State of The Art

Vincent FALCONIERI

February 2019 - August 2019

Contents

1 Introduction **4**

2 Classic Computer vision techniques - White box algorithms **5**

2.1 Classic computer vision algorithm steps 6

2.1.1 Step 1 - Interest / Key Point Detection 6

2.1.2 Step 2 - Descriptor Extraction 7

2.1.3 Step 3 - Feature representation 8

Bag-Of-Words or Bag-Of-Features or Codebook Generation 8

Soft Vector Quantization 8

Hierarchical CodeWords 8

Visual sentences 9

SPM - Spatial Pyramid Matching 9

L-SPM - Latent Spatial Pyramid Matching 9

2.1.4 Step 4 - Feature storage and datastructure 10

LSH - Locally Sensitive Hashing 10

BBF - Best bin first Kd-tree 10

2.1.5 Step 5 - Distance choice 11

Hamming distance / Bruteforce 11

FLANN - Fast Library for Approximate Nearest Neighboors 11

2.1.6 Step 6 - Matching and selection 12

RATIO (Lowe's) 12

GMS - Gird-based Motion Statistics 12

QBH - Quantization Based Hashing 12

2.1.7 Step 7 - Model Fitting 13

RANSAC - Random Sample Consensus 13

Least Meadian 13

3 Global Features Algorithms **14**

3.1 Full region features - FH or CTPH - Fuzzy Hashing Algorithms 15

3.1.1 A-HASH : Average Hash 17

3.1.2 B-HASH : Block Hash 21

3.1.3 D-HASH - Difference Hashing 22

3.1.4 P-HASH - Perceptual Hash 26

3.1.5 W-HASH - Wavelet Hash 31

3.1.6 SimHash - Charikar's simhash 34

3.1.7 R-HASH 35

3.1.8 Spectral-HASH 36

3.1.9 LSH - Locality Sensitve Hashing . . . 37
3.1.10 E2LSH - LSH - Locality Sensitve Hashing . . . 37
3.1.11 Nilsimsa hash - Locality sensitive hash . . . 37
3.1.12 TLSH - Trend Micro Locality Sensitive Hashing . . . 38
3.1.13 SSDeep - Similarity Digest . . . 42
3.1.14 SDHash - Similarity Digest Hash . . . 43
3.1.15 MVHash - Majority Vote Hash . . . 44
3.1.16 MRSH V2 - MultiResolution Similarity Hashing . . . 45
3.1.17 GIST - . . . 46
3.2 Per subregion features . . . 47
3.2.1 HOG - Histogram of Oriented Gradients . . . 47
3.2.2 Contrast context histogramm . . . 47

4 Local Features Algorithms 48
4.1 Comparison overview . . . 49
4.2 Feature detector . . . 53
Harris Detector . . . 53
GoodFeaturesToTrack . . . 53
CSS - Curvature Space Scale . . . 53
Hit and Miss filter . . . 53
Shi/Tomasi . . . 53
MSER . . . 54
SUSAN . . . 54
FAST - Features from Accelerated Segment Test . . . 54
Bag Of Feature . . . 54
4.3 Feature descriptor . . . 55
4.4 Feature detector and descriptor . . . 56
4.4.1 Unsorted . . . 56
BRISK - . . . 56
AKASE - . . . 56
CenSurE . . . 56
KASE - . . . 56
4.4.2 Non-binary features / Vector descriptor . . . 56
SIFT- Scale Invariant Feature Transform . . . 56
SIFT-FLOW . . . 59
Root-SIFT . . . 59
SURF – Speeded-Up Robust Features . . . 60
U-SURF – Upright-SURF . . . 60
TOP-SURF . . . 61
GSSIS - Generalized Scale-Space Interest Points . . . 61
LBP - Local Binary Pattern . . . 61
4.4.3 Binary features / Binary descriptor . . . 62
ORB – Oriented FAST and Rotated BRIEF . . . 62
BRIEF – Binary Robust Independent Elementary Features . . . 72
R-BRIEF – Rotation (?) BRIEF . . . 72
4.5 Feature matching . . . 73
PSO . . . 73

SKF . . . 73
RPM - Robust Point Matching . . . 73
Delaunay Graph Matching . . . 73
Fast Spectral Ranking . . . 73
GHM - Generalized Hierarchical Matching Framework . . . 74
4.6 Image matching . . . 75

5 Algorithms combination **76**
5.1 Phishing . . . 77
5.1.1 Visual similarity - Histogramm . . . 77
5.1.2 Visual similarity - Hash . . . 77
5.1.3 Visual similarity - Keypoints . . . 77
5.1.4 Visual similarity - Text handling . . . 79
Remove text : EAST - Efficient and Accurate Scene Text detection / deep learning text detector . . . 79
Remove text : Tesseract / Bundled text detector . . . 79

6 Neural networks – Black box algorithms **83**
6.1 FAST – Features from Accelerated Segment Test . . . 84
6.2 CNN - Convolutional Neural Network . . . 86
6.3 FRCNN - Faster RCNN . . . 87
6.4 RTSVMs - Robust Transductive Support Vector Machines . . . 88
6.5 RBM - Restricted Boltzmann machine . . . 89
6.6 RPA - Robust Projection Algorith . . . 90
6.7 Boosting SSC . . . 91
6.8 ConvNet - Convolutional Neural Networks . . . 92
6.9 SVM - Support Vector Machine . . . 92

7 Utility algorithms **93**
7.0.1 SWS - Sliding Windows Search . . . 93
7.0.2 ESS - Efficient Subwindow Search . . . 93
7.1 Segmentation . . . 93
7.1.1 SLICO - Simple Linear Iterative Clustering . . . 93
7.1.2 HSNW - ... indexing . . . 95
7.2 Unsorted . . . 96
7.2.1 Block-based approach + KeyPoint approach for Image manipulation . . . 96
7.2.2 Scalable cluster discovery . . . 96
7.3 Raw results . . . 97
7.3.1 Phishing dataset / Hashing based . . . 101

Chapter 1

Introduction

Goal The goal of this document is to provide an overview of the tools and ideas to cluster similar pictures or pictures with similar elements together. This document intents to provide State-of-the-art references, as well as library tests and results review. It may also give insights about combinaison or ideas to improve found ideas.

Methodology 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

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"

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/...)

Chapter 2

Classic Computer vision techniques - White box algorithms

Correspondances can be found between pictures, and so can be used for, e.g. : [Bian et al., 2017]:

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]

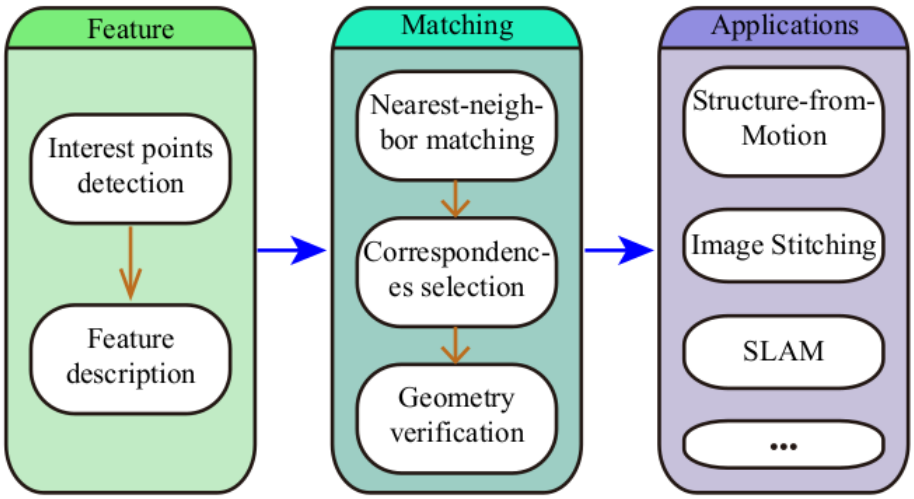


Figure 2.1: Image maching pipeline from [Bian et al., 2017]

Please note that some pictures are old results representation, and may differ from your own experiments. They are kept for their research interest and by lack of time to re-build same pictures for this only purpose.

2.1 Classic computer vision algorithm steps

2.1.1 Step 1 - Interest / Key Point Detection

The goal of this sub-algorithm is to find relevant points in pictures, to encode them later as a representation of the original picture. These **keypoints** are encoded under various format.

PLEASE SEE BELOW THE LIST OF FEATURE DETECTOR

2.1.2 Step 2 - Descriptor Extraction

The goal of this part is to extract a small patch around each keypoint, preserving the most relevant information and discaring unnecessary information (illumination ..)

