval . . . . .	12
2.1.2	Search Engines . . . . .	14
2.2	Problem Statement . . . . .	16
2.3	My Contribution . . . . .	17
2.3.1	Detection of nearly duplicate images . . . . .	17
2.3.2	Recovery of deleted images . . . . .	24
<b>3</b>	<b>Related Work</b>	<b>26</b>
<b>4</b>	<b>Algorithm</b>	<b>30</b>
<b>5</b>	<b>Implementations</b>	<b>33</b>
5.1	Part 1: Recovery of deleted images . . . . .	33
5.2	Part 2: Detection of nearly duplicate images . . . . .	39
5.2.1	Approach 1 . . . . .	39
5.2.2	Approach 2 . . . . .	44
<b>6</b>	<b>Applications</b>	<b>56</b>
6.1	Detection of nearly duplicate images . . . . .	56
6.2	Recovery of deleted images . . . . .	57

<b>7</b>	<b>Conclusion and future work</b>	<b>59</b>
7.1	Future Work . . . . .	59
7.2	Conclusion . . . . .	60

## List of Figures

1	Pixalated image . . . . .	1
2	Bitmap(Raster) Image . . . . .	3
3	Vector Image . . . . .	4
4	Binary Image . . . . .	5
5	Grayscale Image . . . . .	5
6	Representation of Binary Image . . . . .	6
7	Examples of Grayscale Images . . . . .	7
8	R,G,B layers . . . . .	8
9	Example of color image . . . . .	8
10	Spectral information from the same image[?] . . . . .	9
11	Non-identical images with same name . . . . .	13
12	Identical images with different name . . . . .	13
13	CBIR system algorithm . . . . .	16
14	Similar images . . . . .	18
15	Hash of different images . . . . .	19
16	Different Hashing algorithms . . . . .	20
17	MD5 for almost similar images . . . . .	21
18	d-hash for almost similar images . . . . .	22
19	General framework of Content-based image retrieval . . . . .	26
20	Illustration of different query schemes with the corresponding retrieval results . . . . .	27
21	Detection of nearly duplicate images . . . . .	30
22	FTK Imager: Step 1 . . . . .	34
23	FTK Imager: Step 2 . . . . .	34
24	FTK Imager: Steps 3 and 4 . . . . .	35
25	FTK Imager: Steps 5 and 6 . . . . .	35
26	FTK Imager: Steps 7 and 8 . . . . .	36
27	Autopsy: Steps 1 and 2 . . . . .	36
28	Autopsy: Steps 3 and 4 . . . . .	37
29	Autopsy: Step 5 . . . . .	37
30	Autopsy: Image recovery details . . . . .	38
31	Result for Approach 1 to detect similar images . . . . .	43
32	Result for Approach 2 to detect similar images . . . . .	48
33	Input test image . . . . .	49
34	Result images . . . . .	50

35	Result images for dataset of 1000 images for query image as Eiffel tower. . . . .	52
36	Query image: Human face . . . . .	53
37	Results for Query image: Human face . . . . .	53
38	d-hash for similar images where one image inverted.Hamming distance is calculated to be 16. None of the bits match. . . . .	55



# 1 Background

Images are center of attraction for this paper. Main attributes of the paper, 'Recovery of deleted images' and 'Detection of nearly duplicate images', works entirely around digital images. As the paper talks about digital images throughout the paper, we should know what these digital images are and what different file formats exists. For recovery of deleted images, we should know how they are stored in computer and how computer reads them.

## 1.1 Digital Images

Computer files (e.g. digital images) are stored as digital information. When an image is stored, a computer converts this image into digital code for storage. Images are digitized meaning they are stored in bits of 0s and 1s. One byte of information is one pixel. Image is made up of pixels arranged in different fashion. The arrangement of pixels gives the image color, shades and shape.

### 1.1.1 Pixel

A Pixel is the smallest unit of digital image. The name is derived from 'picture element'(?). Many pixels combined together forms a picture. For example, pixelated image is shown in figure 1:

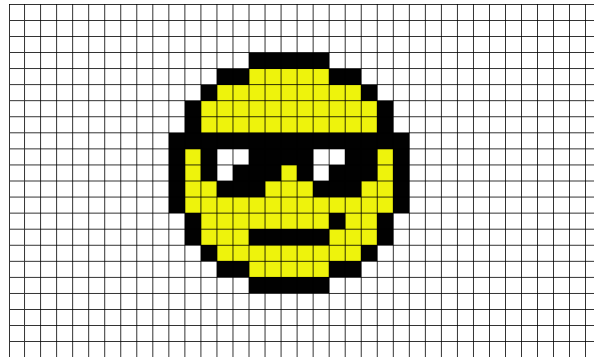


Figure 1: Pixelated image

### 1.1.2 Categories

Images can be divided into two broad categories: Pixel Images and Vector Images[?]. A bitmap is a method to store images. It is a map of bits and location of where they are stored. Most of the file formats that are used for detection nearly duplicate images fall under bitmap images.

#### Pixel Images/Bitmap Images

Bitmap images are pixel-based images. They are typically used for photos and printed design materials. Each pixel store different color. Some of the file types are JPEG, GIF, TIFF, PNG. New formats are constantly developed to address the growing graphic image needs. Each format uses different types of digital data to represent, store and display the graphic image[?]. Example of Bitmap image is shown in figure 2.

Main features for Bitmap images(also known as Raster images) are accounted as follows[?]:

- Bitmap images are pixel based.
- Bitmap images are best for editing photos and creating continuous tone images with soft color blends which Vector images can not do easily.
- Bitmap images are not scalable.
- File sizes are directly proportional to details in the Bitmap images.
- Some processes are not compatible with Bitmap images.
- Conversion to vector images takes more time compared to conversion of vector to bitmap images.
- Bitmap images are the most common image format, including: jpg, gif, png, tif, bmp, psd, eps and pdfs originating from raster programs.
- Common raster(bitmap) programs: photo editing / paint programs such as Photoshop , Paint Shop, GIMP (free).



Figure 2: Bitmap(Raster) Image

## Vector Images

Vector images are the object which are created by mathematical calculations. Unlike pixel images, after enlarging Vector images, they do not lose quality of an image. Vector images are easily edited and re-sized. They are not used for images. Vector images are typically used for logos, cartoons, illustrations, icons etc[?]. Example of Vector images is shown in figure 3.

Main features for Vector images include as[?]:

- Objects are created by mathematical calculations.
- Vector images are best for creating logos, drawings and illustrations, technical drawings and for images that will be applied to physical products.
- Vector images are scalable (without losing quality).
- Vector images can be printed at any resolution.
- Vector images can maintain smaller file sizes.

- Colors can be added or removed easily for various purposes in Vector images.
- Vector images can be easily converted to Raster(Bitmap).
- Common vector graphic file formats are: ai, cdr, svg, and eps , pdfs originating from vector programs.
- Common vector programs: drawing programs such as Illustrator, Corel-Draw, Inkscape (free).

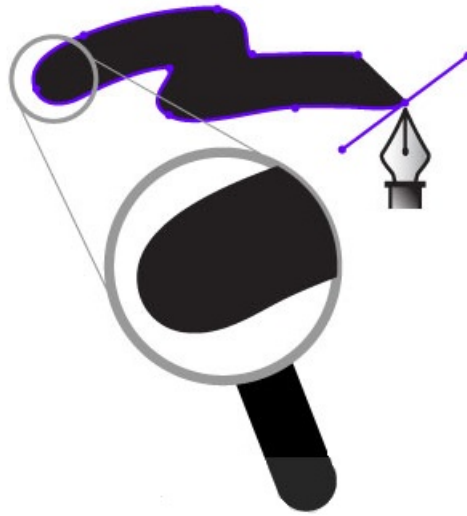


Figure 3: Vector Image

## 1.2 Digital Images types

There are different types of digital images [?]. They are pixel-based images. They can be majorly divided as:

- Binary images
- Grayscale images
- Color images
- Multispectral images

### 1.2.1 Binary Images

Binary images are digital images that take two values, either 0 or 1 for each pixel. It is 1-bit image as its taking only 1 binary digit to represent a pixel. Binary images can be stored in memory as a bitmap. They are often produced as result of thresholding of grayscale or color images to separate the image from background[?]. Figure 4 and figure 5 shows example of binary image and grayscale image respectively.



Figure 4: Binary Image

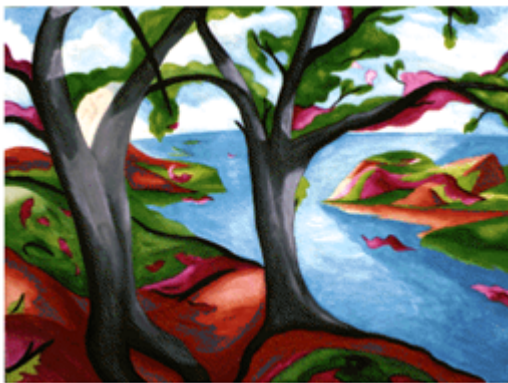


Figure 5: Grayscale Image

Binary images are also called as black-white images. The White part of

an image is considered as foreground and the black part of the image is rest of the image.

The representation and simple explanation of the binary images s shown in Figure 6:

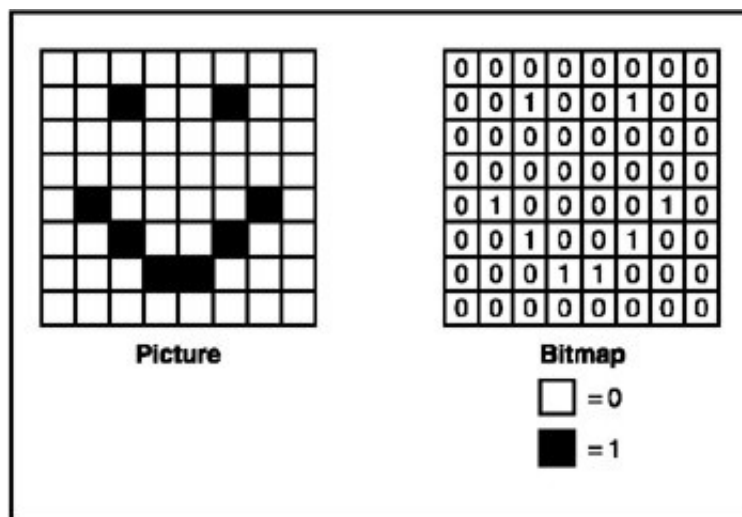


Figure 6: Representation of Binary Image

### 1.2.2 Grayscale images

A grayscale image is one in where the value of each pixel is a single sample representing only an amount of light. That is, the image carries only intensity information[?]. They are also called as one-color images. As the name defines they have one color. Typically, grayscale images contain 8bits/pixel data.

