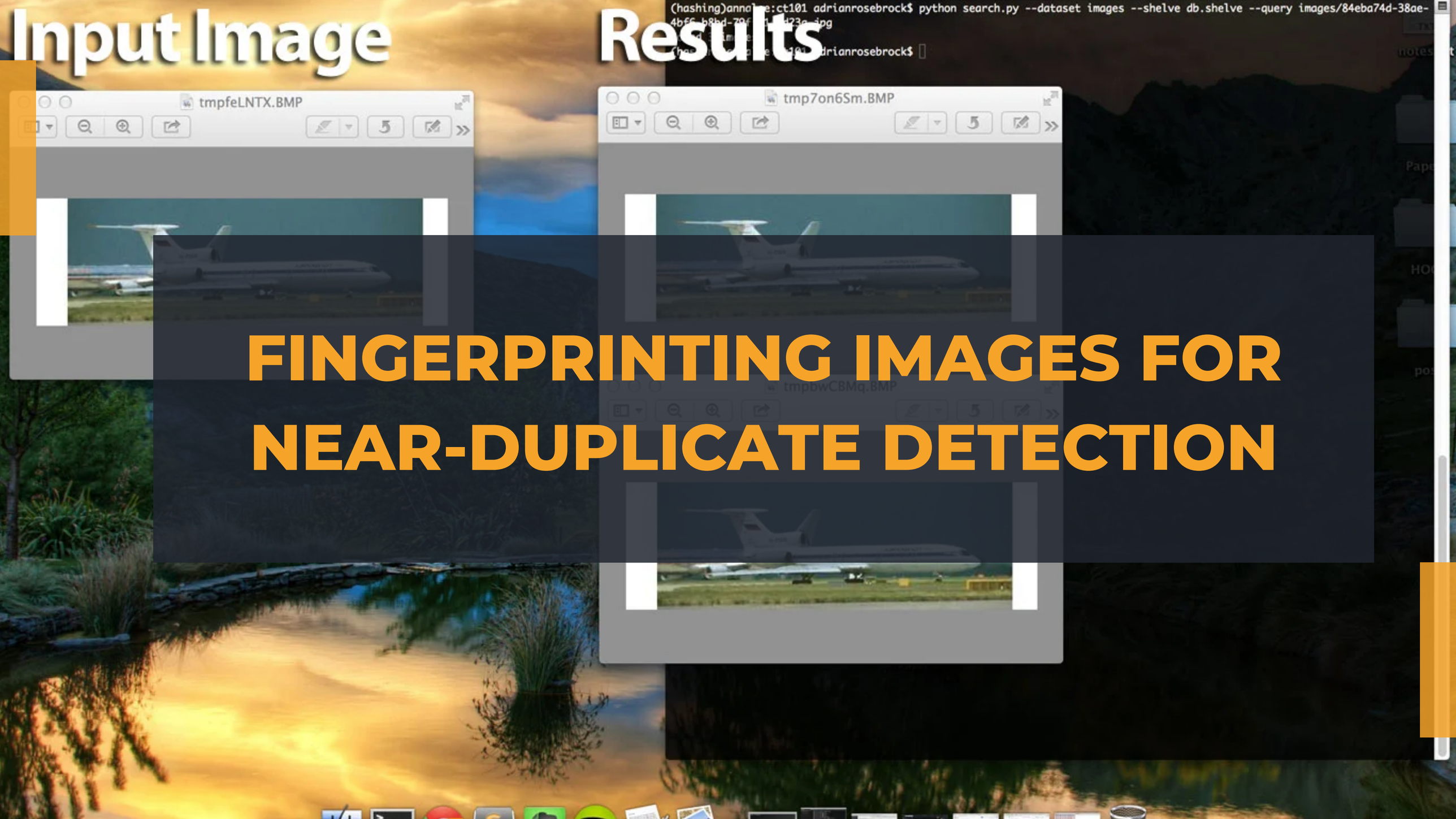

Input Image
Results
(hashing)annalee:ct101 adrianrosebrock$ python search.py --dataset images --shelve db.shelve --query images/84eba74d-38ae-
adrianrosebrock$
tmpfeLNTX.BMP
tmp7on6Sm.BMP
FINGERPRINTING IMAGES FOR
NEAR-DUPLICATE DETECTION

SUMMARY

(a)

