State of The Art

Vincent FALCONIERI

February 2019 - August 2019

Contents

Chapter 1

Introduction

A general overview was made through standard web lookup. [Bupe, 2017] A look was given to libraries, which also provide detailed and useful information. [Fea,]

In the following, we expose :

- The main steps of a Image Matching algorithm
- Few of the most popular Image Matching algorithms

Please, be sure to consider this document is under construction, and it can contain mistakes, structural errors, missing areas .. feel free to ping me if you find such flaw. (Open a PR/Issue/...)

Problem Statement [Cevikalp et al., 2018] states the Image Retrieval problem as "Given a query image, finding and representing (in an ordered manner) the images depicting the same scene or objects in large unordered image collections"

Chapter 2

Classic Computer vision techniques - White box algorithms

Correspondances found between images can be used for [Bian et al.,]:

1. **Similarity Measurement** : probability of two images for showing the same scene
2. **Geometry Estimation** : estimate the transformation between two object views
3. **Data Association** : Sorting pictures by scene (TO CHECK : Same as Similarity measurement

Block based approach : the image is divided into various blocks. These block division is done on the basis of Discrete Wavelet Transform, Discrete Cosine Transform, Zernike moment, and Fourier Mellin Transform. [Raj and Joseph, 2016]

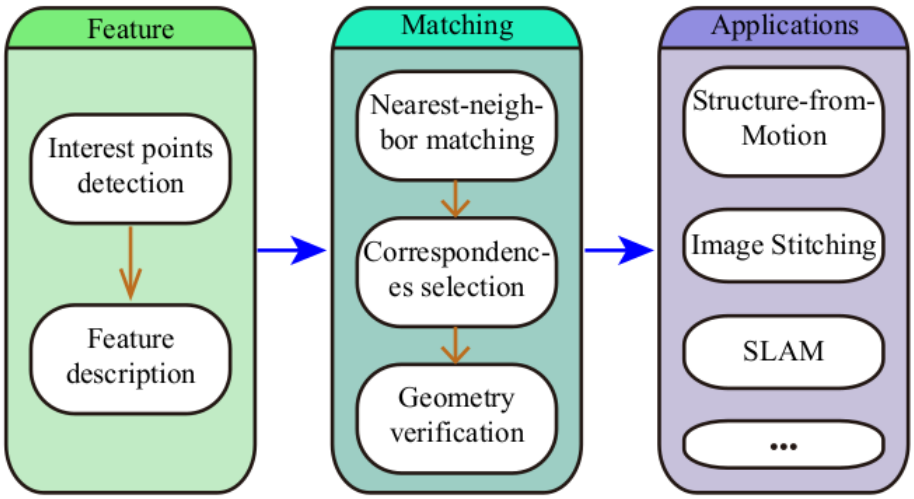


Figure 2.1: Image maching pipeline from [Bian et al.,]

Structure of Classic vision techniques

From [Yu, 2011] :

1. **Global features detection**

   (a) **Full region**
   (b) **Per subregion**

2. **Local features detection**
   Detection should be stable and repeatable. Corners, textured areas, etc. can be interest points. Robust to occlusion and viewpoint changes.

   (a) **Dense sampling over regular grid**
   (b) **Interest points detection**
   Find where interest points are.
      i. Not robust to scale
      Examples : Harris corner detector
      ii. Robust to scale
      Examples : Not robust + Increasing Gaussian blur many time, one for each scale ; Automatic scale selection ; ...

   (c) **Exotics**
      i. Random points sampling
      ii. Segmentation (?)
      iii. Pose estimation
      Example : "pictorial structure" (poselet) More complex

2.1 Step 1 - Key Point Detection

- Corner detectors to find easily localizable points.

Harris Detector

From the original paper [Harris and Stephens, 1988]. Based on the central principle: at a corner, the image intensity will change largely in multiple directions, with a windows shift.
(invariance to rotation, scale, illumination, noise .. said [Yu, 2011])
Distinctive features :

- Rotation-invariant
- NOT scaling invariant
  One point could be a corner in a small scaled neighborhood, or as an edge in a large scaled neighborhood.

CSS - Curvature Space Scale

Hit and Miss filter

Shi/Tomasi

SUSAN

From ... a word in [Rosten and Drummond, 2006] Less accuracy, more speed.

FAST - Features from Accelerated Segment Test

From the original paper [Rosten and Drummond, 2006] cited in [FAS, 2014]
Is a corner detector, based on machine learning. More accuracy, kept with high speed. Based on SUSAN
Distinctive features :

- Not rotation-invariant (no orientation calculation)
- ? scaling invariant

Pro

- High repeatability

Con

- not robust to high levels noise
- can respond to 1 pixel wide lines
- dependent on a threshold
- no scale/rotation invariance (? TO CHECK)

2.2 Step 2 - Descriptor Extraction

Extract a small patch around the keypoints, preserving the most relevant information and discaring necessary information (illumination ..)

Can be :

- Pixels values
- Based on histogram of gradient
- Learnt

Usually :

- Normalized
- Indexed in a searchable data structure

Example Vector descriptors based on our keypoints, each descriptor has size 64 and we have 32 such, so our feature vector is 2048 dimension.

Descriptor's quality A good descriptor code would be, according to [Spe,] :

- easily computed for a novel input
- requires a small number of bits to code the full dataset
- maps similar items to similar binary codewords
- require that each bit has a 50

We should be aware that a smaller code leads to more collision in the hash.

2.3 Step 3 - Feature representation

A local feature needs to be represented. From [Yu, 2011]

2.3.1 Bag-Of-Words or Bag-Of-Features

From [Yu, 2011], representing an image as a set of feature descriptor.

Pro

- Insensitivity of objects location in image

Con

- Loss of spatial information

2.3.2 Codebook Generation

From [Yu, 2011], K-Means clustering over all words describing all pictures. A representative word (=Codeword) of each cluster is chosen (the "mean word"). A list of all representative words is created. A representative vector for each image, is created, as a boolean_list/histogram of representative words linked or not to this image.

Pro

- Shorten the comparisons to do (TO CHECK)

Con

- Representation ambiguity : Codeword may not be representative of a cluster of words (too large, too narrow, more than 1 meaning, ...)

2.3.3 Soft Vector Quantization

From [Yu, 2011], codebook Generation with most and least frequent words removal. Each feature is then represented by a small group of codewords.

Pro

- Mitigate the representation ambiguity problem of CodeBook

Con

- Undo something that has been done ? TO CHECK !

2.3.4 Hierarchical CodeWords

From [Yu, 2011], keep spatial information about the neighboorhood of a codeword.

2.3.5 Visual sentences

Project codeword on a spatial axis. Relative spatial relation between words are kept.

2.3.6 SPM - Spatial Pyramid Matching

From [Yu, 2011], divide a picture into equal partitions (/4, /8, ..), compute a Bag-Of-Word for each partition, its histogram, and concatenate them into one big "ordered" histogram.

Pro

- Keep spatial information of features

Con

- Some "bad translation" can occurs, and split features into different Bag-of-words.

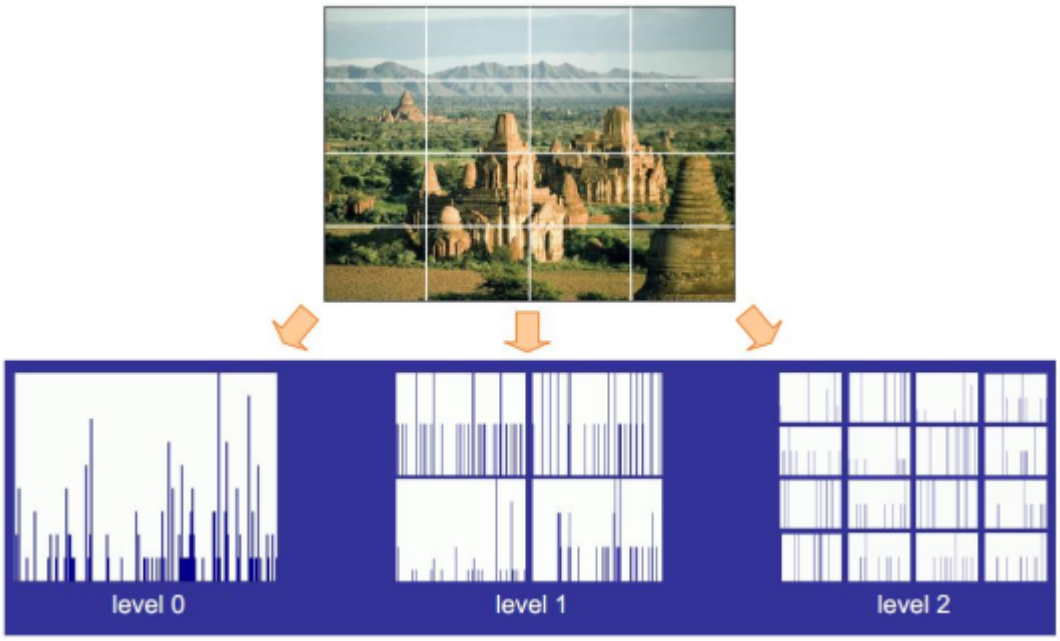


Figure 2.2: 3 levels spatial pyramid from [Yu, 2011]

2.3.7 L-SPM - Latent Spatial Pyramid Matching

From [Yu, 2011], based on SPM but does not split the picture in equal partition = the cell of the pyramid is not spatially fixed.

The cells of the pyramid to move within search regions instead of a predefined rigid partition. Use ESS (See utilities)

Pro

Con

- High computaitonal cost

2.4 Step 4 - Matching

Linked to correspondence problem ?

2.4.1 Datastructure

Compression of descriptors before matching

LSH - Locally Sensitive Hashing

- O(˜N)
- Returns the K (parameter) best matchs
- [Ope,]
- Convert descriptor (floats) to binary strings. Binary strings matched with Hamming Distance, equivalent to a XOR and bit count (very fast with SSE instructions on CPU)

BBF - Best bin first Kd-tree

- O(˜N)
- Example : SIFT – Scale Invariant Feature Tranform

2.4.2 Distance

Hamming distance / Bruteforce

Partially solved by [Manku et al., 2007]
Works with binary features. Can be accelerated with GPU [Bian et al.,].

- $O(N^2)$, N being the number of descriptor per image
- One descriptor of the first picture is compared to all descriptor of a second candidate picture. A distance is needed. The closest is the match.
- Ratio test
- CrossCheck test : list of "perfect match" (TO CHECK)

FLANN - Fast Library for Approximate Nearest Neighboors

From [Bian et al.,], is an approximation for matching in Euclidian space, with KD-Tree techniques.
Work with non-binary features.

- Collections of algorithm, optimized for large dataset/high dimension
- Returns the K (parameter) best matchs
- [Ope,]

Implementation Apache 2 From [Boytsov and Naidan, 2013], available at : `https://github.com/nmslib/nmslib` [Non, 2019] Does not benchmark memory usage.

2.4.3 Selection

Higly noisy correspondences need to be filtered.

Best match

- Returns only the best match
- Returns the K (parameter) best matchs

RATIO -

From [Bian et al.,] recognizes the distinctiveness of features by comparing the distance of their two nearest neighbors.

This test rejects poor matches by computing the ratio between the best and second-best match. If the ratio is below some threshold, the match is discarded as being low-quality.

GMS - Gird-based Motion Statistics

Uses the motion smoothness constraint. Equivalent to RATIO.

Robustness, accuracy, sufficiency, and efficiency of GMS all depend on the number of interest points detected.

QBH - Quantization Based Hashing

From [Song et al., 2018] Incorporates quantization error into the conventional property preserving hashing models to improve the effectiveness of the hash codes

IMI - Inverted Multi-Index

NGS - Neighboorhood Graph Search

HNSW - Hierarchical Navigable Small Worlds Graph based approach. Precise approximate nearest neighbor search in billion-sized datasets.
Highly scalable : "Indexing 1 billion vectors takes about 26 hours with L&C: we can add more than 10,000 vectors per second to the index. We refine at most 10 vectors."

Implementation BSD `https://github.com/facebookresearch/faiss` [Lib, 2019]

2.5 Step 5 - Model Fitting

From [Bian et al.,] and [Fea,], is a step where the geometry of the scene is verified and/or estimated. Given correspondances, the pose of the object is estimated.

- Identify inliers and outliers ˜ Fitting a homography matrix ˜ Find the transformation of (picture one) to (picture two)
- Inliers : "good" points matching that can help to find the transformation
- outliers : "bad" points matching

2.5.1 RANSAC – Random Sample Consensus

Estimation of the homography, searches for the best relative pose between images iteratively and removes outliers that disobey the estimated geometry relation finally. Correspondences that pass the geometry verification are named verified correspondences. Provides a robust estimation of transform matrix.

2.5.2 Least Meadian

Chapter 3

Global Features Algorithms

Generally, weak against occlusion, clutter. Need fixed viewpoint, clear background,fixed pose.

The main assumption is that the similar images in the Euclidean space must have similar binary codes. [Cevikalp et al., 2018]

Categories :

- Locality Sensitive Hashing schemes (LSH)
- Context Triggered Piecewise Hashing (CTPH)

3.1 Full region features - FH or CTPH - Fuzzy Hashing Algorithms

The following algorithms does not intend to match pictures with common part, but to match pictures which are rougly the same. To be clear : If the hashes are different, then the data is different. And if the hashes are the same, then the data is likely the same. There is a possibility of a hash collision, having the same hash values then does not guarantee the same data.

Discrete Cosine Transformation (CDT) may be worst than Discrete Wavelet Transformation (DWT).

From [Sarantinos et al., 2016], also called Context Triggered Piecewise Hashing (CTPH). It is a combination of Cryptographic Hashes (CH), Rolling Hashes (RH) and Piecewise Hashes (PH).

Fuzzy hashing has as a goal of identifying two files that may be near copies of one another.

SDHash seesm the more accurate, but the slower. Cross-reference seems a good way to go.