A grayscale image provides 256 different gray levels. The darkest shade is black and the lightest shade is white. Intermediate colors are defined by equal brightness of RGB for transmitted colors, and CMY colors for refracted lights where RGB stands for Red, Green, Blue and CMY stands for Cyan,

Magenta, Yellow.[?]

Examples of the grayscale image is given in figure 7 as follows:

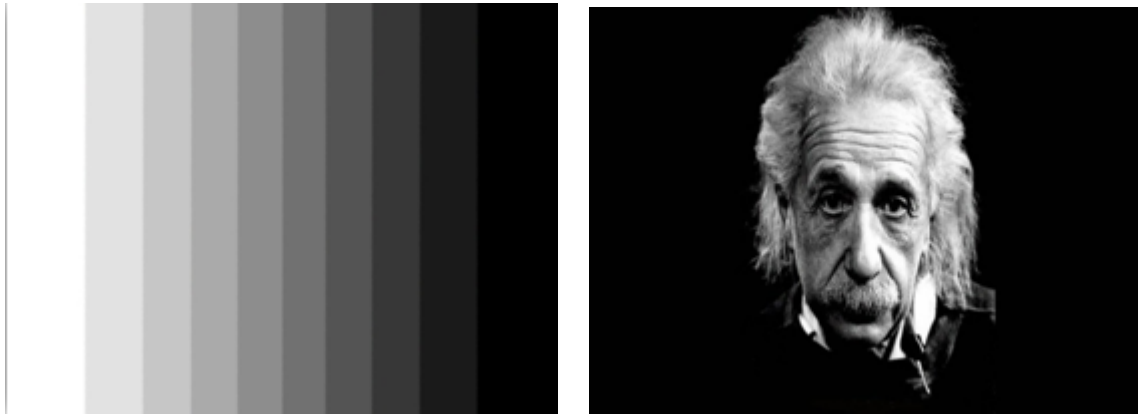


Figure 7: Examples of Grayscale Images

### 1.2.3 Color images

Color images are digital images that include color information for each pixel. Typically, color images are represented as red, green and blue. The information stored in digital image data is brightness information in each spectral band. Color images have 3 values per pixel[?]. Color requires that intensities for every pixel is specified. Figure 8 shows the three layers of the color image.

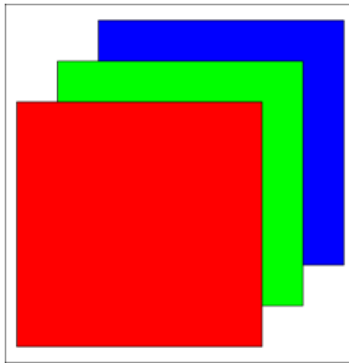


Figure 8: R,G,B layers

It is stored in memory as a raster map. A raster image or bitmap image is stored as a dot matrix data structure. Color images have different file formats[?]. One of the example of color images with the RGB channels is shown in figure 9.

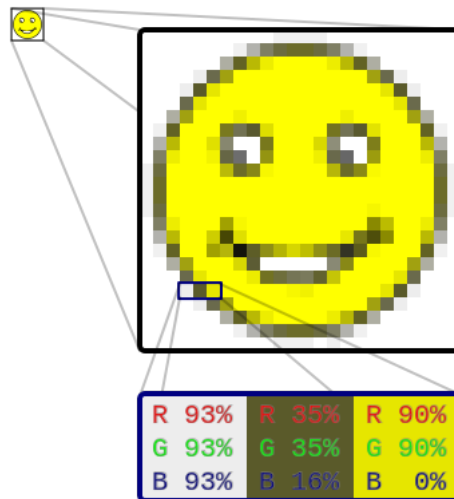


Figure 9: Example of color image

The smiley face in figure 9 is a raster image. When enlarged, individual pixels appear as squares. Zooming in further, they can be analyzed, with



their colors constructed by adding the values for red, green and blue.

#### 1.2.4 Multispectral Images

Multispectral images can be a collection of several monochrome images. It consists of several bands of data. Each band displays a grayscale image or combination of three bands to give color image. The spectral bands used in multispectral images are represented with red, blue and green channel[?]. Typical example is RGB color image. The information in them is not visible to human eyes.

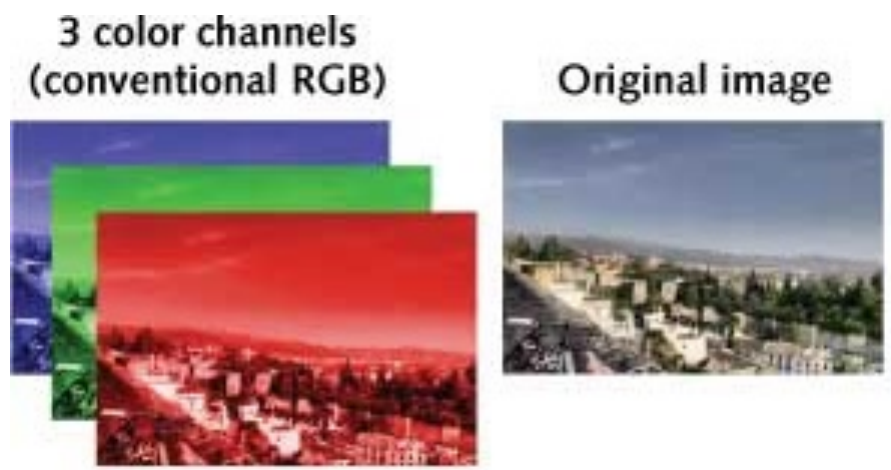


Figure 10: Spectral information from the same image[?]

### 1.3 Image storage in computer

As mentioned in background of digital images, all the digital images are stored as digital information. Images are digitised. One byte of information is one pixel[?]. Number of bits represent how many colors are available for each pixel. In black and white images only 1 bit is required.

0	White
1	Black

2-bits gives four colors:

00	White
01	Light grey
10	Dark grey
11	Black

For  $n$ -bits, there can be  $2^n$  colors. This relationship is defined by color depth. Below is a listing of all of the different color depths over the history of computers.[?]

<b>n</b>	<b><math>2^n</math> colors</b>	<b>Display</b>
1	2	Monochrome Display
2	4	CGA Display
4	16	EGA Display
8	256	VGA Display
16	65539	XGA Display
24	16777216	SVGA Display

## 1.4 How computer reads images

Every computing device have the storage memory that is used for storing information. Different technologies are used to store data by storage device. The technology of storage device decides how you read the image. A controller on the storage device sends the data to the motherboard. The motherboard send the data to operating system. The operating system stores the data on the disk depending on the file system it is using.

Color digital images are made of pixels. Pixels are made of primary colors represented by a series of code. Channel is grayscale made of one primary color with same size as color image. Color image is made of thee channels of Red, Green and Blue. Most images are stored with colors varying from 0-256 for each of Red, Green, Blue channel as data is grouped by 8 bits. 1 byte is 8 bits. 3 groups of 8 is 3 bytes where the first byte represents the amount

of red, the second the amount of green and the third is for the blue. Combination of these bytes gives particular color to pixel and it also composes an image.

Example of raw image in bits where first pixel is Red:

11111111	00000000	00000000
Red	Green	Blue

Similarly, example of raw image in bits where first pixel is Green:

00000000	11111111	00000000
Red	Green	Blue

And example of raw image in bits where first pixel is Blue:

00000000	00000000	11111111
Red	Green	Blue

## 2 Introduction

This thesis is divided into two parts. Small portion of it focuses on recovery of deleted images discussed in 2.3.2 and detection of duplicate images/ nearly duplicate images/ almost identical images discussed in 2.3.1. The idea of detection of nearly duplicate images can be proposed by other established techniques such as content-based image retrieval.

### 2.1 Motivation

#### 2.1.1 Content-based image retrieval

Content-based image retrieval (CBIR) is the application of computer vision techniques to the image retrieval problem. That is, CBIR is the process of searching for digital images in large databases. Content-based means the search carried out based on contents rather than metadata of the image. Metadata refers to data about data. In this case, data is image. Data about image can be keywords, tags, or descriptions associated with the image. CBIR searches image based on the image content than its metadata.[?]

The problem with searching only metadata is that two images can have the same name or keywords but they don't have to be identical. Images are said to be the same if they have same image details.



(a) Calvin and Hobbes Image



(b) Calvin and Hobbes Image

Figure 11: Non-identical images with same name

In figure 11, two images are shown. Each have identical metadata but are different. CBIR will show this as non-identical images even if metadata is same.



(a) Calvin and Hobbes Image



(b) Funny Image

Figure 12: Identical images with different name

In figure 12, two identical images are shown. With traditional text-based searching methods, these images will be considered different images as they don't have same metadata. This will give false negatives which are difficult to avoid.

CBIR fits perfectly in our problem statement. Interest in CBIR increased because metadata based systems gets limited in searching and there is huge data demand to be retrieved. To search an image with name or tags or keywords is easy, but if two similar images have different description then this search will give false results. Thus, there is need of a technique where images can be searched based on their contents.

Many CBIR systems have been developed, but the problem of retrieving images on the basis of their pixel content remains largely unsolved.

### **2.1.2 Search Engines**

There are many text based search engines. One popular example is Google. Others include Bing, DuckDuckGo, Yahoo etc. It works on texts. You have to type words and it gives you result based on those words. Image search engines works differently. You can not use texts. You have to feed image as query. Image search engines can be of different types, but mostly they can be of three types: Search by metadata, by example, or hybrid approach. [?]

#### **1. Search by MetaData**

Search by metadata is like normal keyword text search. It doesn't check the contents of the image. They rely on textual data such as annotations and tagging given to image and contextual hints like texts which are near images. It works as normal text based search. The actual image is hardly processed, which defeats the purpose of searching by the image.

Flickr is one such example where images can be searched with tags, annotations, or the contextual hints. When an image is uploaded to Flickr, tags must also be submitted to an image.

#### **2. Search by Example**

Search by Example depends on contents of the image. Images are analysed, quantified and stored. This is also called Image search engine which is based on the CBIR technique mentioned above. It works

on the contents of the images and not on textual data of the image.

Examples of 'Search by Example' are discussed in Chapter 6. One of those examples is TinEye, is a reverse image search engine. It takes image as query input and it gives all the results of similar images. It searches with the contents of the images. It also gives the webpage or path of the image where the image is posted. This system is hard to develop. But once it is built, human intervention is not required.

### 3. Hybrid search

Some systems can be developed not only on text-based search or only on content-based search, where both techniques are required for some searches. Twitter is one such example, where photos can be uploaded along with tweets.

Common steps for any CBIR system:

Step 1: Defining your image descriptor

Step 2: Indexing your dataset

Step 3: Defining your similarity metric

Step 4: Searching

Figure 13 shows the steps used by CBIR system.