Patches can be :

- Pixels values
- Histogram of gradient
- Learnt To check

And are usually :

- Normalized
- Indexed in a data structure for fast lookup

Example "Based on 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% chance of being one or zero, and that different bits are independent of each other

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

2.1.3 Step 3 - Feature representation

The goal of this step is to choose a relevant storage format for the descriptors. A local feature needs to be represented, precise [Yu, 2011]. To clarify

Bag-Of-Words or Bag-Of-Features or Codebook Generation

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

From [Gorokhovatskyi et al., 2018], "The "**bag of features**" method is a "**bag of words**" analog, which is used in computer vision applications to build strong descriptions of an image. This approach usually includes key point feature detection (typically SIFT), quantization (clustering) of these features (typically with k-means) and distribution of key point features in the space of cluster centers in the form of a histogram (we refer to this procedure as "screening"). Recognition is usually performed by searching for a minimum distance between histograms."

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

- Insensitivity to objects location in image
- Shorten the comparisons to do : less elements presence has to be verified. If a codebook is generated, the presence or absence of each word (= representative feature) in one image, can be encoded as a bitstring. To double check

Con

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

Soft Vector Quantization

From [Yu, 2011], **soft vector quantization** is a **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 ? Check meaning

Hierarchical CodeWords

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

Visual sentences

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

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.

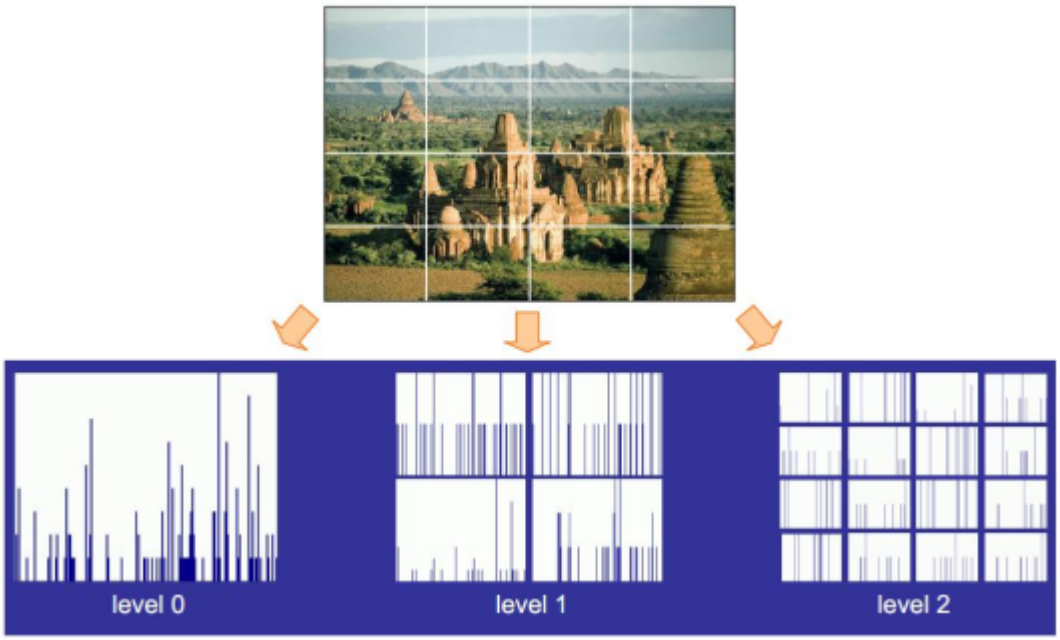


Figure 2.2: 3 levels spatial pyramid from [Yu, 2011]

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

Con

- High computational cost

2.1.4 Step 4 - Feature storage and datastructure

Descriptors can be compressed before matching, especially to allow a fast retrieval.

LSH - Locally Sensitive Hashing

Use a fuzzy hash on feature representation allows a quick approximate lookup on indexed hash.[Ope,]

- O(N)
- Returns the K (parameter) best matchs
- 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 To check ?

2.1.5 Step 5 - Distance choice

This step will find a match between request picture and pictures of the dataset. Linked to correspondence problem ?

Hamming distance / Bruteforce

Partially solved by [Manku et al., 2007]
Works with binary features. Can be accelerated with GPU [Bian et al., 2017].

$O(N^2)$, 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.

Usable with :

- Ratio test Verify link
- CrossCheck test : list of "perfect match" To check

FLANN - Fast Library for Approximate Nearest Neighboors

From [Bian et al., 2017], is an approximation for matching in Euclidian space, with KD-Tree techniques. Work with non-binary features.

Verify following information link [Ope,]

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

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

2.1.6 Step 6 - Matching and selection

This step will find a match between request picture and pictures of the dataset. Linked to correspondence problem ? Verify link Higly noisy correspondences need to be filtered.

Can return :

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

RATIO (Lowe's)

From [Bian et al., 2017] 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 To fill

NGS - Neighboorhood Graph Search To fill

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]

2.1.7 Step 7 - Model Fitting

From [Bian et al., 2017] and [Fea,], Model fitting is a step where the geometry of the scene is verified and/or estimated. Given correspondances, the pose of the object is estimated.

This allows to **identify inliers and outliers** by fitting a homography matrix, which is equivalent to find the transformation of (picture one) to (picture two)

We define :

- Inliers : "good" points matching that can help to find the transformation
- outliers : "bad" points matching

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.

"RANSAC loop involves selecting four feature pairs randomly. It computes Homography H (mapping between any two points with the same center of projection). For each key point, there may be more than one candidate matches in the other processed image. We choose the best matching based on the distance between their descriptors" from [Adel et al.,]

A good illustration video is available at `https://youtu.be/1YNjMxxXO-E?t=95`

Least Meadian

To fill

Chapter 3

Global Features Algorithms

Overview Global feature algorithms try to give a global and general descriptor for a given picture. Generally, this kind of approach is **weak against some transformation as occlusion**, clutter. It needs a fixed viewpoint, a clear background, an 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) : To detail
- Context Triggered Piecewise Hashing (CTPH) : To detail

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

Overview These algorithms do not intend to match pictures with common part, but to match pictures which are roughly the same or are near copies of one another.
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. [Sarantinos et al., 2016] specifies that Context Triggered Piecewise Hashing (CTPH) is a combination of Cryptographic Hashes (CH), Rolling Hashes (RH) and Piecewise Hashes (PH).

Misc. notes

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

An results overview is presented below.

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 of (a/b/d/m/p/w)-Hash from [Tes,] - Lower score is better

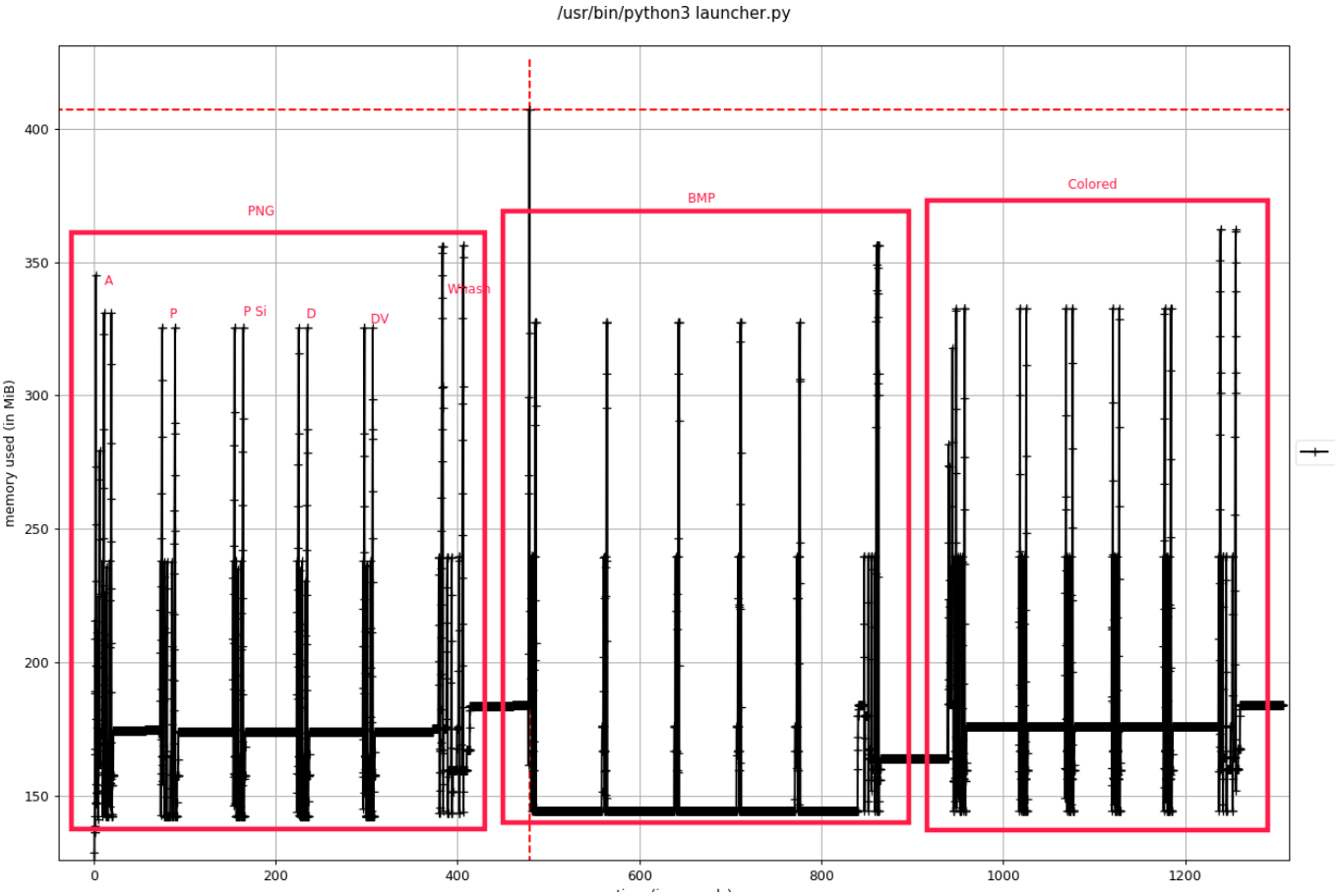


Figure 3.2: Memory consumption of hash-algorithms - from left to right. 84 Mo of pictures are loaded and kept in memory for debug and output purposes. Each "big" block is a different dataset, PNG then BMP, then PNG without text. Each small spike within a block is - from left to right - A, P, P Simple, D, D vertical, W hash. Overhead of the framework is displayed page 103 Figure 7.4

3.1.1 A-HASH : Average Hash

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

There exist relationship between parts of the hash and areas of the input image, that provide 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...). Eahc picture is hashed into a 8 bits vector.