01

02

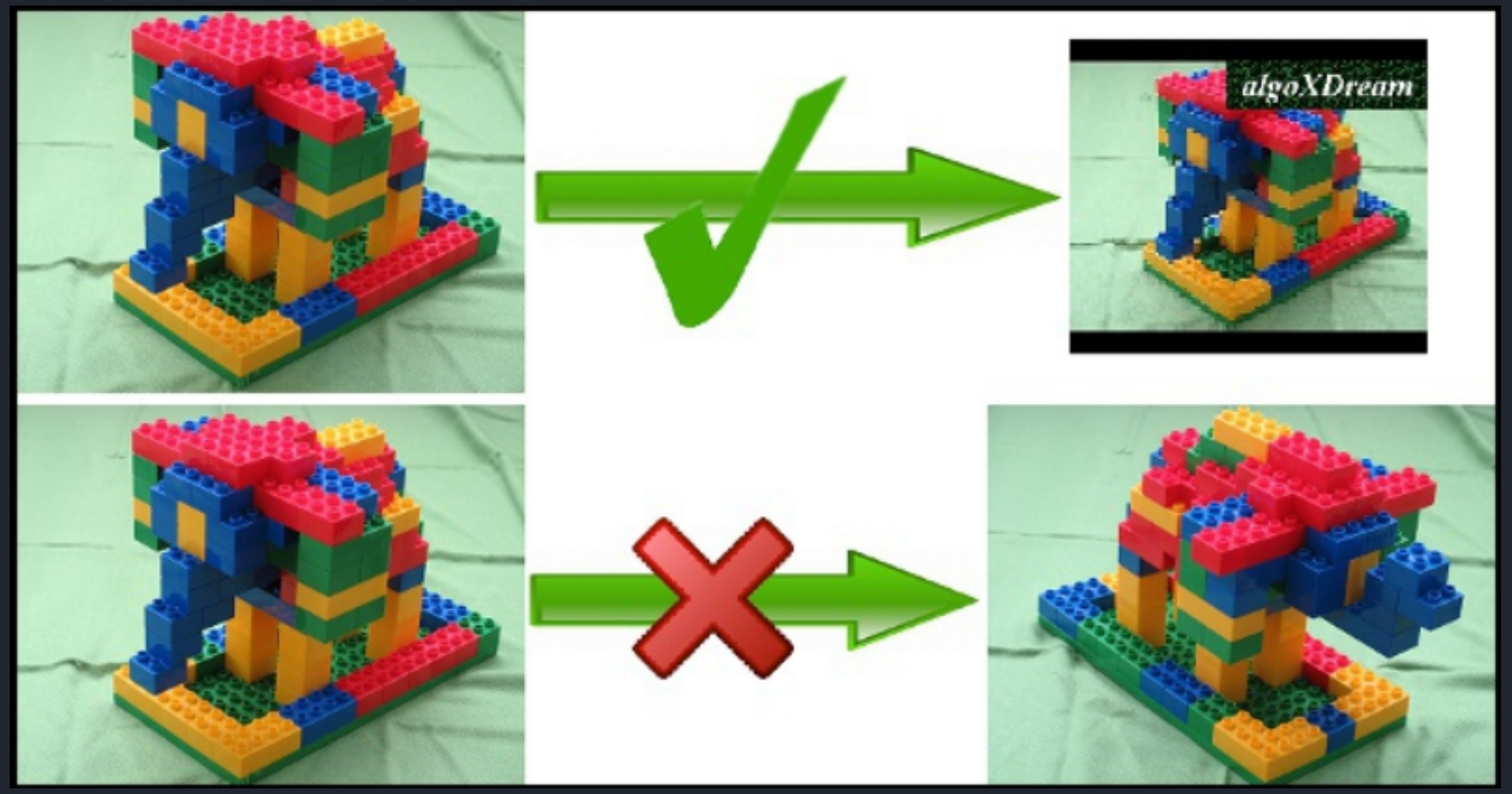
03

The issue of near duplicate images

The fingerprinting method

Our result and demonstration

1-THE ISSUE OF NEAR DUPLICATE IMAGES



CONCRETE EXAMPLE

REAL PHOTOGRAPH :
JEFF WIDENER

THIEVE PHOTOGRAPH :
LOÏS BRUN

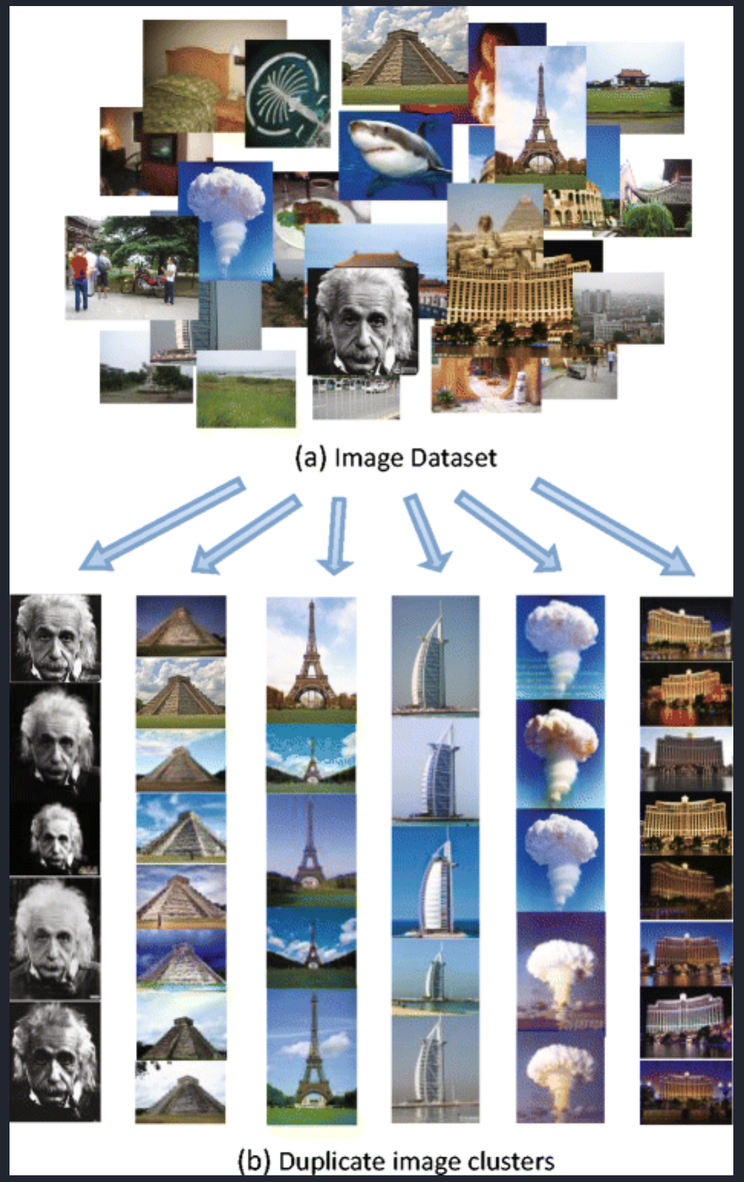
DEFINITIONS

NEAR-DUPLICATE IMAGES