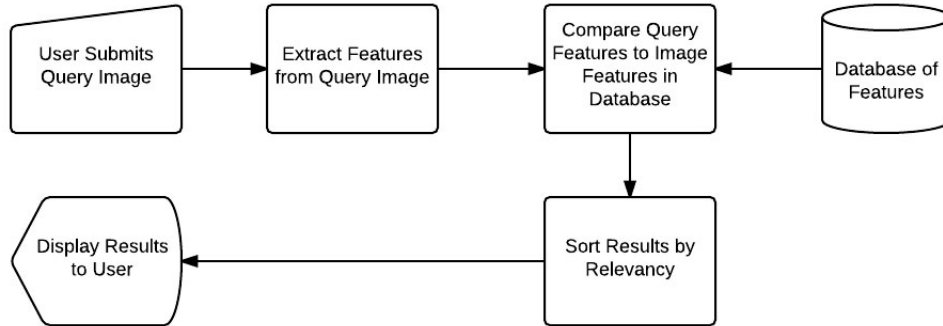


Figure 13: CBIR system algorithm

Figure 13 gives the algorithm for CBIR. When query image is submitted by user. Its features are extracted by different methods, like machine learning or pattern recognition etc. Database of features is maintained seperately. Features from query imageis compared with all the features in database of features. If there is similarity then reults are shown. This algorithm is what we need for the problem statement discussed in 2.2.

## 2.2 Problem Statement

Forensic departments believe that anything can be evidence if it is present at a crime scene. They also believe that criminals ought to leave evidence behind even though they try to clean the place rid of evideneces. Digital images are very difficult to trace. Digital images in many cases can be used as evidence. They are already using digital image forensics to identify suspicious pirated copies of digital images. There are many techniques available too, like object recognition, panoramic image stitching, image mosaicingand near duplicate image detection[?].

There are requirement of image detection which are found at a crime scene. When images are recovered from crime scenes, they can be used as evidence if they are connected to other crimes. Forensic departments maintain huge databases of images saved with keywords and tags. So, if this image



is found in the database, let's say under keyword of kidnapping, then this image can be connected with the crime. This will strengthen the evidence[?].

Also, first thing criminals try to do is wipe all possible evidence including papers, documents, harddisks, thumb-drives, etc. If the hard disks are cleaned in a hurry, any criminal will just shift + delete everything or format the disk or pen-drive. If harddisks are not overwritten and only wiped clean, then the files can be revived. There are many free tools for that, which will be discussed in Chapter 6.

## **2.3 My Contribution**

This thesis focuses on the shortcomings faced using digital images as evidence in cases of crimes. This thesis focuses on two main attributes, 'Detection of nearly duplicate images' and 'Recovery of deleted files'. Recovery of deleted files is easy with the right tools. Steps to recover deleted files are included in chapter 5. This research shows why permanently deleted files can be recovered.

Main focus is on the detection of nearly duplicate images. Dataset includes 1000 images with 101 different categories. This thesis also includes the creation of a dataset with python platform for different images with random sizes. I worked with these images to create shelve (persistent, dictionary-like object) of hashes of images and filenames. But as the use of shelves is deteriorated with new versions of python, it didn't show desired results for identical images with different file formats. This issue was resolved when I created the array and saved hash and filename in the array.

### **2.3.1 Detection of nearly duplicate images**

This section focuses on finding the image in criminal database using image fingerprints. Image fingerprints are also called as hash. There are different types of hashes. I have studied different types of hashes which are discussed in this paper. With their unique features and by testing images with all the techniques, I chose the techniques I have discussed later in the paper. Forensic departments maintain the database of images which are already related

to crimes. If, at crime scene, an image is found and this image can be found in the criminal database, this image can be entered into evidence.

The purpose of this research is to check if an image is in database. This is simple if both the images(one image which we found on crime scene, other image in criminal database)are identical. But two images can be the same even if they are not identical. The human eye can tell if images have similarities, but asking a computer requires more work.



Figure 14: Similar images

Figure 14 shows that these two images have similar details and are connected to each other. However, finding similarity in such images is not straightforward. Image fingerprinting is the solution for such problems. It requires testing of different images with different types of hashing algorithms.

- **Hashing**

From this thesis, I would like to state that hash function is a function that takes relatively arbitrary amount of input and produces output of fixed size. Hash functions help to maintain the security and integrity of input. This topic requires the properties and basic knowledge of hash functions which will be discussed shortly.

My study of hashing included the definition of hashing as use of hash functions to verify that an image is identical to the source image. Hashing is like a digital fingerprint for a file(Image fingerprinting). As two people can not have identical fingerprints, similarly, it is unlikely that two images will have similar hash. The type of hash determines the length of hash. It is mathematically derived. Only identical images have same hash. Following images have similar but very varying details. These images give different hash value. There is need to improvement to show that these images have similar details.

Image	Hash Value
	0b07071193172533
	3e1edf4b4f5b95a3

Figure 15: Hash of different images

Hash is a function that takes as input objects and outputs a string or

number. A hash function is a function that takes an input and produces a value of fixed size.

I calculated hashes of similar images which came out to be different to check their similarity. Both the images are of comic characters, Calvin and Hobbes, but in both the images they are doing different things which changes the details of the images. Hash value of such two images is different as shown in Figure 15.

- **Types of hashing algorithms**

Some of the common cryptographic hashing algorithms for encryption are SHA1, SHA256, MD5. These algorithms are used for encryption. Hashing is used in many other areas of digital study such as download confirmation and encryption. They are used for file verification, password storage.

For image hashing, most known hashing algorithms are a-hash, p-hash, d-hash, w-hash. To apply these hashes to any image, image is converted to grayscale. a-hash (average hash) calculates the mean and binarize the grayscale based on the mean. This binary image is converted into the integer. p-hash (perceptual hash) uses discrete cosine transformation on grayscale image. Similarly, d-hash uses gradients and w-hash uses wavelet transformations.

Cryptographic hashing algorithms	Image hashing algorithms
SHA1	a-hash
SHA256	p-hash
MD5	d-hash
DSA	w-hash

Figure 16: Different Hashing algorithms

Choosing which type of hash depends on its inherent properties and the results we want. We can choose between a-hash, p-hash, d-hash or

w-hash instead of cryptographic hashing algorithms.

- **Image Hashing and Cryptographic Hashing**

During digital investigation, classic hashing techniques such as MD5, SHA1, or SHA256 are commonly used to index large quantities of images in order to detect copies in different archives. These hashing techniques are difficult to use if we want to find other duplicates. These techniques are not suitable if an image has changed by even a bit, then the hash value changes altogether. In case of crime scenes, there is more probability of having two similar content images. These techniques will give totally different hash for both images. Examples for such similar content image is shown in figure 17 and figure 18.



Image	MD5
	6F6C390BD5AB87529613A58EDCB2B57A
	C08D51AD4C203319B8EDDED0FDF4449C

Figure 17: MD5 for almost similar images

I calculated the Hash for two almost identical images as shown in figure

17. One is the colored image and the other is black and white image which makes both images different. This lead to change the MD5 hash for both the images as shown above. As the first bit is changed , the whole hash value changed for above image. I tested different types of images with different algorithms. With the observation, it can be concluded conclude that these types of hash functions are helpful to observe the changes in data like message exchanging securely. In such cases, these functions are known to provide security , authenticity and integrity of messages sent or received.

We need hash value that should not change over small detail to increase the accuracy of searching similar images. Hash value for an image should remain same irrespective of its format or resolution. Even if the details of the image are changed but images are almost same then hash value should not change much.



Image	d-hash
	3e1edf4b4f5b95a3
	3e16df4b4b5b9183

Figure 18: d-hash for almost similar images

I calculated d-hash of two images as shown in figure 18. We can see that images are identical. The difference between them is that one is colored image and other one is Black and White image. Observe the hash value. Starting from first bit, 4th, 10th, 14th, 15th bit is changed in 16 bit hash value. Hamming distance is 4 as only 4 bits are changed. Hamming distance is discussed shortly. Test results can verify that these two are identical images even if one image has gray background.

The techniques that are developed are perceptual hash, difference hash, average hash, wavelet hash. They are used to:

- find duplicates very fast. Instead of searching for the whole image, you will look for the hash of the image.
- finding similar images.

The second image is black and white version of the first image in figure 18. They are the same. MD5 gives the hash totally different for two images whereas d-hash gives almost similar hash. This property of d-hash makes it suitable for the research purpose of this subject. Python has all the hashes implemented in the library.

## • Hamming Distance

To compare hash values, hamming distance is used. The Hamming Distance is a number used to denote the difference between two binary strings.

### Calculation of Hamming distance

There are some steps we have to take to calculate hamming distance of two binary strings.

1. Two strings should be of equal length

2. Compare the bits in each string. If they are the same, then write '0' and if they are different, then write '1'.
3. Add all 0s and 1s , we got in step 2 above. This is Hamming Distance.

Basically Hamming distance means how many bits are different in given two strings.

Example:

string1	:	0011	1100	0001
string2	:	0110	1010	0101
Compare	:	0101	0101	0100

$$\text{Hamming Distance} = 0+1+0+1+0+1+0+1+0+1+0+0 = 5$$

Hamming distance 5 means , string is different in 5 bits. This can be used when images are similar but not identical. We can put threshold on hamming distance (i.e. if hamming distance is less than, let's say, 6, then images contain similar details).

### 2.3.2 Recovery of deleted images

This thesis also discusses recovery of deleted images. To understand how images are recovered, we have to know how they are deleted and what happens when images are deleted.

#### The deleted images

Generally if files are deleted, they end up in temporary holding space i.e. recycle bin. They can either be recovered or deleted from Recycle Bin to reclaim the disk space it was using previously. But if they are deleted even from Recycle Bin, they are considered to be permanently deleted. But this doesn't happen in reality. Images or files can be recovered even if they are not present in Recycle Bin.

Operating systems keep track of files with the help of 'pointers'. Pointers tell where the file is. It tells where the file starts and where it ends. Every file on the computer has pointer. When the file is deleted, operating system



simply removes the pointer and marks the sector 'Available' which contains that file. Practically, a file is not present but it technically is still there.

The file is there until the OS overwrites it with another data. If the files are recently deleted then they can be recovered. In order to recover a deleted file every sector that included data for that file must be in its original condition. If small part is overwritten, then file may get corrupt. But there is chance with images. Even if they are partially overwritten, then also there are tools that can recover them partially which still can be helpful as evidence. The human eye can analyse partially recovered photo.

### **Restoring Images**

As we discussed in previous section, we can recover these images if they are not overwritten by the OS. There are ways that these images can remain deleted and un-recoverable. But we will be discussing about recovery of such images. I recovered deleted images with different scenarios such as

- When they are deleted from computer.
- When they are deleted from HDD.
- When they are deleted from pendrive.
- When they are deleted from SSD.