Examples : Holistic features (="Spatial enveloppe" = naturalness, openness, roughness, ruggedness, expansion ..), colors histograms, "Global Self-Similarity" (=spatial arrangement)

DIFFERENT HASH THAN THE SOURCE IMAGE

Type of error	aHash	bHash	dHash	mHash	pHash	wHash
Gaussian smoothing	1828 (20.0%)	1241 (13.6%)	3807 (41.6%)	2122 (23.2%)	786 (8.6%)	971 (10.6%)
Grayscale	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)
Increased brightness	3874 (42.4%)	3206 (35.1%)	5357 (58.6%)	3451 (37.7%)	3986 (43.6%)	2844 (31.1%)
Decreased brightness	1717 (18.8%)	809 (8.8%)	4935 (54.0%)	2115 (37.7%)	1030 (11.3%)	1420 (15.5%)
JPEG compression	455 (5.0%)	598 (6.5%)	1616 (17.7%)	658 (7.2%)	546 (6.0%)	514 (5.6%)
Increased contrast	2559 (28.0%)	2062 (22.6%)	4568 (50.0%)	2474 (27.1%)	2460 (26.9%)	2197 (24.0%)
Decreased contrast	2056 (22.5%)	766 (8.4%)	5223 (51.1%)	2154 (23.6%)	1063 (11.6%)	2026 (22.2%)
Scaled	554 (6.1%)	1297 (14.2%)	2091 (22.9%)	851 (9.3%)	664 (7.3%)	614 (6.7%)
Watermark	3600 (39.4%)	5046 (55.2%)	7167 (78.4%)	4029 (44.1%)	4242 (46.4%)	2896 (31.7%)
Cropped	8543 (93.4%)	8750 (95.7%)	9075 (99.2%)	8514 (93.1%)	9088 (99.4%)	7840 (85.7%)
Total	**25,186 (25.0%)**	**23,775 (23.6%)**	**43,839 (43.6%)**	**26,368 (26.2%)**	**23,865 (23.7%)**	**21,322 (21.2%)**

Figure 3.1: Results from [Tes,] - Lower score is better

3.1.1 A-HASH : Average Hash

From ... [Loo, 2011] : "the result is better than it has any right to be."

relationship between parts of the hash and areas of the input image = ability to apply "masks" (like "ignore the bottom 25% of the image".) and "transformations" at comparison time. (searches for rotations in 90degree steps, mirroring, inverts...)

8 bits for a image vector.

Idea to be faster (achieve membw-bound conditions) : Batch search (compare more than one vector to all others) = do X search at the same time.

More than one vector could be transformation of the initial image (rotations, mirrors).

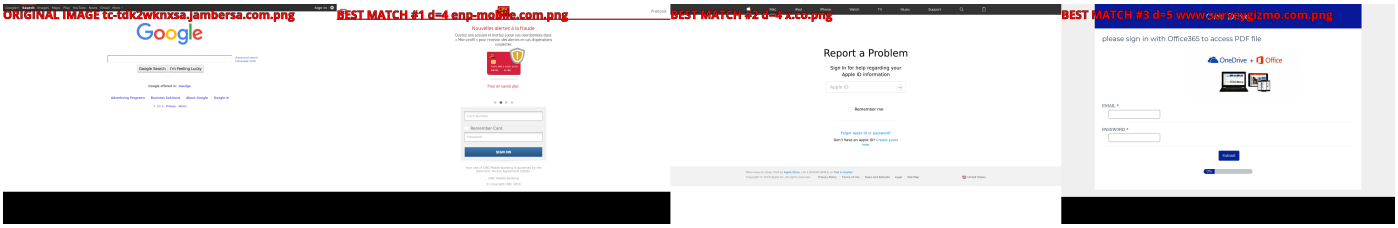
Pro

- Masks and transformation available
- Ability to look for modified version of the initial picture
- Only 8 bits for a image vector.

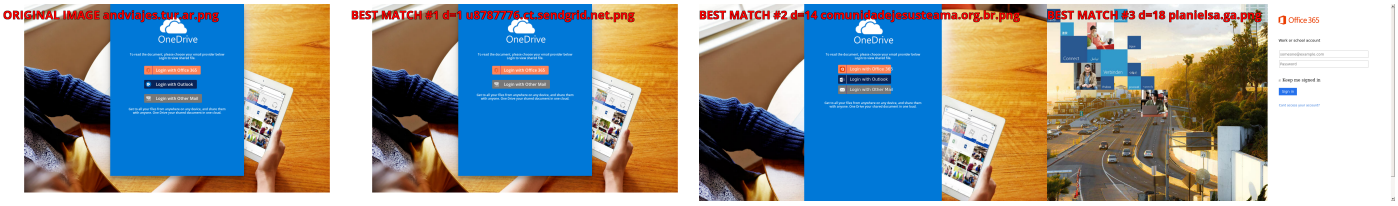
Implementation ImageHash 4.0 `https://pypi.org/project/ImageHash/`
Javascript Implementation : [Aluigi, 2019]

Results Results given by the ImageHash implementation of A-hash algorithm does not provides reliable results.

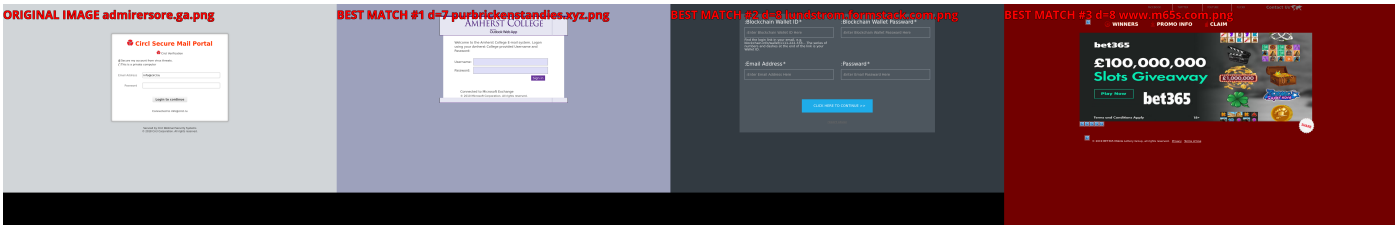
Time Hashing time : 16.796968936920166 sec for 207 items (0.081s per item)
Matching time : nobs=207, minmax=(0.023s, 1.58s), mean=0.08s, variance=0.025s, skewness=6.62s, kurtosis=50.37s



(a) Structural matching leading to mismatch



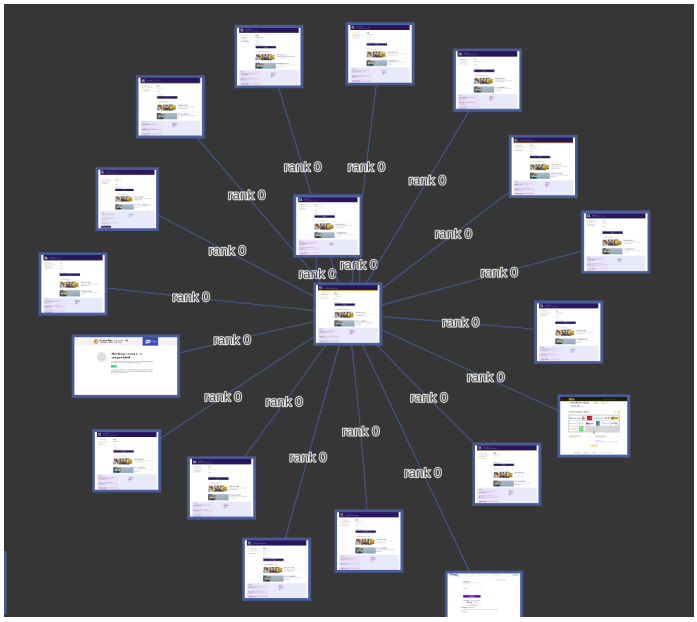
(b) Structural matching leading to match



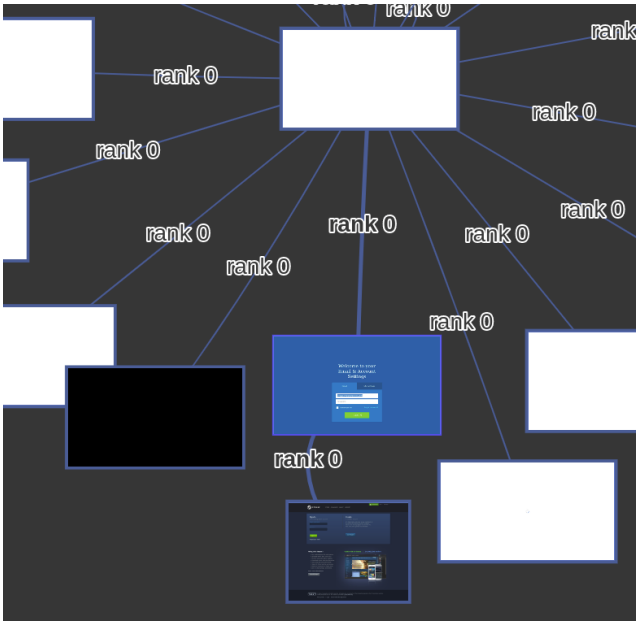
(c) Structural matching seeing "a form"

(d) Missing logo in the match

Figure 3.2: A-hash



(a) Good structural matching



(b) Strange matching (black/white)

(c) Bad matching

Figure 3.3: A-hash

3.1.2 B-HASH : Block Hash

Seems worst than A-hash.

Implementation [Blo,] `http://blockhash.io` and `https://github.com/commonsmachinery/blockhash-pyth`

3.1.3 D-HASH - Difference Hashing

From [Hahn, 2019], DHash is a very basic algorithm to find nearly duplicate pictures.
The hash can be of length 128 or 512 bits. The delta between 2 "matches" is a Hamming distance (# of different bits.)

Pro

- Detecting near or exact duplicates : slightly altered lighting, a few pixels of cropping, or very light photoshopping

Con

- Not for similar images
- Not for duplicate-but-cropped

Steps of the algorithm

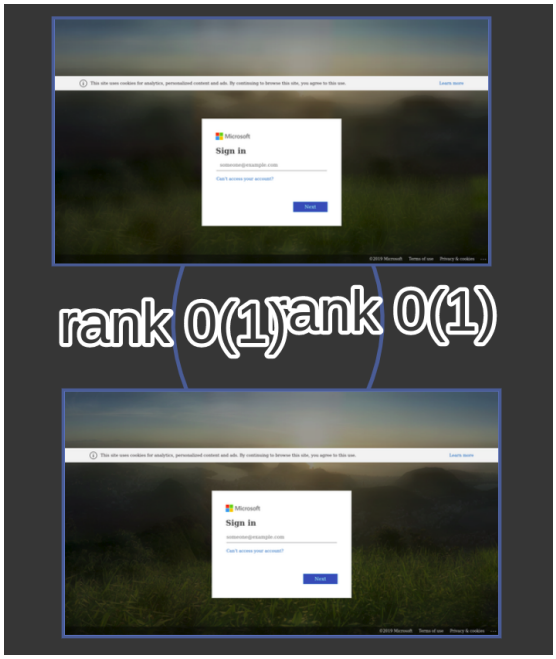
1. Convert the image to grayscale
2. Downsize it to a 9x9 thumbnail
3. Produce a 64-bit "row hash": a 1 bit means the pixel intensity is increasing in the x direction, 0 means it's decreasing
4. Do the same to produce a 64-bit "column hash" in the y direction
5. Combine the two values to produce the final 128-bit hash value

Implementation ImageHash 4.0 `https://pypi.org/project/ImageHash/`

Results Results given by the ImageHash implementation of D-hash algorithm does not provides reliable results, but better than a-hash results.

Time nobs : 207s min time : 0.00211s max time : 0.00379s mean :0.0024236169990134123s variance : 1.3200716728248016e-07s skewness : 2.412448208288436s kurtosis : 5.039791588290182

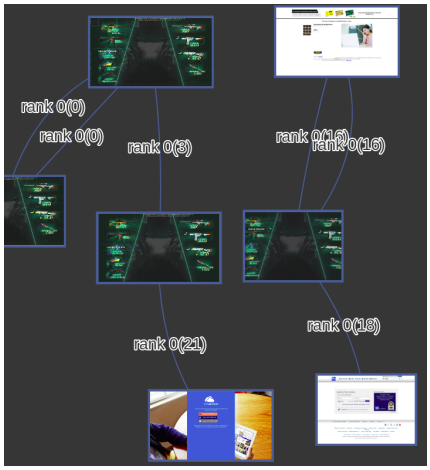
"Vertical" variation A variation of the d-hash algorithm is available, as d-hash vertical.



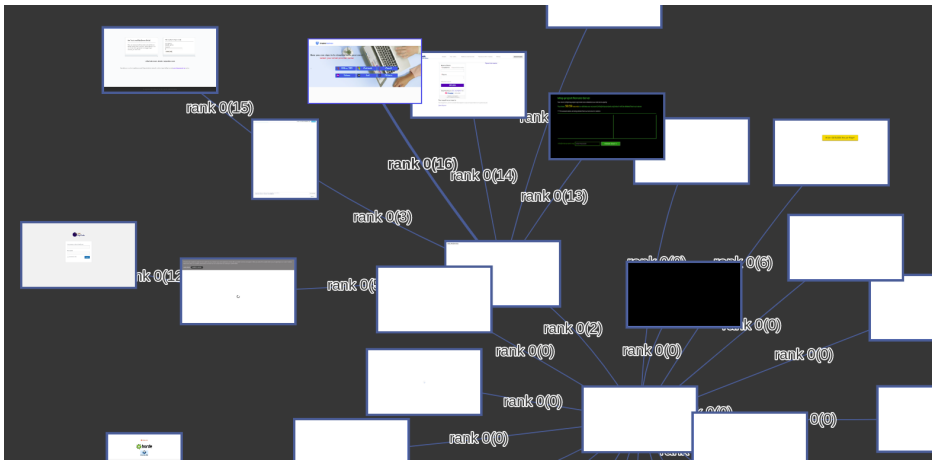
(a) Better than A-hash result

(b) Better than A-hash result

Figure 3.4: d-hash



(a) Wrong result



(b) Wrong result

Figure 3.5: d-hash

(a) Mismatch (b) Mismatch

Figure 3.6: Vertical d-hash

3.1.4 P-HASH - Perceptual Hash

From ... [Loo, 2011] and [Igor, 2011] and [PHa, 2013]
Exist in mean and median flavors
8 bits for a image vector.
Java implementation : [PHa, 2011]

Pro

- Robustness to gamma
- Robustness to color histogram adjustments
- Should be robust to rotation, skew, contrast adjustment and different compression/formats.
- Open Source (GPLv3), C++, API

