Aneka- Detecting various forms of the same Wavelet Image Hashing Algorithm

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

Geetanjali Agarwal

Advisor: Dr. Michael Soltys

December 2018

APPROVED FOR MS IN COMPUTER SCIENCE

Advisor: Dr. Michael Soltys **Date**

Dr. Brian Thoms **Date**

Dr. Houman Dallali **Date**

APPROVED FOR THE UNIVERITY

Dr. Osman Ozturgut **Date**

Aneka – Detecting various forms of the same Wavelet Image Hashing Algorithm

Geetanjali Agarwal

December 3, 2018

Abstract

Digital imaging has experienced tremendous growth in recent decades, and have been used in a growing number of applications. With such increasing popularity and the availability of low-cost image editing software, the integrity of digital image content can no longer be taken for granted. This thesis introduces a new methodology for the forensic analysis of digital images. It proposes a novel hashing method using scale-invariant feature transform (SIFT) features points and Discrete Wavelet Transform (DWT) approximation coefficients for image authentication. Experimental results show that the proposed method is robust to various content-preserving operations. In addition, the performance of the proposed method is compared to existing methods. The comparison results show that the proposed method performs better than the existing methods. This thesis also mentions about the

Amazon Web Services that are being used in detail. Also, the name of this thesis — *Aneka* means that which have many variations. This thesis also talks about recognizing nearly duplicate/similar images or detecting differents variations of an image present in a database. *Aneka* is also one of the names of Lord *Vishnu*.

Contents

List of Figures

1 Introduction

Due to the ever-increasing digitalization, the authentication of multimedia content is becoming more and more important. Authentication in general means deciding whether an object is authentic or not. That is, if it matches a given original object. The authentication depends heavily on the type of the object. When authenticating an executable file, it is important that every single bit exactly matches the original executable. Cryptographic hash functions are adequate for such tasks. To check the authenticity of multimedia content, other methods are better suited. A multimedia object, e.g. an image, can have different digital representations that all look the same to the human perception. Powerful multimedia tools make digital images, manipulate it easily and therefore bring a series of problems, such as image authentication, image copy detection and image forensics.

In 1996, Schneider and Chang[20] introduced the pioneer work of image hashing. From then on, many researchers have devoted themselves to developing image hashing. In terms of the used techniques, the existing algorithms can be roughly classified into the following categories.

1. Discrete wavelet transform (DWT): Statistics of DWT coefficients are firstly exploited to generate hashes. This method is sensitive to contrast adjustment and gamma correction. Researchers used the end-stopped wavelet transform to detect visually significant points for hash construction.

Their scheme is robust against JPEG compression and small-angle rotation. Ahmed et al. [15] presented a hash scheme based on DWT and SHA-1. It is fragile to brightness adjustment, contrast adjustment and rotation. The MH-based hashing is robust against rotation.

2. Discrete cosine transform (DCT): DCT coefficients can indicate visual content and used it to build robust hashes for digital watermarking. This scheme is also sensitive to rotation. later, a method was designed based on invariant relation between DCT coefficients at the same position in separate blocks. This method can distinguish JPEG compression from malicious attacks, but it is vulnerable to some perceptually insignificant modifications.

3. Radon transform (RT): Motivated by RT, Roover et al.[9] designed a scheme called RASH method which extracts robust features from a set of radial projections of pixels. This method is also resilient to rotation, but its discrimination needs to be improved. Ou et al. exploited the RT and DCT to construct robust hashes. The RT-DCT hashing is resistant to JPEG compression and filtering, but its discrimination is not desirable either.

4. Fourier transform: Swaminathan et al. [5] exploited Fourier coefficients to generate image hashes. This method is resistant to moderate geometric transforms and filtering. He also designed a method combining RT, DWT and Fourier transform. This hashing is robust against print-scan

attack.

5. Matrix factorisation: Kozat et al. [5] viewed images and attacks as a sequence of linear operators, and calculated hashes using Singular Value Decompositions (SVDs). The SVD- SVD hashing is robust against rotation at the cost of significantly increasing misclassification. He first used non-negative matrix factorisation (NMF) to derive hashing. Their hashing is resilient to geometric attacks, but sensitive to watermark embedding. He found invariant relation in NMF coefficient matrix and used it to construct hashes. The method is robust against normal digital operations, but fragile to rotation.

Besides the above techniques, other strategies have also been reported. For example, Khelifi and Jiang [6] proposed a hashing with the theory of watermark detection. Lu and Wu [10] designed a hashing based on visual words. Zhao and Wei [19] exploited Zernike moments to generate hashes. Tang et al. [9] used structural features to design hashing. A common weakness of is sensitive to rotation. Recently, Liu et al. [5] calculated hashes by wave atom transform and grey code. Lv and Wang [9] introduced an algorithm based on scale invariant feature transform and Harris detector. Li et al. [9] calculated hashes using random Gabor filtering (GF) and dithered lattice vector quantisation (LVQ). The GFLVQ hashing is resistant to JPEG compression and rotation, but its discrimination is not good enough. Qin et al. extracted hashes with Fourier transform and non-uniform sampling. Zhao et al. [10] used Zernike moments and statistics of salient region to construct hashes. The algorithms [9] only tolerate small-angle rotation. Although many algorithms have been designed, there are still some practical problems. For example, some algorithms can resist normal digital operations, but their discriminations are not desirable. This means that when they are applied to copy detection, the retrieved images will include some unexpected different images, leading to a low retrieval efficiency. This paper discusses about the use of colour vector angle in image hashing, and then proposes a robust hashing with colour vector angles. Since vector angle is effective in evaluating colour differences, the proposed algorithm can effectively extract colour features and then make generated hash discriminative. Many experiments are conducted and the results show that this algorithm can reach a good balance between robustness and discrimination and outperforms some well-known algorithm.

1.1 Motivation

In the last decade, an emerging multimedia technology called image hashing attracts many researchers attentions. It uses a short string called hash to represent an input image and finds applications in image retrieval, image authentication, digital watermarking, copy detection and so on. In practice, digital images often undergo normal digital operations, such as JPEG compression, brightness adjustment, contrast adjustment, geometric transform, watermark embedding, image filtering and gamma correction. These operations change digital representations of images, but keep their visual appearances unchanged. Each of these image processing steps changes the binary representation of the image. This means that image hash should be a visual content-based representation. Therefore, although classical cryptographic hash functions, for example, MD5 and SHA-1,[9] can map any message into a fixed-size string, they are not suitable for image hashing because of their sensitivity to bit-level change. In general, image hashing must satisfy two basic properties,

(1) **Perceptual robustness**: visually identical images have the same or very similar hashes no matter what their digital representations are. In other words, image hashing should be robust against normal digital operations.

(2) **Discriminative capability**: different images have different hashes. It means that hash distance between different images should be big enough.

In addition, image hashing should satisfy additional properties when it is applied to some applications. For example, it should be key-dependent for image forensics. Note that the two basic properties conflict with each other. Perceptual robustness amounts to robustness under small perturbations whereas the discriminative capability needs minimization of collision for different images. High performance algorithms should keep a good balance between them.

Forensic departments believe that anything can be treated as an evidence if it was found at a crime scene. They also believe that criminals ought to leave at least some evidence behind even though they try to clean the place rid of evidences. 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 mosaic and 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, lets say under keyword of kidnapping, then this image can be connected with the crime. This will strengthen the evidence. This paper concentrates on fulfiling these tasks.

This paper focuses on designing an algorithm that will be useful in image authentication and forensics. It is divided into two parts. First part of it focuses on detection of duplicate images/ nearly duplicate images through Wavelet Image Hashing Algorithm. This algorithm for detection of nearly duplicate images was tested through other previously implemented algorithms- Perceptual Hashing, Average Hashing, Difference Hashing.

2 Background

This chapter briefly discusses the theoretical and practical aspects of the technologies used for creation of the solution proposed in this paper. This chapter also explains basics of Image processing and the previously implemented algorithms for Image hashing. Starting with Amazon Web Services(AWS), the software calculates the hash of the image and then compares it with the hashes already present in the database to check for similar hashes. If it finds one, its a hit!

2.1 Amazon Web Services(AWS)

1. **EC2 instance:**

Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides secure, resizable compute capacity in the cloud. It is designed to make web-scale cloud computing easier for developers. Amazon EC2s simple web service interface allows user to obtain and configure capacity up or down according to conditions you define. It provides user with complete control of user computing resources and lets user run on Amazons proven computing environment. Amazon EC2 reduces the time required to obtain and boot new server instances to minutes, allowing user to quickly scale capacity, both up and down, as user computing requirements change. Amazon EC2 changes the economics of computing by allowing user to pay only for capacity that user actually use. Amazon EC2 provides developers the tools to build failure resilient applications and isolate them from common failure scenarios.

Benefits of EC2:

a. Elastic web-scale computing:

Amazon EC2 enables user to increase or decrease capacity within minutes, not hours or days. User can commission one, hundreds, or even thousands of server instances simultaneously. User can also use amazon EC2 auto scaling to maintain availability of EC2 fleet and automatically scale fleet up and down depending on its needs in order to maximize performance and minimize cost. To scale multiple services, user can use aws auto scaling.

b. Completely controlled:

User have complete control of the instances including root access and the ability to interact with them as user would any machine. User can stop any instance while retaining the data on the boot partition, and then subsequently restart the same instance using web service apis. Instances can be rebooted remotely using web service application interface', and user also have access to their console output.

c. Flexible cloud hosting services:

User have the choice of multiple instance types, operating systems, and software packages. Amazon ec2 allows user to select a configuration of memory, cpu, instance storage, and the boot partition size that is optimal for users choice of operating system and application. For example, choice of operating systems includes numerous linux distributions and microsoft windows server.

d. Integrated:

Amazon EC2 is integrated with most aws services such as amazon simple storage service (amazon S3), Amazon Relational database service (Amazon RDS), and Amazon Virtual Private cloud (Amazon VPC) to provide a complete, secure solution for computing, query processing, and cloud storage across a wide range of applications.

e. Reliable:

Amazon EC2 offers a highly reliable environment where replacement instances can be rapidly and predictably commissioned. The service runs within amazons proven network infrastructure and data centers. The amazon EC2 service level agreement commitment is 99.99% availability for each amazon EC2 region.

f. Secure:

Cloud security at aws is the highest priority. As an aws customer, user will benefit from a data center and network architecture built to meet the requirements of the most security-sensitive organizations. Amazon ec2 works in conjunction with amazon vpc to provide security and robust networking functionality to compute resources.

g. Inexpensive:

Amazon ec2 passes on to user the financial benefits of amazons scale. User pay a very low rate for the compute capacity user actually consume. See amazon ec2 instance purchasing options for more details.

h.Easy to start:

There are several ways to get started with amazon ec2. User can use the aws management console, the aws command line tools (cli), or aws sdks. Aws is free to get started. To learn more, please visit our tutorials.

2. **S3 bucket:**

Industries today need the ability to simply and securely collect, store, and analyze their data at a massive scale. Amazon S3 is object storage built to store and retrieve any amount of data from anywhere web sites and mobile apps, corporate applications, and data from IoT sensors or devices. It is designed to deliver 99.99% durability, and stores data for millions of applications used in every industry. S3 provides comprehensive security and compliance capabilities that meet even the most stringent regulatory requirements. It gives customers flexibility in the way they manage data for cost optimization, access control, and compliance. S3 provides query-in-place functionality, allowing you to run powerful analytics directly on your data at rest in S3. And Amazon S3 is the most supported cloud storage service available, with integration from the largest community of third-party solutions, systems integrator partners, and other AWS services.

Benefits of S3 bucket:

a. Flexible management:

Amazon S3 offers the most flexible set of storage management and administration capabilities. Storage administrators can classify, report, and visualize data usage trends to reduce costs and improve service levels. Objects can be tagged with unique, customizable metadata so customers can see and control storage consumption, cost, and security separately for each workload. The s3 inventory feature delivers scheduled reports about objects and their metadata for maintenance, compliance, or analytics operations. Since amazon S3 works with AWS Lambda, customers can log activities, define alerts, and invoke workflows, all without managing any additional infrastructure.

b. Query in place:

Amazon S3 allows you to run sophisticated big data analytics on your data without moving the data into a separate analytics system. Amazon Athena gives anyone who knows sql on-demand query access to vast amounts of unstructured data. Amazon RedShift spectrum lets you run queries spanning both your data warehouse and S3. And only AWS offers Amazon S3 select, a way to retrieve only the subset of data you need from an S3 object, which can improve the performance of most applications that frequently access data from S3 by up to 400%.

c. Unmatched durability, availability and scalability:

Amazon S3 runs on the worlds largest global cloud infrastructure and is designed from the ground up to deliver 99.99% of durability. Data in amazon S3 Standard, S3 Standard-IA, and Amazon Glacier Storage classes is automatically distributed across a minimum of three physical Availability Zones (AZs) that are typically miles apart within an AWS region. The amazon S3 one zone-IA storage class stores data in a single AZ, and is ideal for customers who want a lower cost option for infrequently accessed data and do not require the availability and resilience of S3 standard storage. Amazon S3 can also automatically replicate data to any other AWS region.

3. **Amazon DynamoDB:**

Amazon DynamoDB is a nonrelational database that delivers reliable performance at any scale. It's a fully managed, multi-region, multi-master database that provides consistent single-digit millisecond latency, and offers built-in security, backup and restore, and in-memory caching.