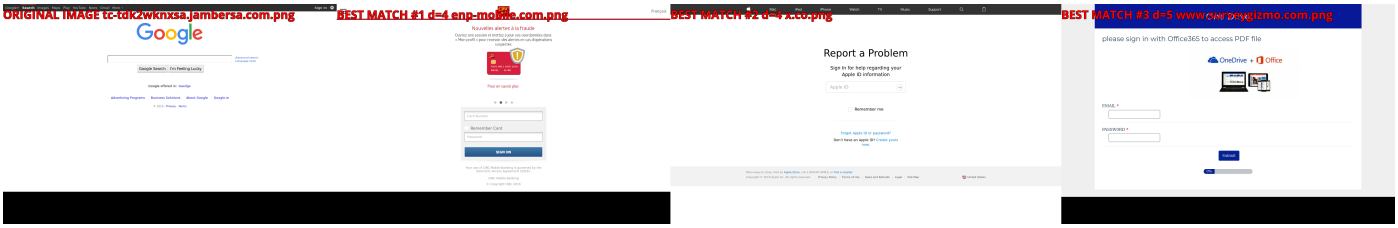
Idea to be faster (achieve membw-bound conditions) : Batch search (compare more than one vector to all others) = do X search at the same time. More than one vector could be transformation of the initial image (rotations, mirrors). Therefore a "rotation, mirrored, etc." search of a request picture can be performed.

Pro

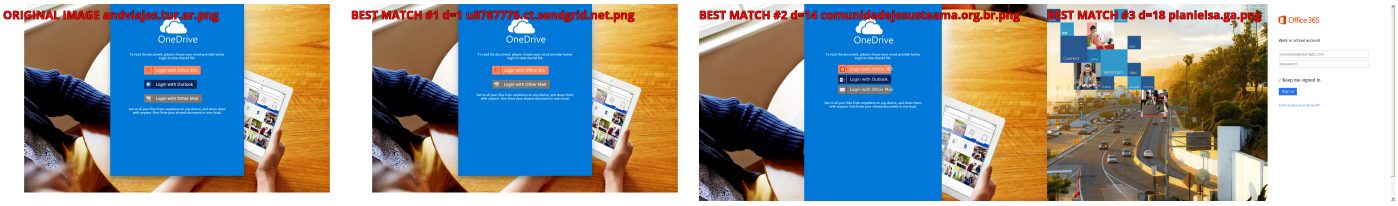
- Masks and transformation available
- Ability to look for modified version of the initial picture
- Only 8 bits for a image vector.

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

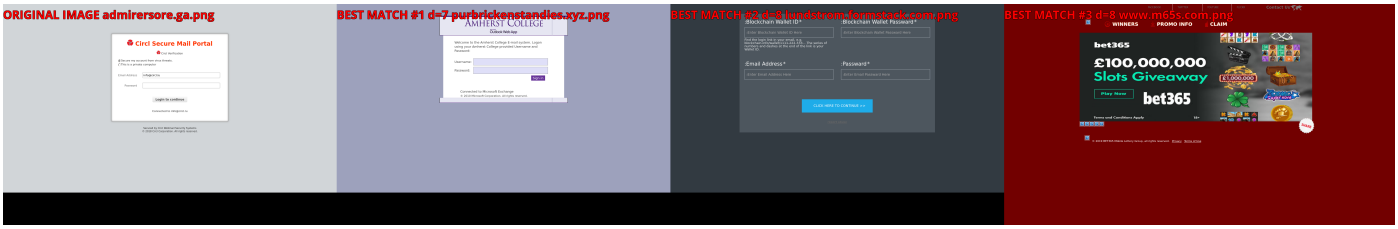
Results Results given by the ImageHash implementation of A-hash algorithm are not reliable. The influence of a threeshold does improve the true-positive rate (at the cost of a not-measured increase of false-negative) ; see Figure 3.15.



(a) Structural matching leading to mismatch



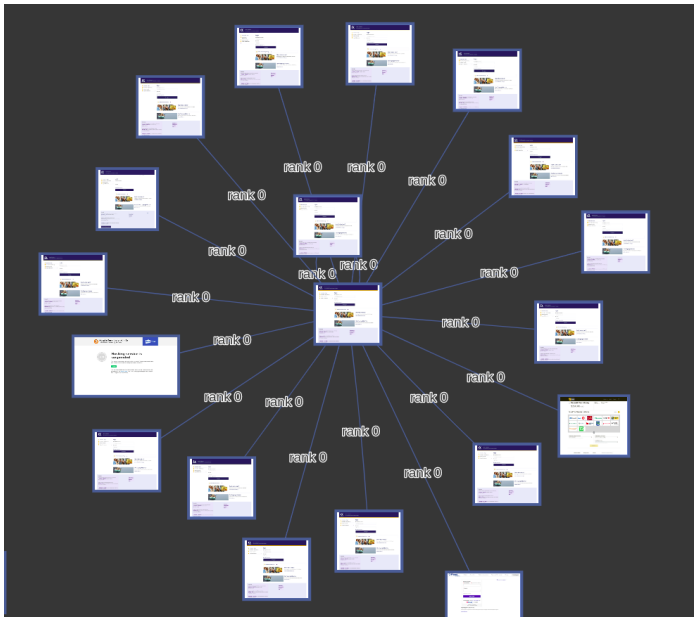
(b) Structural matching leading to match



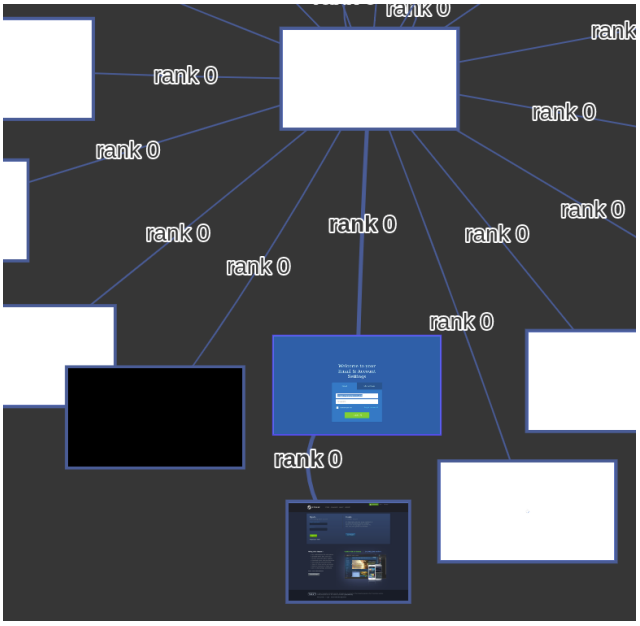
(c) Structural matching seeing "a form"

(d) Missing logo in the match

Figure 3.3: A-hash



(a) Good structural matching



(b) Strange matching (black/white/color)