Steps of the algorithm

1. Reduce size of the input image to 32x32 (needed to simplify DCT computation)
2. Reduce color to grayscale (same)
3. Compute the DCT : convert image in frequencies, a bit similar to JPEG compression
4. Reduce the DCT : keep the top-left 8x8 of the DCT, which are the lowest frequencies
5. Compute the average DCT value : without the first term (i.e. solid colors)
6. Further reduce the DCT : 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.
7. Construct the hash : create a 64 bits integer from the hash
8. Comparing with Hamming Distance (threeshold = 21)

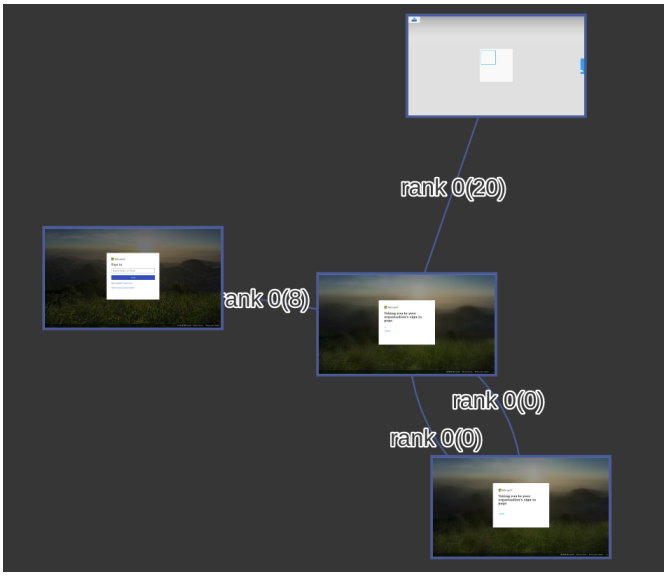
Implementation ImageHash 4.0 `https://pypi.org/project/ImageHash/`

Results Still some strange results

Time nobs : 207 min time : 0.00253s max time : 0.63245s mean :0.009314882582512455s variance : 0.0020187846451833486s skewness : 13.117889676404076s kurtosis : 177.8720654998952

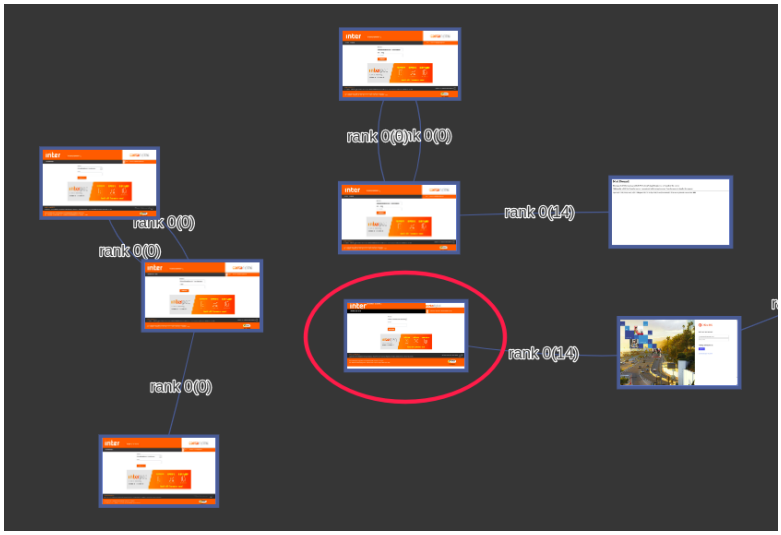
"Simple" variation A variation of the p-hash algorithm is available, as p-hash simple.

(a) Wrong result

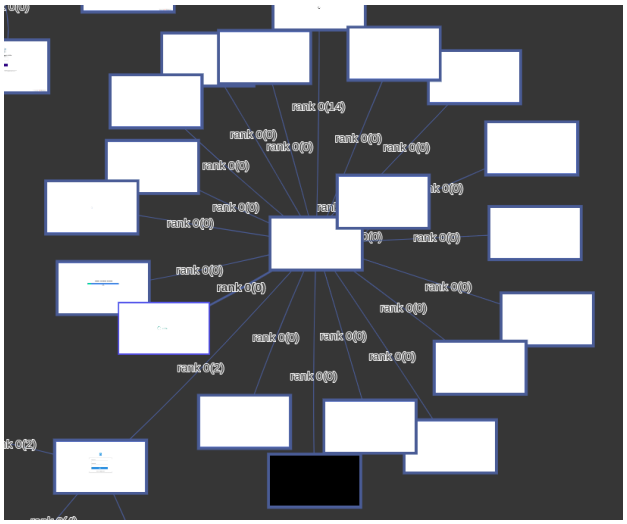


(b) Wrong result

Figure 3.7: P-hash



(a) Wrong result



(b) Good result

Figure 3.8: P-hash - Simple

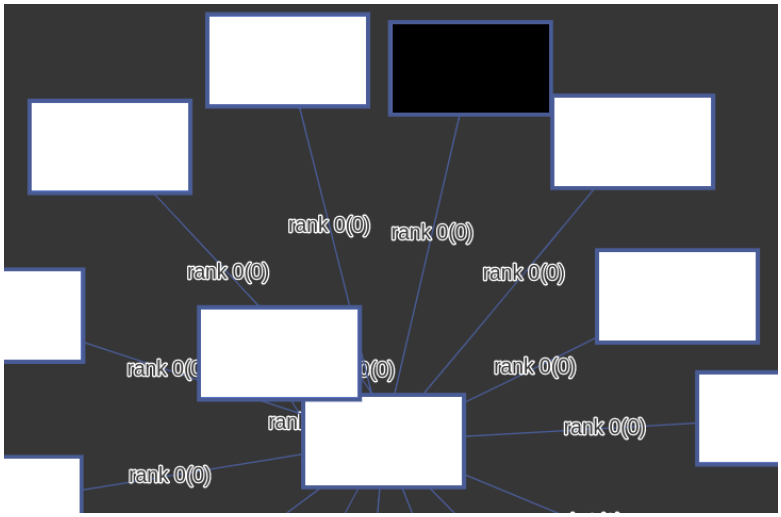
3.1.5 W-HASH - Wavelet Hash

From ... Uses DWT instead of DCT. [TO LOOK]

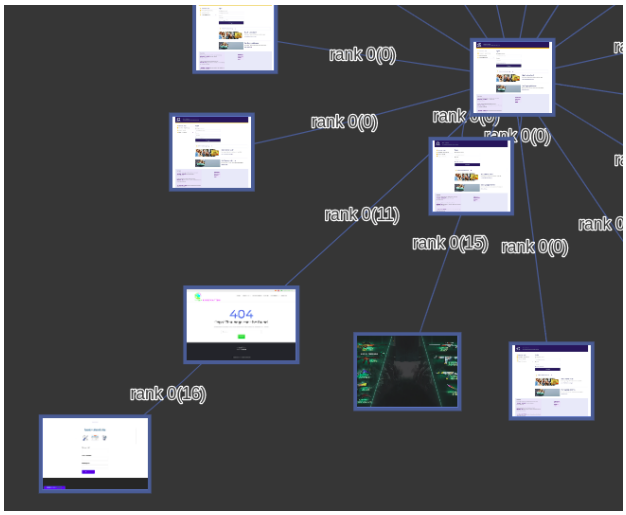
Implementation ImageHash 4.0 `https://pypi.org/project/ImageHash/`

Results Better than others, but still some strange/wrong results.

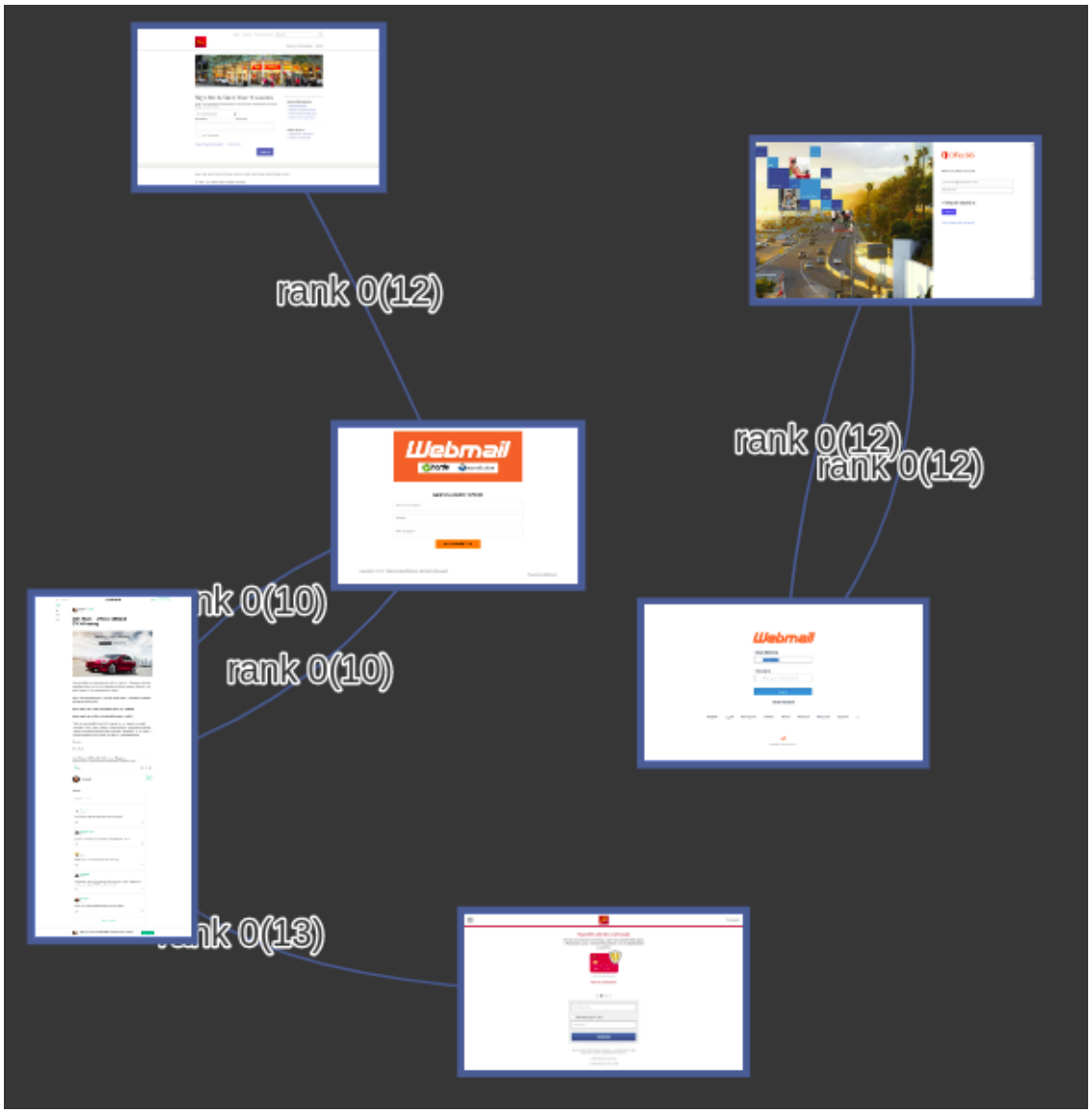
Time nobs : 207 min time : 0.00214s max time : 0.53841s mean :0.005320862295546 97s variance : 0.0013867749582104625s skewness : 14.274196161282449s kurtosis : 201.83699346303464



(a) Strange result



(b) Wrong result



(c) Wrong result

Figure 3.9: W-hash

3.1.6 SimHash - Charikar's simhash

From ... [Manku et al., 2007]
repository of 8B webpages, 64-bit simhash fingerprints and k = 3 are reasonable.
C++ Implementation

3.1.7 R-HASH

From ... [Loo, 2011]

Equivalent to A-Hash with more granularity of masks and transformation. Ability to apply "masks" (color channel, ignoring (f.ex. the lowest two) bits of some/all values) and "transformations" at comparison time. (color channel swaps)

48 bits for a rgb image vector

Pro

- Masks and transformation available
- More precise masks (than A-hash)
- More precise transformations (than A-hash)

Con

- Larger memory footprint

Steps of the algorithm

1. Image scaled to 4x4
2. Compute vector
3. Comparison = sum of absolute differences: abs(a[0]-b[0]) + abs(a[1]-b[1]) + ... + abs(a[47]-b[47]) = 48 dimensional manhattan distance

3.1.8 Spectral-HASH

From [Spe,]. A word is given in [Loo, 2011]

The bits are calculated by thresholding a subset of eigenvectors of the Laplacian of the similarity graph

Similar performance to RBM

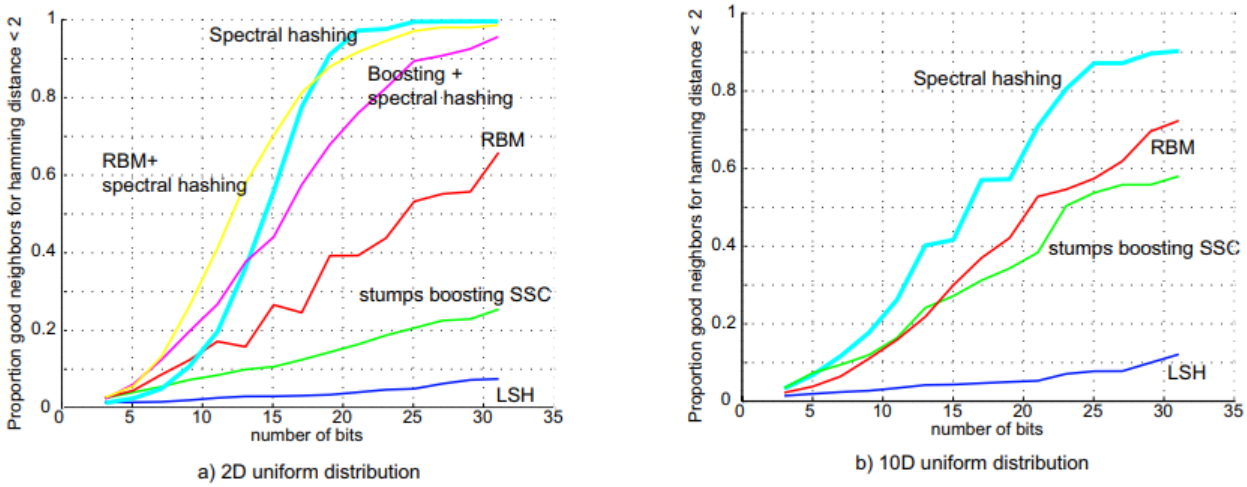


Figure 3.10: Spectral Hashing comparison from [Spe,]

3.1.9 LSH - Locality Sensitve Hashing

Same as E2LSH ? Chooses random projections so that two closest image samples in the feature space fall into the same bucket with a high probability, from [Cevikalp et al., 2018]

3.1.10 E2LSH - LSH - Locality Sensitve Hashing

From [Gionis et al.,] a word is given in [Spe,] and [Cevikalp et al., 2018].
The code is calculated by a random linear projection followed by a random threshold, then the Hamming distance between codewords will asymptotically approach the Euclidean distance between the items.