Benefits of Amazon DynamoDB:

a. Performance at scale:

DynamoDB delivers consistent, single-digit millisecond responsiveness at any scale. Build apps with virtually unlimited throughput and storage. Add an in-memory cache that reduces response times from milliseconds to microseconds, without any app changes.

b. Fully managed

Dynamodb is a serverless database that automatically scales throughput up or down, and continuously backs up your data for protection. Dynamodb gives your globally distributed applications fast access to local data by replicating tables across multiple aws regions.

c. Enterprise-ready

Built for mission-critical workloads. Your data is secured with encryption and guaranteed reliability with a service level agreement. You have full oversight of your tables with fine-grained access control, integrated monitoring tools, and support for private connections over VPN.

2.2 Database shelve

A shelf is a persistent, dictionary-like object. The difference with dbm databases is that the values (not the keys!) 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. 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 (see Example). If the optional writeback parameter is set to True, all entries accessed are also cached in memory, and written back on sync() and close(); this can make it handier to mutate mutable entries in the persistent dictionary, but, if many entries are accessed, it can consume vast amounts of memory for the cache, and it can make the close operation very slow since all accessed entries are written back (there is no way to determine which accessed entries are mutable, nor which ones were actually mutated).

Shelf objects support all methods supported by dictionaries. This eases the transition from dictionary based scripts to those requiring persistent storage.

The two important terms that are important to be understand are pickle and shelve-

1. **pickle** is for serializing some object (or objects) as a single byte stream

in a file.

2. **shelve** builds on top of pickle and implements a serialization dictionary where objects are pickled, but associated with a key (some string), so we can load the shelved data file and access the pickled objects via keys. This could be more convenient were you to be serializing many objects.

2.3 phpMyAdmin

phpMyAdmin is a free software tool written in PHP that is intended to handle the administration of a MySQL or MariaDB database server. You can use phpMyAdmin to perform most administration tasks, including creating a database, running queries, and adding user accounts.

Supported features:

Currently phpMyAdmin can:

1. create, browse, edit, and drop databases, tables, views, columns, and indexes
2. display multiple results sets through stored procedures or queries
3. create, copy, drop, rename and alter databases, tables, columns and indexes

4. maintenance server, databases and tables, with proposals on server configuration
5. execute, edit and bookmark any SQL-statement, even batch-queries
6. load text files into tables
7. create and read dumps of tables
8. export data to various formats: CSV, XML, PDF, ISO/IEC 26300 - OpenDocument Text and Spreadsheet, Microsoft Word
9. 2000, and LATEX formats
10. import data and MySQL structures from OpenDocument spreadsheets, as well as XML, CSV, and SQL files
11. administer multiple servers
12. add, edit, and remove MySQL user accounts and privileges
13. check referential integrity in MyISAM tables
14. using Query-by-example (QBE), create complex queries automatically connecting required tables

2.4 Image Preprocessing

Image preprocessing is form of signal processing for which the input is an image, such as a picture; the output of image pre-processing may be either an image or, a set of characteristics or parameters related to the image. Most image pre-processing techniques involve treating the image as a two-dimensional signal and applying standard signal-processing techniques to it. segmentation refers to the process of partitioning a digital image into multiple segments (sets of pixels, also known as super pixels). The goal of segmentation is to simplify and/or change the representation of an image into something that is more meaningful and easier to analyze. Image segmentation is typically used to locate objects and boundaries in images.

Preprocessing consists of those operations that prepare data for subsequent analysis that attempts to correct for systematic errors. The digital images are subjected to several corrections. After the pre-processing is complete, the original images are pre-processed to make the dimensionality more adaptable to processing which also helps to make the processing faster.

1. **Lab Colour Space**

A Lab colour space is a colour opponent space with dimension L for lightness and a and b for the colour opponent dimensions, based on nonlinearly compressed CIE XYZ colour space coordinates. "Lab" colour spaces is to create a space which can be computed via simple formulas from the XYZ space,but is more perceptually uniform than XYZ. Perceptually uniform means that a change of the same amount in a colour value should produce a change of about the same visual importance. When storing colours in limited precision values, this can improve the reproduction of tones. Both Lab spaces are relative to the white point of the XYZ data they were converted from. Lab values do not define absolute colours unless the white point is also specified.[3]The goal is to identify different colours in image by analyzing the $L^*a^*b^*$ colour space. The image was acquired using the Image Acquisition Toolbox.

Step 1: **Acquire Image** Read the image, which is an colourful image instead of using gray image.

Step 2:**Calculate Sample Colours in $L^*a^*b^*$ Colour Space for each region.** The $L^*a^*b^*$ colour space is derived from the CIE XY tristimulus values. The $L^*a^*b^*$ space consists of a luminosity 'L^*' layer, chromaticity layer 'a^*' indicating where colour falls along the red-green axis, and chromaticity layer 'b^*' indicating where the colour falls along the blue-yellow axis. [18]Your approach is to choose a small sample region for each colour and to calculate each sample region's average colour in 'a^*b^*' space.

Step 3: **Classify Each Pixel Using the Nearest Neighbour rule each colour marker now has an 'a^*' and a 'b^*' value.** The smallest distance will tell you that the pixel most closely matches that colour marker.

Step 4: **Display Results of Nearest Neighbour Classification** The label matrix contains a colour label for each pixel in the fabric image. Use the label matrix to separate objects in the original fabric image by colour.

Step 5: **Display ’a*’ and ’b*’ Values of the Labelled Colours.** The nearest neighbour classification separated the different colour populations by plotting the ’a*’ and ’b*’ values of pixels that were classified into separate colours. For display purposes, label each point with its colour label.

The three coordinates of LAB represent the lightness of the color ($L^* = 0$ yields black and $L^* = 100$ indicates diffuse white; specular white may be higher), its position between red/magenta and green (a^*, negative values indicate green while positive values indicate magenta) and its position between yellow and blue (b^*, negative values indicate blue and positive values indicate yellow) coordinate ranges from 0 to 100.

The possible range of a^* and b^* coordinates is independent of the colour space that one is converting from, since the conversion uses X and Y which come from RGB the red/green and yellow/blue opponent channels are computed as differences of lightness transformations of cone responses, CIELAB is a chromatic value colour space The nonlinear relations for L^*, a^*, and b^* are intended to mimic the nonlinear response of the eye. [4]Furthermore, uniform changes of components in the $L^*a^*b^*$ colour space aim to correspond to uniform changes in perceived colour, so the relative perceptual differences between any two colours in $L^*a^*b^*$ can be approximated by treating each colour as a point in a three dimensional space. The L*a*b* colour space includes all perceivable colours which means that its gamut exceeds those of the RGB and CMYK colour models.[12] One of the most important attributes of the L*a*b*-model is the device independency. This means that the colours are defined independent of their nature of creation or the device they are displayed on. The L*a*b* color space is used e.g. in Adobe Photoshop when graphics for print have to be converted from RGB to CMYK, Your goal is to identify different colours in image by analyzing the L*a*b* colour space.

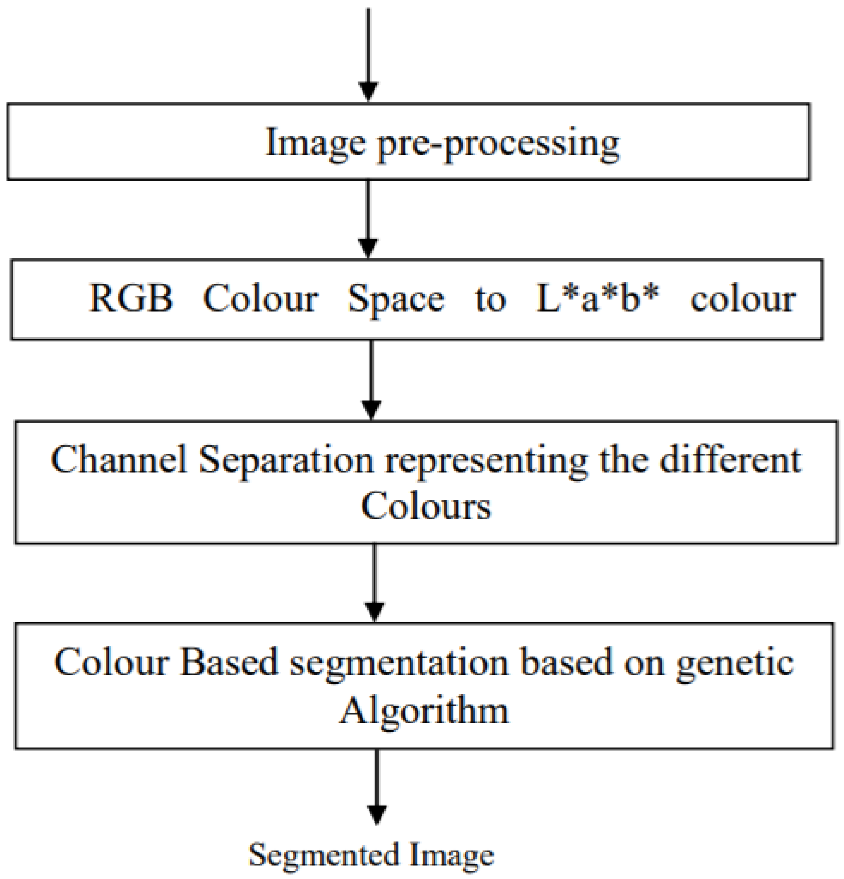


Image pre-processing
RGB Colour Space to L*a*b* colour
Channel Separation representing the different Colours
Colour Based segmentation based on genetic Algorithm
Segmented Image

Figure 1: Fig. Scheme of Segmentation process

The difference between the two points in the $L^*a^*b^*$ colour space is same with the human visual system. Since the $L^*a^*b^*$ model is a three-dimensional model, it can only be represented properly in a three-dimensional space.[13] The solution to convert digital images from the RGB space to the $L^*a^*b^*$ colour space is given by the following formula:

$$L^* = 116 f\left(Y/Y_{\mathrm{n}}\right) - 16$$

$$a^{*} = 500\big[f\,(X/X_{\mathrm{n}}) - f\,(Y/Y_{\mathrm{n}})\big]$$

$$b^{*} = 200\big[f\,(Y/Y_{\mathrm{n}}) - f\,(Z/Z_{\mathrm{n}})\big]$$

X, Y, Z, X_n, Y_n, and Z_n are the coordinates of CIEXYZ colour space. The solution to convert digital images from the RGB space to the CIEXYZ colour space is as the following formula.

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.608 & 0.174 & 0.201 \\ 0.299 & 0.587 & 0.114 \\ 0.000 & 0.066 & 1.117 \end{bmatrix} \times \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

X_n, Y_n, and Z_n are respectively corresponding to the white value of the parameter.

$$\begin{bmatrix} f\,(x) \end{bmatrix} = \begin{bmatrix} X^{1/3} & x\rangle 0.008856 \\ 7.787x + 16/116 & x \geq 0.008856 \end{bmatrix}$$

Colour space conversion is the translation of the representation of a colour from one basis to another. This typically occurs in the context of converting an image that is represented in one colour space to another colour space.

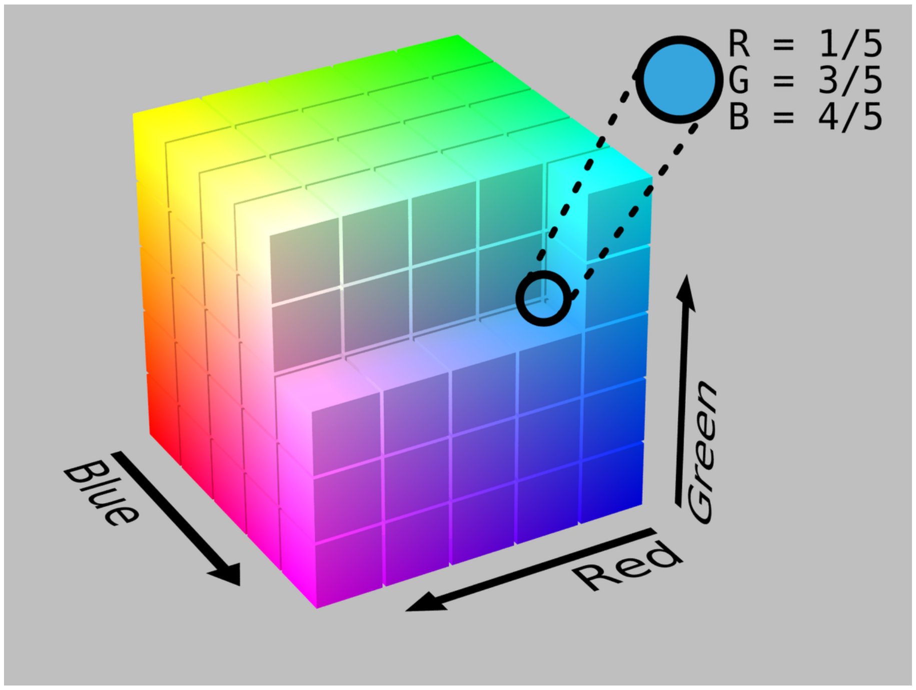


Figure 2: Fig. L*a*b* color space converted image

2.5 Image Processing

Image processing is a method to convert an image into digital form and perform some operations on it, in order to get an enhanced image or to extract some useful information from it. It is a type of signal dispensation in which input is image, like video frame or photograph and output may be image or characteristics associated with that image. Usually Image Processing system includes treating images as two dimensional signals while applying already set signal processing methods to them. It is among rapidly growing technologies today, with its applications in various aspects of a business. Image Processing forms core research area within engineering and computer science disciplines too.

Image processing basically includes the following three steps.

Step 1: Importing the image with optical scanner or by digital photography.

Step 2: Analyzing and manipulating the image which includes data compression and image enhancement and spotting patterns that are not to human eyes like satellite photographs.