(c) Bad matching

Figure 3.4: A-hash

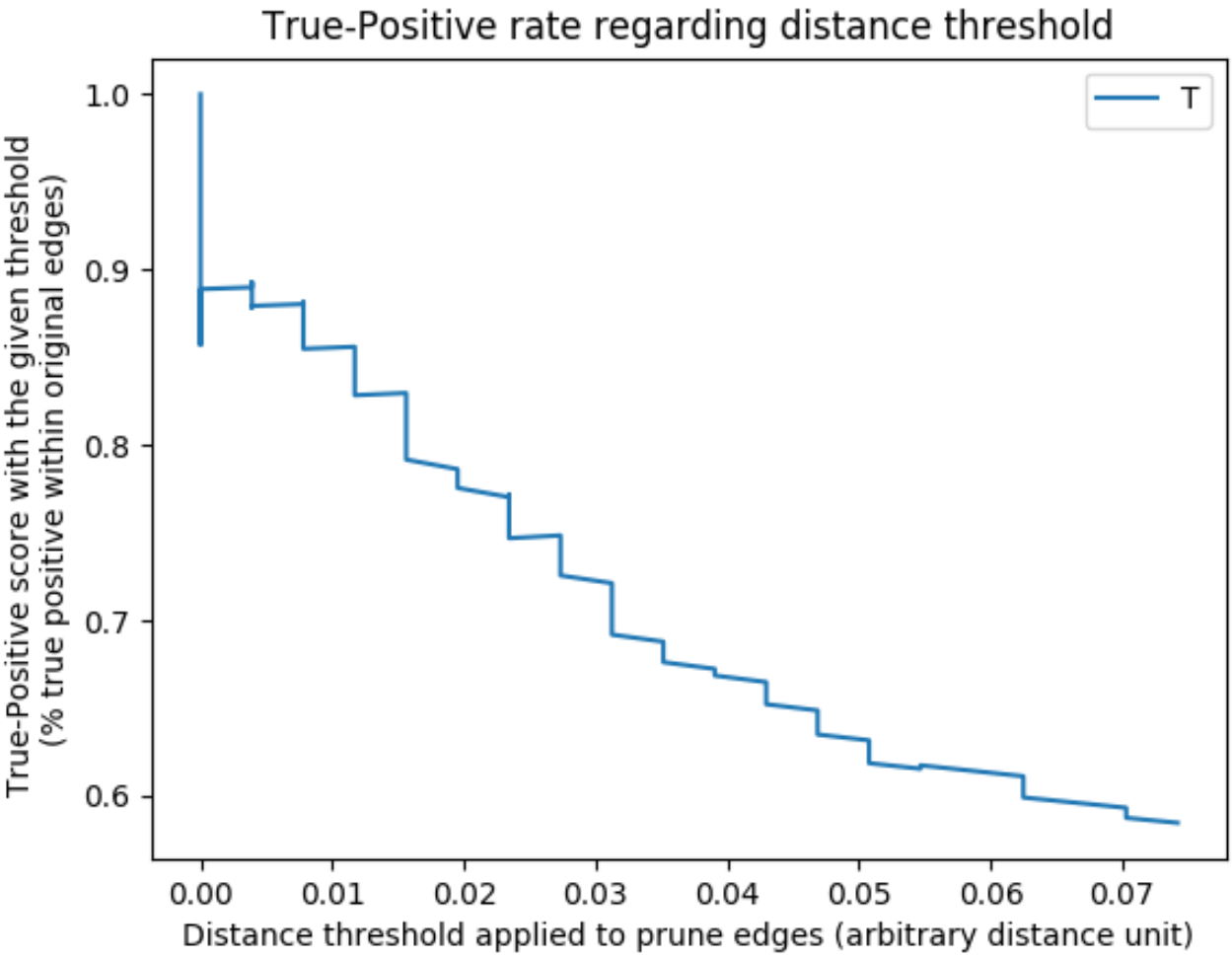


Figure 3.5: Threshold influence on True-positive rate. This figure presents the true-positive rate of images matched, regarding the threshold applied to the edge (= matches) between pictures. All matches with a distance below the absciss (threshold) value are removed ; and the generated graph true-positive rate is computed.

3.1.2 B-HASH : Block Hash

Seems worst than A-hash. Not tested, then.

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 istance between 2 vector is a Hamming distance (number 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

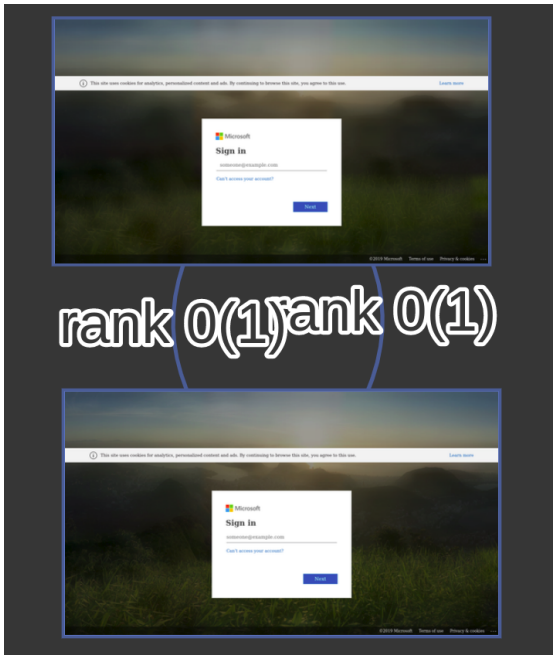
Steps of the algorithm

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.

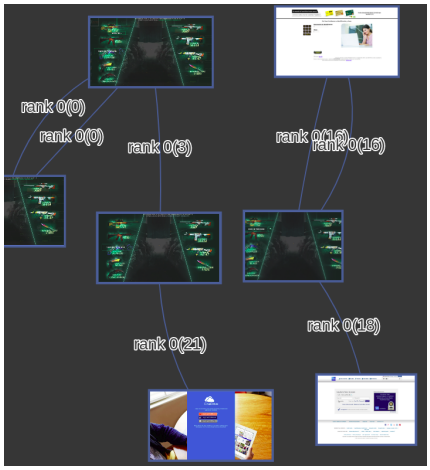
"Vertical" variation A variation of the d-hash algorithm is available, as d-hash vertical.