Not so far from Machine Learning Approaches, but outperformed by them.

Pro

- Faster than Kdtree

Con

- Very inefficient codes (512 bits for a picture (TO CHECK))

3.1.11 Nilsimsa hash - Locality sensitive hash

A word in [Oliver et al., 2013]

Pro

- Open Source

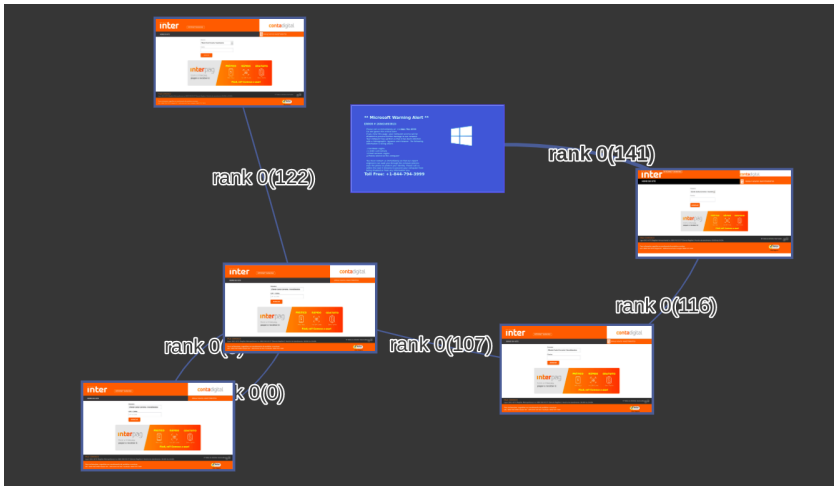
3.1.12 TLSH - Trend Micro Locality Sensitive Hashing

From [Oliver et al., 2013] directly performed on file, not only pictures.

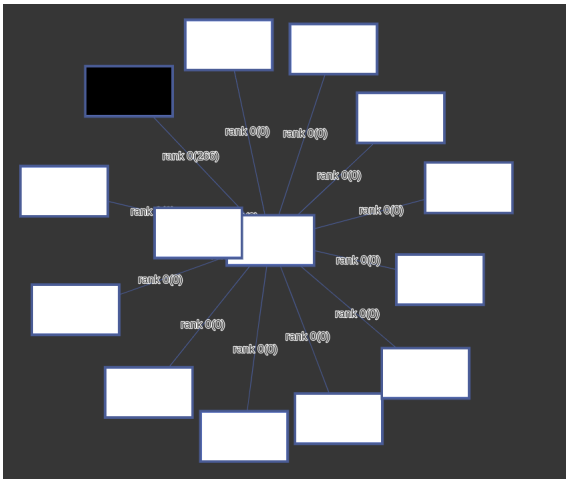
Pro

- Parametered threeshold (below 30 in original paper)
- Open Source

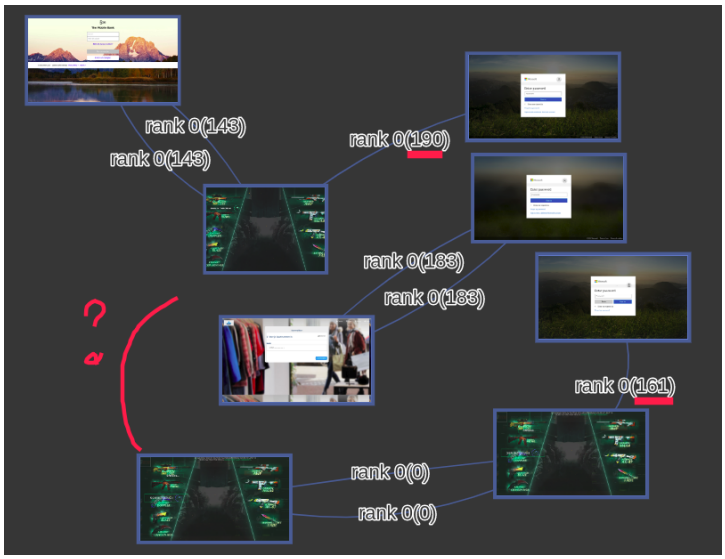
Time Normal version : nobs : 207s min time : 0.00083s max time : 0.00165s mean :0.00095s variance : 0.0s skewness : 2.7512s kurtosis : 7.24537 No-length version : nobs : 207s min time : 0.00095s max time : 0.00455s mean :0.00115s variance : 0.0s skewness : 7.60762s kurtosis : 67.0044



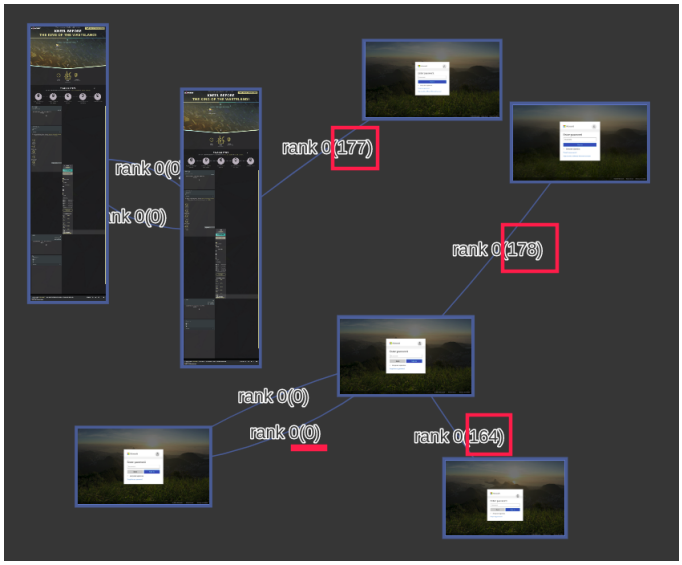
(a) High distance for close pictures



(b) Perfect match for colors



(c) Strange mismatches



(d) Diverses matches

Figure 3.11: TLSH (normal)

(a) Strange result

(b) Strange result

Figure 3.12: Inconsistent distance regarding pictures - TLSH (no length)

3.1.13 SSDeep - Similarity Digest

From ... few words on it in [Sarantinos et al., 2016]
Implementation (C) at [Fuz, 2019]
Historically the first fuzzing algorithm. CTPH type.

Table 1 – Time to hash pseudo-random data

Algorithm	1 MB	10 MB	50 MB
MD5	9	49	223
SHA256	24	184	897
Ssdeep	71	669	6621
Whirlpool	156	1505	7518

All times are measured in milliseconds.

Figure 3.13: Hashing time from [Kornblum, 2006]

Pro

- Effective for text (Spam, ..)
- Open Source

Con

- Not effective for Images, Videos, ...
- Les effective than Sdhash

Steps of the algorithm

1. Rolling hashing to split document into "6 bits values segments"
2. Uses hash function (MD5, SHA1, ..) to produce a hash of each segment
3. Concatenate all hashes to create the signature (= the fuzzy hash)

3.1.14 SDHash - Similarity Digest Hash

From ... Roussev in 2010 few words on it in [Sarantinos et al., 2016]

Uses Bloom Filters to identify similarities between files on condition with common features. (Quite blurry)

Pro

- More accurate than VHash, SSDeep, MRSHV2
- Options available (TO CHECK) - See a particular implementation used in [Sarantinos et al., 2016]
- Open Source

Con

- Slow compared to MVHash, SSDeep, MRSHV2

Steps of the algorithm

1. Perform a hash/entropy (TO CHECK) calculation with a moving window of 64 bits.
2. Features (? How are they recognized?) are hashed with SHA-1
3. Features are inserted into a Bloom Filter

3.1.15 MVHash - Majority Vote Hash

From ... few words on it in [Sarantinos et al., 2016]
It is Similarity Preserving Digest (SPD) Uses Bloom Filters

Pro

- almost as fast as SHA-1 (paper)

Steps of the algorithm

1. Majority vote on bit level (transformation to 0s or 1s)
2. RLE = Run Length Encoding, represents 0s and 1s by their length
3. Create the similarity digest (? TO CHECK)

3.1.16 MRSH V2 - MultiResolution Similarity Hashing

From ... few words on it in [Sarantinos et al., 2016] Variation of SSDeep, with polynomial hash instead of rolling hash (djb2)

Pro

- Fast than SDHash

Con

- Slow compared to MVHash, SSDeep

3.1.17 GIST -

From [Oliva and Torralba,] a word in [Li et al., 2018]

Holistic feature which is based on a low dimensional representation of the scene that does not require any form of segmentation, and it includes a set of perceptual dimensions (naturalness, openness, roughness, expansion, ruggedness)

3.2 Per subregion features

Per subregion Example : Histogram of Oriented Gradients (HOG)
Holistic feature ...

3.2.1 HOG - Histogram of Oriented Gradients

From ... A word in [Yu, 2011] The idea is to describe shape information by gradient orientation in localized sub-regions.

Chapter 4

Algorithms combination

4.1 Block-based approach + KeyPoint approach for Image manipulation

From [Raj and Joseph, 2016]

Chapter 5

Local Features Algorithms

Goal is to transform visual information into vector space

5.1 Comparison overview

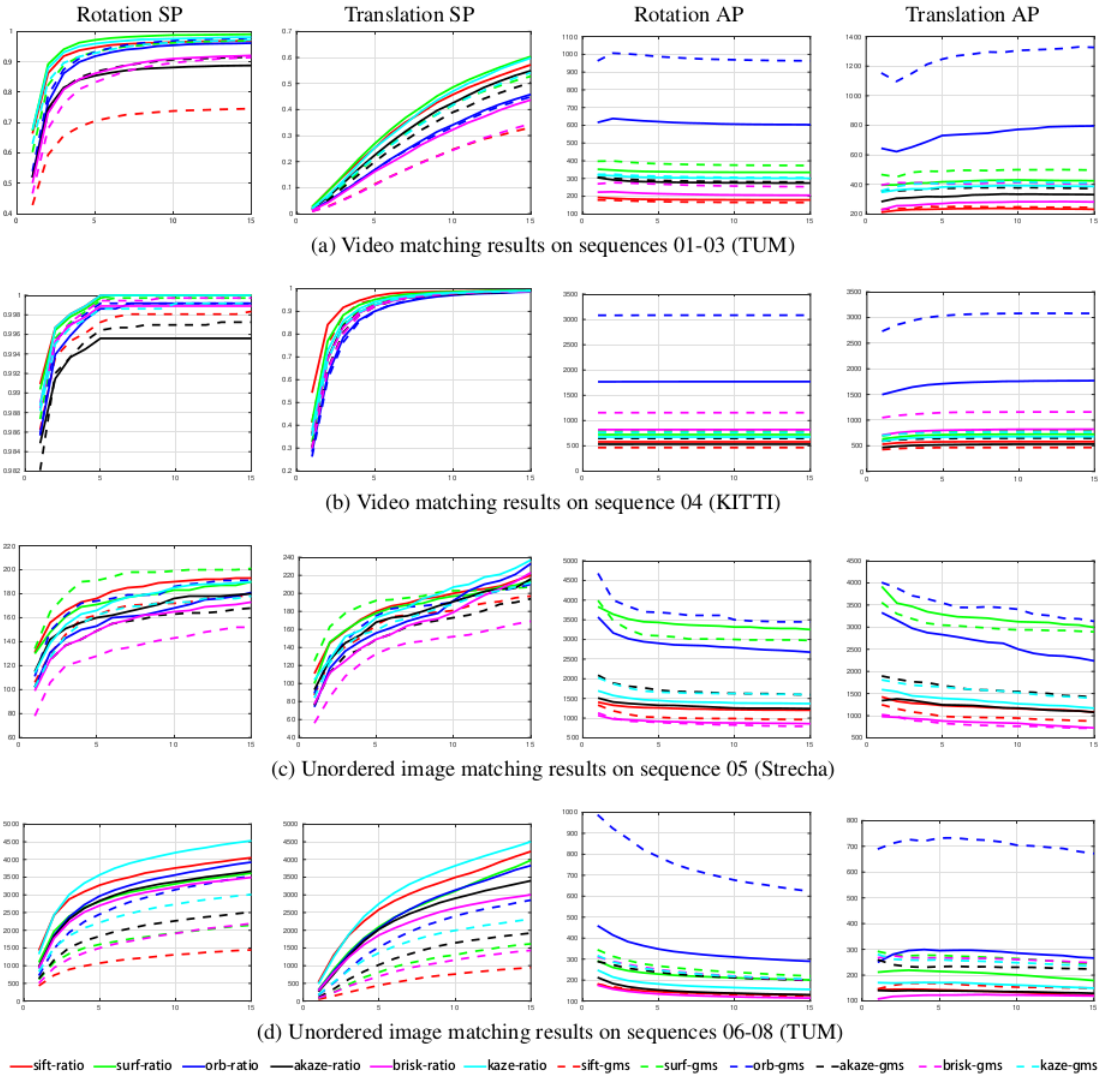


Figure 5.1: Benchmarking of SIFT, SURF, ORB, AKAZE with RATIO and GMS selection ; FLANN or Hamming for distance. SP curves show the success ratio or success number (number of correspondance for AP) with thresholds. X-Axis being the threeshold. AP curves illustrate the mean number of verified correspondences with thresholds.[Bian et al.,]

In few words :

- **Robustness**
  Success ratio (15°difference max from real position) = Succes to match pairs
  Non-binary are better than binaries algorithms. Number of interest points change the best matching method to choose.
- **Accuracy**
  Success ratio (5°difference max from real position) = Are pairs are matched "for sure"
  Non-binary are better than binaries algorithms

Table 3. The time consumption of algorithms, tested in CPU (i7-4930K) with 16GB memory. Best viewed in colors or fonts.