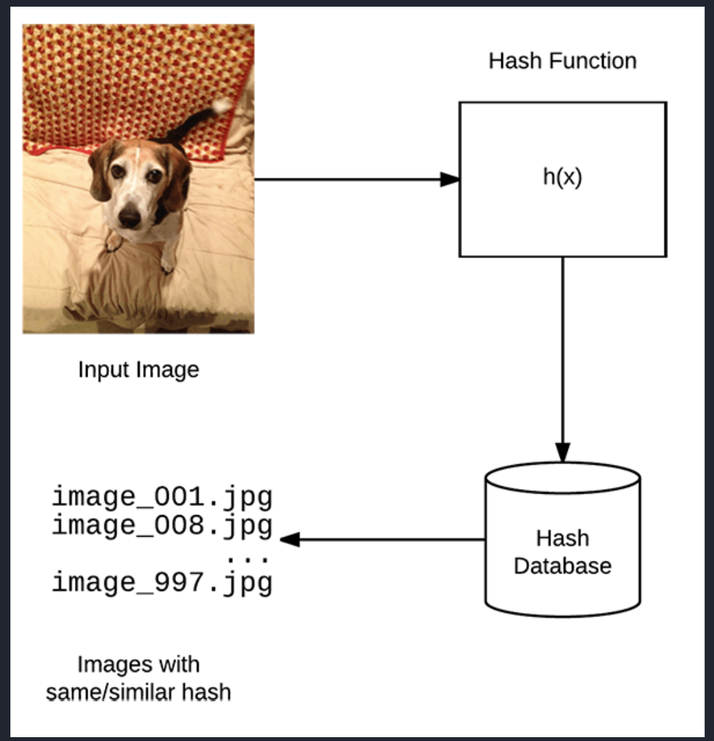
After images are posted on the internet, other web users can modify them and then repost their versions, thereby generating near-duplicate images

FINGERPRINT

The process of examining the contents of an image and then constructing a value that uniquely identifies an image based on these contents.