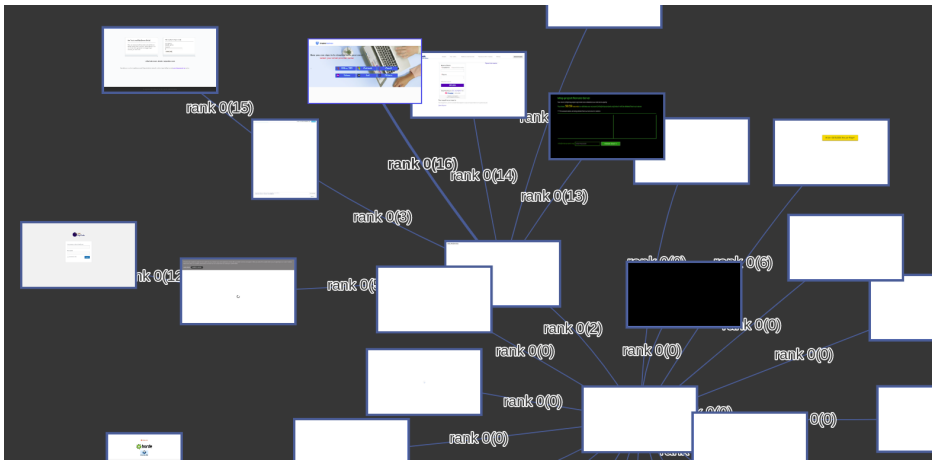
(a) Better than A-hash result

(b) Better than A-hash result

Figure 3.6: d-hash



(a) Wrong result



(b) Wrong result

Figure 3.7: d-hash

(a) Mismatch (b) Mismatch

Figure 3.8: Vertical d-hash

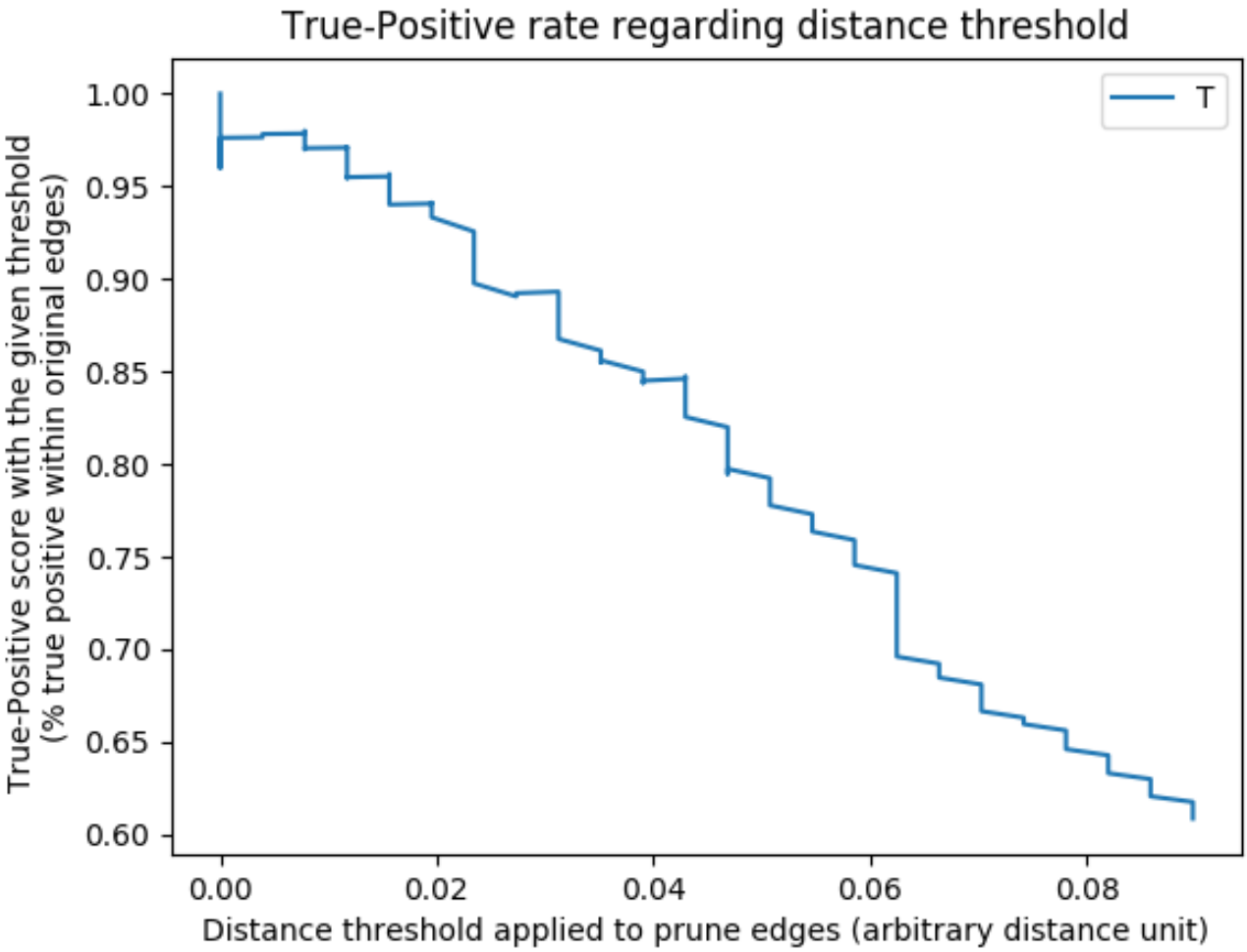


Figure 3.9: Threshold influence on True-positive rate. This figure presents the true-positive rate of images matched, regarding the threshold applied to the edge (= matches) between pictures. All matches with a distance below the absciss (threshold) value are removed ; and the generated graph true-positive rate is computed.

3.1.4 P-HASH - Perceptual Hash

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

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

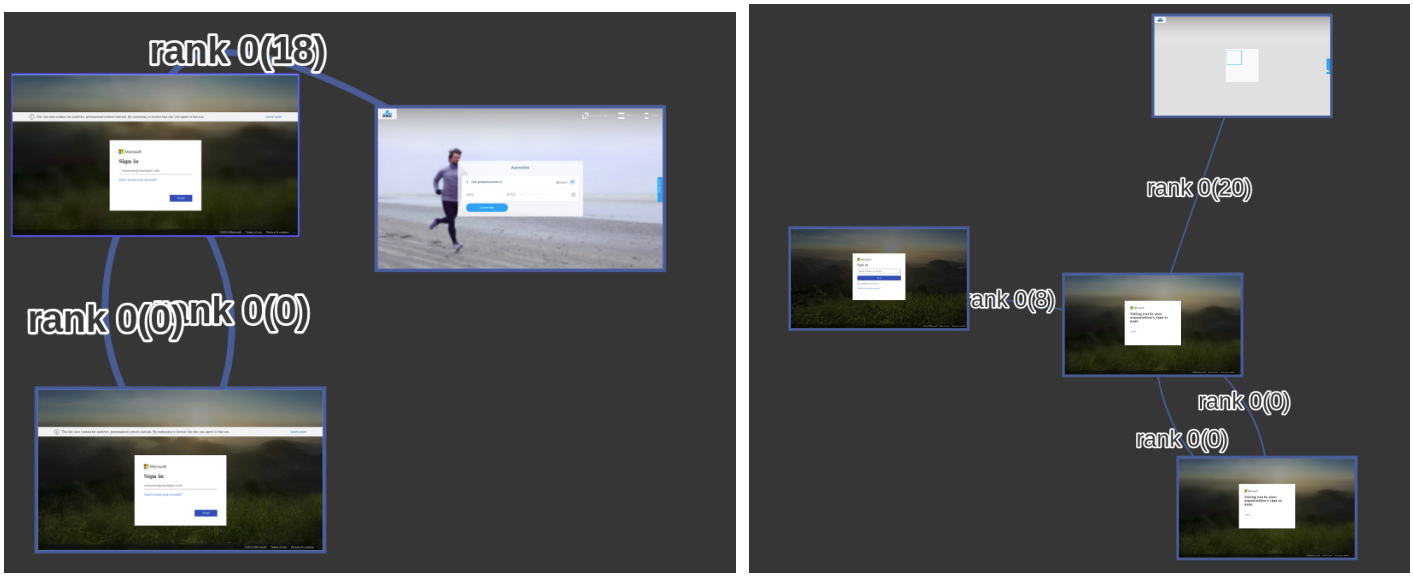
Steps of the algorithm

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/`
Java implementation : [PHa, 2011]

Results Still some strange results

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



(a) Wrong result

(b) Wrong result

Figure 3.10: P-hash

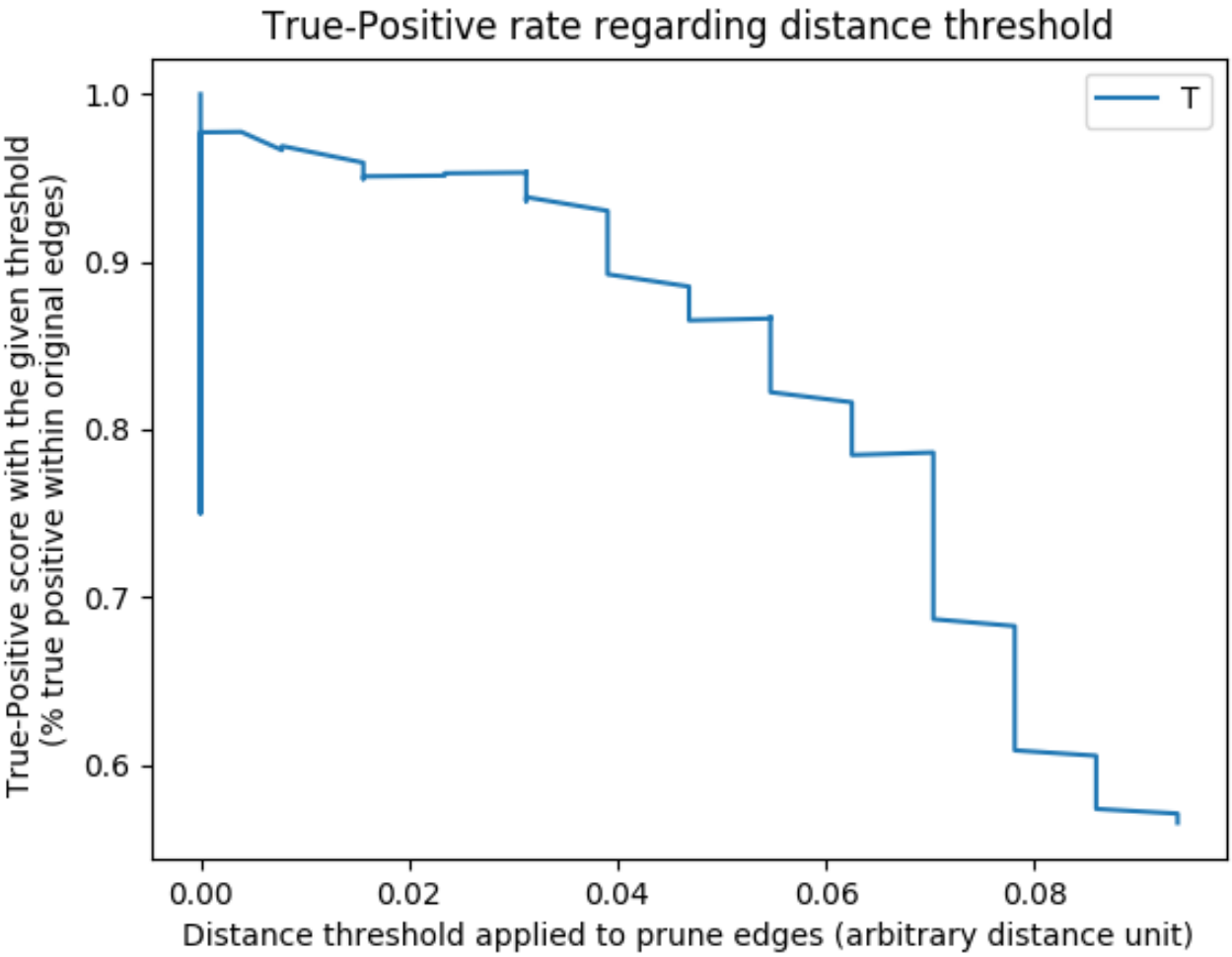
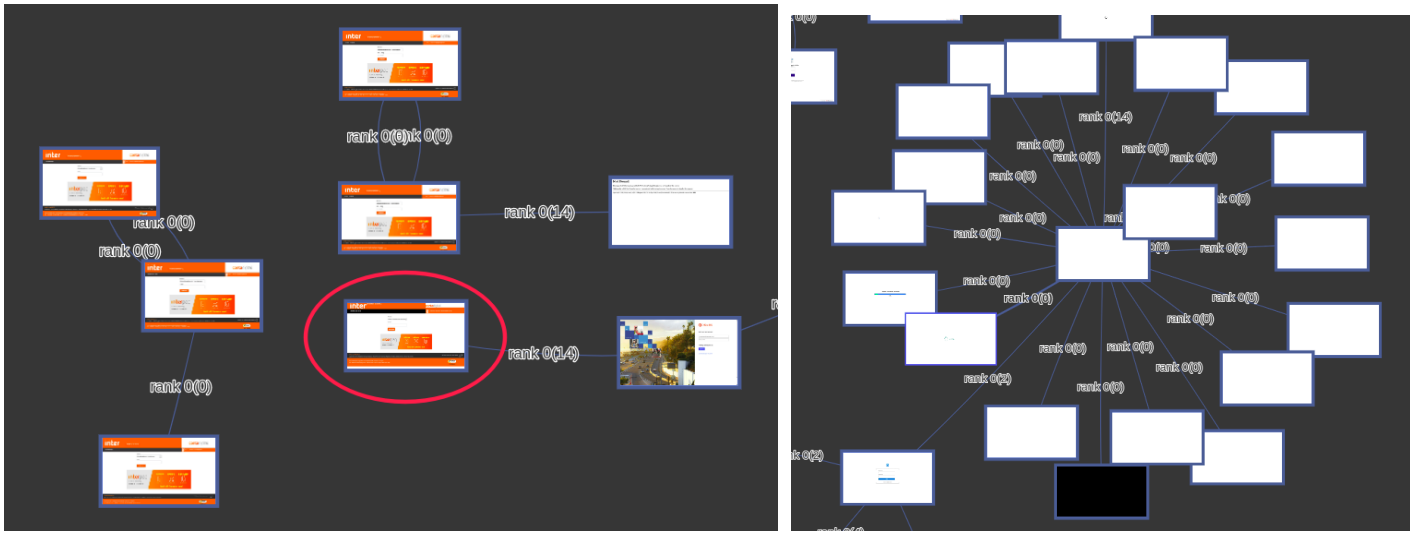