Sequences	Features	Results			
		Feature Numbers	Extraction Time (ms)	NN+RATIO+RANSAC (ms)	NN+GMS+RANSAC (ms)
01-03	SIFT	1196.66	147.25	41.07	**40.57**
	SURF	1410.23	90.86	**51.15**	53.81
	ORB	3927.09	30.69	**102.46**	112.88
	AKAZE	952.07	125.89	**18.34**	23.17
06-08	BRISK	1314.57	22.24	**15.42**	20.90
	KAZE	1130.88	186.48	**24.30**	25.16
04	SIFT	2717.55	142.26	58.43	**58.23**
	SURF	2922.41	94.34	64.99	**61.47**
	ORB	10645.30	46.46	**303.35**	333.75
	AKAZE	1964.16	118.82	**26.53**	35.23
	BRISK	3787.43	61.29	**54.65**	64.18
	KAZE	2303.47	260.85	**47.07**	52.86
05	SIFT	8555.45	1582.86	277.17	**195.97**
	SURF	23631.80	1302.59	647.83	**477.72**
	ORB	29792.90	182.62	2330.34	**2250.56**
	AKAZE	9920.55	668.57	461.79	**448.07**
	BRISK	7720.12	159.87	427.55	**285.94**
	KAZE	8613.15	3680.88	**364.56**	375.489

Figure 5.2: Benchmarking of SIFT, SURF, ORB, AKAZE, BRISK, KAZE with computation time. Ordered best time from best to worst : red, green, blue, black. [Bian et al.,]

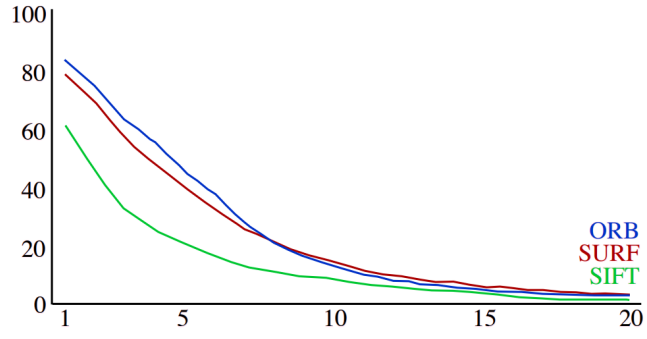
Table 2. The score of robustness, accuracy, and sufficiency exported from results on rotation cases. Best viewed in colors.

Matchers	Fig 3(a)			Fig 3(b)			Fig 3(c)			Fig 3(d)		
	RS	AS	SS	RS	AS	SS	RS	AS	SS	RS	AS	SS
SIFT-RATIO	0.970	0.977	182	1	1	580	193	0.912	1261	4050	0.808	144
SIFT-GMS	0.746	0.946	170	0.998	0.999	466	178	0.910	1023	1446	0.741	145
SURF-RATIO	0.991	0.981	339	1	0.999	723	190	0.9	3433	3615	0.777	229
SURF-GMS	0.970	0.962	383	1	1	795	201	0.950	3094	2143	0.747	267
ORB-RATIO	0.962	0.955	621	0.999	0.999	1770	181	0.845	2903	3924	0.756	348
ORB-GMS	0.978	0.953	988	0.999	1	3083	191	0.911	3689	3535	0.694	788
AKAZE-RATIO	0.888	0.962	280	0.996	1	533	180	0.889	1327	3668	0.773	153
AKAZE-GMS	0.916	0.946	289	0.997	0.999	648	168	0.887	1713	2515	0.728	238
BRISK-RATIO	0.921	0.936	214	0.999	1	821	173	0.861	917	3494	0.772	134
BRISK-GMS	0.915	0.910	268	1	1	1157	152	0.842	893	2191	0.683	251
KAZE-RATIO	0.981	0.978	306	1	1	675	190	0.874	1449	4529	0.787	182
KAZE-GMS	0.972	0.955	313	0.999	0.999	762	178	0.910	1661	3010	0.736	247

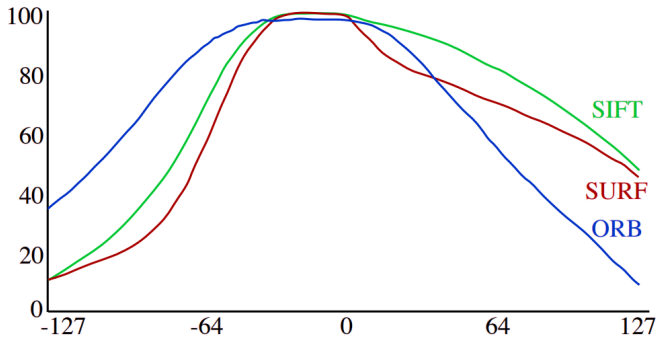
Figure 5.3: Benchmarking of SIFT, SURF, ORB, AKAZE, BRISK, KAZE on robustness (RS), accuracy (AS), sufficiency (SS). Ordered best time from best to worst : red, green, blue, black. [Bian et al.,]

- **Sufficiency**
  Mean number of correctly geometric matched pairs.
  ORB-GMS is the best.
- **Efficiency**
  Feature detection time + Feature matching time.
  ORB and BRISK are the fastest, KASE the slowest.

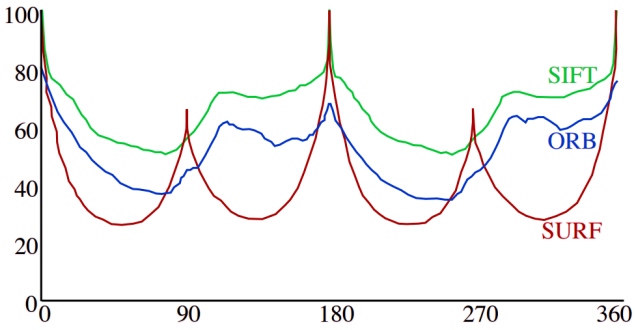
Some more comparisons are performed in [Key, 2014] about robustness of different algorithms.



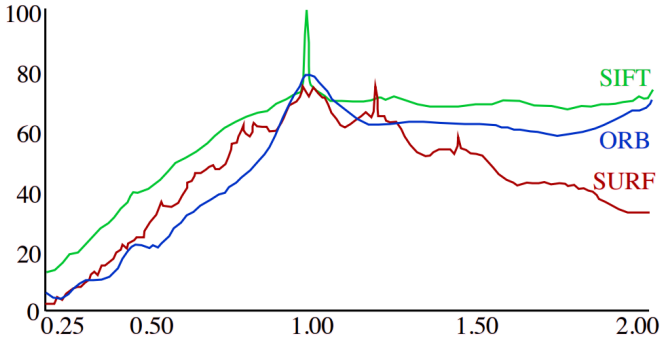
(a) Blurring repeatability (parameter)



(b) Brightness repeatability (offset)



(c) Rotation repeatability (angle)



(d) Scaling repeatability (factor)

Figure 5.4: Robustness (nb of matches/nb points in original image in %) comparison from [Key, 2014]

5.2 Non-binary features

5.2.1 SIFT- Scale Invariant Feature Transform

From the original paper [Lowe, 2004] and a concise explanation from [Int, 2014] 3x less fast than Harris Detector

SIFT detects scale-invariant key points by finding local extrema in the difference-of-Gaussian (DoG) space. [Li et al., 2018]

Each key point is described by a 128-dimensional gradient orientation histogram. Subsequently, all SIFT descriptors are modeled/quantized using a bag-of-words (BoW). The feature vector of each image is computed by counting the frequency of the generated visual words in the image.

Interesting "usability" notices are presented in [Suri et al., 2010], as skiping first octave features, ...

Con

- Patented algorithm and not included in OpenCV (only non-free module)
- Slow (HOW MUCH TO CHECK)

Steps of the algorithm

1. Extrema detection
   Uses an approximation of LoG (Laplacian of Gaussian), as a Difference of Gaussion, made from difference of Gaussian blurring of an image at different level of a Gaussian Pyramid of the image. Kept keypoints are local extrema in the 2D plan, as well as in the blurring-pyramid plan.
2. Keypoint localization and filtering
   Two threesholds has to be set :
   - Contract Threeshold : Eliminate low contract keypoint (0.03 in original paper)
   - Edge Threeshold : Eliminate point with a curvature above the threeshold, that could match edge only. (10 in original paper)
3. Orientation assignement
   Use an orientation histogram with 36 bins covering 360 degrees, filled with gradient magnitude of given directions. Size of the windows on which the gradient is calculated is linked to the scale at which it's calculated. The average direction of the peaks above 80% of the highest peak is considered to calculate the orientation.
4. Keypoint descriptors
   A 16x16 neighbourhood around the keypoint is divided into 16 sub-blocks of 4x4 size, each has a 8 bin orientation histogram. 128 bin are available for each Keypoint, represented in a vector of float, so 512 bytes per keypoint. Some tricks are applied versus illumination changes, rotation.
5. Keypoint Matching
   Two distance calculation :
   - Finding nearest neighboor.
   - Ratio of closest distance to second closest is taken as a second indicator when second closest match is very near to the first. Has to be above 0.8 (original paper) (TO CHECK what IT MEANS)

5.2.2 SIFT-FLOW

From [Freeman et al., 2010], realize in motion prediction from a single image, motion synthesis via object transfer and face recognition.

Implementation SIFT Flow (modified version of SIFT) C++ [SIF,] at `http://people.csail.mit.edu/celiu/SIFTflow/`

5.2.3 Root-SIFT

From [Arandjelović and Zisserman, 2012]
Better performances as SIFT, but no implementation found.

5.2.4 SURF – Speeded-Up Robust Features

[Bay et al., 2006] Use the BoWs to generate local features.

Pro

- Faster than SIFT (x3) : Parralelization, integral image ..
- Tradeoffs can be made :
  - Faster : no more rotation invariant, lower precision (dimension of vectors)
  - More precision : **extended** precision (dimension of vectors)
- Good for blurring, rotation

Con

- Patented algorithm
- Not good for illumination change, viewpoint change

Steps of the algorithm

1. Extrema detection
   Approximates Laplacian of Guassian with a Box Filter. Computation can be made in parrallel at different scales at the same time, can use integral images ... Roughly, does not use a gaussian approximation, but a true "square box" for edge detection, for example.
   The sign of the Laplacian (Trace of the Hessian) give the "direction" of the contrast : black to white or white to black. So a negative picture can match with the original ? (TO CHECK)
2. Keypoint localization and filtering
3. Orientation assignement
   Dominant orientation is computed with wavlet responses with a sliding window of 60°
4. Keypoint descriptors
   Neighbourhood of size 20sX20s is taken around the keypoint, divided in 4x4 subregions. Wavelet response of each subregion is computed and stored in a 64 dimensions vector (float, so 256 bytes), in total. This dimension can be lowered (less precise, less time) or extended (e.g. 128 bits ; more precise, more time)
5. Keypoint Matching

5.2.5 U-SURF – Upright-SURF

Rotation invariance can be "desactivated" for faster results, by bypassing the main orientation finding, and is robust up to 15°rotation.

5.2.6 GSSIS - Generalized Scale-Space Interest Points

From [Ima, 2015], generalized interest point, with colors exension, of SIFT and SURF.

Roughly : uses more complicated way of generating local interest points.

Pro

- Scale-invariant

5.2.7 LBP - Local Binary Pattern

From [Li et al., 2018], use the BoWs to generate local features

5.3 Binary features

5.3.1 ORB – Oriented FAST and Rotated BRIEF

From [Rublee et al., 2011] which is rougly a fusion of FAST and BRIEF. See also [ORB, 2014]

Pro

- Not patented

Steps of the algorithm

1. Extrema detection
   FAST algorithm (no orientation)
2. Keypoint localization and filtering
   Harris Corner measure : find top N keypoints
   Pyramid to produce multi scale features
3. Orientation assignement
   The direction is extracted from the orientation of the (center of the patch) to the (intensity-weighted centroid fo the patch). The region/patch is circular to improve orientation invariance.
4. Keypoint descriptors
   R-BRIEF is used, as Brief Algorithm is bad at rotation, on rotated patches of pixel, by rotating it accordingly with the previous orientation assignement.
5. Keypoint Matching
   Multi-probe LSH (improved version of LSH)

Option explanation Mostly from [BFM, 2015], [Jav,] :

To be clear :

- **Target picture** = request picture = new picture = picture we want to find correlation on
- **Candidate picture** = stored picture = old pictures = pictures already parsed, prepared and stored, to find correlation on.

Perform a feature matching :

- **Brute-Force matcher**, with bf.match(), is simple. It takes the descriptor of one feature of the target picture and is evaluated with all other features of all candidate pictures, using some distance calculation. The closest one is returned. It returns a list of matches. Use crossCheck=True.
- **KnnMatch**, with bf.knnMatch(), returns for each descriptor of the target picture, the n-best matches with descriptors of a candidate picture. It searches for N-best candidate for each descriptor. It returns list of a list of matches. Use crossCheck=False and then do the ratio test.

Remove outliers and bad matches :

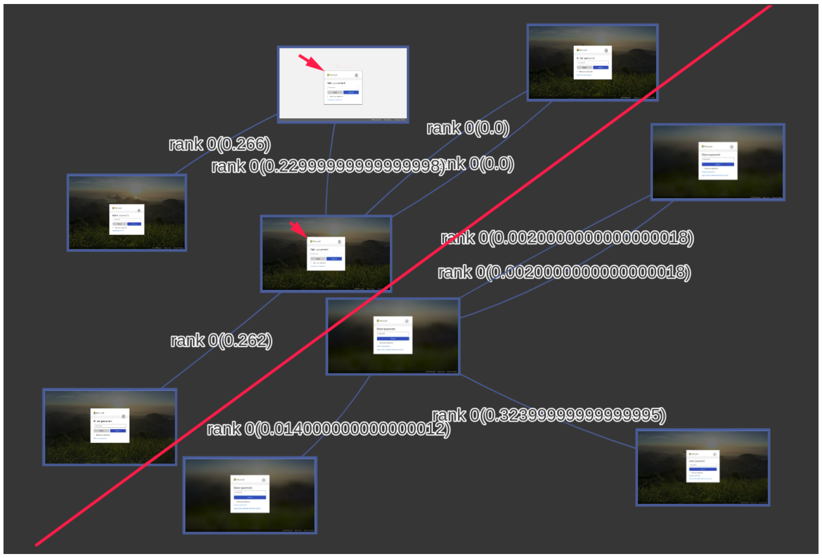
- **CrossCheck** is an alternative to the ratio test. Cross-check does matching of two sets of descriptors D1 and D2 in both directions (D1 to D2 and D2 to D1) retaining matches that exists in both.