Step 3: Output is the last stage in which result can be altered image or report that is based on image analysis.

Purpose of Image processing:
The purpose of image processing is divided into 5 groups. They are:

1. Visualization - Observe the objects that are not visible.
2. Image sharpening and restoration - To create a better image.
3. Image retrieval - Seek for the image of interest.
4. Measurement of pattern Measures various objects in an image.
5. Image Recognition Distinguish the objects in an image.

Types:

The two types of methods used for Image Processing are Analog and Digital Image Processing. Analog or visual techniques of image processing can be used for the hard copies like printouts and photographs. Image analysts use various fundamentals of interpretation while using these visual techniques.[2] The image processing is not just confined to area that has to be studied but on knowledge of analyst. Association is another important tool in image processing through visual techniques. So analysts apply a combination of personal knowledge and collateral data to image processing.

Digital Processing techniques help in manipulation of the digital images by using computers. As raw data from imaging sensors from satellite platform contains deficiencies. To get over such flaws and to get originality of information, it has to undergo various phases of processing. The three general phases that all types of data have to undergo while using digital technique are Pre- processing, enhancement and display, information extraction.[14]

The diagram below depicts the steps involved in Image processing-

2.6 Image Recognition

Image recognition, in the context of machine vision, is the ability of software to identify objects, places, people, writing and actions in images. Com-

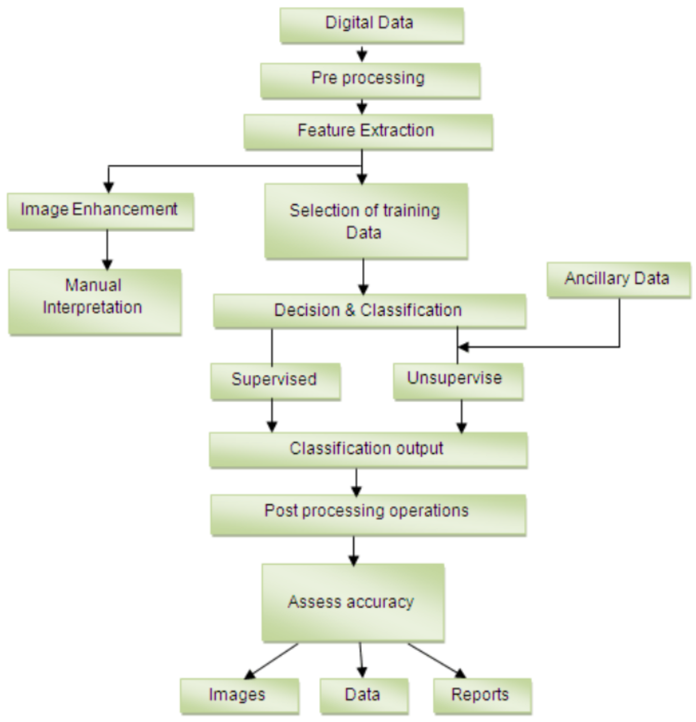
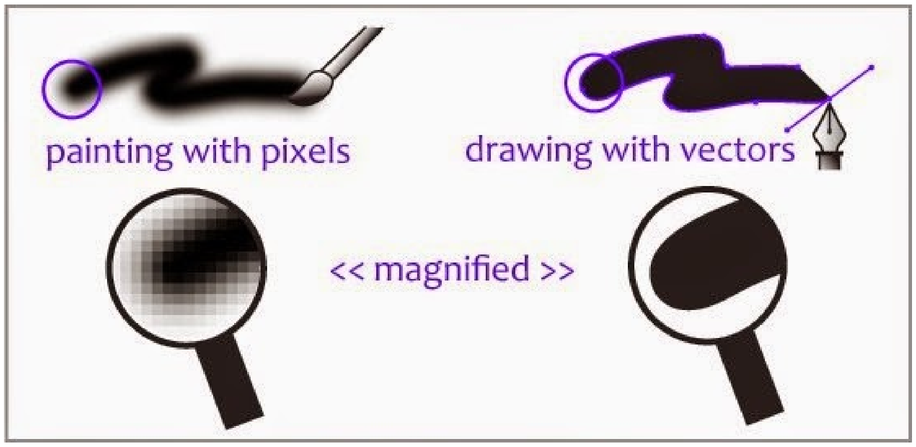


Figure 3: Fig. Steps involved in Image processing

puters can use machine vision technologies in combination with a camera and artificial intelligence software to achieve image recognition.[1] Image recognition is used to perform a large number of machine-based visual tasks, such as labeling the content of images with meta-tags, performing image content search and guiding autonomous robots, self-driving cars and accident avoidance systems.



2.6.1 Working of image recognition

1. Gather and Organize Data

The human eye perceives an image as a set of signals which are processed by the visual cortex in the brain. This results in a vivid experience of a scene, associated with concepts and objects recorded in ones memory. Image recognition tries to mimic this process. Computer perceives an image as either a raster or a vector image. Raster images are a sequence of pixels with discrete numerical values for colors while vector images are a set of color-annotated polygons.

To analyze images the geometric encoding is transformed into constructs depicting physical features and objects. These constructs can then be logically analyzed by the computer. Organizing data involves classification and feature extraction. The first step in image classification is to simplify the image by extracting important information and leaving out the rest. For example, while extracting main figure from the background you will notice a significant variation in RGB pixel values.[16]

However, it can be simplified by running an edge detector on the image. It is easy to discern the circular shape of the face and eyes in these edge images and so it can be concluded that edge detection retains the essential information while throwing away non-essential information. Some well-known feature descriptor techniques are Haar-like features introduced by Viola and Jones, Histogram of Oriented Gradients (HOG), Scale-Invariant Feature Transform (SIFT), Speeded Up Robust Feature (SURF) etc.

2. Build a Predictive Model

This section mentions about how a classification algorithm takes this feature vector as input and outputs a class label (e.g. brain/no brain). Before a classification algorithm can do its magic, we need to train it by showing thousands of brain images. The general principle in machine learning algorithms is to treat feature vectors as points in higher dimensional space. Then it tries to find planes or surfaces (contours) that separate higher dimensional space in a way that all examples from a particular class are on one side of the plane or surface.

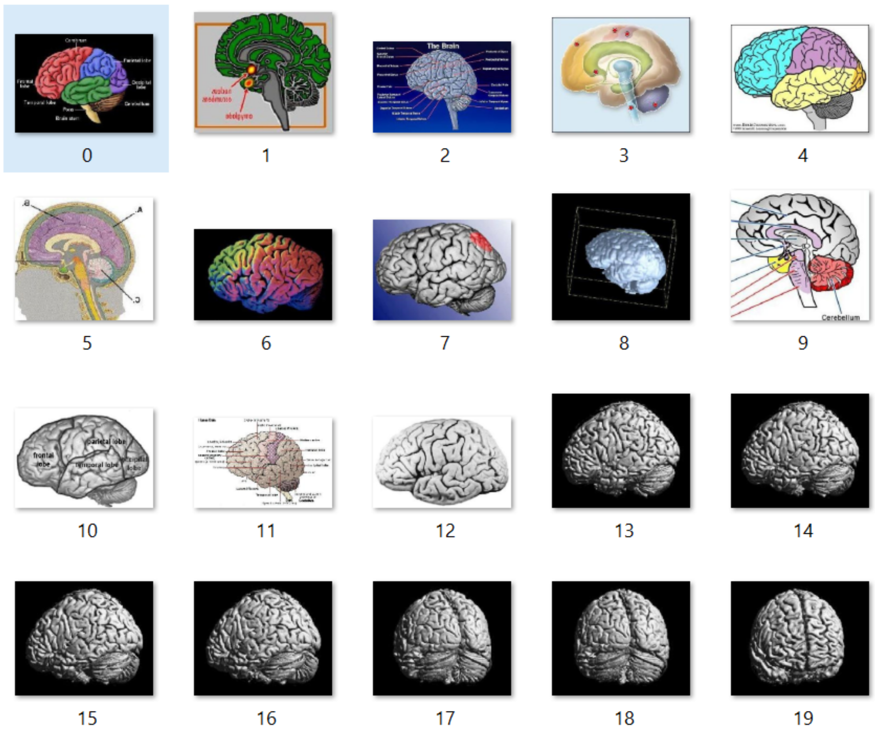


Figure 4: Brain images

3. Recognize Images

While the above two steps take up most of the effort, this step to recognize image is pretty easy. The image data, both training, and test are organized. Training data is different from test data, which also means we remove duplicates (or near duplicates) between them. This data is fed into the model to recognize images. We have to find the image of a cat in our database of known images which has the closest measurements to our test image. All we need to do is train a classifier that can take the measurements from a new test image and tells us about the closest match with a cat. Running this classifier takes milliseconds. The result of the classifier is all the brain images that look similar to the test image.

The major challenges in building an image recognition model are hardware processing power and cleansing of input data. It can be possible that most of the images might be high definition. While dealing with large images of size more than 500 pixels, it becomes 250,000 pixels (500 X 500) per image.[7] A training data of mere 1000 images will amount to 0.25 billion values for the machine learning model. Moreover, the calculations are not easy addition or multiplication, but complex derivatives involving floating point weights and matrices.

There are some quick hacks to overcome the above challenges:

1. Image compression tools to reduce image size without losing clarity
2. Use grayscale and gradient version of colored images
3. Graphic processor units (GPU) To train the neural networks containing large data sets in less time and with less computing infrastructure.

Image recognition is used in many applications like systems for factory automation, toll booth monitoring, and security surveillance.

This paper concentrates on detecting similar images that have been resized, cropped or adulterated with filters. This can be done through a technique called Image Hashing.

2.7 Image Hashing

While normally hashing a file hashes the individual bits of data of the file, image hashing works on a slightly higher level. The difference is that with image hashing, if two pictures look practically identical but are in a different format, or resolution (or there is minor corruption, perhaps due to compression) they should hash to the same number. Despite the actual bits of their data being totally different, if they look parctically identical to a human, they hash to the the same thing.

Hashing is a function that applies to an arbitrary data and produces the data of a fixed size (mostly a very small size). There are many different types of hashes, but talking about image hashing, it is used either to:

a. find duplicates very fast. Instead of searching for the whole image, look for the hash of the image.

b. finding similar images

Images that look identical to us, can be very different if you will just compare the raw bytes. This can be due to:

1. resizing,

2. rotation,
3. slightly different color gamma,
4. different format,
5. some minor noise, watermarks and artifacts.

2.8 Crypographic Hash functions

A cryptographic hash function is a special class of hash function that has certain properties which make it suitable for use in cryptography. It is a mathematical algorithm that maps data of arbitrary size to a bit string of a fixed size (a hash) and is designed to be a one-way function, that is, a function which is infeasible to invert. The only way to recreate the input data from an ideal cryptographic hash function's output is to attempt a brute-force search of possible inputs to see if they produce a match, or use a rainbow table of matched hashes. Bruce Schneier[11] has called one-way hash functions "the workhorses of modern cryptography". The input data is often called the message, and the output (the hash value or hash) is often called the message digest or simply the digest.

The ideal cryptographic hash function has five main properties:

1. it is deterministic so the same message always results in the same hash.
2. it is quick to compute the hash value for any given message.

3. it is infeasible to generate a message from its hash value except by trying all possible messages.
4. a small change to a message should change the hash value so extensively that the new hash value appears uncorrelated with the old hash value.
5. it is infeasible to find two different messages with the same hash value.

Cryptographic hash functions have many information-security applications, notably in digital signatures, message authentication codes (MACs), and other forms of authentication. They can also be used as ordinary hash functions, to index data in hash tables, for fingerprinting, to detect duplicate data or uniquely identify files, and as checksums to detect accidental data corruption. Indeed, in information-security contexts, cryptographic hash values are sometimes called (digital) fingerprints, checksums, or just hash values, even though all these terms stand for more general functions with rather different properties and purposes.

So, there is a need a hash function which will create a similar (or even identical) hash for similar images. The most well known algorithms are mentioned in Section 2.6

2.9 Hamming distance

Given two vectors u, v $\in F^n$, we define the hamming distance between u and v, d (u, v) to be the number of places where u and v differ.

Thus the Hamming distance between two vectors is the number of bits we must change to change one into the other.

Example Find the distance between the vectors 01101010 and 11011011.

$$01101010$$
$$11011011$$

They differ in four places, so the Hamming distance
d (01101010, 11011011) = 4

2.10 Error correcting codes

Error correcting codes are used in many places, wherever there is the possibility of errors during transmission. Some examples are NASA probes (Galileo), CD players and the Ethernet transmission protocol.

We assume that the original message consists of a series of bits, which can be split into equal size blocks and that each block is of length n, i.e. a member of F^n

The usual process consists of the original block $x \in F^n$ this is then encoded by some encoding function to $u \in F^{n+k}$ which is then sent across some (noisy) channel. At the other end the received value $v \in F^{n+k}$ is decode by

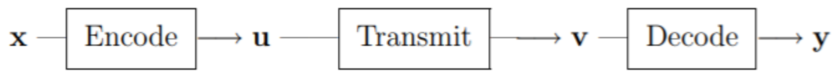


Figure 5: Error Correcting Codes

means of the corresponding decoding function to some $y \in F^n$.

If there are no errors in the channel $u = v$ and $x = y$.

2.11 Image Hashing Algorithms (Previously implemented)

1. **Average Hashing (aHash):**

With pictures, high frequencies give you detail, while low frequencies show you structure. A large, detailed picture has lots of high frequencies. A very small picture lacks details, so it is all low frequencies. This approach crushes the image into a grayscale 8x8 image and sets the 64 bits in the hash based on whether the pixel's value is greater than the average color for the image.[13]

Following are the steps to show how the Average hash algorithm works-

Step 1: Reduce size. The fastest way to drastically remove high frequencies and detail is to shrink the image. In this case, shrink it to 8x8 so that there are 64 total pixels. Don't bother keeping the aspect ratio, just crush it down to fit an 8x8 square. This way, the hash will match any variation of the image, regardless of scale or aspect ratio.