Figure 3.11: Threshold influence on True-positive rate. This figure presents the true-positive rate of images matched, regarding the threshold applied to the edge (= matches) between pictures. All matches with a distance below the absciss (threshold) value are removed ; and the generated graph true-positive rate is computed.



(a) Wrong result (b) Good result

Figure 3.12: P-hash - Simple

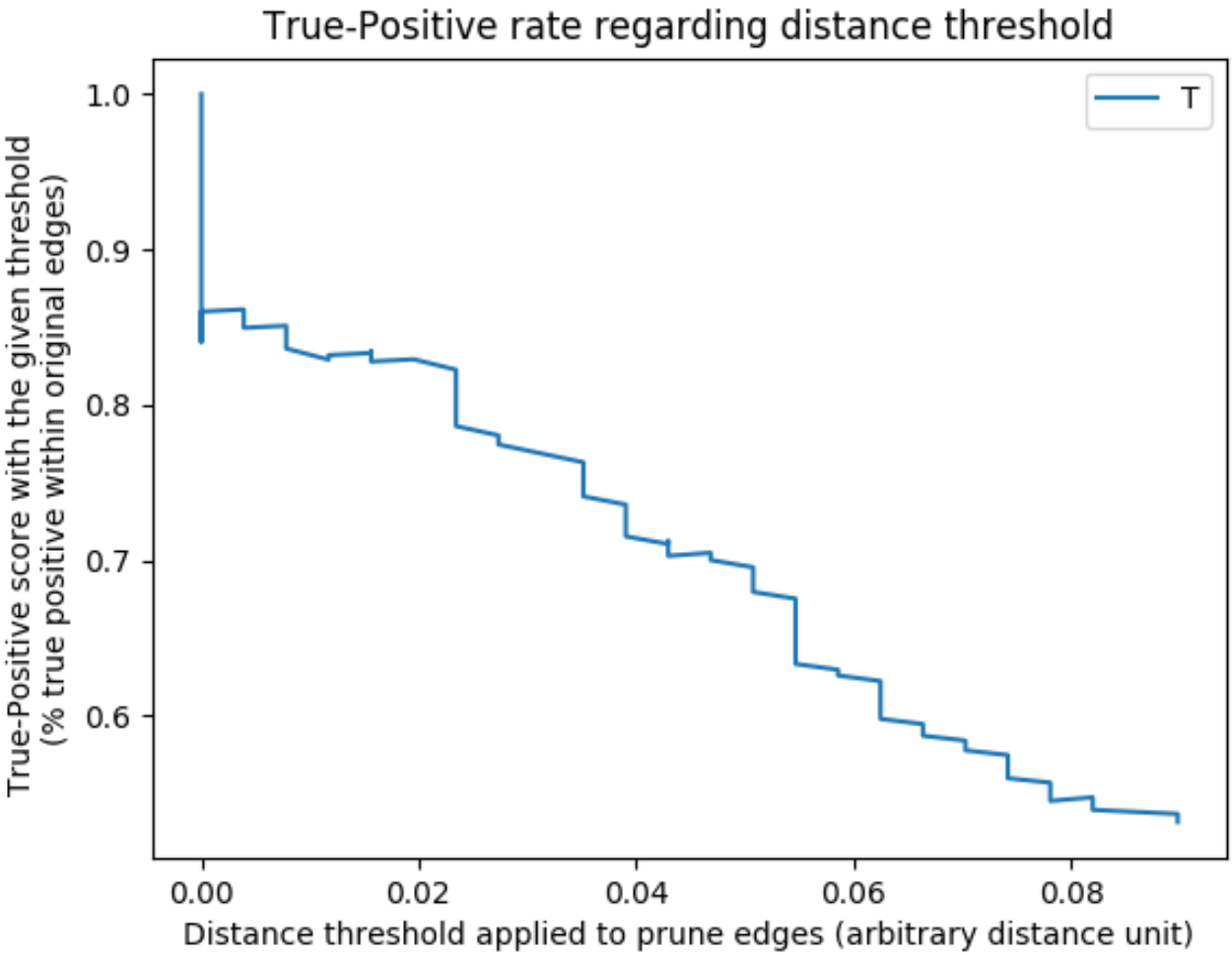


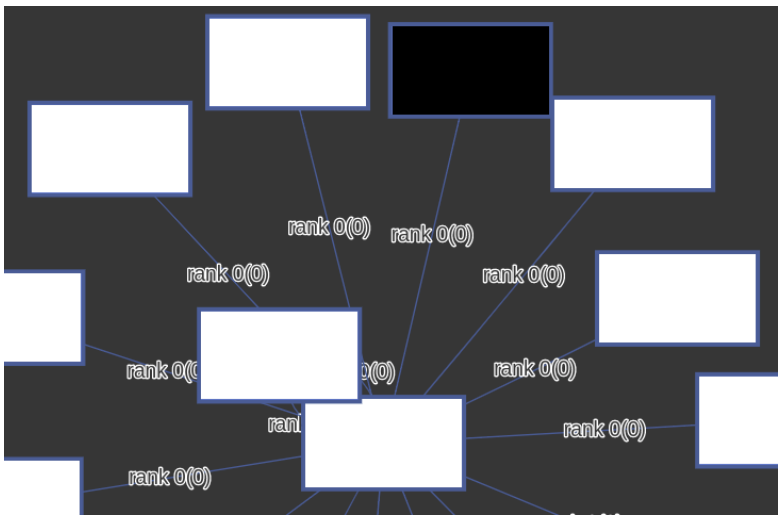
Figure 3.13: Threshold influence on True-positive rate. This figure presents the true-positive rate of images matched, regarding the threshold applied to the edge (= matches) between pictures. All matches with a distance below the absciss (threshold) value are removed ; and the generated graph true-positive rate is computed.