- A **ratio test** can be performed on each k-uplets. Repetitive patterns are detected if the distance between one descriptor of the target picture is the same with the two best descriptors of the candidate picture. 2 points on the candidate picture matched 1 point on the target picture.

RANSAC filter matches. (TO CHECK)
Print nice graphs :

- **cv2.drawKeypoints()** to draw keypoints
- **cv2.drawMatches()** helps us to draw the matches. It stacks two images horizontally and draw lines from first image to second image showing best matches.
- **cv2.drawMatchesKnn** draws all the k best matches. If k=2, it will draw two match-lines for each keypoint.

Time MIN-version : nobs : 207 min time : 0.00022s max time : 1.22806s mean :0.78938s variance : 0.06399s skewness : -2.6371s kurtosis : 5.44295
MAX-version : nobs : 207 min time : 0.00022s max time : 1.89294s mean :0.81612s variance : 0.0911s skewness : -1.02273s kurtosis : 3.77603
RATIO-version : nobs : 207 min time : 0.00022s max time : 2.25978s mean :0.92306s variance : 0.14716s skewness : -0.13801s kurtosis : 2.98186



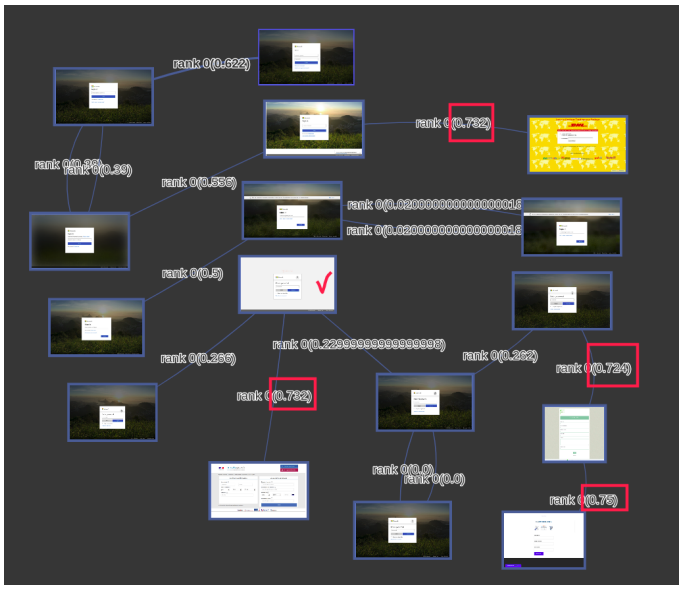
(a) Good Microsoft matching despite color change.

(b) Attractor problem : an image with a low descriptors number will "attract" a lot of other pictures

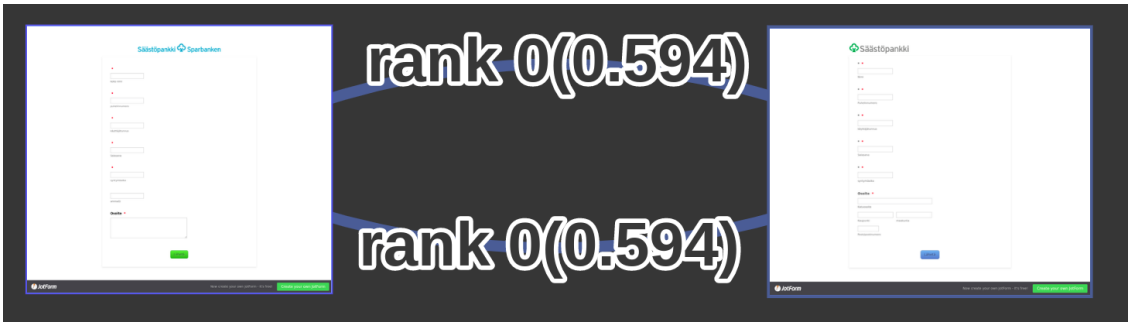
(c) No descriptor problem : pictures without descriptor should be handled differently (here : fixed distance, still matching to the last evaluated picture)

Figure 5.5: Results - ORB - min version

(a) Good brand matching even with different backgrounds. Logo is present on each picture



(b) Good matching on brand even with disturbed background

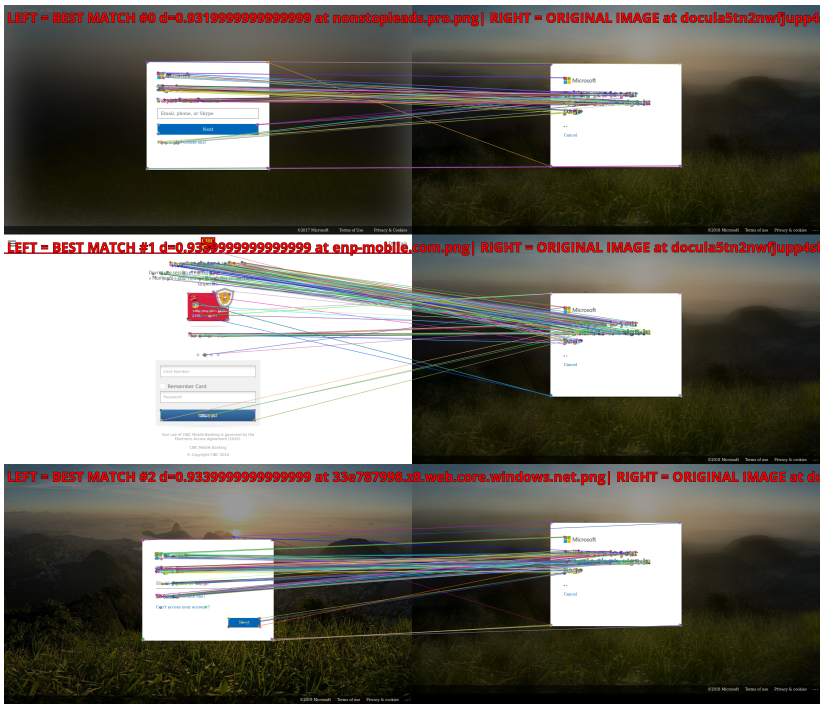


(c) Good matching on brand

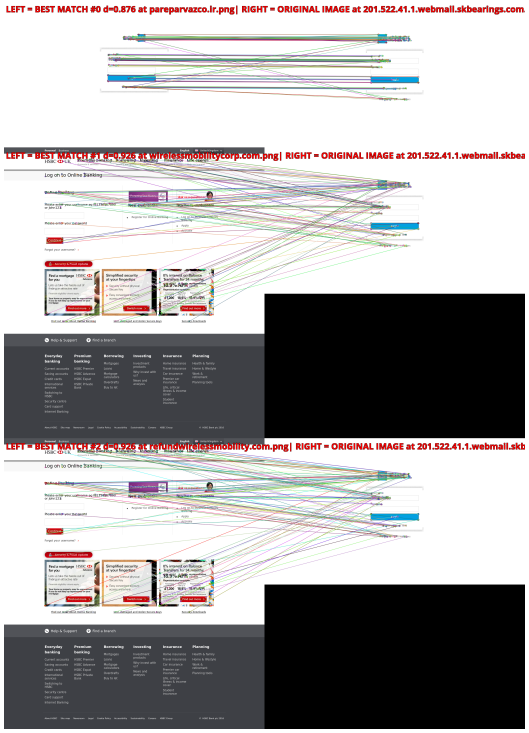
Figure 5.6: Results - ORB - max version

Few tips to analyze following pictures :

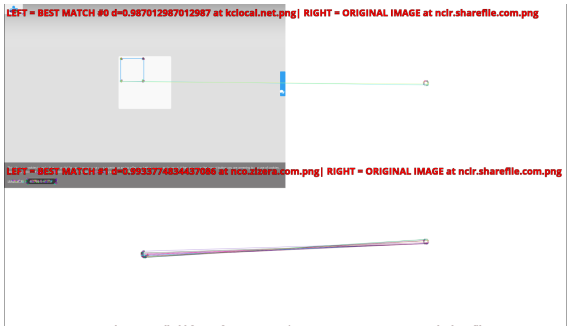
- **Parrallel lines** (if there is not rotation) are a indicator of quality matching. It keeps the spatial consistency between source and candidate pictures.
- **Text** seems to be a problem. Letters are matched to letters, generating false positive. It also "uses" descriptor space (number of descriptors is artificially limited), and so, hinders true logo (for example) to be described and used.



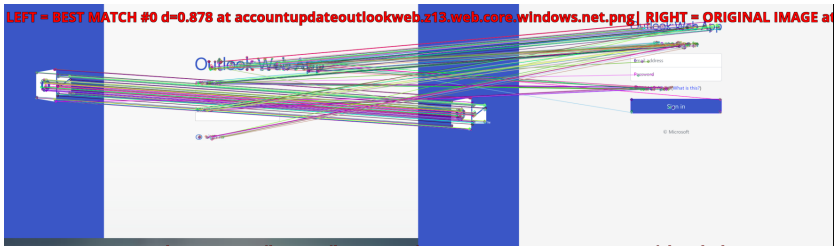
(a) Good first match, bad second match, good third match. Bad overall ranking. Text matching problem.



(b) Good and clear first match

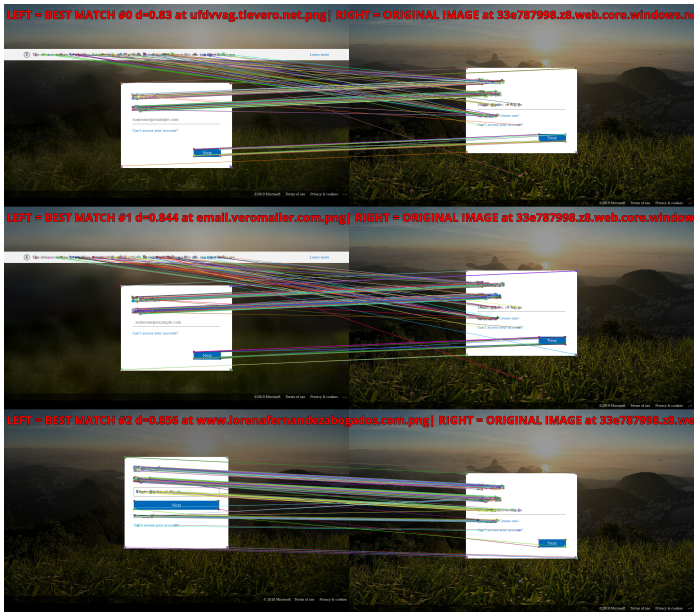


(c) True edge case : a header on a "almost-white" page does not match with the original "almost white-page"

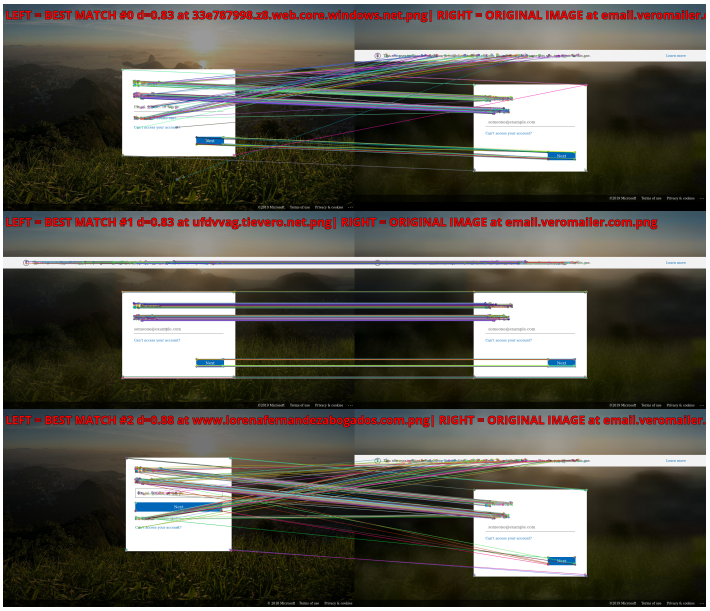


(d) Good matching on brand even with disturbed background

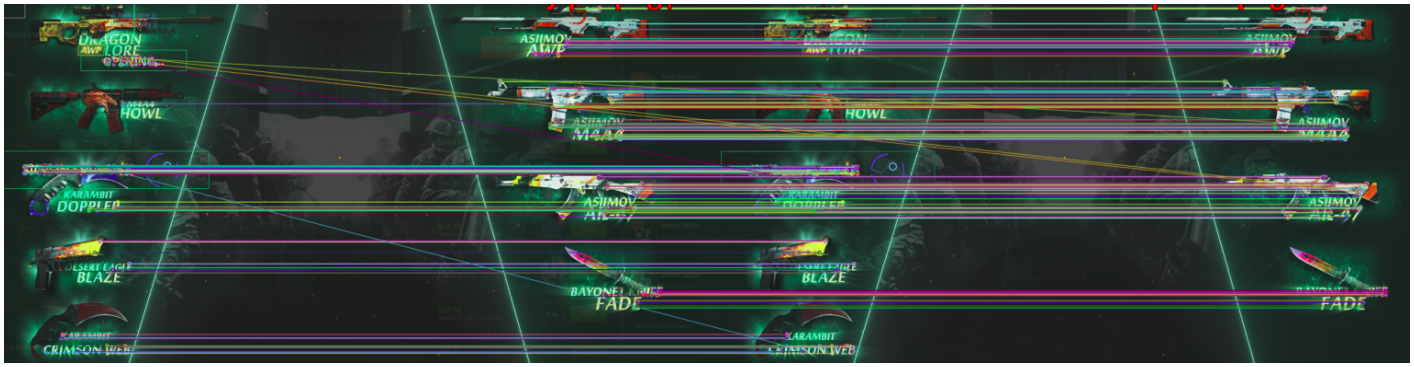
Figure 5.7: Results - ORB - ratio



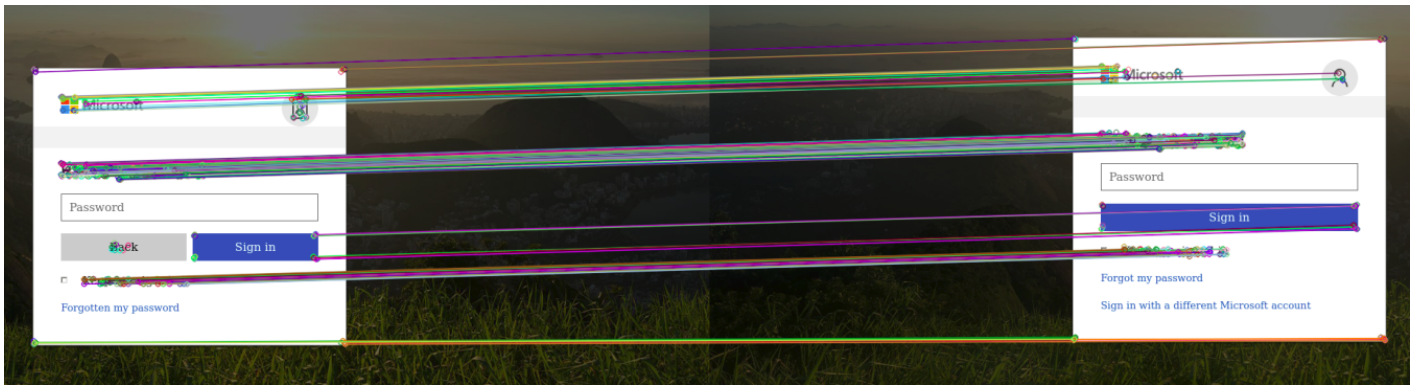
(a) Good parrallel matching on a microsoft form