Step 2: Reduce color. The tiny 8x8 picture is converted to a grayscale. This changes the hash from 64 pixels (64 red, 64 green, and 64 blue) to 64 total colors.

Step 3: Average the colors. Compute the mean value of the 64 colors.

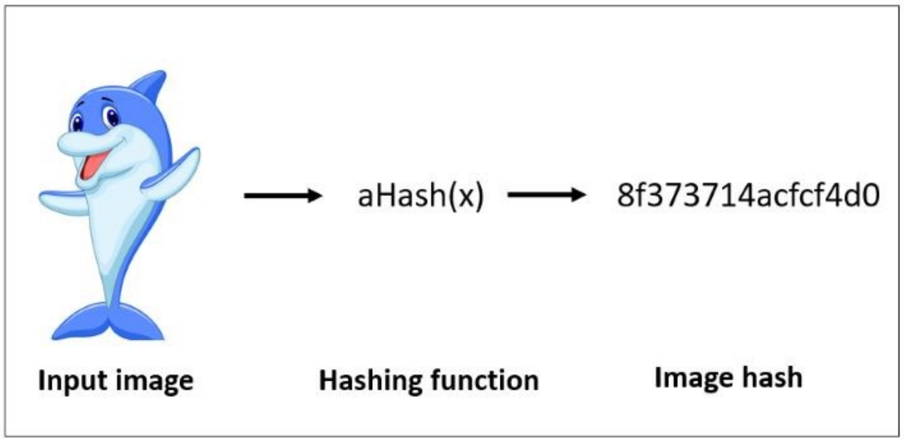


Figure 6: Flow diagram for Average Hash algorithm

Step 4: Compute the bits. Each bit is simply set based on whether the color value is above or below the mean.

Step 5: Construct the hash. Set the 64 bits into a 64-bit integer. The order does not matter, just as long as you are consistent.

2. **Perceptual Hashing (pHash):**

A perceptual hash is a fingerprint of a multimedia file derived from various features from its content. Unlike cryptographic hash functions which rely on the avalanche effect of small changes in input leading to drastic changes in the output, perceptual hashes are ”close” to one another if the features are similar. [17]This algorithm is similar to aHash but use a discrete cosine transform (DCT) and compares based on frequencies rather than color values.

Following are the steps to show how the Perceptual hash algorithm works-

Step 1: Reduce size. Like Average Hash, pHash starts with a small image. However, the image is larger than 8x8; 32x32 is a good size. This is really done to simplify the DCT computation and not because it is needed to reduce the high frequencies.

Step 2: Reduce color. The image is reduced to a grayscale just to further simplify the number of computations.

Step 3: Compute the DCT. The DCT separates the image into a collection of frequencies and scalars. While JPEG uses an 8x8 DCT, this algorithm uses a 32x32 DCT.

Step 4: Reduce the DCT. While the DCT is 32x32, just keep the top-left 8x8. Those represent the lowest frequencies in the picture.

Step 5: Compute the average value. Like the Average Hash, compute the mean DCT value (using only the 8x8 DCT low-frequency values and excluding the first term since the DC coefficient can be significantly different from the other values and will throw off the average).

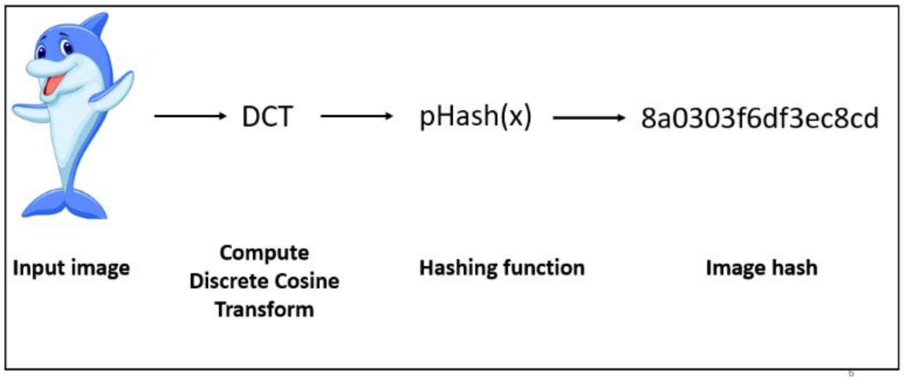


Figure 7: Flowdiagram for Perceptual Hashing algorithm

Step 6: Further reduce the DCT. This is the magic step. Set the 64 hash bits to 0 or 1 depending on whether each of the 64 DCT values is above or below the average value.The result doesn't tell us the actual low frequencies; it just tells us the very-rough relative scale of the frequencies to the mean. The result will not vary as long as the overall structure of the image remains the same.

Construct the hash. Set the 64 bits into a 64-bit integer. The order does not matter, just as long as you are consistent. To see what this fingerprint looks like, simply set the values (this uses +255 and -255 based on whether the bits are 1 or 0) and convert from the 32x32 DCT (with zeros for the high frequencies) back into the 32x32 image.

3. **Difference Hashing (dHash):**

Like aHash and pHash, dHash is pretty simple to implement and is far more accurate than it has any right to be. As an implementation, dHash is nearly identical to aHash but it performs much better. While aHash focuses on average values and pHash evaluates frequency patterns, dHash tracks gradients.

Following are the steps to show how the Average Hash algorithm works-

Step 1: Reduce size. The fastest way to remove high frequencies and detail is to shrink the image. In this case, shrink it to 9x8 so that there are 72 total pixels. (I'll get to the "why" for the odd 9x8 size in a moment.) By ignoring the size and aspect ratio, this hash will match any similar picture regardless of how it is stretched.

Step 2: Reduce color. Convert the image to a grayscale picture. This changes the hash from 72 pixels to a total of 72 colors. (For optimal performance, either reduce color before scaling or perform the scaling and color reduction at the same time.)

Step 3: Compute the difference. The dHash algorithm works on the difference between adjacent pixels. This identifies the relative gradient direction. In this case, the 9 pixels per row yields 8 differences between adjacent pixels. Eight rows of eight differences becomes 64 bits.

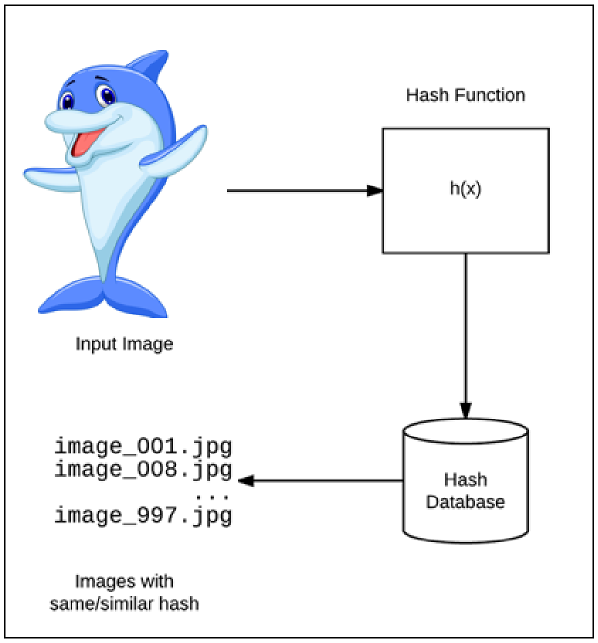


Figure 8: Flowdiagram of Difference Hash algorithm

Step 4: Assign bits. Each bit is simply set based on whether the left pixel is brighter than the right pixel. The order does not matter, just as long as you are consistent. (I use a ”1” to indicate that P[x] ¡ P[x+1] and set the bits from left to right, top to bottom using big-endian.)

2.12 Proposed algorithm

The proposed image hashing is composed of four steps. The input image is firstly preprocessed to produce a normalised image. Colour vector angle of each pixel is then calculated. Next, block division is performed, and block means are extracted to form a feature matrix. A single-level two-dimensional (2D) DWT is finally used to produce a short hash. Colour vector angle is introduced in Section 2.5.1 and the detailed steps are described in Section 2.5.1 and 2.5.2

2.12.1 Colour vector angle

In general, a colour image can be depicted by its hue, saturation and luminance, where the hue represents colour appearance, the saturation describes the amount of white contained in the colour, and the luminance also called intensity is an indicator of brightness. In practice, normal digital operations, such as brightness/contrast adjustment, only change intensity and keep the hue and saturation almost unchanged. Colour vector angle is a useful feature and has been successfully used in edge detection and image retrieval. It is insensitive to intensity variations, but sensitive to differences in hue and saturation. This property is good for image hashing. Moreover, comparing with the Euclidean distance in RGB colour space, colour vector angle is more effective in evaluating perceptual differences between two colours.

The advantage of colour vector angle is attributed to its sensitiveness to hue differences. Let $P_1 = [R_1 , G_1 , B_1]^T$ and $P_2 = [R_2 , G_2 , B_2]^T$ be vectors of two colours, where R_1 and R_2 , G_1 and G_2 , B_1 and B_2 , are their red, green and blue components, respectively. Thus, the angle θ between P_1 and P_2 can be calculated by,

$$\theta = \arcsin\left(1 - \frac{\left(P^{\mathrm{T}}{}_1 P_2\right)^2}{P^{\mathrm{T}}{}_1 P_1 P^{\mathrm{T}}{}_2 P_2}\right)^{\left(\frac{1}{2}\right)}$$

To reduce computational cost, we use the $\sin\theta$ for representation which is defined as follows,

$$\sin\theta = \left(1 - \frac{\left(P^{\mathrm{T}}{}_1 P_2\right)^2}{P^{\mathrm{T}}{}_1 P_1 P^{\mathrm{T}}{}_2 P_2}\right)^{\left(\frac{1}{2}\right)}$$

2.12.2 Scale-invariant feature transform(SIFT)

The scale-invariant feature transform (SIFT) is a feature detection algorithm in computer vision to detect and describe local features in images. Applications include object recognition, robotic mapping and navigation, image stitching, 3D modeling, gesture recognition, video tracking, individual identification of wildlife and match moving.

SIFT keypoints of objects are first extracted from a set of reference images and stored in a database. An object is recognized in a new image by individually comparing each feature from the new image to this database and finding candidate matching features based on Euclidean distance of their feature vectors. From the full set of matches, subsets of keypoints that agree on the object and its location, scale, and orientation in the new image are identified to filter out good matches. The determination of consistent clusters is performed rapidly by using an efficient hash table implementation of the generalised Hough transform. Each cluster of 3 or more features that agree on an object and its pose is then subject to further detailed model verification and subsequently outliers are discarded. Finally the probability that a particular set of features indicates the presence of an object is computed, given the accuracy of fit and number of probable false matches. Object matches that pass all these tests can be identified as correct with high confidence.

SIFT is quite an involved algorithm. Here's an outline of what happens in SIFT-

1. Constructing a scale space

This is the initial preparation. You create internal representations of the original image to ensure scale invariance. This is done by generating a "scale space".

2. LoG Approximation

The Laplacian of Gaussian is great for finding interesting points (or key points) in an image. But it's computationally expensive. So we cheat and approximate it using the representation created earlier.

3. Finding keypoints

With the super fast approximation, we now try to find key points. These are maxima and minima in the Difference of Gaussian image we calculate in step 2

4. Get rid of bad key points

Edges and low contrast regions are bad keypoints. Eliminating these makes the algorithm efficient and robust. A technique similar to the Harris Corner Detector is used here.

5. Assigning an orientation to the keypoints

An orientation is calculated for each key point. Any further calculations are done relative to this orientation. This effectively cancels out the effect of orientation, making it rotation invariant.

6. Generate SIFT features

Finally, with scale and rotation invariance in place, one more representation is generated. This helps uniquely identify features.

The detailed steps of the wHash algorithm are as follows:

2.12.3 Preprocessing:

The input image is converted to $S \times S$ by bi-cubic interpolation. The image resizing is to make our hash resistant to those images with different resolutions. Gaussian low-pass filtering is then applied to the resized image. This operation can be achieved by a convolution mask.

Let $\mathrm{T}_{\mathrm{Gaussian}}\left(i,j\right)$ be the element in the ith row and the jth column of the convolution mask. Thus, it can be obtained by,

$$\mathrm{T}_{\mathrm{Gaussian}}\left(i,j\right) = \frac{T\left(^{1}\right)(i,j)}{\sum_{\mathrm{i}}} \sum{}_{\mathrm{j}} T^{1}\left(i,j\right)$$

in which σ is the standard deviation of all elements in the convolution mask. For example, $1 \leq \mathrm{i} \leq 1$ and $1 \leq \mathrm{j} \leq 1$ if the mask is 3×3. The filtering manipulation is to alleviate influences of minor modifications on the hash, such as noise contamination.