Deleted images can be recovered from all the above scenarios. There are different tools which will be discussed in 5.1. Tools such as Piriform Recuva, IOBit undelete, TestDisk and PhotoRec, Autopsy, FTK Imager etc. Yodot Hard Drive recovery tool is easier to recover deleted data of SSD. Data from phones can be recovered too. Some programs are with good GUI and some have good reliability and speed. Since many of the tools are free, its easier to recover images. Only creating image of hard disk can be time consuming. It depends on the size of hard disk. The more the size, more time it takes to create image.[?]

### 3 Related Work

Detection of Near duplicate images is part of image recognition. The ways that it can be carried are varied. It can be seen as clustering problem. There are different techniques which influenced detection of nearly duplicate images.

#### 1. CBIR: Content-based image retrieval

As popularity of digital devices with camera has increased, digital photos have gained importance in every field. Forensics also give importance to digital image and use it as evidence against many cases. They use CBIR for forgery of images, duplication of images, etc. Textual information is inconsistent when searching for images. CBIR has been preferred over the year and is making advances.[?]

CBIR works on content based search. Key problem is measuring similarity between images efficiently, in this case, feature(content) extraction. In early CBIR system, global features are image content described by color, shape, texture, structure, etc. Similarly, instead of feature or content matching we can take some features of the image that won't change. Image fingerprint can be used as content of the image. We can compare hash (image fingerprint) of the image to detect similarity.

[?] discusses about key terms of content based image retrieval. Figure 29 shows general framework of CBIR and figure 20 shows the different query schemes with corresponding query results.

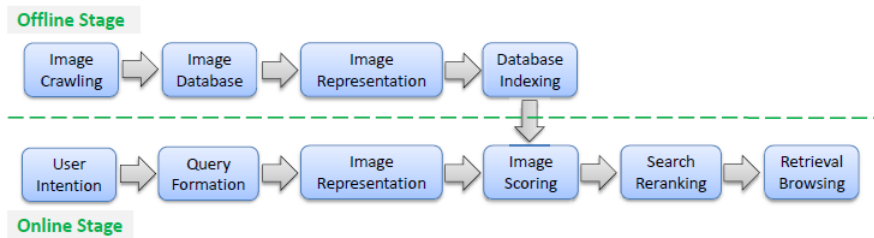


Figure 19: General framework of Content-based image retrieval

paper [?] talks about the query formation as query by example image is most intuitive query formation. User has an example image at hand and would like to retrieve more or better images about the same or similar semantics.

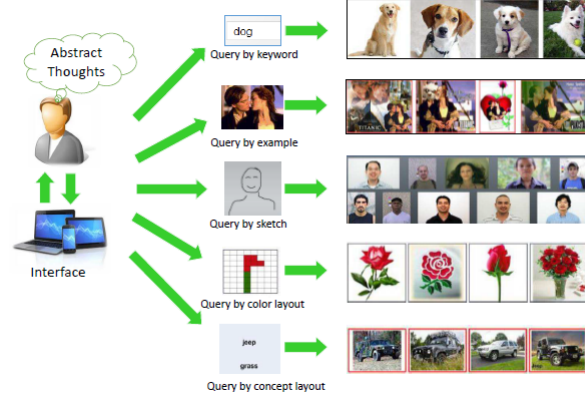


Figure 20: Illustration of different query schemes with the corresponding retrieval results

According to author of the paper[?], The above query formations, figure 20, are convenient for uses to input but may still be difficult to express the user's semantic intention. This kind of query is specially suitable for searching 1generalized objects or scenes with context when the object recognition results are ready for the database images and the queries. As the problem statement discussed in 2.2, We can use part of CBIR algorithm shown in figure 20. Query formation can be carried out with example image shown in figure 20.

Paper on 'Content based Image Retrieval (CBIR) System for Diagnosis of Blood Related Diseases'[?] shows the working of how CBIR is used to find similar images. They used similar algorithm given in figure 13. According to paper[?], Despite various CBIR developments, medical images in different fields are very particular and require a specific design of CBIR systems. There exists a large number of medical image acquisition devices among which computed tomography scanners (CT), magnetic resonance imagers (MRI), ultrasound probes (US) and

nuclear imagers are the most widely used. Images are very different in case of resolution, contrast, signal to noise ratio.

Paper on 'Similarity-Based Online Feature Selection in Content-Based Image Retrieval'[?] discusses CBIR and feature selection. To apply the proposed feature selection method in region-based image retrieval systems, we propose a novel region-based representation to describe images in a uniform feature space with real-valued fuzzy features. Our system is suitable for online relevance feedback learning in CBIR by meeting the three requirements: learning with small size training set, the intrinsic asymmetry property of training samples, and the fast response requirement. Extensive experiments, including comparisons with many state-of-the-arts, show the effectiveness of our algorithm in improving the retrieval performance and saving the processing time.

## 2. Duplicate image detector

There are sites which are required to implement a duplicate image detector to avoid importing dupes into large image store. For example, Jetsetter[?], they have huge database of high resolution of travel photos. For storage purpose, this algorithm helps to minimize all the duplicates. They follow same d-hash perceptual hashing algorithm. There are some other examples too, like IconFinder[?]. Many icons are uploaded at iconfinder.com. It increases risk of piracy too. To avoid it, they came up with duplicate image detection technique where they use d-hash algorithm.

Python has image hashing library called 'ImageHash'. It supports average hashing, perception hashing, difference hashing, wavelet hashing. All the hashing algorithms are designed differently for different uses. Algorithm of difference hashing helped me improve the algorithm for better results.

## 3. Duplicate photo cleaner

Duplicate Photo Cleaner[?] is an advanced app to find and manage duplicate and similar images. The algorithm for duplicate photo cleaner compares query images just like a human would, which means that it finds more duplicates and similar photos. App lets the user control

program's settings and can specify the level of similarity that is sufficient to consider photos to be duplicates. Then they can be compared manually to keep best shots and delete the rest in a click.

Duplicate Photo Cleaner uses Content-Based Image Retrieval. CBIR analyzes actual image content and uses the information gathered to compare images. This way Duplicate Photo Cleaner compares images that were re-sized, edited, cropped and converted to other formats.

## 4 Algorithm

The algorithm for image recognition is simple. The steps are divided into three parts which is shown in figure 19. First, there will be database of images. A new database is created where hash values of these images are stored. When we have index image, the hash value of this image is calculated and that hash value is checked in database. If this value is present in database then it is present in the database . But in the case of near duplicate images, we can compare this hash value with other hash values and will present those which are almost similar.

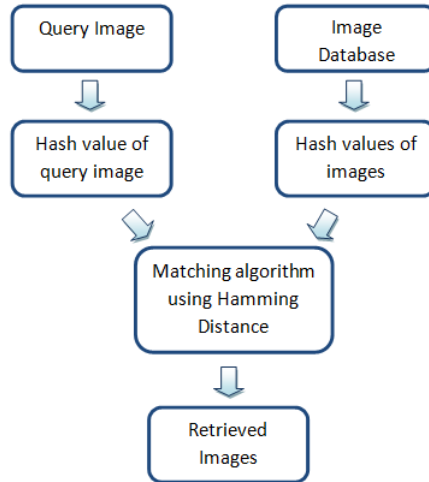


Figure 21: Detection of nearly duplicate images

What type of hash value we should take is an important question. There are different perceptual hashing algorithms. d-Hash algorithm is most suitable for my thesis. d-Hash computes the difference in brightness between vertically adjacent pixels[?].

The algorithm includes:

- Step 1: Grayscale the image.
- Step 2: Normalization of the image.
- Step 3: Comparing adjacent pixels vertically.
- Step 4: Convert the difference in bits.
- Step 5: Compare hash by Hamming Distance.
- Step 6: Show the results for those whose Hamming distance is less than 6.

### **Grayscale the image:**

This is done to reduce each pixel value to luminous intensity value. First three pixels give intensity value for (red, green, blue), let's say, (255,255,255). If the value is (255,255,255), it is white pixel as it is highest gray-scale value. It will give highest intensity as 255. On the other hand, (0,0,0), a black pixel, will give lowest intensity on 0. By gray-scaling, we are reducing each pixel value to its luminous intensity.

### **Normalization of the image:**

We can normalize the image to a standard base size to remove unnecessary details and high frequencies which can obstruct later in calculation of hash value. Fastest way to remove high frequencies is shrinking the image. An image is shrunk to 9x8 to get 72 pixels. After this re-sizing or stretching may not affect the hash value.

### **Comparing vertical pixels:**

When we compare vertical pixels of these intensity values, we get an array of binary values. There are 9 pixels in a column as we discussed in Normalization of the image. As d-hash works on a difference between vertically adjacent pixels, 9 pixels per column gives 8 differences between vertical pixel. There are 8 columns of 8 differences. This is calculated to be 64 bits.

### **Convert the difference in bits:**

This steps make it easy to store and use hash. The image is converted into hexadecimal string. If the left pixel is brighter than the right pixel, it is written as '1' otherwise it is '0'. This is done for every row left to right.

### **Compare hash by Hamming Distance:**

The above step gives the d-hash value. The database is composed of hash values of all the images. The hamming distance is calculated for a hash value of the query image and a hash values in database.

### **Show the results for those whose Hamming distance is less than 6:**

If the hamming distance is 0 means there is perfect match. If the hamming distance is 6 or less than 6, then they are almost similar. Hamming distance of 6 is taken by trial and error method, which can be changed with respect to details we are looking for in a image.



## 5 Implementations

### 5.1 Part 1: Recovery of deleted images

Extracting files and/or recovery of critical forensic information is key within the process of computer forensics. There are many tools that a forensic examiner may choose to utilize in order to do so. Although this does not directly relate to recovery of files from a forensic stand point, it can also be utilized for users who have lost data and want to try their hand at recovery of information. The focus of this thesis is on Autopsy and how to use the free tool to recover deleted files.

Before we start, we need to download a few files. These files are free and enable you to obtain some of the basic bits of information that you will need in order to obtain files from a forensic image. Be aware that we are also mounting the images with other software to provide to you that the files that were deleted are still on the disk we are performing our analysis on.

Autopsy is an easier tool to use and it is free. Before the recovery procedure starts, we need an image of the harddisk. To take an image of the Harddisk one can use FTK imager.

Steps to create an image of th harddisk using FTK Imager along with the images:

- In FTK Imager, select Physical drive as Source. Click Next as shown in figure 20.

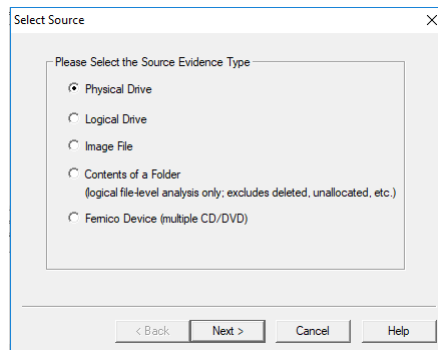


Figure 22: FTK Imager: Step 1

- select the drive as shown in figure 21.