(b) Not-the-best first matching. A white header with text is creating noise/mismatches



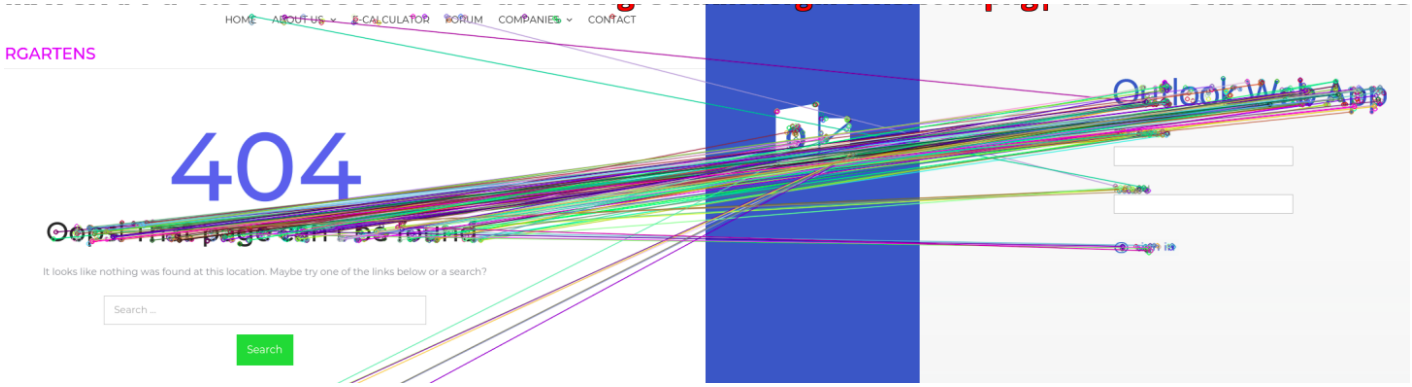
(c) Specific Typo and pictures allow a good matching



(d) Perfect match on Microsoft form

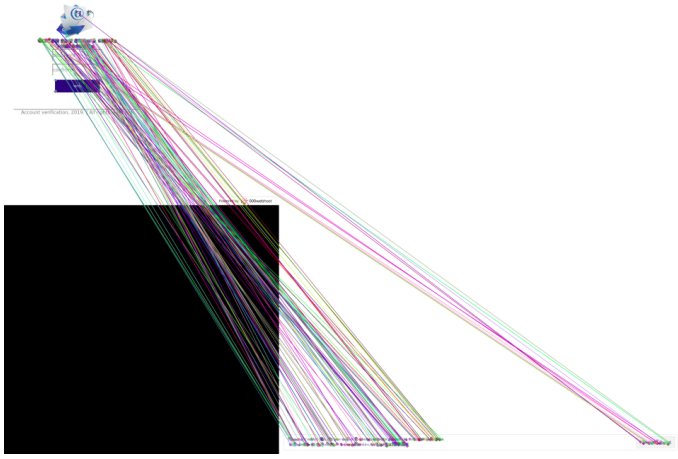
Figure 5.8: Results - ORB - ratio

(a) Logo are matching, form text is mismatching with title text



(b) Mismatch due to text and font matching

(c) Glaring mismatch due to long text



(d) Glaring mismatch due to text only presence

Figure 5.9: Results - ORB - ratio

5.3.2 BRISK -

5.3.3 AKASE -

5.4 Unsorted

5.4.1 PSO

From ... few words in [Nurnajmin Qasrina et al., 2018]

5.4.2 SKF

From [Nurnajmin Qasrina et al., 2018]
Faster than PSO.

5.4.3 RPM - Robust Point Matching

From ... Few words in [Yang et al., 2014] Unidirectional matching approach. Does not "check back" if a matching is correct. Seems to achieve only the transformation (geometry matching) part.

5.4.4 BRIEF – Binary Robust Independent Elementary Features

Extract binary strings equivalent to a descriptor without having to create a descriptor
See BRIEF [BRI,]

Pro

- Solve memory problem

Con

- Only a keypoint descriptor method, not a keypoint finder
- Bad for large in-plan rotation

Steps of the algorithm

1. Extrema detection
2. Keypoint localization and filtering
3. Orientation assignement
4. Keypoint descriptors
   Compare pairs of points of an image, to directly create a bitstring of size 128, 256 ou 512 bits. (16 to 64 bytes)
   Each bit-feature (bitstring) has a large variance ad a mean near 0.5 (TO VERIFY). The more variance it has, more distinctive it is, the better it is.
5. Keypoint Matching Hamming distance can be used on bitstrings.

5.4.5 R-BRIEF – Rotation (?) BRIEF

Variance and mean of a bit-feature (bitstring) is lost if the direction of keypoint is aligned (TO VERIFY : would this mean that there is a preferential direction in the pair of point selection ?)
Uncorrelated tests (TO CHECK WHAT IT IS) are selected to ensure a high variance.

5.4.6 CenSurE

5.4.7 KASE -

Shipped in OpenCV library. Example can be found at [Nikishaev, 2018]

Steps of the algorithm

1. Extrema detection
2. Keypoint localization and filtering
3. Orientation assignement
4. Keypoint descriptors
5. Keypoint Matching

5.4.8 Delaunay Graph Matching

Algorithm from 2012, quite advanced. Would need some tests or/and review See M1NN [Fang, 2012] that is presenting 3 algorithms :

- **M1NN Agglomerative Clustering**

Different types of data,robust to noise, may 'over' cluster. Better clustering performance and is extendable to many applications, e.g. data mining, image segmentation and manifolding learning.

- **Modified Log-likelihood Clustering**

Measure and compare clusterings quantitatively and accurately. Energy of a graph to measure the complexity of a clustering.

- **Delaunay Graph Characterization and Graph-Based Image Matching**

Based on diffusion process and Delaunay graph characterization, with critical time. Graph-based image matching method. SIFT descriptors also used. Patent problem ? Outperforms SIFT matching method by a lower error rate.

Pro

- Lower error
- Extensible to 3D (but not done yet ?)

Con

- Lower number of matches

5.4.9 Fast Spectral Ranking

From [Iscen et al., 2018] Seems to have quite fast result, ranking algorithm. Still dark areas over the explanations.

5.4.10 GHM - Generalized Hierarchical Matching Framework

From [Chen et al., 2012] Roughly, the algorithm split the input picture into interest areas, and then do matching on these different areas.

This tries to achieve a object-oriented recognition. It uses Saliency Map.

This (TO CHECK) is a non-rectangle version of SPM.

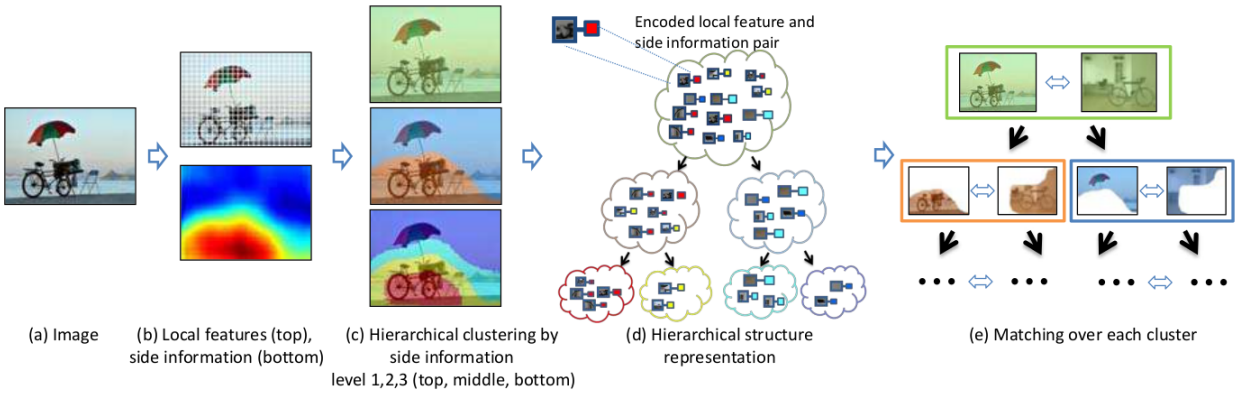


Figure 5.10: Hierarchical Hashing as showed in [Chen et al., 2012]

Steps of the algorithm

1. Multiple scale detection is performed in each image and the obtained multi-scale scores are averaged to get final single object confidence map.

Chapter 6

Neural networks – Black box algorithms

See [Li et al., 2018] to get a larger overview of deeplearning capabilities, applied to a particular field.

6.1 FAST – Features from Accelerated Segment Test

From [FAS, 2014] the algorithm is mainly Machine Learning, but as far as I can see, there is no direct need of machine learning in the algorithm, but for speed.

It seems that the selection of candidate pixel, and the selection of a threeshold is holded by Machine Learning. It also seems, that "mostly brighter", "similar" and "mostly darker" pixels are used to feed a decision tree (ID3 algorithm - decision tree classifier) to allow a fast recognition of a corner.

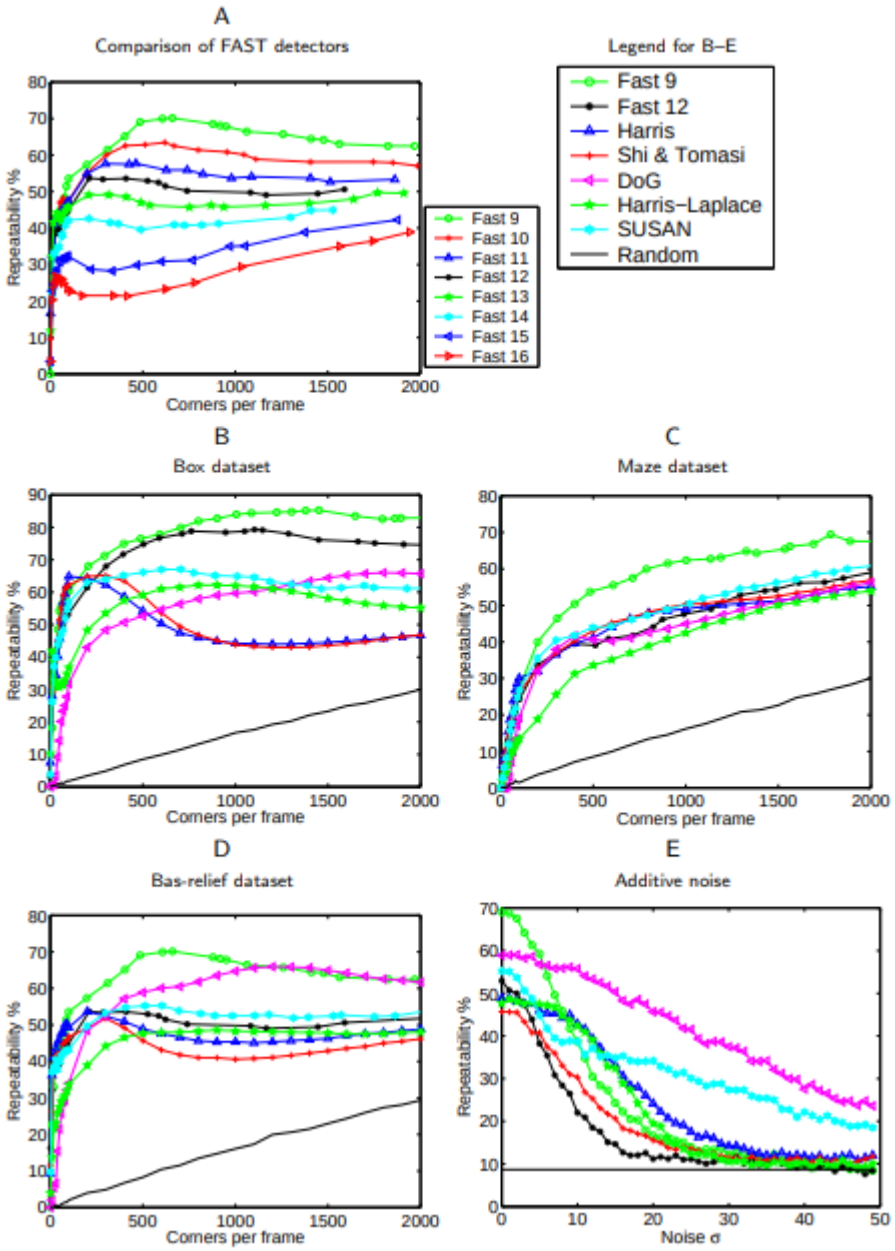


Figure 6.1: Corner detector from [Rosten and Drummond, 2006]

Pro

- "High performance" (HOW MUCH, TO CHECK)

Con

- "Too" sensitive if n¡12 : increase in false-positive
- Many calculation just to "throw away" a pixel.
- Many True-postive around the same position
- Not robust to high levels of noise
- Dependant on a threshold

Steps of the algorithm

1. Extrema detection For each pixel, select a cicle-patch (not disk-patch, not a surface!) of 16 pixels around it. The pixel is a corner if there is n (n=12) contiguous pixels parts of the circle, which are all brighter or all darker than the center-pixel.
   It's easy to remove all "not-corner" points, by checking only few (1, 9, 5 and 13) pixels of the circle.
2. Keypoint localization and filtering
3. Orientation assignement
4. Keypoint descriptors
5. Keypoint Matching

6.2 CNN - Convolutional Neural Network

From ... [Gionis et al.,]

6.3 FRCNN - Faster RCNN

From ... [Sun et al., 2018] Mainly for faces detection.