2.12.4 Colour vector angle calculation:

As angle calculation needs two colours, we generate a reference colour $\mathrm{P}_{\mathrm{ref}} =$

$$R_{\mathrm{ref}}, G_{\mathrm{ref}}, B_{\mathrm{ref}}$$

T , where $\mathrm{R}_{\mathrm{ref}}$, $\mathrm{G}_{\mathrm{ref}}$ and $\mathrm{B}_{\mathrm{ref}}$ are the means of red, green and blue components of all pixels. Thus, for each pixel, we calculate its colour vector angle between

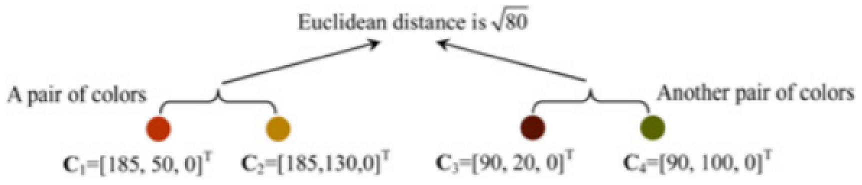


Figure 9: Two colour pairs having perceptual difference with the same Euclidean distance

its RGB vector and P_{ref}. After computation, we obtain a matrix A of colour vector angles. Fig. is an example of conversion from the normalised image to colour vector angles.

2.12.5 Feature extraction

The L^* component of the Ilab color image is used to extract robust features. The step-by-step feature extraction process is explained as follows.

Step 1. Computation of SIFT feature points: SIFT is a computer vision technique used to detect and describe invariant features points on digital images.

The proposed method computes SIFT feature points from the L^* component. These points are denoted as FPi , $1 \geq i \geq t$ where t signifies the total number of feature points. The i^{th} SIFT feature point is represented as: F $\mathrm{P}^{\mathrm{i}}(x^{\mathrm{i}}, y^{\mathrm{i}}, \sigma, \theta)$ where the coordinates $(\mathrm{x}^{\mathrm{i}}, \mathrm{y}^{\mathrm{i}})$ denote the location of the feature point on the L^* component, and σ and θ signify the scale and orientation,

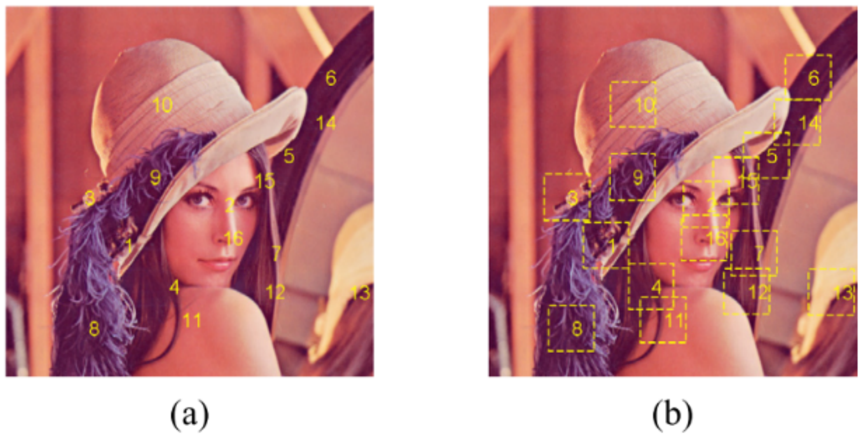


Figure 10: SIFT feature points: (a) n distinct feature points selected on Lena and (b) Extracrtion of overlapped blocks using n feature points

respectively.

Step 2. Selection of distinct SIFT feature points: To select distinct feature points from the list of t points, the points are sorted in descending order based on the scale, and then duplicate points are removed. The first n points are then selected as feature points. The selected distinct points are denoted as $\mathrm{DFP}^{\mathrm{j}}$, $1 \geq j \geq n$. The n = 16 distinct points selected on the Lena image are shown in Fig. 6(a). The feature points are labeled with numbers.

2.12.6 DWT:

To make a short hash, we apply a single-level 2D DWT to M. A single-level 2D DWT decomposes an input matrix, for example, an image, into four sub-bands, that is, LL, LH, HL and HH sub-bands. DWT coefficients in the LL sub-band contain most information of the input matrix and depict the coarse characteristic, whereas those in other sub-bands preserve the edge information in different directions. This implies that DWT coefficients in the LL sub-band can be used to approximately represent the input matrix. Therefore we extract those DWT coefficients in the LL sub-band and randomly permute them to form a compact representation. Clearly, the use of DWT can reduce nearly 75% matrix elements and then make our hash short enough. Let L be the total number of DWT coefficients in the LL sub-band, where L = $(\lceil N \backslash 2 \rceil)^2$ and $\lceil . \rceil$ denotes upward rounding. Concatenate the columns of LL sub-band and obtain a compact representation r = [r(1), r(2),. . ., r(L)]. The random permutation is controlled by a secret key, which is used as the seed of a random generator. It is achieved as follows. Generate L pseudo-random numbers by the secret key, sort these L numbers to make an ordered sequence, and record the original positions of these ordered numbers in an array E. Suppose that c = [c(1), c(2), . . . , c(L)] is the permuted result of r. Thus, it can be obtained by ,

$$c\ (l) = r(E[l])$$

where E[l] is the lth element of E and l = 1, 2,. . ., L. In general, the bigger the available number of c, the more unpredictable our hash. Obviously, the available number of c is equivalent to the number of permutations, that is, L! For example, when L is 16 or 64, the available number of c is about 2.09

1013 or 1.27 1089, respectively. Next we quantise c as follows,

$$h(l) = [c(l) \times 10000 + 0.5]$$

where l = 1, 2, ..., L and [.] is the rounding operation. Therefore our hash is an integer representation labelled as h = [h(1), h(2),..., h(L)]. In experiments, we find that each element can be represented by 15 bits. Thus, the length of our hash is 15L bits. This will be validated in Section 3.2. Note that hash length is related to block size. The smaller the block size, the longer the hash length. For example, if the normalised image size is 512 512 and the block size is 32 32, that is, S = 512 and b = 32, then N = 512/32 = 16, L =$(\lceil 16 \backslash 2 \rceil)^2$ = 64 and therefore the hash length is 960 bits.

2.12.7 Generation of Intermediate hash

The intermediate hash (IH) vector is generated by computing the row-wise average of approximation coefficients of matrix $AC_{Q \times R}$ as described in the following equation. To produce a secure and key-dependent hash, the intermediate hash vector is randomly permuted using a pseudorandom procedure.

$$I\,H_k = sum_{m=1}^{R} AC(k,m)/R, 1 \geq k \geq Q$$

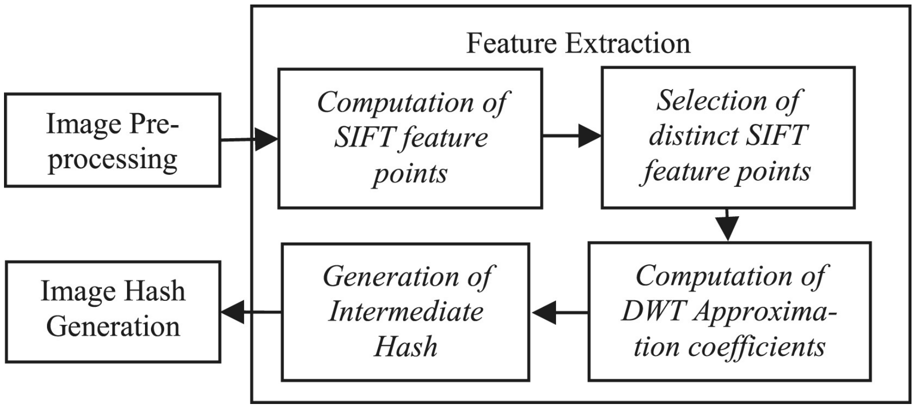

Feature Extraction
Image Pre-processing
Computation of SIFT feature points
Selection of distinct SIFT feature points
Computation of DWT Approximation coefficients
Generation of Intermediate Hash
Image Hash Generation

Figure 11: Framework of wHash algorithm

2.12.8 Hamming distance

the hash similarity is measured using the normalized hamming distance. Let h_1 and h_2 be two hashes. Thus, NHD is defined as,

$$\mathbf{NHD(h_1,\, h_2)} = \frac{1}{L}\sum_{z=1}^{L} |h_{\mathbf{1}}(z) - h_{\mathbf{2}}(z)|$$

The more similar the images of the input hashes, the smaller the d value. If d is smaller than a threshold T, the two images are considered as visually identical images. Otherwise, they are different images.

3 Algorithm:

Step 1: The selected RGB color images were scaled to 512 × 512 ($N \times N$) pixels using bi-cubic interpolation and converted to L*a*b* color images. [15]

The L*a*b* space consists of a luminosity 'L*' layer, chromaticity layer 'a*' indicating where color falls along the red-green axis, and chromaticity layer 'b*' indicating where the color falls along the blue-yellow axis. The unique goal of the L*a*b* model is to be device-independent. The colors should not be dependent on the device they are displayed on. To calculate L*, a*, b*,

$$L^* = 116\, f\,(Y / Y_n)\ \ 16$$

$$a^* = 500[f(X / X_n)\text{-}f(Y / Y_n)]$$

$$b^* = 200\, [f\,(Y / Y_n)\text{-}f(Z / Z_n)]$$

X_n, Y_n, and Z_n are the tristimulus values of the reference white object, $R(\lambda) = 1$. The tristimulus values X_n, Y_n, and Z_n represent the white adaptation point (i.e. the chromaticity of the illuminant). A positive a^* value corresponds to a reddish color and negative a_* indicates a greenish color. Similarly, a positive b^* value indicates yellowish color and a negative b^*

value signifies bluishness.

Step 2: The n = 16 SIFT feature points were chosen to extract image content from the L* component. The block size 64×64 ($P \times P$) was considered to extract content from around the chosen SIFT feature points.

Scale Invariant Feature Transform(SIFT)

The most basic of feature detectors focuses on finding basic features in the images, this can be in the form of corners (harris detector) or edges (canny detector), these are often features that are affected by scale transform.

Step 3: The approximation coefficients were computed by applying level 2 (l = 2) 2D Daubechie wavelet transform on each extracted block of pixels.

Step 4: Finally, the binary hash of length 256 ($n \times (P/2^l)$) was generated.

Similarity between images:

The similarity between original and suspect images was measured using the normalized hamming distance (NHD) as defined in the following equation.

$\mathbf{NHD(h_1,\, h_2)} = \frac{1}{L}\sum_{z=1}^{L} |h_{\mathbf{1}}(z) - h_{\mathbf{2}}(z)|$

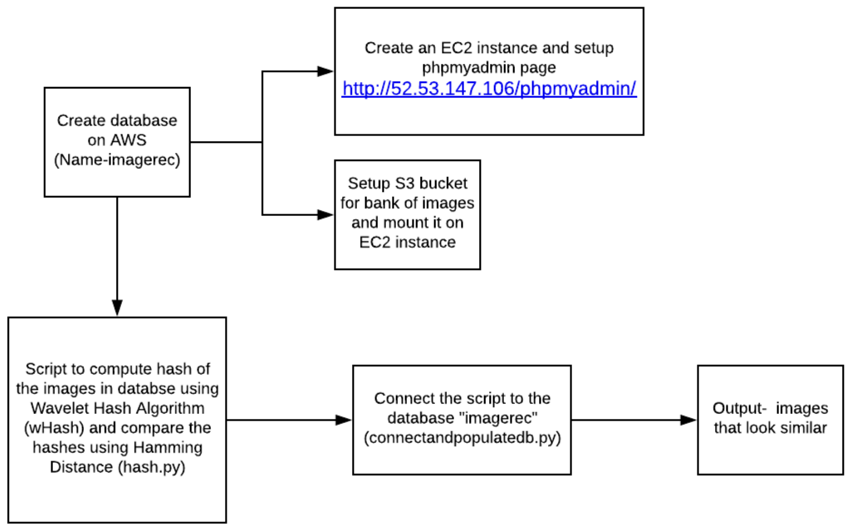

Create database on AWS (Name-imagerec)
Create an EC2 instance and setup phpmyadmin page http://52.53.147.106/phpmyadmin/
Setup S3 bucket for bank of images and mount it on EC2 instance
Script to compute hash of the images in databse using Wavelet Hash Algorithm (wHash) and compare the hashes using Hamming Distance (hash.py)
Connect the script to the database "imagerec" (connectandpopulatedb.py)
Output- images that look similar

Figure 12: Flow-diagram for Approach 1

4 Implementation

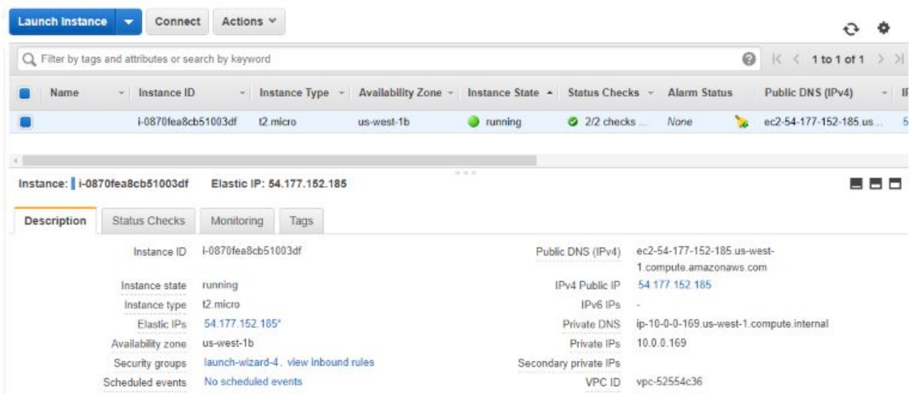
The algorithm was implemented using two different approaches figure 12 and 17

4.1 Working with Amazon web services- EC2 instance, S3 bucket and DynamoDB

4.1.1 Setting up EC2 instance

We created an EC2 instance and connected it to SSH using Private key.