2 - THE FINGERPRINTING METHOD



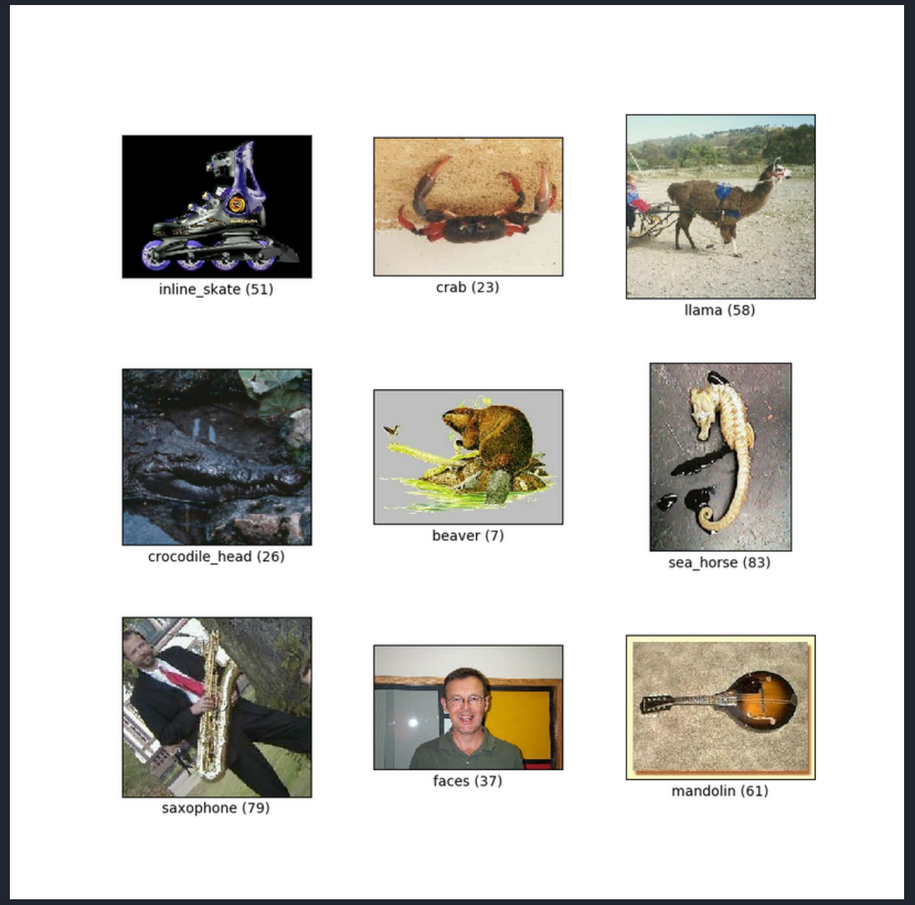
THE DATASET

CALTECH 101

7,500 images from 101 different categories, including people, motorcycles, and airplanes.

OUR GOAL

Randomly select 17 images, and create N new images by randomly resizing them by +/- a few percentage points. Then find these near-duplicate images