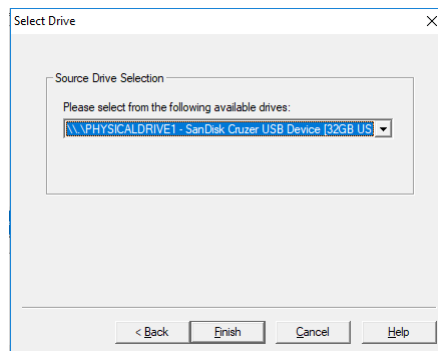


Figure 23: FTK Imager: Step 2

- Add Image destination and select destination image type as Raw(dd) as shown in figure 22.

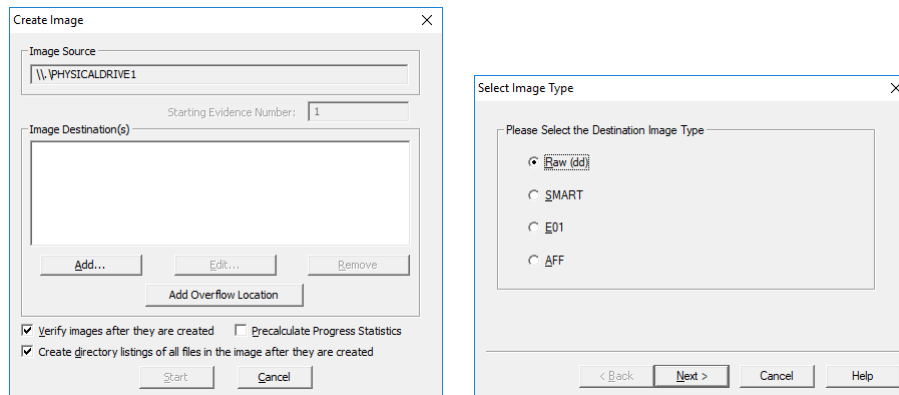


Figure 24: FTK Imager: Steps 3 and 4

- Fill out general information. Select destination folder and click Next as shown in figure 23.

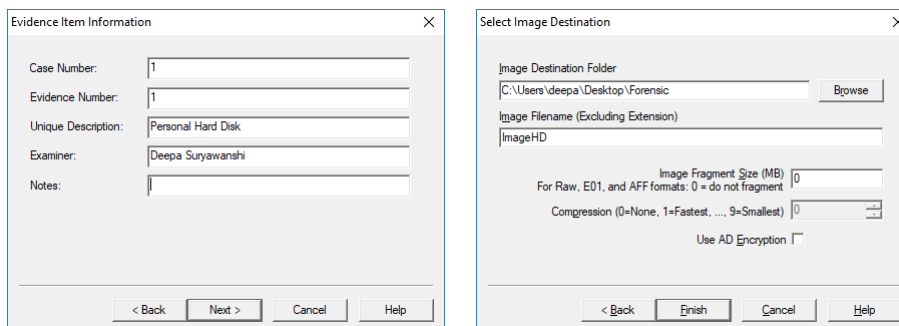


Figure 25: FTK Imager: Steps 5 and 6

This will give the Image of hard disk. This image can help us recover the deleted files from this hard disk. This is shown in figure 24.

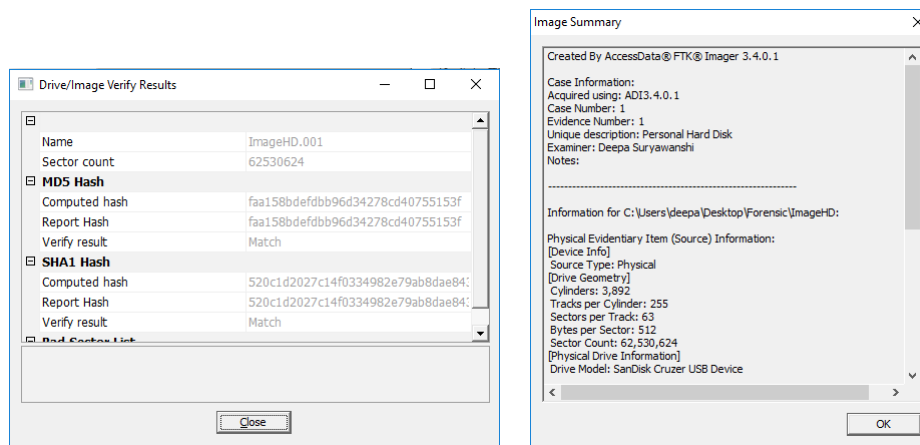


Figure 26: FTK Imager: Steps 7 and 8

Steps to recover deleted files from Autopsy:

- Create new case with case name and path. Fill out the information as examiner as shown in figure 25.

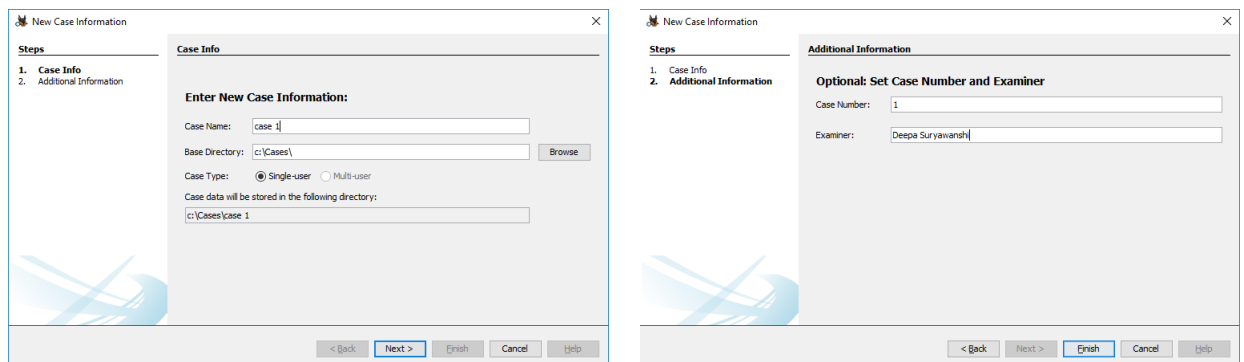


Figure 27: Autopsy: Steps 1 and 2

- Select 'Disk Image' in type of data source and path to data source as shown in figure 26.

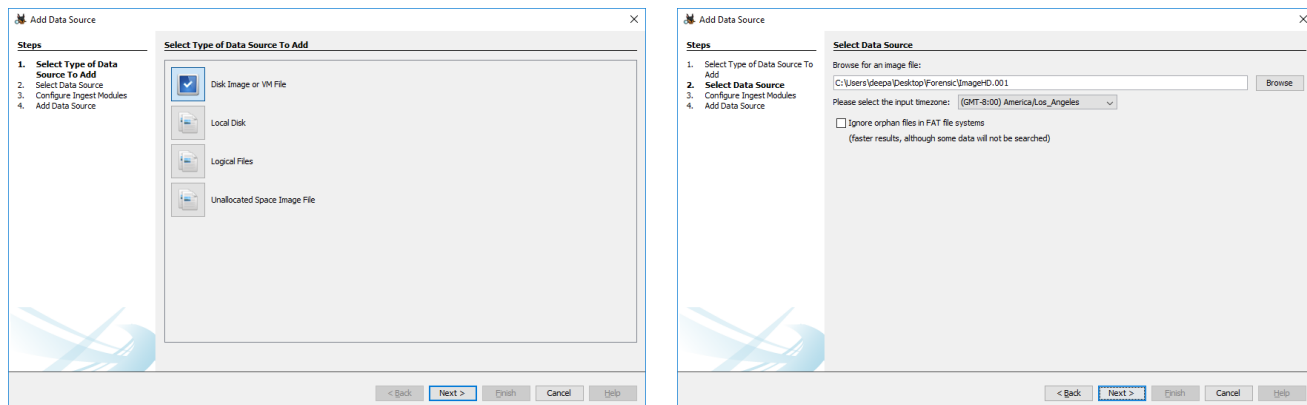


Figure 28: Autopsy: Steps 3 and 4

- Configure Ingest modules. Click Next. It will add the source to local. This takes some time depending on the size of hard disk as shown in figure 27.

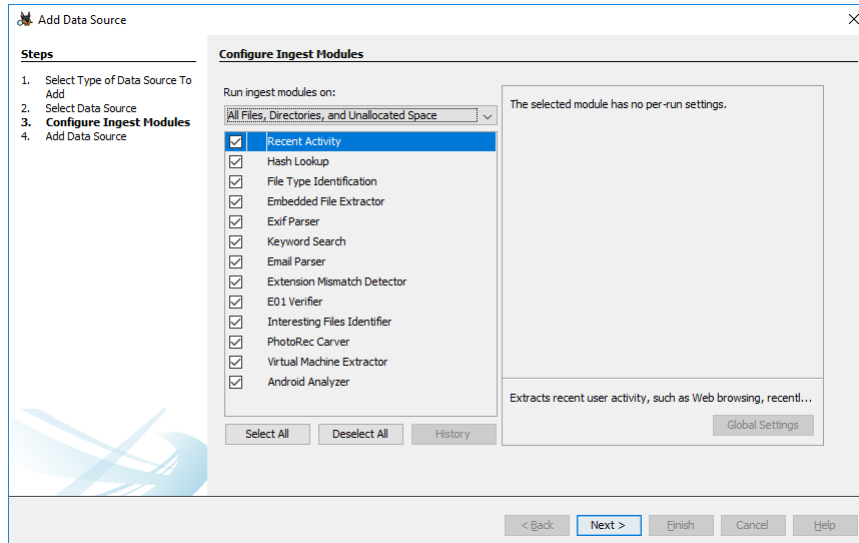


Figure 29: Autopsy: Step 5

After data source has been added, it will give you information about how

many files were deleted and how much they are recovered which is shown in figure 28.