SSH command: ssh -i ”newinstance.pem” ec2-user@ec2-54-215-234-30.us-west-1.compute.amazonaws.com

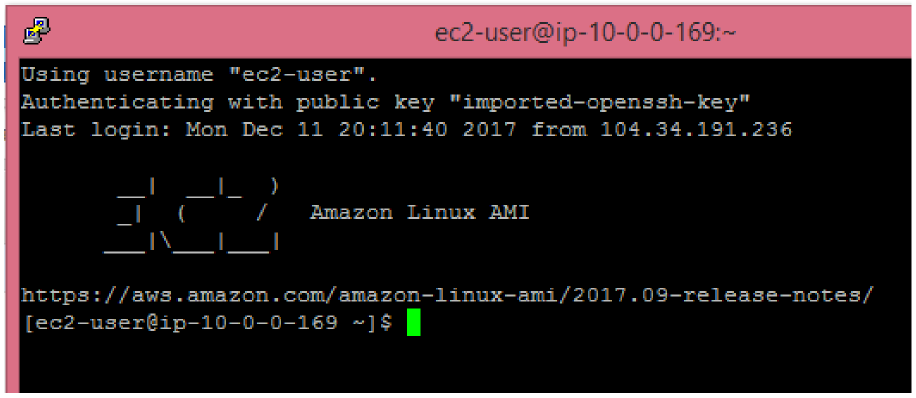


Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status	Public DNS (IPv4)
	i-0870fea8cb51003df	t2.micro	us-west-1b	running	2/2 checks ...	None	ec2-54-177-152-185.us...

Instance: i-0870fea8cb51003df Elastic IP: 54.177.152.185

Instance ID	i-0870fea8cb51003df	Public DNS (IPv4)	ec2-54-177-152-185.us-west-1.compute.amazonaws.com
Instance state	running	IPv4 Public IP	54.177.152.185
Instance type	t2.micro	IPv6 IPs	-
Elastic IPs	54.177.152.185*	Private DNS	ip-10-0-0-169.us-west-1.compute.internal
Availability zone	us-west-1b	Private IPs	10.0.0.169
Security groups	launch-wizard-4 . view inbound rules	Secondary private IPs	
Scheduled events	No scheduled events	VPC ID	vpc-52554c36

Figure 13: EC2 instance (AWS)



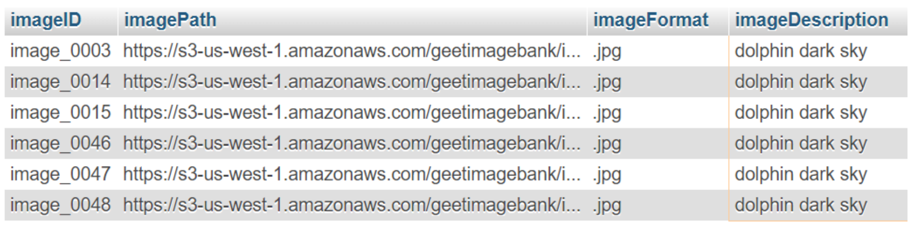
```
Using username "ec2-user".
Authenticating with public key "imported-openssh-key"
Last login: Mon Dec 11 20:11:40 2017 from 104.34.191.236

       __|  __|_  )
       _|  (     /   Amazon Linux AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-ami/2017.09-release-notes/
[ec2-user@ip-10-0-0-169 ~]$
```

Figure 14: SSH using Private key

The figure below depicts an EC2 instance (Fig. 8) and succesful SSH using Private key(Fig.9)

4.1.2 Creating phpmyadmin page to access the database

For this, firstly MySQL server and php needs to be installed to directly access the Database.



imageID	imagePath	imageFormat	imageDescription
image_0003	https://s3-us-west-1.amazonaws.com/geetimagebank/i...	.jpg	dolphin dark sky
image_0014	https://s3-us-west-1.amazonaws.com/geetimagebank/i...	.jpg	dolphin dark sky
image_0015	https://s3-us-west-1.amazonaws.com/geetimagebank/i...	.jpg	dolphin dark sky
image_0046	https://s3-us-west-1.amazonaws.com/geetimagebank/i...	.jpg	dolphin dark sky
image_0047	https://s3-us-west-1.amazonaws.com/geetimagebank/i...	.jpg	dolphin dark sky
image_0048	https://s3-us-west-1.amazonaws.com/geetimagebank/i...	.jpg	dolphin dark sky

Figure 15: Databse 'imagerec'

Below mentioned is the link for my phpMyAdmin page - imageRec DB (Fig. 10)

Database - imagerec

The database created contains table ImageDetails which include attributes-ImageID, ImageName, ImagePath, ImageDescription and ImageFormat. Another table to store the HASH value named ImageHash with attributes- ImageId and ImageHash

4.1.3 Connect and Populate the database

This section discusses about script to populate the database. To populate the database on AWS, we first established the connection to the imagerec database we created on the phpmyadmin page.

Program Description:

1. The code take as input the IP address of the phpmyadmin page, username and password.

2. Then using pymysql library it runs a query to request for connection

3. If the connection is successful it returns the data/values in table imageDetails as the output (This query can be changed as per the need)

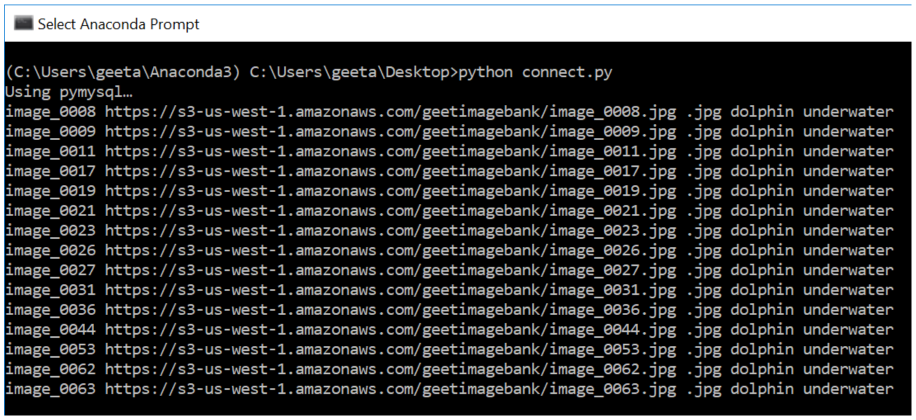
Below is the program for the same.

Code Listing 1: Connect and Populate the database

```
#To compile this script through command prompt- python
    connect_populate_db.py


import xlrd

import pymysql


# Open the workbook and define the worksheet


book = xlrd.open_workbook("image.xlsx") #open the excel file which
    contains 4 rows- imageID, imagePath, imageFormat,
    imageDescription

sheet = book.sheet_by_name("image")  #name of the sheet


# Establish a MySQL connection

database = pymysql.connect (host="52.53.147.106", user = "geet",
    passwd = "image", db = "imagerec")
```

```
# Get the cursor, which is used to traverse the database, line by
    line
cursor = database.cursor()

# Create the INSERT INTO sql query
query = """INSERT INTO imageDB
    (imageID,imagePath,imageFormat,imageDescription) VALUES (%s,
    %s, %s, %s)"""
#query = """SELECT * FROM imageDB"""


# Create a For loop to iterate through each row in the XLS file,
    starting at row 2 to skip the headers
for r in range(0, sheet.nrows):
      imageID    = sheet.cell(r,0).value
      imagePath  = sheet.cell(r,1).value
      imageFormat = sheet.cell(r,2).value
      imageDescription = sheet.cell(r,3).value



      # Assign values from each row
      values = (imageID, imagePath, imageFormat, imageDescription)
```

```
# Get the number of rows in the resultset

numrows = cursor.rowcount


# Get and display one row at a time

for x in range(0, numrows):

   row = cursor.fetchone()

   print (row[0], "-->", row[1])


      # Execute sql Query

   cursor.execute(query, values)


# Close the cursor

cursor.close()


# Commit the transaction

database.commit()



# Close the database connection

database.close()


# Print results

print ("")

print ("All Done! Bye, for now.")
```



```
(C:\Users\geeta\Anaconda3) C:\Users\geeta\Desktop>python connect.py
Using pymysql…
image_0008 https://s3-us-west-1.amazonaws.com/geetimagebank/image_0008.jpg .jpg dolphin underwater
image_0009 https://s3-us-west-1.amazonaws.com/geetimagebank/image_0009.jpg .jpg dolphin underwater
image_0011 https://s3-us-west-1.amazonaws.com/geetimagebank/image_0011.jpg .jpg dolphin underwater
image_0017 https://s3-us-west-1.amazonaws.com/geetimagebank/image_0017.jpg .jpg dolphin underwater
image_0019 https://s3-us-west-1.amazonaws.com/geetimagebank/image_0019.jpg .jpg dolphin underwater
image_0021 https://s3-us-west-1.amazonaws.com/geetimagebank/image_0021.jpg .jpg dolphin underwater
image_0023 https://s3-us-west-1.amazonaws.com/geetimagebank/image_0023.jpg .jpg dolphin underwater
image_0026 https://s3-us-west-1.amazonaws.com/geetimagebank/image_0026.jpg .jpg dolphin underwater
image_0027 https://s3-us-west-1.amazonaws.com/geetimagebank/image_0027.jpg .jpg dolphin underwater
image_0031 https://s3-us-west-1.amazonaws.com/geetimagebank/image_0031.jpg .jpg dolphin underwater
image_0036 https://s3-us-west-1.amazonaws.com/geetimagebank/image_0036.jpg .jpg dolphin underwater
image_0044 https://s3-us-west-1.amazonaws.com/geetimagebank/image_0044.jpg .jpg dolphin underwater
image_0053 https://s3-us-west-1.amazonaws.com/geetimagebank/image_0053.jpg .jpg dolphin underwater
image_0062 https://s3-us-west-1.amazonaws.com/geetimagebank/image_0062.jpg .jpg dolphin underwater
image_0063 https://s3-us-west-1.amazonaws.com/geetimagebank/image_0063.jpg .jpg dolphin underwater
```

Figure 16: Output for connect and populate database

```
print ("")

columns = str(sheet.ncols)

rows = str(sheet.nrows)

print ("I just imported " ,columns, " columns and ", rows, " rows
    to MySQL!")
```

Fig. 11 depicts the output for the above Python script-

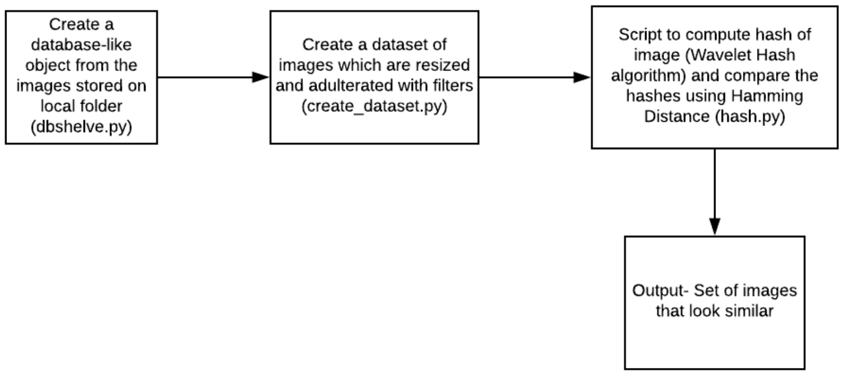

Create a database-like object from the images stored on local folder (dbshelve.py)
Create a dataset of images which are resized and adulterated with filters (create_dataset.py)
Script to compute hash of image (Wavelet Hash algorithm) and compare the hashes using Hamming Distance (hash.py)
Output- Set of images that look similar

Figure 17: Flow-diagram for Approach 2

4.1.4 Database shelve- alternative approach for reading images as input

We created a database shelve to test the code.