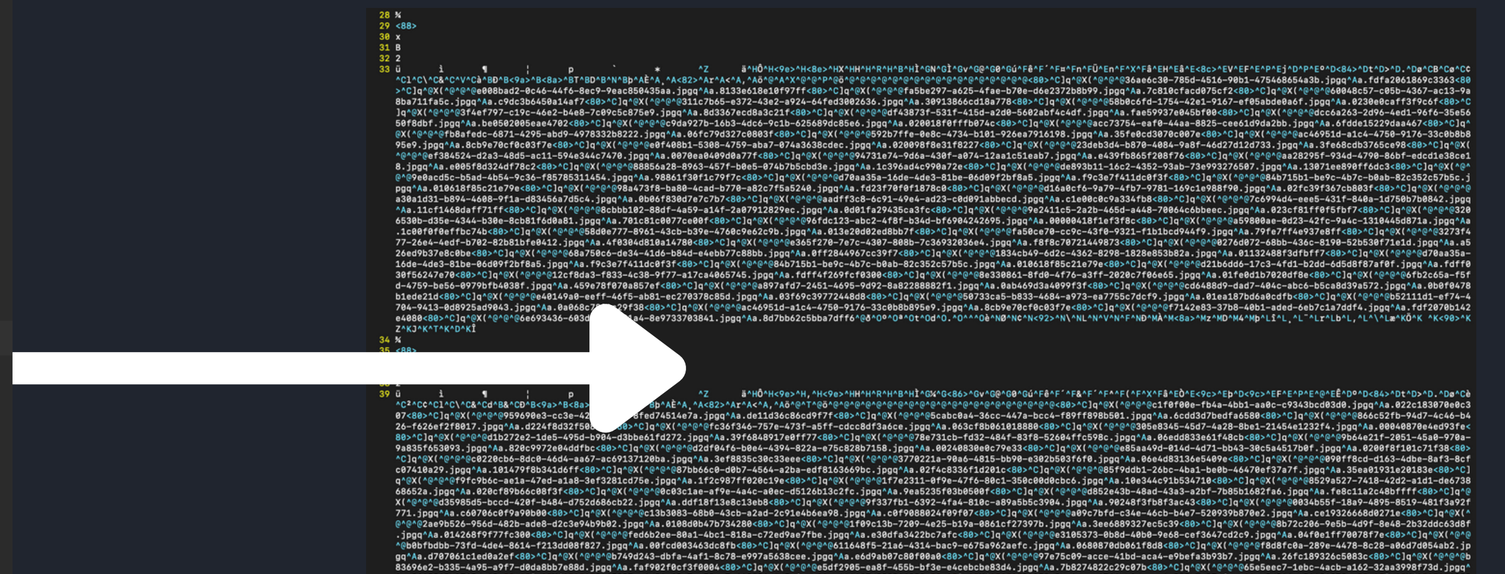


STEP 1 : CREATION OF A NEW DATASET

```
    help = "output directory")
ap.add_argument("-c", "--csv", required=True,
    help= "path to CSV file for image URL")
ap.add_argument("--csv2", required = True,
    help = "path to CSV file for image counts")
args = vars(ap.parse_args())

# open the output file for writing
output = open(args["csv"], "w")
output2 = open(args["csv2"], "w")


# loop over the input images
for imagePath in glob2.iglob(args["input"] + "/*/*.jpg"):
    # generate a random filename for the image and copy it to
    # the output location
    filename = str(uuid.uuid4()) + ".jpg"
    shutil.copy(imagePath, args["output"] + "/" + filename)
    fileUrl="http://"+''.join(random.choice(string.ascii_lowercase) for i in range(10))+".com"
    output2.write("%s,%s\n" % (filename, fileUrl))

    # there is a 1 in 500 chance that multiple copies of this
    # image will be used
    if random.randint(0, 500) == 0:
        # initialize the number of times the image is being
        # duplicated and write it to the output CSV file
        numTimes = random.randint(1, 8)
        output.write("%s,%d\n" % (filename, numTimes))

        #index.py --dataset images --shelve db.shelve loop over a random number of times for this image to
        # be duplicated
        for i in range(0, numTimes):
            image = Image.open(imagePath)

            # randomly resize the image, perserving aspect ratio
            factor = random.uniform(0.95, 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), Image.ANTIALIAS)

            # generate a random filename for the image and copy
            # it to the output directory
            adjFilename = str(uuid.uuid4()) + ".jpg"
            shutil.copy(imagePath, args["output"] + "/" + adjFilename)
            adjFileUrl="http://"+''.join(random.choice(string.ascii_lowercase) for i in range(10))+".com"
            output2.write("%s,%s\n" % (adjFilename,adjFileUrl))
```

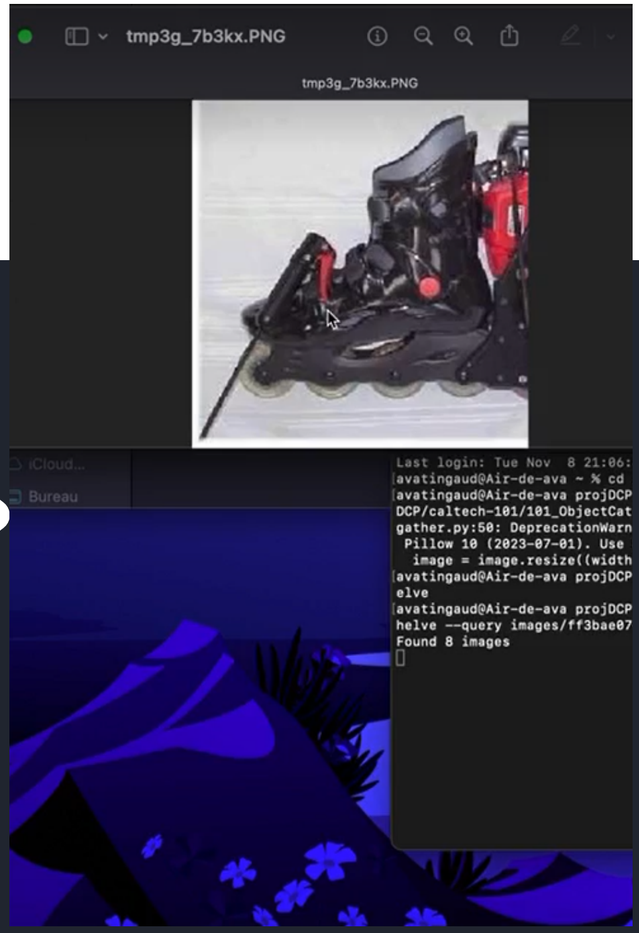
STEP 2 : EXTRACT IMAGE FINGERPRINTS FROM OUR DATASET AND BUILD OUR DATABASE OF FINGERPRINTS

```
# USAGE
# python index.py --dataset images --shelve db.shelve

# import the necessary packages
import ...

# 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"):
    # load the image and compute the difference hash
    image = Image.open(imagePath)
    h = str(imagehash.dhash(image))

    # extract the filename from the path and update the database
    # using the hash as the key and the filename append to the
    # list of values
    filename = imagePath[imagePath.rfind("/") + 1:]
    db[h] = db.get(h, []) + [filename]

# close the shelf database
db.close()
```