3.1.5 W-HASH - Wavelet Hash

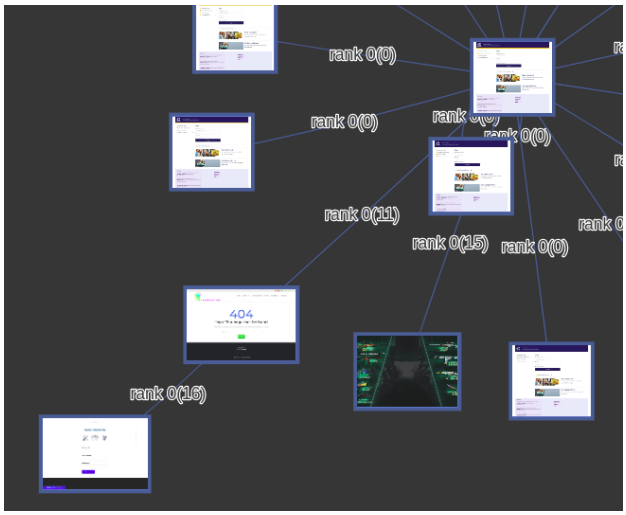
From ... Uses DWT instead of DCT. To detail

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

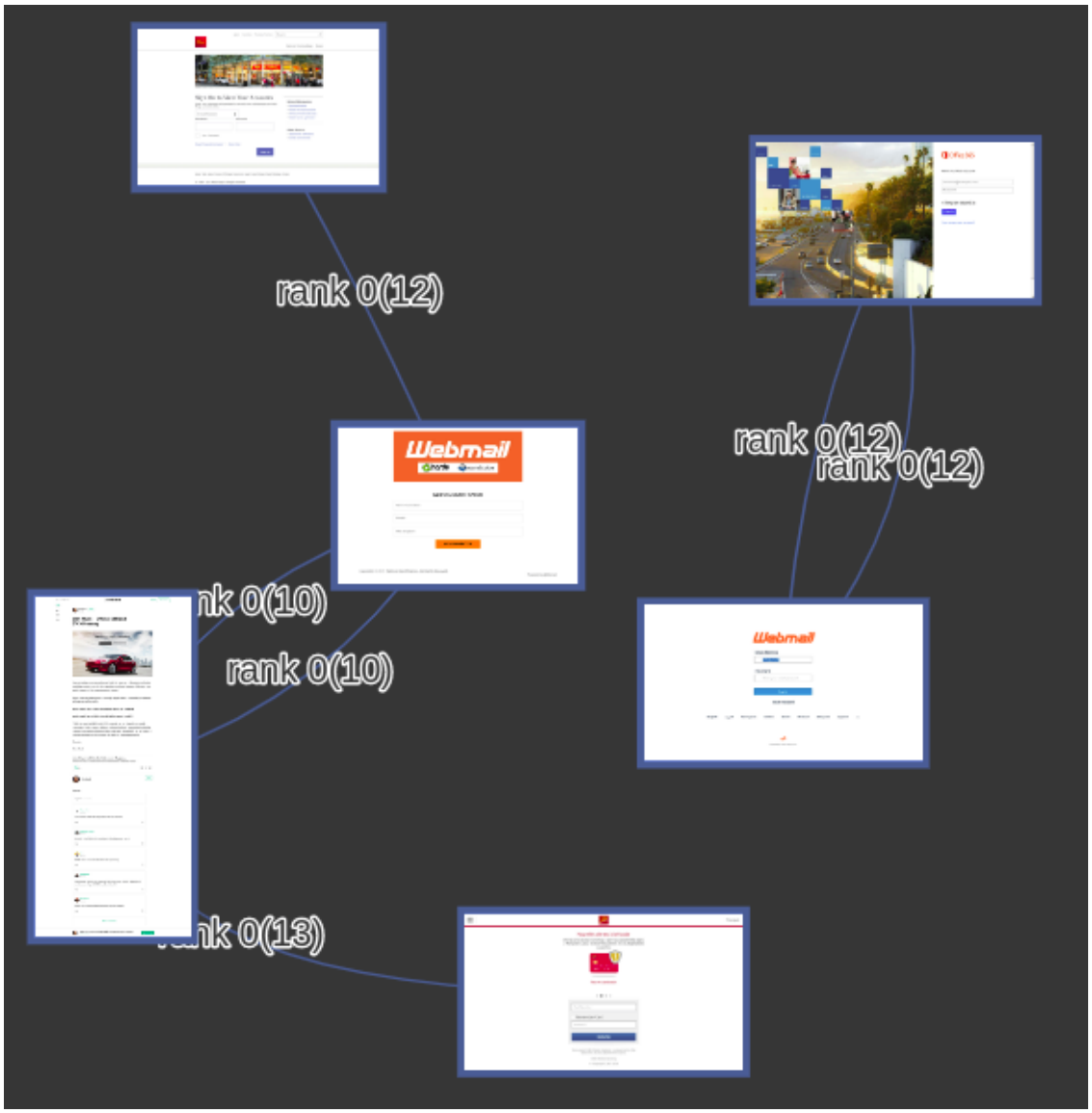
Results Better than others, but still some strange/wrong results.



(a) Strange result



(b) Wrong result



(c) Wrong result

Figure 3.14: W-hash

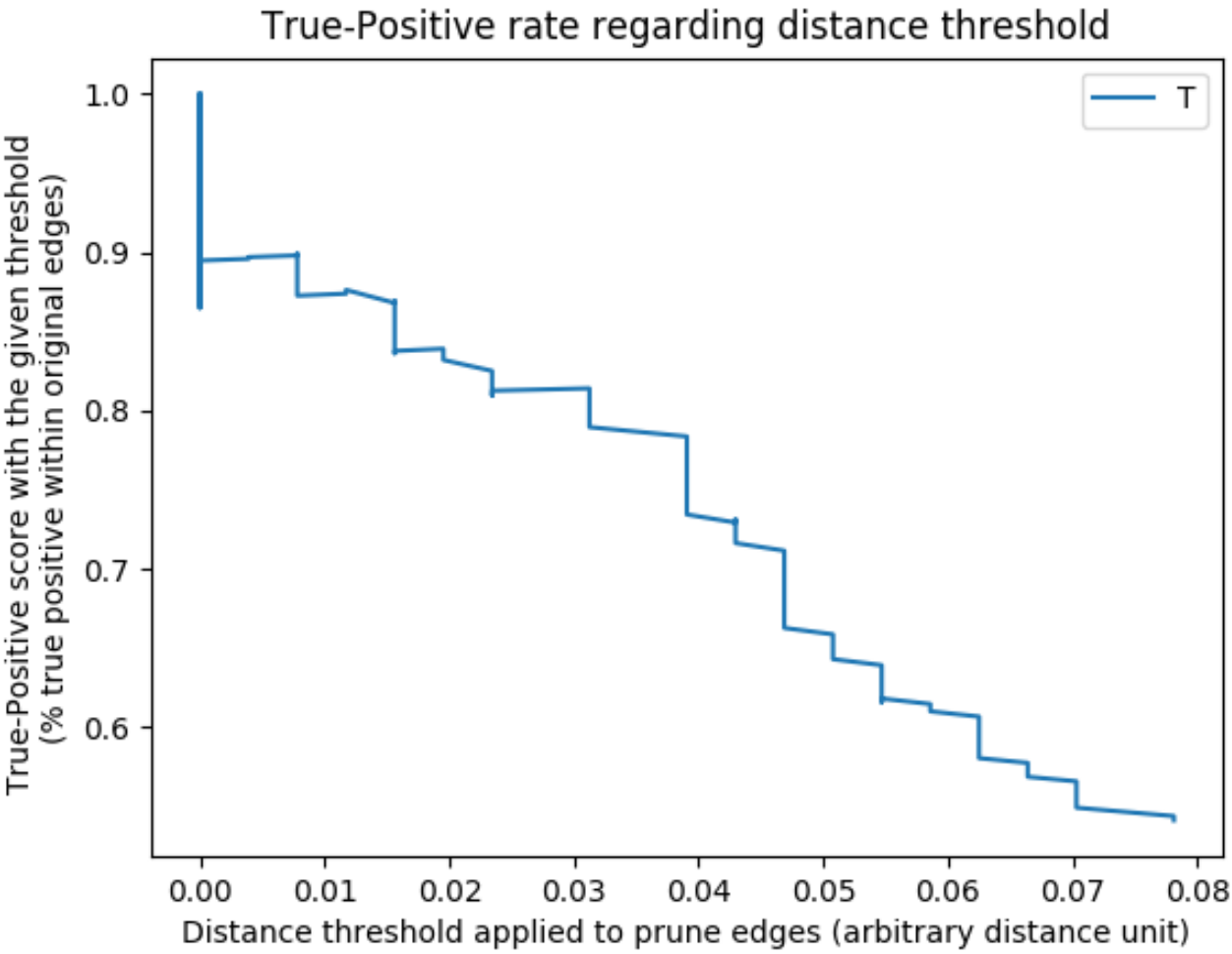


Figure 3.15: Threshold influence on True-positive rate. This figure presents the true-positive rate of images matched, regarding the threshold applied to the edge (= matches) between pictures. All matches with a distance below the absciss (threshold) value are removed ; and the generated graph true-positive rate is computed.