Below is the Python script that allows the creation of a database on localhost

Program Description:

1. ‘dbshelve.py‘: code for creating a ‘shelve‘ object database from the contents of a local folder

2. ‘–dataset dataset‘: this indicates the local folder where the images are stored

3. ‘–shelve db.shelve‘: is the output dataset using this Python shelve library

Code Listing 2: Creating a Database Shelve

```
#db.shelve
#To run this code use following in command prompt:
# python dbshelve.py --dataset dataset --shelve db.shelve

from PIL import Image
import imagehash
import argparse
import shelve
import glob

# construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-d", "--dataset", required = True ,
help = "path to input dataset of images")
ap.add_argument("-s", "--shelve", required = True ,
help = "output shelve database")
args = vars(ap.parse_args())

# open the shelve database
db = shelve.open(args["shelve"], writeback = True )
# loop over the image dataset
for imagePath in glob.glob(args["dataset"] + "/*.jpg"):
#compute the difference hash
   image = Image.open(imagePath)
```

```
  h = str(imagehash.whash(image))
  print(h)

 #Save the hash as the key and filename
  filename = imagePath[imagePath.rfind("/") + 1:]
  db[h] = db.get(h, []) + [filename]
  print(db[h])
 # close the shelf database
db.close()
```

4.2 Python script to create dataset

Program Description:

1. ‘create_dataset.py‘: code for creating an output folder containing images that are adulterated with filters or resized or cropped

2. ‘–input faces1‘: this indicates the local folder where the original images are stored

3. ‘–output dataset‘: creates output folder ‘dataset‘ which contains images in adulterated form

Code Listing 3: Creating a dataset with images adulterated with modications

```
# This code be run with command:

# python create_database.py --input faces1 --output dataset


from PIL import Image

import os, os.path

import argparse

import random

import shutil

import glob2

import uuid

import shutil

from shutil import copyfile


# construct the argument parse and parse the arguments

ap = argparse.ArgumentParser()

ap.add_argument("-i", "--input", required = True ,

help = "input directory of images")

ap.add_argument("-o", "--output", required = True ,

help = "output directory")


args = vars(ap.parse_args())


#finding image in the given folder
```

```
for imagePath in glob2.iglob(args["input"] + "/**/*.jpg"):
   filename = str(uuid.uuid4()) + ".jpg"
   shutil.copy(imagePath ,os.path.abspath(args["output"] + "/" +
       filename))
   numTimes = random.randint(1, 8)
   for i in range(0, numTimes):
      image = Image.open(imagePath)
      #changing the size of the image randomly
      factor = random.uniform(0.90, 1.05)
      width = int(image.size[0] * factor)
      ratio = width / float(image.size[0])
      height = int(image.size[1] * ratio)
      image = image.resize((width , height),1)

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

4.3 Python script for aHash, pHash, dHash and wHash algorithms

'hash.py'- Script to calculate hash of images already stored in the dataset and retrieve similar images

Program Description:

1. '–dataset dataset': this will let script to access dataset to calculate hash of all the images present in the dataset
2. '–query brain\ 87·jpg': this is the query image for which hashes will be compared to retrieve all the similar images.

Below is the Python script for Wavelet hash($wHash$), Difference hash($dHash$), Average hash ($aHash$) and Perceptual hash ($pHash$) algorithm.

Code Listing 4: Computing hash of images

```
#IMAGE HASHING ALGORITHMS- AVERAGE HASH, PERCEPTUAL HASH,
   DIFFRENCE HASH, WAVELET HASH


#To compile this script through command prompt
#python hash.py --dataset dataset --query brain/87.jpg


from __future__ import (absolute_import, division, print_function)
from PIL import Image
import os.path
import imagehash
import argparse
import glob
import numpy as np
import matplotlib.pyplot as plt
import uuid


# construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-d", "--dataset", required = True,
    help = "path to input dataset of images")
ap.add_argument("-q", "--query", required = True,
      help = "path to the query image")
args = vars(ap.parse_args())

def _binary_array_to_hex(arr):
   bit_string = ''.join(str(b) for b in 1 * arr.flatten())
   width = int(np.ceil(len(bit_string)/4))
   return '{:0>{width}x}'.format(int(bit_string, 2), width=width)

def hex_to_hash(hexstr):
   hash_size = int(np.sqrt(len(hexstr)*4))
   binary_array = '{:0>{width}b}'.format(int(hexstr, 16), width =
       hash_size * hash_size)
   bit_rows = [binary_array[i:i+hash_size] for i in range(0,
       len(binary_array), hash_size)]
   hash_array = np.array([[bool(int(d)) for d in row] for row in
      bit_rows])
   return ImageHash(hash_array)


def old_hex_to_hash(hexstr, hash_size=8):
   l = []
   count = hash_size * (hash_size 4)
   if len(hexstr) != count:
      emsg = 'Expected hex string size of {}.'
      raise ValueError(emsg.format(count))
   for i in range(count // 2):
      h = hexstr[i*2:i*2+2]
      v = int("0x" + h, 16)
   return ImageHash(np.array(l))


# string1 and string2 should be the same length.
def hamming_distance(string1, string2):
    # Start with a distance of zero, and count up
    distance = 0
    # Loop over the indices of the string
    L = len(string1)
    for i in range(L):
        # Add 1 to the distance if these two characters are not
            equal
        if string1[i] != string2[i]:
            distance += 1
    # Return the final count of differences
    return distance

def average_hash(image, hash_size=8):
   if hash_size < 2:
      raise ValueError("Hash size must be greater than or equal to
          2")

   # reduce size and complexity, then covert to grayscale
   image = image.convert("L").resize((hash_size, hash_size),
       Image.ANTIALIAS)

   # find average pixel value; 'pixels' is an array of the pixel
       values, ranging from 0 (black) to 255 (white)
   pixels = numpy.asarray(image)
   avg = pixels.mean()

   # create string of bits
   diff = pixels > avg
   # make a hash
   return str(ImageHash(diff))

def phash(image, hash_size=8, highfreq_factor=4):
   if hash_size < 2:
      raise ValueError("Hash size must be greater than or equal to
          2")

   import scipy.fftpack
   img_size = hash_size * highfreq_factor
   image = image.convert("L").resize((img_size, img_size),
       Image.ANTIALIAS)
   pixels = numpy.asarray(image)
   dct = scipy.fftpack.dct(scipy.fftpack.dct(pixels, axis=0),
       axis=1)
   dctlowfreq = dct[:hash_size, :hash_size]
   med = numpy.median(dctlowfreq)
   diff = dctlowfreq > med
   return str(ImageHash(diff))

def dhash_vertical(image, hash_size=8):
   # resize(w, h), but np.array((h, w))
   image = image.convert("L").resize((hash_size, hash_size + 1),
       Image.ANTIALIAS)
   pixels = np.asarray(image)
   # compute differences between rows
   diff = pixels[1:, :] > pixels[:-1, :]
   return str(ImageHash(diff))
```

```
def whash(image, hash_size = 8, image_scale = None, mode = 'haar',
    remove_max_haar_ll = True)$\colon$
   import pywt
   dwt_level = ll_max_level - level

   image = image.convert("L").resize((image_scale, image_scale),
       Image.ANTIALIAS)
   pixels = np.asarray(image) / 255
   hex_string = []
   if remove_max_haar_ll:
      coeffs = pywt.wavedec2(pixels, 'haar', level = ll_max_level)
      coeffs = list(coeffs)
      coeffs[0] *= 0
      pixels = pywt.waverec2(coeffs, 'haar')

   coeffs = pywt.wavedec2(pixels, mode, level = dwt_level)
   #print(coeffs)
   dwt_low = coeffs[0]

   med = np.median(dwt_low)
   diff = dwt_low > med
   return str(ImageHash(diff))
```

```
# loop over the image dataset
hashes = []
for imagePath in glob.glob(args["dataset"] + "/*"):
    # load the image and compute the difference hash
    image = Image.open(imagePath)
    k = whash(image)
    hashes.append([k,imagePath])
    #print(hashes)

wbhash = [x[0] for x in hashes] # gives the hash for image
path = [x[1] for x in hashes] # gives the path+filename for the
    image
#print(wbhash)

#open input image and calculate difference hash
query = Image.open(args["query"])
ohash = whash(query)
print(ohash)

#calculate hamming distance for image
for hashes, path in zip(wbhash, path):
   ham = hamming_distance(ohash,hashes)
   #print(ham)
```

```
    # Hamming Distance is zero means duplicate image is detected
if (ham == 0):
   image = Image.open(path)
   image.show()
   print("hamming distance is ", ham)
   print(path)
     # hamming distance < 6 gives those images which are almost
         alike
elif (ham <6):
   num = 0
   image = Image.open(path)
   image.show()
   print("hamming distance is ", ham)
   print(path)
```

5 Experimental results

In the experiments, all images are resized to 512×512 and blurred by a 3×3 Gaussian low-pass mask with a unit standard deviation and the Haar wavelet transform is exploited to produce hashes. The hamming distance we set was <6 because for high accuracy of the results, 2 / 3 of the hash bits should be same which means approximately 45 bits out of 64 should be same and we got fairly good result.

The results are as follows-

5.1 Result obtained for wHash algorithm

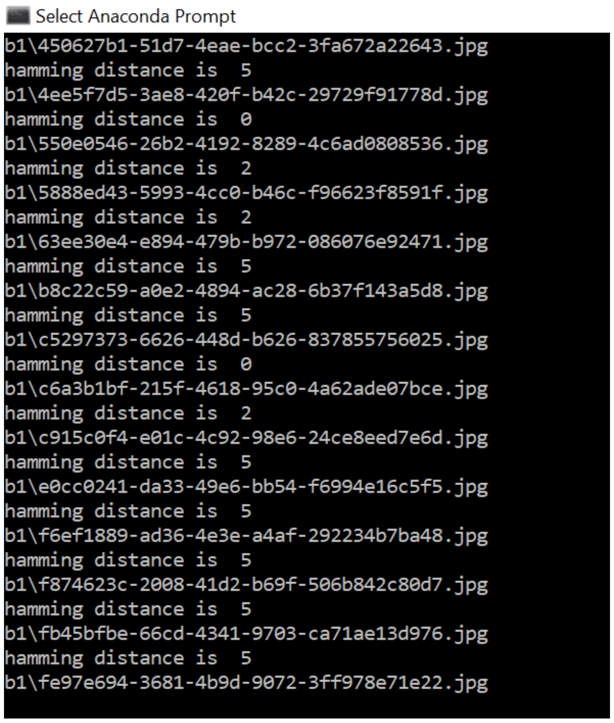
From the fig.12 we can say that images with different variations were obtained. Fig. 12 depicts the hamming distance of the retrieved images and fig. 13 depicts the variety of images obtained against query image.

5.2 Result obtained for dHash algorithm

From the fig.14, we can say that images that were exactly similar were retrieved because the Hamming distance is 0

5.3 Result obtained for pHash algorithm

From the fig.15, we can say that images that were exactly similar were retrieved because the Hamming distance is 0



```
b1\450627b1-51d7-4eae-bcc2-3fa672a22643.jpg
hamming distance is  5
b1\4ee5f7d5-3ae8-420f-b42c-29729f91778d.jpg
hamming distance is  0
b1\550e0546-26b2-4192-8289-4c6ad0808536.jpg
hamming distance is  2
b1\5888ed43-5993-4cc0-b46c-f96623f8591f.jpg
hamming distance is  2
b1\63ee30e4-e894-479b-b972-086076e92471.jpg
hamming distance is  5
b1\b8c22c59-a0e2-4894-ac28-6b37f143a5d8.jpg
hamming distance is  5
b1\c5297373-6626-448d-b626-837855756025.jpg
hamming distance is  0
b1\c6a3b1bf-215f-4618-95c0-4a62ade07bce.jpg
hamming distance is  2
b1\c915c0f4-e01c-4c92-98e6-24ce8eed7e6d.jpg
hamming distance is  5
b1\e0cc0241-da33-49e6-bb54-f6994e16c5f5.jpg
hamming distance is  5
b1\f6ef1889-ad36-4e3e-a4af-292234b7ba48.jpg
hamming distance is  5
b1\f874623c-2008-41d2-b69f-506b842c80d7.jpg
hamming distance is  5
b1\fb45bfbe-66cd-4341-9703-ca71ae13d976.jpg
hamming distance is  5
b1\fe97e694-3681-4b9d-9072-3ff978e71e22.jpg
```

Figure 18: Output of wHash algorithm- Hamming distance

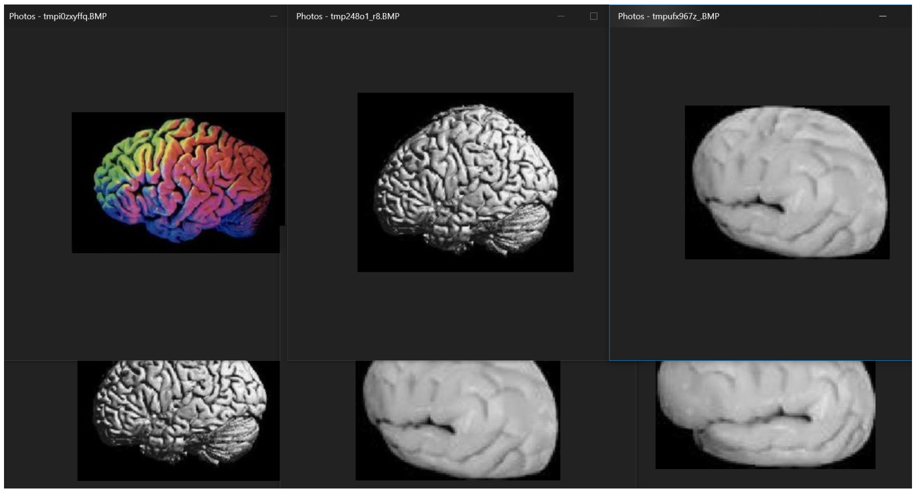
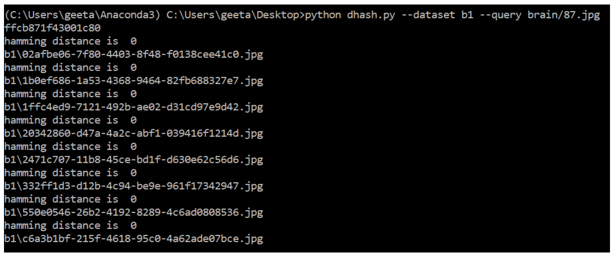
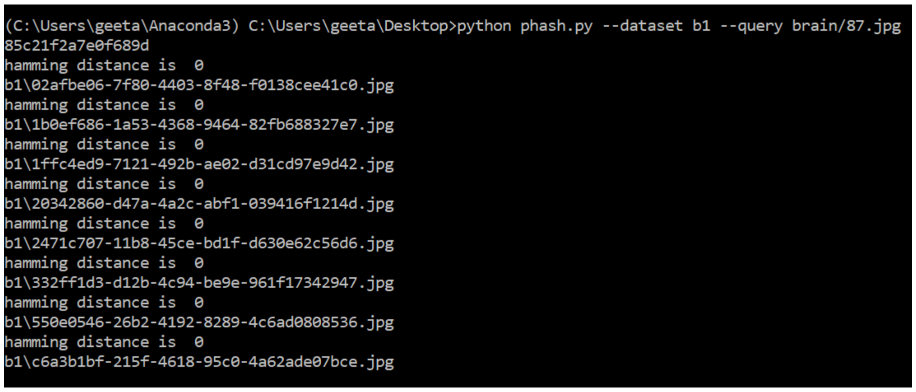


Figure 19: Output of wHash algorithm-Images Retrieved