STEP 3 : PERFORM THE SEARCH TO DETERMINE IF AN IMAGE ALREADY EXISTS IN THE DATASET WITH THE SAME FINGERPRINT VALUE.

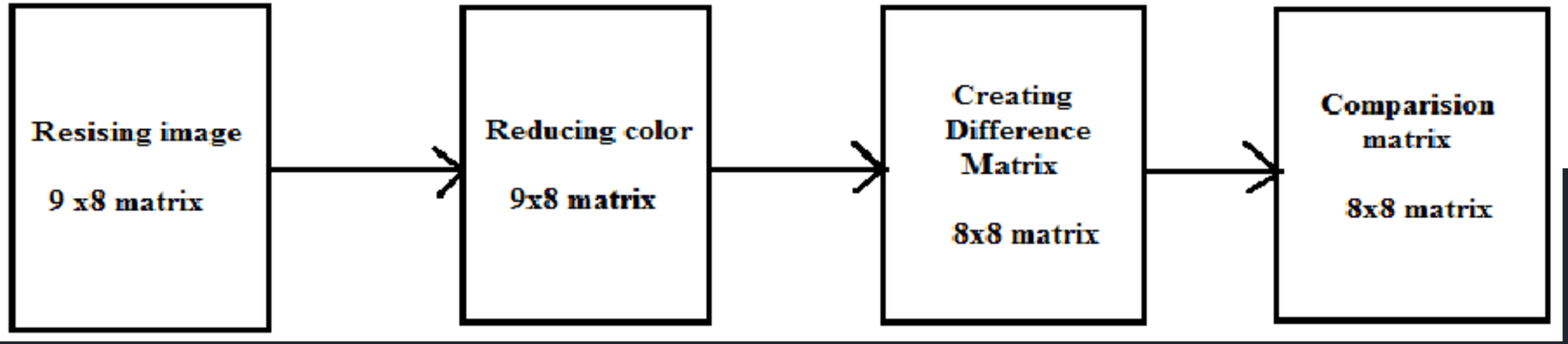
```
# python search.py --dataset images --shelve db.shelve --query images/84eba74d-38ae-4bf6-b8bd-79ffa1dad23a.jpg

# import the necessary packages
import ...

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

# open the shelve database
db = shelve.open(args["shelve"])

# load the query image, compute the difference image hash, and
# and grab the images from the database that have the same hash
# value
query = Image.open(args["query"])
h = str(imagehash.dhash(query))
filenames = db[h]
print ("Found %d images" % (len(filenames)))

# loop over the images
for filename in filenames:
    image = Image.open(args["dataset"] + "/" + filename)
    image.show()

# close the shelve database
db.close()
```



If there are any images with the same hash value, we loop over these images and display them to our screen.

MORE ABOUT IMAGEHASH

TO CALCULATE THE HASH OF AN IMAGE, DHASH USES A DIFFERENCE FIELD MATRIX.