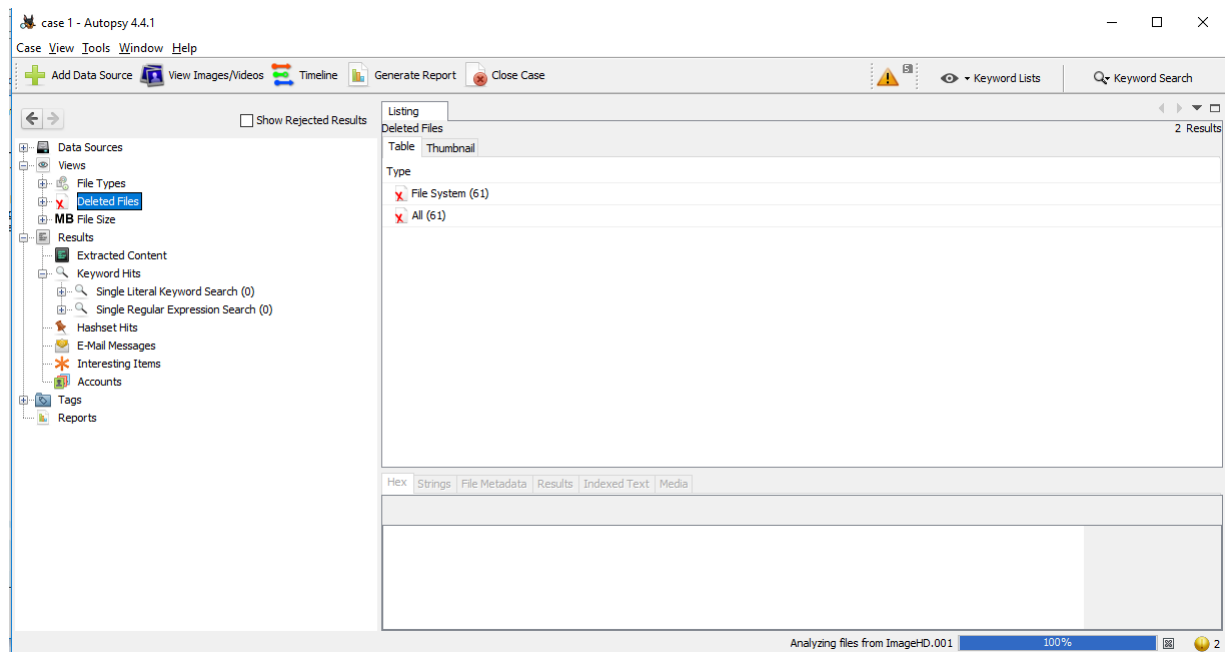


Figure 30: Autopsy: Image recovery details

## 5.2 Part 2: Detection of nearly duplicate images

For implementation for detection of nearly duplicate images, I created dataset of images of different categories. The dataset I found online is of computational Vision at Caltech, called Caltech 101[?]. This dataset contains 101 categories and each category contains at least 50 images. This helped me to work on diverse set and find the similar images.

The common python libraries that I used for the implementation of detection of nearly duplicate images are:

- PIL/Pillow to facilitate reading and loading images. PIL development is stagnated in 2009. Now, PIL works under Pillow library. Necessary changes are made to Pillow for it to work properly.
- ImageHash, which contains our implementation of dHash. It gives the hash of the given images. Hash for different types of images (jpg, bmp, png etc) gives same hash for given image. If the resolution of the image is changed then the difference between two hashes comes out to be very small.
- NumPy/SciPy, which are required by ImageHash. NumPy is the fundamental package for scientific computing with Python.
- Argparse module, it makes it easy to write user-friendly command-line interfaces. The program defines what arguments it requires, and argparse will figure out how to parse those out of sys.argv. The argparse module also automatically generates help and usage messages and issues errors when users give the program invalid arguments.

Working on the thesis, I followed two approaches.

### 5.2.1 Approach 1

In approach 1, coding is divided into three parts:

1. Collection of images and creating dataset using python.
2. Calculating image Hash

### 3. Display image as a result

#### collect.py

```
1
2
3 # This code be run with command:
4 # python collect.py --input inputimages --output dataset
5 #inputimages is the folder where I have downloaded 20 random images. da
6
7 from PIL import Image
8 import os, os.path
9 import argparse
10 import random
11 import shutil
12 import glob2
13 import uuid
14 from shutil import copyfile
15
16 # construct the argument parse and parse the arguments
17 ap = argparse.ArgumentParser()
18 ap.add_argument("-i", "--input", required = True,
19                 help = "input directory of images")
20 ap.add_argument("-o", "--output", required = True,
21                 help = "output directory")
22
23 args = vars(ap.parse_args())
24
25 #finding image in the given folder
26 for imagePath in glob2.iglob(args["input"] + "/*/*.jpg"):
27     filename = str(uuid.uuid4()) + ".jpg"
28     shutil.copy(imagePath, '\\\\?\\'+os.path.abspath(args["output"])
29     numTimes = random.randint(1, 8)
30     for i in range(0, numTimes):
31         image = Image.open(imagePath)
32         #changing the size of the image randomly
33         factor = random.uniform(0.90, 1.05)
34         width = int(image.size[0] * factor)
35         ratio = width / float(image.size[0])
36         height = int(image.size[1] * ratio)
37         image = image.resize((width, height),1)
```



```

38         #saving the image with random name
39         adjFilename = str(uuid.uuid4()) + ".jpg"
40         image = image.save (args["output"] + "/" + adjFilename,

```

This code gave me the dataset of images with different sizes created randomly[?]. Line 26-40 searches images in folder and changes their size randomly and saves in the folder. This is how I created dataset for the code given below to find the similar images. Note: all the image formats are similar too.

### hashdataset.py

```

1
2 # This code can be run with the command:
3 # python hashdataset.py --dataset dataset --shelve db.shelve
4 # Dataset is the folder of all random images. Shelve will store the hash
5
6 from PIL import Image
7 import imagehash
8 import argparse
9 import shelve
10 import glob
11
12 # construct the argument parse and parse the arguments
13 ap = argparse.ArgumentParser()
14 ap.add_argument("-d", "--dataset", required = True,
15                 help = "path to input dataset of images")
16 ap.add_argument("-s", "--shelve", required = True,
17                 help = "output shelve database")
18 args = vars(ap.parse_args())
19
20 # open the shelve database
21 db = shelve.open(args["shelve"], writeback = True)
22
23 # loop over the image dataset
24 for imagePath in glob.glob(args["dataset"] + "/*.jpg"):
25     # compute the difference hash
26     image = Image.open(imagePath)
27     h = str(imagehash.dhash(image))
28
29     #Save the hash as the key and filename

```

```

30         filename = imagePath[imagePath.rfind("/") + 1:]
31         db[h] = db.get(h, []) + [filename]
32
33 # close the shelf database
34 db.close()

```

After running this code, I created dictionary of hashes in db.shelve. A shelf is a persistent, dictionary-like object. Values in a shelf can be essentially arbitrary Python objects. This includes most class instances, recursive data types, and objects containing lots of shared sub-objects. The keys are ordinary strings. Now main code is to see if given image is in our dataset. I calculated hash of the given image and then i checked if that hash value is in db.shelve. If it is there, then it will show the results.[?]

### find.py

```

1
2 #To run this code use following in command prompt:
3 # python find.py --dataset images --shelve db.shelve --query 101_Object
4 #I am trying to search 18.jpg image, which is in 101_ObjectCategories f
5 from PIL import Image
6 import imagehash
7 import argparse
8 import shelve
9
10
11 ap = argparse.ArgumentParser()
12 ap.add_argument("-d", "--dataset", required = True,
13                 help = "path to dataset of images")
14 ap.add_argument("-s", "--shelve", required = True,
15                 help = "output shelve database")
16 ap.add_argument("-q", "--query", required = True,
17                 help = "path to the query image")
18 args = vars(ap.parse_args())
19
20 #open shelve database
21 db = shelve.open(args["shelve"])
22
23
24 query = Image.open(args["query"])
25 h = str(imagehash.dhash(query)) #hash of input image

```

```

26 filenames = db[h] #check the hash in database
27 filenames = list(set(filenames)) #removes the repeated images
28 print ("Found %d images" % (len(filenames)))
29 for filename in filenames:
30     #image = Image.open(args["dataset"] + "/" + filename)
31     image = Image.open(filename)
32     image.show()
33 db.close()

```

This code gives the result for the duplicate images with different sizes. Line 25 and line 26 are significant lines of the code which compare the query image hash and compare it with dataset image hashes which are saved using shelves.

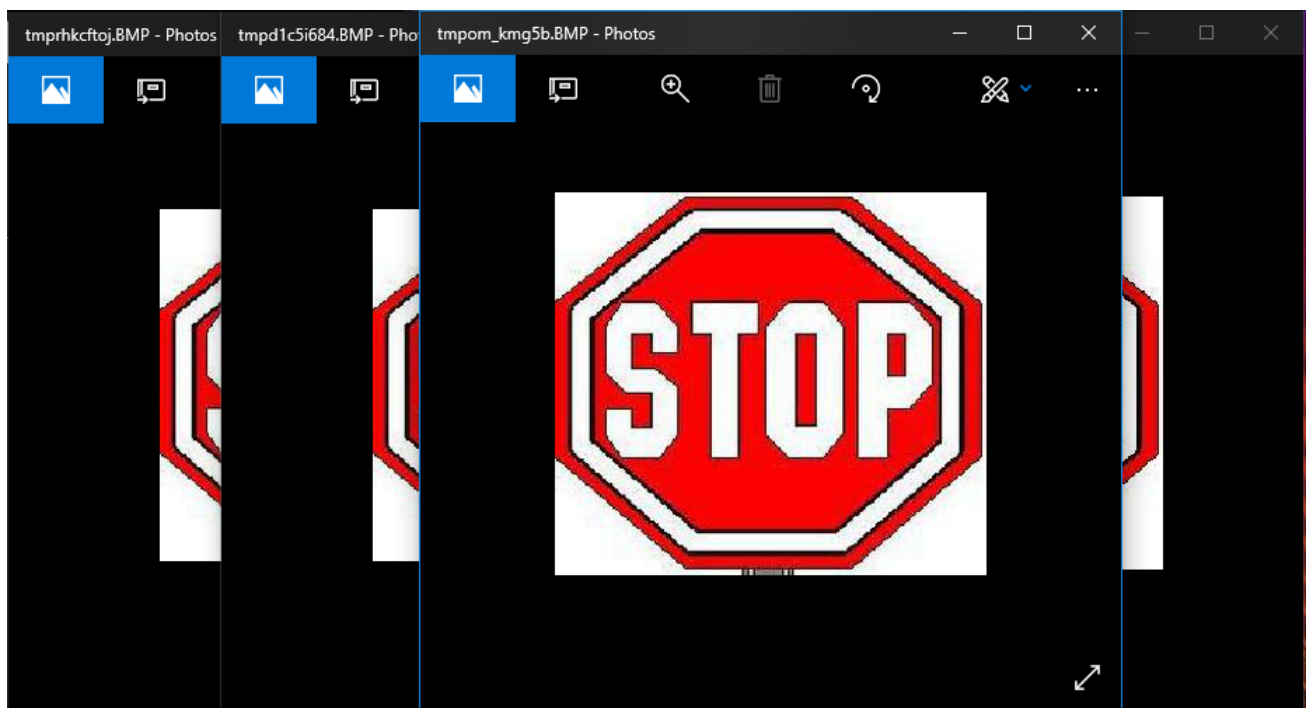


Figure 31: Result for Approach 1 to detect similar images

Figure 31 shows the result of all .jpg image files as the input image was

.jpg digital image. But my dataset had 8 similar images. Other images were of different file formats. Only these 4 images are of .jpg format. The problem with this code is that it doesn't work for different types of images.