```
(C:\Users\geeta\Anaconda3) C:\Users\geeta\Desktop>python dhash.py --dataset b1 --query brain/87.jpg
ffcb871f43001c80
hamming distance is  0
b1\02afbe06-7f80-4403-8f48-f0138cee41c0.jpg
hamming distance is  0
b1\1b0ef686-1a53-4368-9464-82fb688327e7.jpg
hamming distance is  0
b1\1ffc4ed9-7121-492b-ae02-d31cd97e9d42.jpg
hamming distance is  0
b1\20342860-d47a-4a2c-abf1-039416f1214d.jpg
hamming distance is  0
b1\2471c707-11b8-45ce-bd1f-d630e62c56d6.jpg
hamming distance is  0
b1\332ff1d3-d12b-4c94-be9e-961f17342947.jpg
hamming distance is  0
b1\550e0546-26b2-4192-8289-4c6ad0808536.jpg
hamming distance is  0
b1\c6a3b1bf-215f-4618-95c0-4a62ade07bce.jpg
```

Figure 20: Output of dHash algorithm-Hamming distance

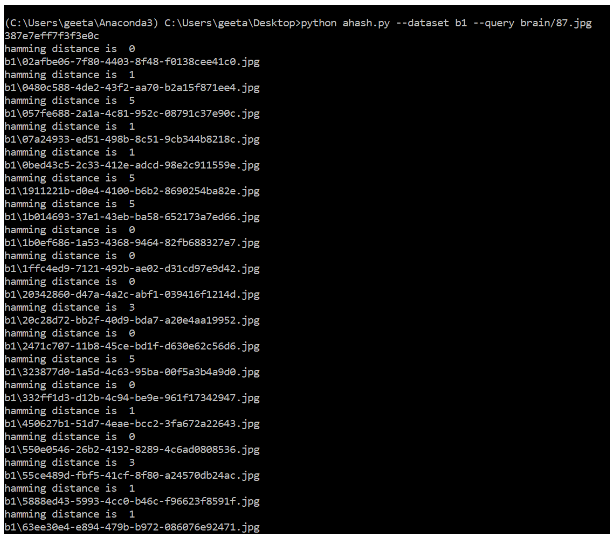


```
(C:\Users\geeta\Anaconda3) C:\Users\geeta\Desktop>python phash.py --dataset b1 --query brain/87.jpg
85c21f2a7e0f689d
hamming distance is  0
b1\02afbe06-7f80-4403-8f48-f0138cee41c0.jpg
hamming distance is  0
b1\1b0ef686-1a53-4368-9464-82fb688327e7.jpg
hamming distance is  0
b1\1ffc4ed9-7121-492b-ae02-d31cd97e9d42.jpg
hamming distance is  0
b1\20342860-d47a-4a2c-abf1-039416f1214d.jpg
hamming distance is  0
b1\2471c707-11b8-45ce-bd1f-d630e62c56d6.jpg
hamming distance is  0
b1\332ff1d3-d12b-4c94-be9e-961f17342947.jpg
hamming distance is  0
b1\550e0546-26b2-4192-8289-4c6ad0808536.jpg
hamming distance is  0
b1\c6a3b1bf-215f-4618-95c0-4a62ade07bce.jpg
```

Figure 21: Output of pHash algorithm-Hamming distance

5.4 Result obtained for aHash algorithm

From the fig.16 and 17, we can say that images that were exactly similar were retrieved but there was large difference in their hamming distance.



```
(C:\Users\geeta\Anaconda3) C:\Users\geeta\Desktop>python ahash.py --dataset b1 --query brain/87.jpg
387e7eff7f3f3e0c
hamming distance is  0
b1\02afbe06-7f80-4403-8f48-f0138cee41c0.jpg
hamming distance is  1
b1\0480c588-4de2-43f2-aa70-b2a15f871ee4.jpg
hamming distance is  5
b1\057fe688-2a1a-4c81-952c-08791c37e90c.jpg
hamming distance is  1
b1\07a24933-ed51-498b-8c51-9cb344b8218c.jpg
hamming distance is  1
b1\0bed43c5-2c33-412e-adcd-98e2c911559e.jpg
hamming distance is  5
b1\1911221b-d0e4-4100-b6b2-8690254ba82e.jpg
hamming distance is  5
b1\1b014693-37e1-43eb-ba58-652173a7ed66.jpg
hamming distance is  0
b1\1b0ef686-1a53-4368-9464-82fb688327e7.jpg
hamming distance is  0
b1\1ffc4ed9-7121-492b-ae02-d31cd97e9d42.jpg
hamming distance is  0
b1\20342860-d47a-4a2c-abf1-039416f1214d.jpg
hamming distance is  3
b1\20c28d72-bb2f-40d9-bda7-a20e4aa19952.jpg
hamming distance is  0
b1\2471c707-11b8-45ce-bd1f-d630e62c56d6.jpg
hamming distance is  5
b1\323877d0-1a5d-4c63-95ba-00f5a3b4a9d0.jpg
hamming distance is  0
b1\332ff1d3-d12b-4c94-be9e-961f17342947.jpg
hamming distance is  1
b1\450627b1-51d7-4eae-bcc2-3fa672a22643.jpg
hamming distance is  0
b1\550e0546-26b2-4192-8289-4c6ad0808536.jpg
hamming distance is  3
b1\55ce489d-fbf5-41cf-8f80-a24570db24ac.jpg
hamming distance is  1
b1\5888ed43-5993-4cc0-b46c-f96623f8591f.jpg
hamming distance is  1
b1\63ee30e4-e894-479b-b972-086076e92471.jpg
```

Figure 22: Output of aHash algorithm-Hamming distance

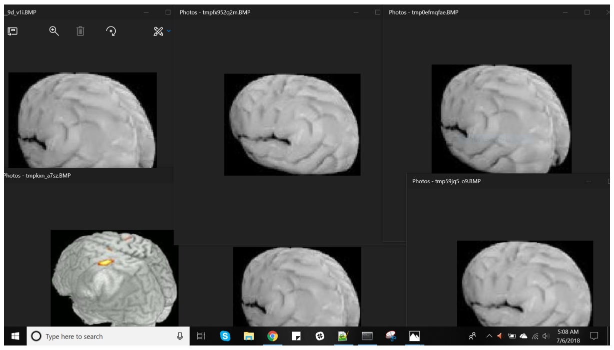


Figure 23: Output of aHash algorithm-Images retrieved

5.5 Comparison

The hash functions were tested against each other using the image database imagerec and dbshelve which contains 9000+ images. For every image created, 10 individual test images with slight, randomized modifications.

The modifications applied were-

1. increased and decreased brightness
2. increased and decreased contrast
3. added a watermark
4. converted the image to grayscale
5. scaled it down
6. cropped the borders
7. applied JPEG compression

This resulted in 10,000+ test images.
Comparing above results, we can see that wHash retrieved similar images with minimal bit difference, dHash retrieved only exactly similar images which also means there was large difference between the generated hash bits, pHash also retrieved exactly similar images and aHash retrieved exactly similar images but the bits generated for those images varied largely.

To evaluate the performance of the proposed method, it was comapred with existing methods. The performance comparisons are shown using a receiver operating characteristics (ROC) curve (Fig. 24) by using the true positive rate (TPR) on the Y axis and false positive rate (FPR) on the X axis. The TPR is defined as the ratio between the number of pairs of similar images considered as similar images (N_{SS}) and the total number of pairs of similar images (N_{TS}). Similarly, the FPR is defined as the ratio between the number of pairs of distinct images misclassified as similar images (N_{DS}) and the total number of pairs of distinct images (N_{TD}).

$TPR\ (T) = N_{SS}/\ N_{TS}$

$FPR(T) = N_{DS}\ /\ N_{TD}$

The robust hashing method must produce higher TPR values to show strong robustness and yields smaller FPR values to show good anti-collision capabilities. In this experiment, the TPR and FPR were computed for various thresholds . The ROC comparisons are presented in Fig. 24.

Fig. 24 depicts a line-graph for False positive rate versus True positive rate plotted for Average, Perceptual, Difference and Wavelet hashing algorithms.

From the above graph,(Fig. 24), it is clear that wHash gave better results as compared to previously implemented algorithms.

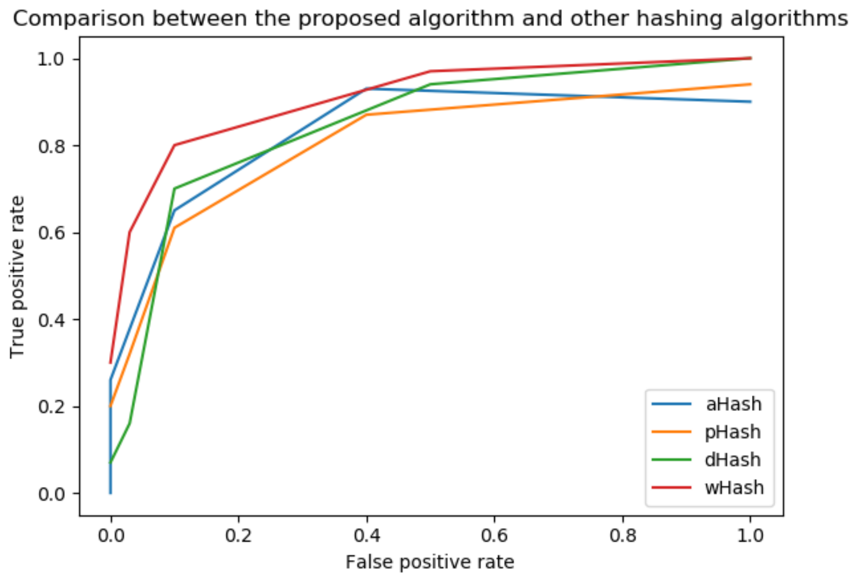


Figure 24: Comparison graph between proposed algorithm(wHash) and other hashing algorithms

6 Experimental Analysis

The proposed method was implemented using Python on a computer with an Intel Core i5-4200M CPU of 2.50 GHz, and RAM of 4.0 GB. The performance of the proposed method on robustness and anti-collision capabilities was tested using our database of RGB images that were selected randomly from various sources. The selected RGB color images were scaled to 512×512 $(N \times N)$ pixels using bi-cubic interpolation and converted to L*a*b* color images. The n = 16 SIFT feature points were chosen in order to extract image content from the L* component. The block size 64×64 $(P \times P)$ was considered to extract content from around the chosen SIFT feature points.

The approximation coefficients were computed by applying level 2 (l = 2) 2D Daubechie wavelet transform on each extracted block of pixels. Finally, the binary hash of length 256 ($n \times (P/2l)$) was generated. The similarity between original and suspect images was measured using the normalized hamming distance (NHD).

6.1 Key-dependent hash

The pseudorandom procedure was incorporated in the feature extraction to produce a secure hash. To show that the hash of the proposed method was key-dependent, the following three experiments were conducted: (1) generation of hash for 100 different images with the same key,(2) generation of hash for one image with different 100 keys, and (3) generation of hash for different 100 images with different 100 keys. The results show that the hash varied when the key was changed.

7 Conclusion and future work

In this study, a robust image hashing method was proposed using SIFT feature points and DWT approximation coefficients for image authentication. The performance of the proposed method was tested based on various image manipulations. Our experiments showed that the proposed method was robust to various content-preserving distortions such as compression, scaling, filtering, additive noise, brightness, and contrast adjustment. The proposed method was compared with existing methods using a line-graph. The comparison results indicate that the proposed method outperformed the other methods. The produced hash proved to be short in length as well as key-dependent.

The future work may include an algorithm for Video Hashing which may include automatic video clip identification in a video database or in broadcasting,online search in a streaming video, authentication of the video content or content-based watermarking.

References

[1] Abbas Ahmed, Siyal. A secure and robust hash-based scheme for image authentication.

[2] Xiao Chen, Wan. Robust audio hashing based on discrete-wavelet-transform and non-negative matrix factorisation.

[3] G. Lowe David. Distinctive image features from scale-invariant key points.

[4] et al. Han. Content-based image authentication: Current status, issues and challenges.

[5] Ben Hoyt. Duplicate image detection with perceptual hashing in python, 2017.

[6] F.Jiang Khelifi. Perceptual image hashing based on virtual watermark detection, 2010.

[7] Chang Lin. A robust image authentication system distinguishing jpeg compression from malicious manipulation.

[8] Leung Liu, Cheng. Wave atom transform generated strong image hashing scheme.

[9] Vasumathi Devarac Lokanadham Naidu Vadlamudia, Rama Prasad V. Vaddellab. Robust image hashing using sift feature points and dwt approximation coefficients.

[10] W. Lu. Multimedia forensic hash based on visual words, 2010.

[11] Mihcak Monga. Robust and secure image hashing via non-negative matrix factorizations.

[12] et al. Ouyang. Robust hashing for image authentication using sift feature and quaternion zernike moments.

[13] et al. Patel. Color image segmentation for medical images using l*a*b* color space.

[14] Chang Schneider. A robust content based digital signature for image authentication.

[15] et al. Tang. Robust image hashing via colour vector angles and discrete wavelet transform.

[16] Huang Tang, Z. Robust image hashing based on multiple histograms.

[17] Zhang Tang, Wang. Robust image hashing for tamper detection using non-negative matrix factorization.

[18] et al. Zhao. Robust hashing for image authentication using zernike moments and local features.

[19] Wei Zhao, Y. perceptual image hash for tampering detection using zernike moments, 2010.

[20] Yumin Dai Zhenjun Tang. Robust image hashing via colour vector angles and discrete wavelet transform, 2013.