Pro

- M

6.4 RTSVMs - Robust Transductive Support Vector Machines

From [Gionis et al.,] Seems to scale very well (> 1 Million data)
Uses a hashing method, binary hierarchical trees and TSVM classifier.

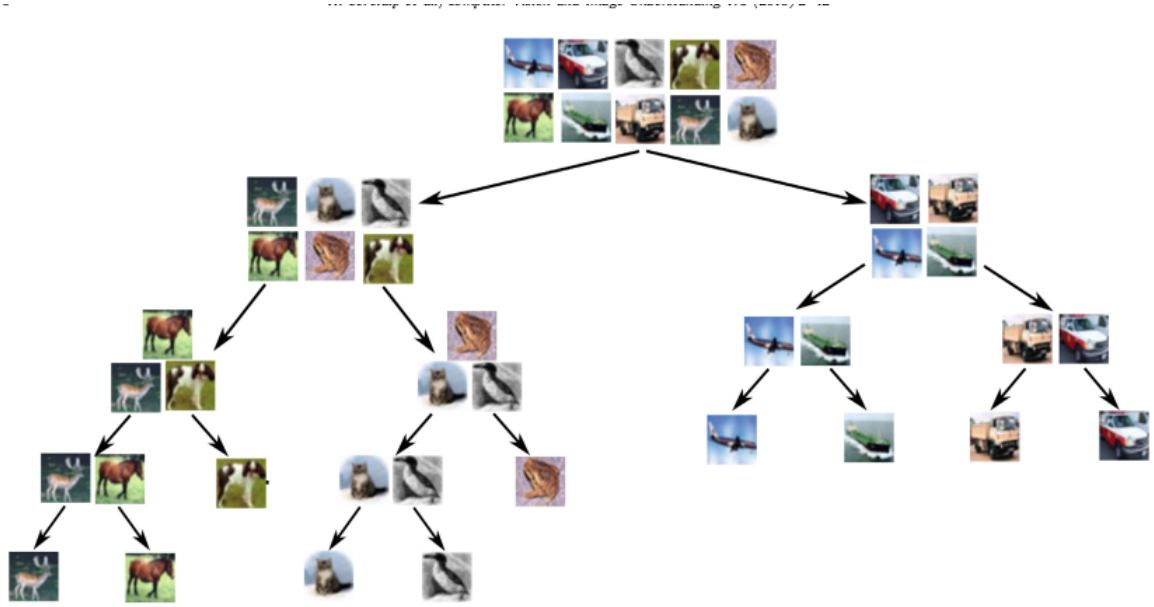


Fig. 2. Binary hierarchical tree obtained for CIFAR10 dataset using convex hull modeling of the classes. Each image represents an object class where it comes from.

Figure 6.2: Biary hierarchical tree from [Gionis et al.,]

6.5 RBM - Restricted Boltzmann machine

From ... A word is given in [Spe,]

To learn 32 bits, the middle layer of the autoencoder has 32 hidden units Neighborhood Components Analysis (NCA) objective function = refine the weights in the network to preserve the neighborhood structure of the input space.

Pro

- More compact outputs code of picture than E2LSH = Better performances

6.6 RPA - Robust Projection Algorith

From ... [Igor, 2011]

6.7 Boosting SSC

From ... A word is given in [Spe,]

Pro

- Better than E2LSH

Con

- Worst than RBM

6.8 ConvNet - Convolutional Neural Networks

Learn a metric between any given two images. The distance can be threesholded to decide if images match or not.

Training phase

Goal :

- Minimizing distance between "same image" examples
- Maximizing distance between "not same image" examples

Evaluation phase

Apply an automatic threeshold.

SVM - Support Vector Machine

Chapter 7

Utility algorithms

7.1 SWS - Sliding Windows Search

From ... [Yu, 2011] A bounding box is sliding on the picture, and an objet-existence score in the bounding box is computed for each position, and each rectangle size.

Pro

- B

Con

- Too complex ! $O(N^4)$ windows to evaluate, with N = resolution on one axis of the picture

Heuristics can be used to reduce the expected complexity of the algorithm. The picture is reduced in size, with a constant size bounding box, to find objects at different scales. These heuristics may miss objects.

7.2 ESS - Efficient Subwindow Search

From [Yu, 2011] Based on a branch-and-bound algorithm. The algorithm does not evaluate all subrectangle of rectangle with a low evaluation of the best chance they have to contain an object.

Pro

- Sublinear to number of pixels. (below $O(N)$)

7.3 SLICO - Simple Linear Iterative Clustering

Cluster a picture into smaller chunks. For example, used in [Raj and Joseph, 2016] for Copy Move detection.

7.4 HSNW - ... indexing

From ... A word in [Douze et al., 2018]

Bibliography

[Blo,] Blockhash. http://blockhash.io/.

[BRI,] BRIEF (Binary Robust Independent Elementary Features) — OpenCV 3.0.0-dev documentation. https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_brief/py_brief.html#brief.

[Fea,] Feature Matching + Homography to find Objects — OpenCV 3.0.0-dev documentation. https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_feature_homography/py_feature_homograph feature-homography.

[Jav,] Java OpenCV - extracting good matches from knnMatch. https://stackoverflow.com/questions/35428440/java-opencv-extracting-good-matches-from-knnmatch.

[Ope,] OpenCV: Feature Matching. https://docs.opencv.org/3.4.3/dc/dc3/tutorial_py_matcher.html.

[SIF,] SIFT Flow: Dense Correspondence across Scenes and its Applications. http://people.csail.mit.edu/celiu/SIFTflow/.

[Spe,] Spectralhashing.pdf. http://people.csail.mit.edu/torralba/publications/spectralhashing.pdf.

[Tes,] Testing different image hash functions – Content Blockchain.

[Loo, 2011] (2011). Looks Like It - The Hacker Factor Blog. http://www.hackerfactor.com/blog/index.php?/archives/432-Looks-Like-It.html.

[PHa, 2011] (2011). pHash-like image hash for java. https://pastebin.com/Pj9d8jt5.

[PHa, 2013] (2013). pHash.org: Home of pHash, the open source perceptual hash library. http://www.phash.org/.

[FAS, 2014] (2014). FAST Algorithm for Corner Detection — OpenCV 3.0.0-dev documentation. https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_fast/py_fast.html.

[Int, 2014] (2014). Introduction to SIFT (Scale-Invariant Feature Transform) — OpenCV 3.0.0-dev documentation. https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_sift_intro/py_sift_intro.html#sift-intro.

[Key, 2014] (2014). Keypoints and Descriptors 1. https://www.cs.auckland.ac.nz/~rklette/CCV-Dalian/pdfs/E02_Features.pdf.

[ORB, 2014] (2014). ORB (Oriented FAST and Rotated BRIEF) — OpenCV 3.0.0-dev documentation. https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_orb/py_orb.html#orb.

[BFM, 2015] (2015). BFMatcher raises error for Python interface when crossCheck option is enabled · Issue #46 · MasteringOpenCV/code. https://github.com/MasteringOpenCV/code/issues/46.

[Ima, 2015] (2015). Image Matching Using Generalized Scale-Space Interest Points.

[Fuz, 2019] (2019). Fuzzy hashing API and fuzzy hashing tool. Contribute to ssdeep-project/ssdeep development by creating an account on GitHub. ssdeep-project.

[Lib, 2019] (2019). A library for efficient similarity search and clustering of dense vectors.: Facebookresearch/faiss. Facebook Research.

[Non, 2019] (2019). Non-Metric Space Library (NMSLIB): An efficient similarity search library and a toolkit for evaluation of k-NN methods for generic non-metric spaces.: Nmslib/nmslib. nmslib.

[Aluigi, 2019] Aluigi, V. (2019). JavaScript implementation of the Average Hash using HTML5 Canvas.

[Arandjelović and Zisserman, 2012] Arandjelović, R. and Zisserman, A. (2012). Three things everyone should know to improve object retrieval. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2911–2918.

[Bay et al., 2006] Bay, H., Tuytelaars, T., and Van Gool, L. (2006). SURF: Speeded Up Robust Features. In Leonardis, A., Bischof, H., and Pinz, A., editors, *Computer Vision – ECCV 2006*, volume 3951, pages 404–417. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Bian et al.,] Bian, J., Zhang, L., Liu, Y., Lin, W.-Y., Cheng, M.-M., and Reid, I. D. Image Matching: An Application-oriented Benchmark. page 11.

[Boytsov and Naidan, 2013] Boytsov, L. and Naidan, B. (2013). Engineering Efficient and Effective Non-metric Space Library. In Hutchison, D., Kanade, T., Kittler, J., Kleinberg, J. M., Mattern, F., Mitchell, J. C., Naor, M., Nierstrasz, O., Pandu Rangan, C., Steffen, B., Sudan, M., Terzopoulos, D., Tygar, D., Vardi, M. Y., Weikum, G., Brisaboa, N., Pedreira, O., and Zezula, P., editors, *Similarity Search and Applications*, volume 8199, pages 280–293. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Bupe, 2017] Bupe, C. (2017). What algorithms can detect if two images/objects are similar or not? - Quora. https://www.quora.com/What-algorithms-can-detect-if-two-images-objects-are-similar-or-not.

[Cevikalp et al., 2018] Cevikalp, H., Elmas, M., and Ozkan, S. (2018). Large-scale image retrieval using transductive support vector machines. *Computer Vision and Image Understanding*, 173:2–12.

[Chen et al., 2012] Chen, Q., Song, Z., Hua, Y., Huang, Z., and Yan, S. (2012). Hierarchical matching with side information for image classification.

[Douze et al., 2018] Douze, M., Sablayrolles, A., and Jegou, H. (2018). Link and Code: Fast Indexing with Graphs and Compact Regression Codes. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3646–3654, Salt Lake City, UT. IEEE.

[Fang, 2012] Fang, Y. (2012). Data Clustering and Graph-Based Image Matching Methods.

[Freeman et al., 2010] Freeman, W. T., Torralba, A., Yuen, J., and Liu, C. (2010). SIFT Flow: Dense Correspondence across Scenes and its Applications.

[Gionis et al.,] Gionis, A., Indyk, P., and Motwani, R. Similarity Search in High Dimensions via Hashing. page 12.

[Hahn, 2019] Hahn, N. (2019). Differentiate images in python: Get a ratio or percentage difference, and generate a diff image - nicolashahn/diffimg.

[Harris and Stephens, 1988] Harris, C. and Stephens, M. (1988). A Combined Corner and Edge Detector. In *Procedings of the Alvey Vision Conference 1988*, pages 23.1–23.6, Manchester. Alvey Vision Club.

[Igor, 2011] Igor (2011). Nuit Blanche: Are Perceptual Hashes an instance of Compressive Sensing ?

[Iscen et al., 2018] Iscen, A., Avrithis, Y., Tolias, G., Furon, T., and Chum, O. (2018). Fast Spectral Ranking for Similarity Search. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7632–7641, Salt Lake City, UT. IEEE.

[Kornblum, 2006] Kornblum, J. (2006). Identifying almost identical files using context triggered piecewise hashing. *Digital Investigation*, 3:91–97.

[Li et al., 2018] Li, Z., Zhang, X., Müller, H., and Zhang, S. (2018). Large-scale retrieval for medical image analytics: A comprehensive review. *Medical Image Analysis*, 43:66–84.

[Lowe, 2004] Lowe, D. G. (2004). Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60(2):91–110.

[Manku et al., 2007] Manku, G. S., Jain, A., and Das Sarma, A. (2007). Detecting near-duplicates for web crawling. In *Proceedings of the 16th International Conference on World Wide Web - WWW '07*, page 141, Banff, Alberta, Canada. ACM Press.

[Nikishaev, 2018] Nikishaev, A. (2018). Feature extraction and similar image search with OpenCV for newbies.

[Nurnajmin Qasrina et al., 2018] Nurnajmin Qasrina, A., Pebrianti, D., Zuwairie, I., Luhur, B., and Mohd Falfazli, M. J. (2018). Image Template Matching Based on Simulated Kalman Filter (SKF) Algorithm.

[Oliva and Torralba,] Oliva, A. and Torralba, A. Modeling the Shape of the Scene: A Holistic Representation of the Spatial Envelope. page 31.

[Oliver et al., 2013] Oliver, J., Cheng, C., and Chen, Y. (2013). TLSH – A Locality Sensitive Hash. In *2013 Fourth Cybercrime and Trustworthy Computing Workshop*, pages 7–13, Sydney NSW, Australia. IEEE.

[Raj and Joseph, 2016] Raj, R. and Joseph, N. (2016). Keypoint Extraction Using SURF Algorithm for CMFD.

[Rosten and Drummond, 2006] Rosten, E. and Drummond, T. (2006). Machine Learning for High-Speed Corner Detection. In Leonardis, A., Bischof, H., and Pinz, A., editors, *Computer Vision – ECCV 2006*, volume 3951, pages 430–443. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Rublee et al., 2011] Rublee, E., Rabaud, V., Konolige, K., and Bradski, G. (2011). ORB: An efficient alternative to SIFT or SURF. In *2011 International Conference on Computer Vision*, pages 2564–2571, Barcelona, Spain. IEEE.

[Sarantinos et al., 2016] Sarantinos, N., Benzaid, C., Arabiat, O., and Al-Nemrat, A. (2016). Forensic Malware Analysis: The Value of Fuzzy Hashing Algorithms in Identifying Similarities. In *2016 IEEE Trustcom/BigDataSE/ISPA*, pages 1782–1787, Tianjin, China. IEEE.

[Song et al., 2018] Song, J., Gao, L., Liu, L., Zhu, X., and Sebe, N. (2018). Quantization-based hashing: A general framework for scalable image and video retrieval. *Pattern Recognition*, 75:175–187.

[Sun et al., 2018] Sun, X., Wu, P., and Hoi, S. C. (2018). Face detection using deep learning: An improved faster RCNN approach. *Neurocomputing*, 299:42–50.

[Suri et al., 2010] Suri, S., Schwind, P., Uhl, J., and Reinartz, P. (2010). Modifications in the SIFT operator for effective SAR image matching.

[Yang et al., 2014] Yang, X., Pei, J., and Shi, J. (2014). Inverse consistent non-rigid image registration based on robust point set matching.

[Yu, 2011] Yu, P. (2011). Image classification using latent spatial pyramid matching.