For examples, if query image is .jpg type and dataset contains of .jpg, .bmp, .tif, .png. In this case query image of .jpg type will show results of .jpg types. It won't show the same image of different type. The problem is with shelve. Because of Python semantics, a shelf cannot know when a mutable persistent-dictionary entry is modified. By default modified objects are written only when assigned to the shelf They discontinued practice of shelve in python 3. Some of the features give different results with shelve.

The problem above was resolved in approach 2. This leads to approach 2.

### 5.2.2 Approach 2

We will be working on larger database. It's easier to create tuple for hash values and image path instead of shelve[?].

#### imagerec.py

```
1
2 #To run this code through command prompt:
3 # python imagerec.py --dataset dataset --query 101_ObjectCategories/18.
4 # 18.jpg image is searched through image dataset.
5
6 from PIL import Image
7 import imagehash
8 import argparse
9 import glob
10 import numpy as np
11 import matplotlib.pyplot as plt
12
13
14 # construct the argument parse and parse the arguments
15 ap = argparse.ArgumentParser()
16 ap.add_argument("-d", "--dataset", required = True,
17                 help = "path to input dataset of images")
```

```

18 ap.add_argument("-q", "--query", required = True,
19                 help = "path to the query image")
20 args = vars(ap.parse_args())
21
22
23 def hamming2(s1, s2):
24     #Calculate the Hamming distance between two bit strings
25     assert len(s1) == len(s2)
26     return sum(c1 != c2 for c1, c2 in zip(s1, s2))
27
28
29
30 def dhash(image, hash_size = 8):
31     # Grayscale and shrink the image in one step.
32     image = image.convert('L').resize((hash_size, hash_size + 1), Image.ANTIALIAS)
33     pixels = list(image.getdata())
34
35     # Compare vertically adjacent pixels.
36     difference = []
37     for col in range(hash_size):
38         for row in range(hash_size):
39             pixel_left = image.getpixel((col, row))
40             pixel_right = image.getpixel((col, row + 1))
41             difference.append(pixel_left > pixel_right)
42
43     # Convert the binary array to a hexadecimal string.
44     decimal_value = 0
45     hex_string = []
46     for index, value in enumerate(difference):
47         if value:
48             decimal_value += 2**(index % 8)
49         if (index % 8) == 7:
50             hex_string.append(hex(decimal_value)[2:].rjust(2, '0'))
51             decimal_value = 0
52
53     return ''.join(hex_string)
54
55 # loop over the image dataset
56 hashes = []
57 for imagePath in glob.glob(args["dataset"] + "/*"):

```

```

58     # load the image and compute the difference hash
59     image = Image.open(imagePath)
60     k = dhash(image)
61     hashes.append([k,imagePath])
62
63 dbhash = [x[0] for x in hashes] # gives the hash for image
64 path = [x[1] for x in hashes]   # gives the path+filename for the image
65
66 #open input image and calculate difference hash
67 query = Image.open(args["query"])
68 ohash = dhash(query)
69 # print(ohash)
70 # print(dbhash)
71
72 #calculate hamming distance for image
73 for hashes,path in zip(dbhash,path):
74     ham = hamming2(ohash,hashes)
75     # Hamming Distance is zero means duplicate image is detected
76     if (ham == 0):
77         image = Image.open(path)
78         image.show()
79         print("hamming distance is ", ham)
80         print(path)
81     # hamming distance < 11 gives those images which are almost ali
82     elif (ham < 11):
83         image = Image.open(path)
84         image.show()
85         print("hamming distance is ", ham)
86         print(path)

```

This code gives the results for any type of image as it focuses on the hash. Even if image has different type(jpg, png, tif, bmp), then hash changes by smaller factor. Hamming distance in this case will be less. I have created dataset of thousand images. Images are of 100 different categories. The code works for near duplicate images.

Line 15-20 creates the argument parsing. The argparse module makes it easy to write user-friendly command-line interfaces. The program defines what arguments it requires, and argparse will figure out how to parse those out of sys.argv. The argparse module also automatically generates help and

usage messages and issues errors when users give the program invalid arguments.

Line 23-26 is function to give hamming distance for two images. Line 30-53 gives difference hash. To calculate the difference hash, images should be standardized as we want similar hash for near duplicate images. To standardize images, images are grayscaled and shrunk in line 32. They are converted to pixels in line 33. These pixels are converted vertically in line 37-41. They are converted to string to return the hash value.

Next step is to read the images in dataset provided and calculate hash. Line 59 and 60 reads the dataset image and calculates its d-hash in k. They are saved in 'hashes' in 61. Query image is read and hash is calculated in line 67 and line 68.

Comparison of the query image hash and dataset images hashes is done from line 76-86. If hamming distance is 0, then images are similar. If hamming distance is more than 0 and less than 11, then images are near duplicate. Hamming distance can be changed in line 82.



Figure 32: Result for Approach 2 to detect similar images

This code was run for same input image and same database that is used in approach 1. Figure 32 shows 8 image results for the same input image. It gives result irrespective of the file type which supports our purpose. Since the background is same for all the images in figure 8, the results are accurate. For images which are nearly duplicate figure 32 shows the result.



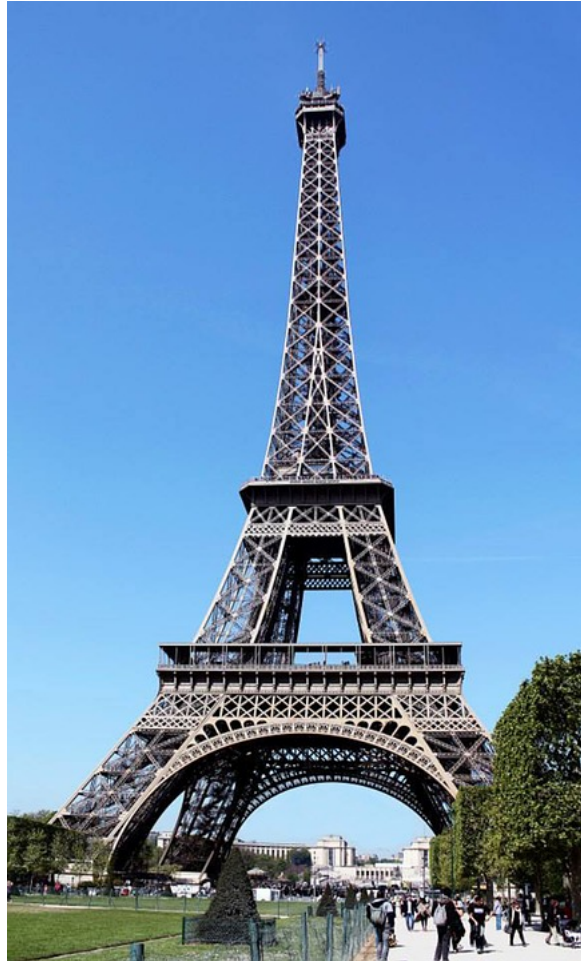


Figure 33: Input test image

Following were test results:



Figure 34: Result images

For input test image shown in figure 33, these all were resulting images for hamming distance of 11 and less. As I have said earlier, we can change

the hamming distance according to our results. There is scope to improve the hamming distance in searching nearly identical images. We can see that resulting images have different details and all are of different sizes and mixed formats, but results were satisfactory. I tested my code over many different image types and many different detailing images. There are some false positives. For example, if both the images have blue sky, then more hamming distance gives false positives. False positives can be removed with human intervention.

Results in figure 32 and 34 are for smaller database of 200 images. I tested the code for bigger dataset of 1000+ images. Results for test query image shown in figure 33 are shown in figure 35. In figure 35, result shows false positives. They can be removed with human intervention. To get more result on Eiffel tower with different contents, i increased hamming distance to 10. Hamming distance of 10 gave me result of other images with white background or black background or similar background. In case of background pixel difference i calculated to be same as the background is consistent for whole image. In case of sky, images have similarity of sky. But that is not the content we want to retrieve. Solution for this is dataset can be divided with the help of contents. These results are with the objects. In case of humans, it shows more accurate results than objects or places as shown in figure 36.

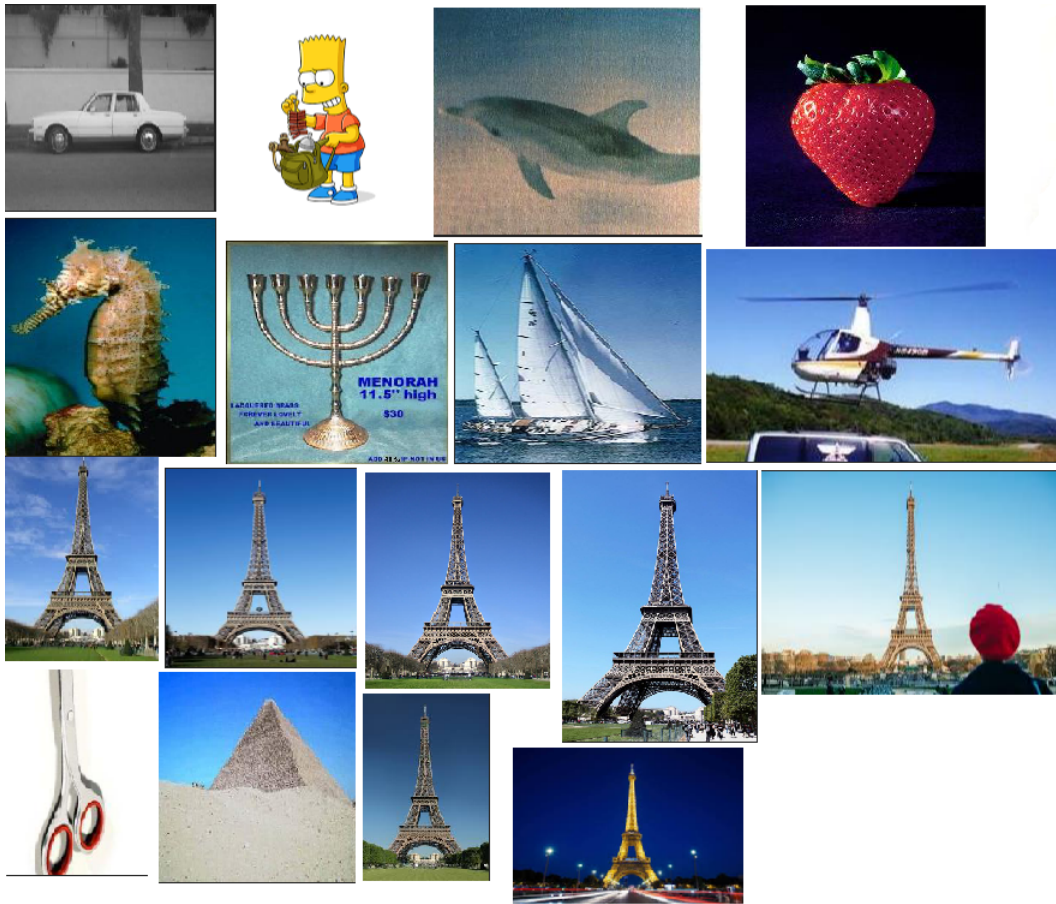


Figure 35: Result images for dataset of 1000 images for query image as Eiffel tower.