1. RESIZING THE IMAGE
2. GRAYSCALING THE IMAGE
3. GENERATING A DIFFERENCE FIELD
4. GENERATING THE HASH

THE UNIQUENESS OF THE SOLUTION

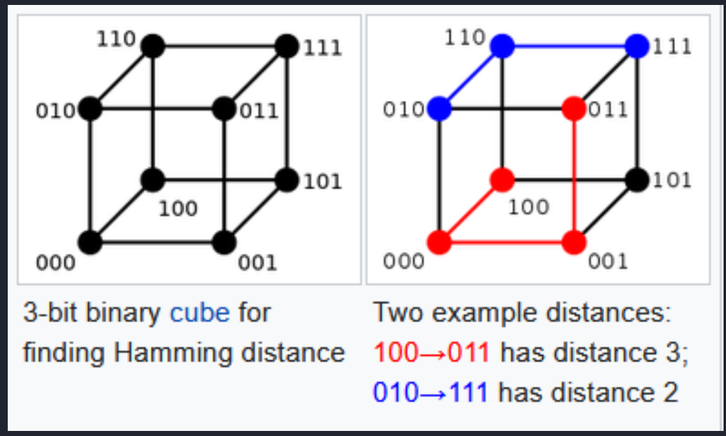
EXPERIMENT OVER 150 000 IMAGES

HASH FUNCTIONS	Comparaison Method	COMPUTING TIME	ACCURACY
aHash	Pixel value compared to the average color of the image	3,75 hours	High number of false positives
pHash	Pixel value compared to the frequencies in the image	7 hours	Perfect
dHash	Pixel brightness value compared to the adjacent pixel	< 3,75 hours	Very few false positives

IMPROVEMENTS

HAMMING DISTANCE

n order to find similar but not identical images we compute the Hamming distance in order to calculate the number of bits in a hash that are different

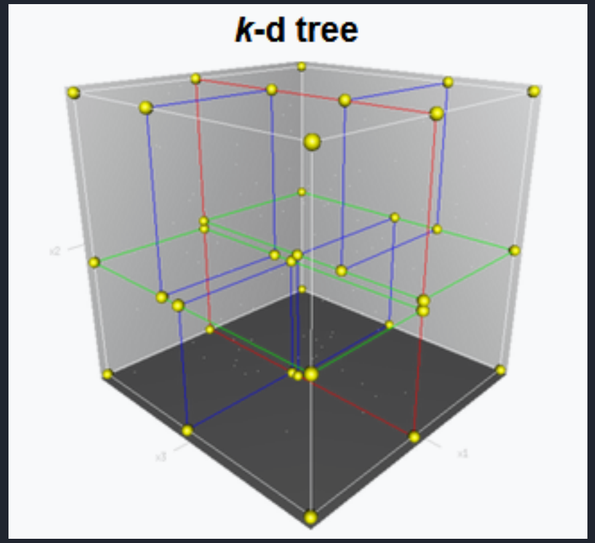


3-bit binary cube for finding Hamming distance

Two example distances: 100→011 has distance 3; 010→111 has distance 2

SPACE-PARTITIONING DATA STRUCTURE

To reduce the complexity of the search problem from linear to sub-linear we use k-d tree or Vantage-point tree



APPLICATION SCOPES

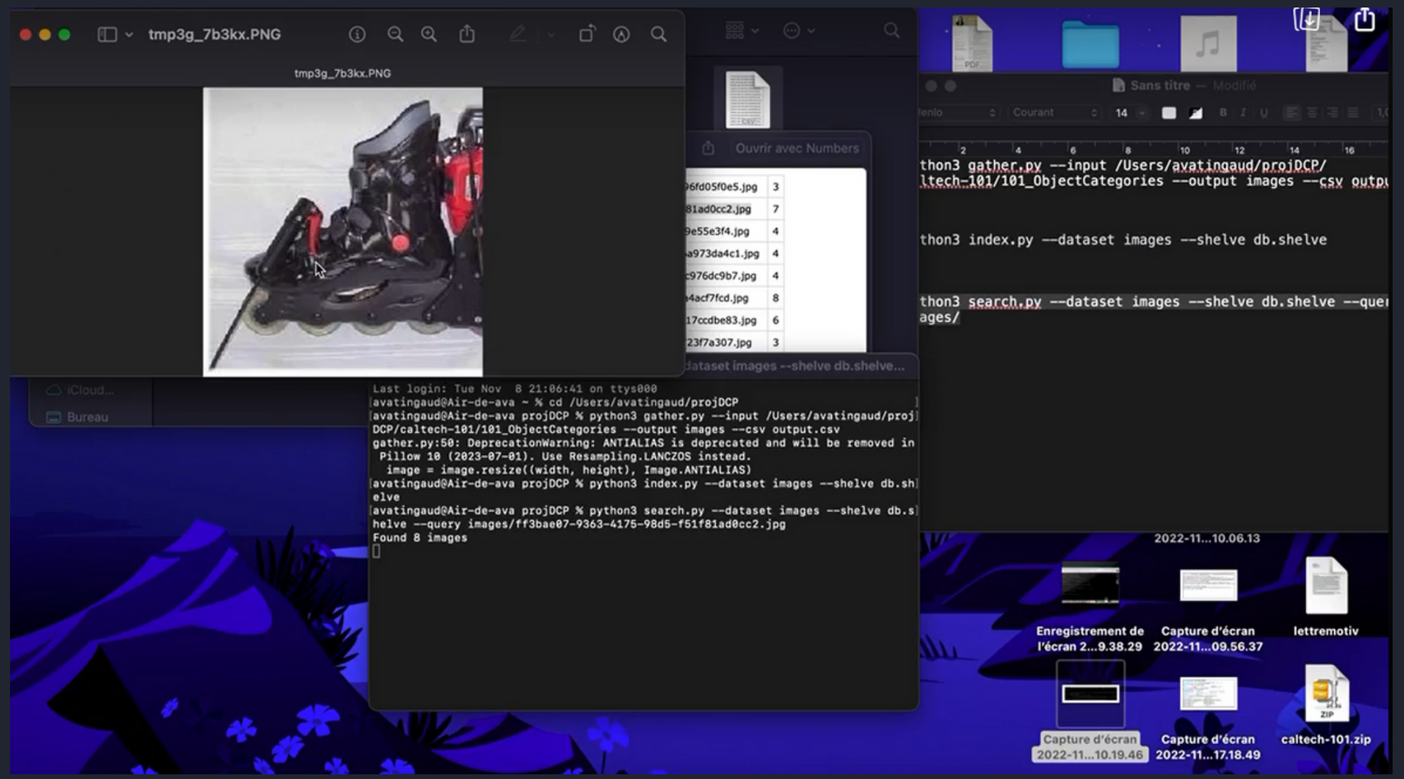
"DATING" ISSUES

Unwelcome images on certain websites can become pandemic if removed by hand. Moreover they can seriously damage the brand reputation of the app

E-TERRORISM

Terrorism on internet can be mentally damaging to the standard user but even more to moderators who have to go through thousands of traumatizing images per day

3 - RESULTS AND DEMONSTRATION

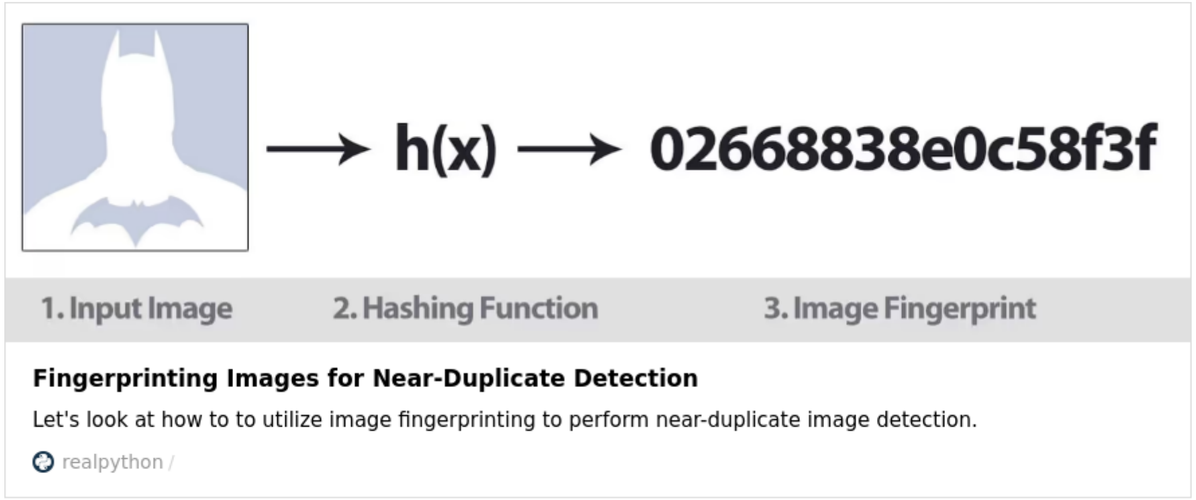


CONCLUSION :

image fingerprints are constructed using the visual contents of an image.

image fingerprint can uniquely identify an image.

Credit :



we built a system to find and recognize images with similar content using nothing but the image fingerprint

image fingerprint can be used to rapidly find images with near-duplicate content.