Figure 36: Query image: Human face

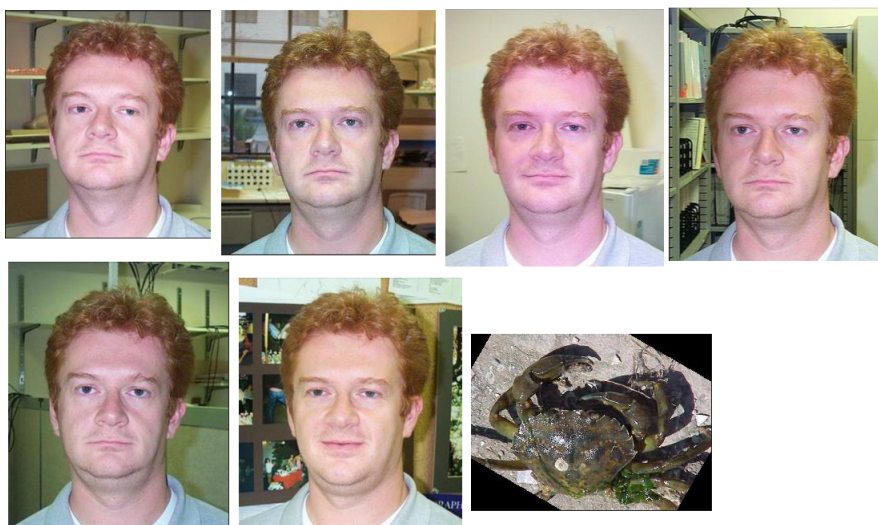


Figure 37: Results for Query image: Human face

In figure 37, we can see the results for figure 36. One of the image is of crab which is false positive. This result because of colors in the image. Human head images have light background and walls are of light olive green

shade. If i reduce hamming distance then we won't get false positive. But in that case we might end up with false negative.

Accuracy can be tested with inverted images. The algorithm is used for d-hash. In d-hash difference between adjacent pixel is calculated. Thus, when image is inverted, the hash of both the images is calculated to be different. In figure 38, we can see hashes of inverted images.


Image	d-hash
	c0c0603889386060
	f9f9e32ee3f9fcfc

Figure 38: d-hash for similar images where one image inverted. Hamming distance is calculated to be 16. None of the bits match.



## 6 Applications

It is pivotal to note that when conducting an investigation, every piece of evidence can be used even if it is remotely related. There is demand for detection of nearly duplicate images in not only Forensics but in other departments. Same goes with recovery of deleted files. Free tools made it easier to recover the files you accidentally deleted as well as in case of crimes.

### 6.1 Detection of nearly duplicate images

The applications of image recognition are wide. There are different ways through which this can be carried out. The relatively recent simplicity with which digital contents can be produced, processed, and distributed has opened a new era, the all-digital world. Different techniques depending on the application of image fingerprinting are introduced. These techniques brought the new edge to the digital innovations. There are some sites that uses this algorithm.

Some of them are: Google reverse image search, tineye.com, ImageWiKi, Bing, eBay, Pixsy, Pinterest etc.

Some 'elite' dating sites use this algorithm to keep away obscene images to keep the accounts clean. This algorithm can be used to find the image results in google database. tineye.com uses this algorithm for image verification, matching, or reverse image-search solutions. They state the application for this MatchEngine algorithm as:[?]

- Travel and booking sites use MatchEngine to verify and update listings by finding duplicate, low-resolution, and modified images.
- Online marketplaces use MatchEngine to visually identify products, find duplicates and derivatives, and match source images.
- Dating sites use MatchEngine to build profile verification and blacklist search solutions.



- Insurance companies and resellers use MatchEngine to build fraud detection and prevention tools.
- Trademark offices use MatchEngine to speed up the process of identifying infringing trademarks and facilitating new trademark registrations.

Similarly purpose of Google's reverse image search is to use a picture as your search to find related images from around the web. When you search using an image, your search results may include:

- Similar images
- Sites that include images
- other sizes of image which is searched

Basic algorithm for reverse image search follows creation of 'unique and compact digital signature or fingerprint' of said image and matching it with indexed images.

Biggest application of this technique is used in law enforcement by forensic department. As said earlier, forensic departments maintain the huge database of images which are related to different crimes. If certain image found in crime scene matches with the image in such databases then this image can be used as evidence to support other evidences.

## **6.2 Recovery of deleted images**

Even if the images are deleted permanently from PC, laptop or any hard disk, then also it can be recovered. Only condition is that it is not a solid state drive and after deleting file, the disk is not overwritten by other file. We have discussed why these files are recoverable.

There are many free tools that can be used to recover deleted files. There are two steps to recovery. First, we need disk image from where the file is deleted. Second, we need the tool to recover the image or files.

Why do they create image of the hard disk? To maintain the authenticity and integrity of criminal evidence, its data should not be tampered. They don't work on original evidence. Imaging helps evidence to be preserved securely without data being tampered. Some of the tools that are best to create the image of hard disk are:

**For creating image of hard disk**

Linux dd	SafeBack	SnapBack
Xplico	SANS SIFT	ProDiscover Basic
Volatility	CAINE	Oxygen Forensics Suite

Some of the free tools that are available are:

**For recovering deleted files using image of hard disk**

Recuva	Puran File Recovery	Disk Drill
Glary Undelete	SoftPerfect File Recovery	EaseUS Data Recovery Wizard
Wise Data Recovery	Restoration	FreeUndelete

## 7 Conclusion and future work

### 7.1 Future Work

Once we have database of all the image hashes, we can call the database with python code.

We will need MySQL driver. The driver can be found at:

<https://sourceforge.net/projects/mysql-python/files/>

After installing MySQLdb, we can import this library in the code to access the images. Here is example of how we can use database in python:

**db.py**

```
1 import MySQLdb
2
3 db = MySQLdb.connect(host="localhost",      # your host, usually localhost
4                       user="john",          # your username
5                       passwd="megajonhy",   # your password
6                       db="jonhydb")         # name of the data base
7
8 # you must create a Cursor object. It will let
9 # you execute all the queries you need
10 cur = db.cursor()
11
12 # Use all the SQL you like
13 cur.execute("SELECT * FROM YOUR_TABLE_NAME")
14
15 # print all the first cell of all the rows
16 for row in cur.fetchall():
17     print row[0]
18
19 db.close()
```

Also, same concept can be extended to videos. It can follow the same algorithm as image fingerprints.

## 7.2 Conclusion

In this paper, we have presented the study and implementation of nearly duplicate images and recovery of deleted files. We have evaluated images of all different types. This paper presents noval approach towards finding nearly duplicate images and can be extended to study of audio and video fingerprinting.

Different hash can be used for this study, e.g. a-hash, p-hash, d-hash. a-hash takes longer time to load and scale images as averaging pixels takes more time than taking difference which is carried out in d-hash. In case of accuracy, p-hash gives best results. But in case of performance it lags behind d-hash. d-hash may give some false positives which can be disregarded by human intervention.

I believe this can be used for more than thousands of images. Future work might include some strategies which can find duplicates with minimal similarity. There is scope to extend the study of this thesis to finding images with their similarity content calculated in percentages.

## References

- [1] BBC: Bitesize. Encoding images. *Webpage*, 2018.
- [2] CoderSource.net. Conversion of a color image to a binary image. *Archives*, 2005.
- [3] Adobe Photoshop User Guide. Color mode. *Webpage*, 2017.
- [4] Ben Hoyt. Duplicate image detection with perceptual hashing in python. *Webpage*, 2017.
- [5] M.S. Rao H.R. Chennamma, Lalitha Rangarajan. Robust near duplicate image matching for digital image forensics. *International Journal of Digital Crime and Forensics*, 2009.
- [6] MacUpdate: <https://www.macupdate.com/app/mac/51196/duplicate-photo-cleaner>. Duplicate photo cleaner. *App*, 2018.
- [7] Deepali Kayande Jignyasa Sanghavi. Content based image retrieval (cbir) system for diagnosis of blood related diseases. *NCIPET*, 2013.
- [8] R. Fergus L. Fei-Fei and P. Perona. Learning generative visual models from few training examples: an incremental bayesian approach tested on 101 object categories,iee,cvpr 2004. *IEEE*, 2004.
- [9] The IconFinder Blog-Martin LeBlanc. Detecting duplicate images using python. *Webpage*, 2014.
- [10] Sam Lundquist. Image file formats: everything youve ever wanted to know. *Blog:https://99designs.com/blog/tips/image-file-types/*, 2017.
- [11] Real Python. Fingerprinting images for near-duplicate detection. *Webpage*, 2014.
- [12] A. Walker R. Fisher, S. Perkins and E. Wolfart. Grayscale images. *Webpage*, 2003.
- [13] Adrian Rosebrock. The complete guide to building an image search engine with python and opencv. *Tutorials*, 2014.
- [14] S.Farooq. Multispectral images. *Webpage*, 2018.

- [15] Onilne tech tips. 5 photo recovery tools tested and reviewed. *Webpage*, 2018.
- [16] Ultimate Photo Tips. What is pixel? *Webpage*, 2018.
- [17] Wesley Tucker. How are graphics and images stored in a computer? *Techwalla Webpage*, 2018.
- [18] Vector-conversion.com. Raster(bitmap) vs. vector. *Webpage*, 2018.
- [19] John Wallace. Multispectral imaging: Transverse-field-detector sensor has 36 color channels. *Webpage*, 2014.
- [20] Conny Wallstrom. 8-bit vs 16-bit what color depth you should use and why it matters. *Webpage*, 2015.
- [21] Qionghai Dai Jinwei Gu Wei Jiang, Guihua Er. Similarity-based online feature selection in content-based image retrieval. *IEEE*, 2006.
- [22] Houqiang Li Wengang Zhou and Qi Tian. Recent advance in content-based image retrieval: A literature survey, ieee. *IEEE*, 2018.