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.
Has a 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

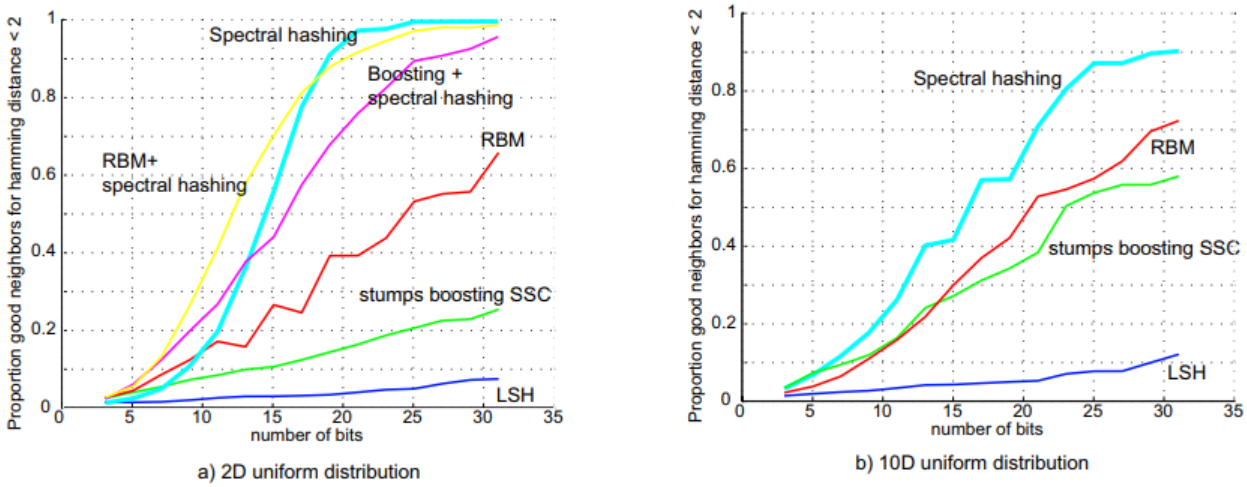


Figure 3.16: Spectral Hashing comparison from [Spe,]

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] To detail

Pro

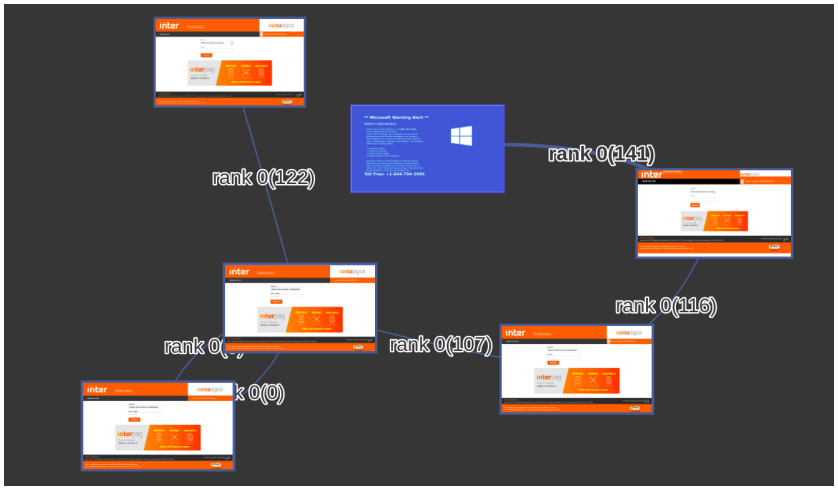
- Open Source

3.1.12 TLSH - Trend Micro Locality Sensitive Hashing

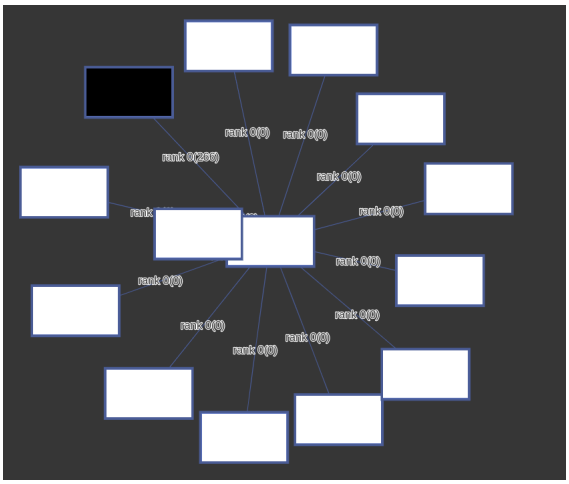
From [Oliver et al., 2013] directly performed on file, not only pictures. Therefore, running TLSH on PNG can't provide consistent results, as the structure of the file is taken into account (and so it's meaningless compression). Files have to be uncompressed (e.g. some types of BMP files) to have consistent resuls with TLSH.

Pro

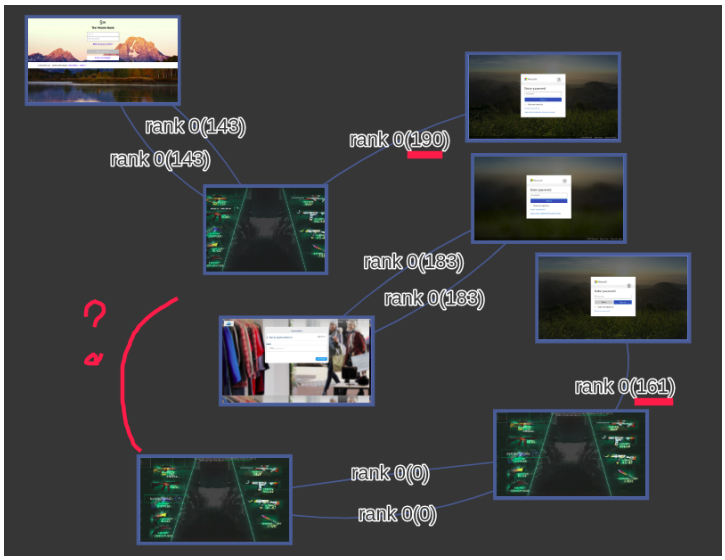
- Parametered threeshold (below 30 in original paper)
- Open Source



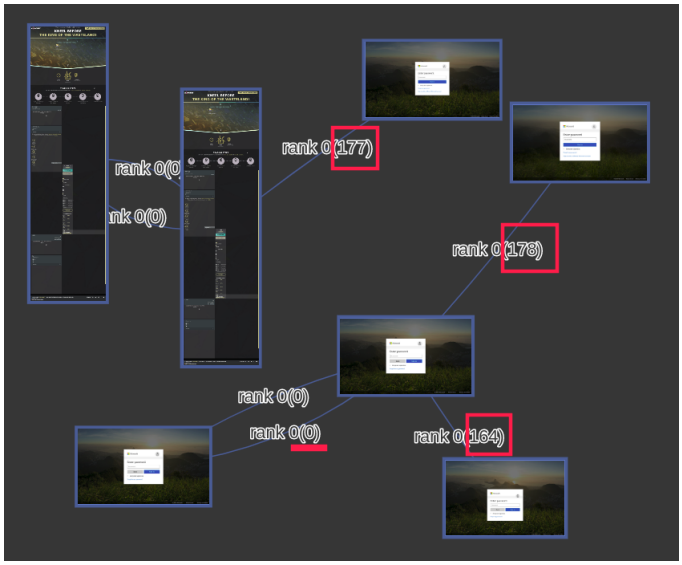
(a) High distance for close pictures



(b) Perfect match for colors



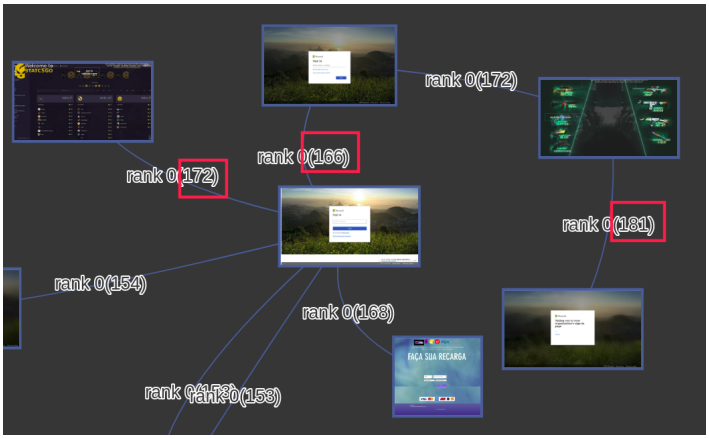
(c) Strange mismatches



(d) Diverses matches

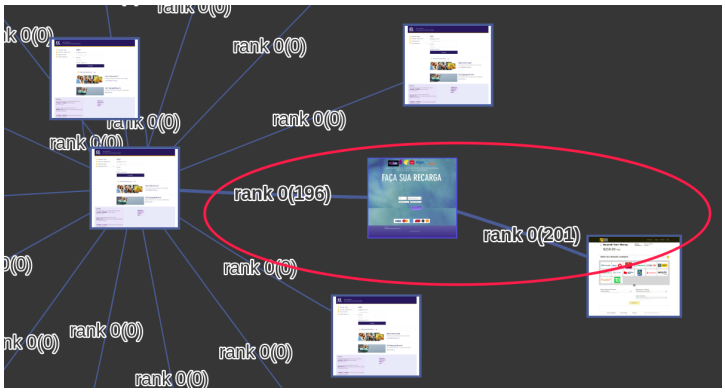
Figure 3.17: TLSH (normal) with PNG

(a) Strange result



(b) Strange result

Figure 3.18: Inconsistent distance regarding pictures - TLSH (no length) with PNG



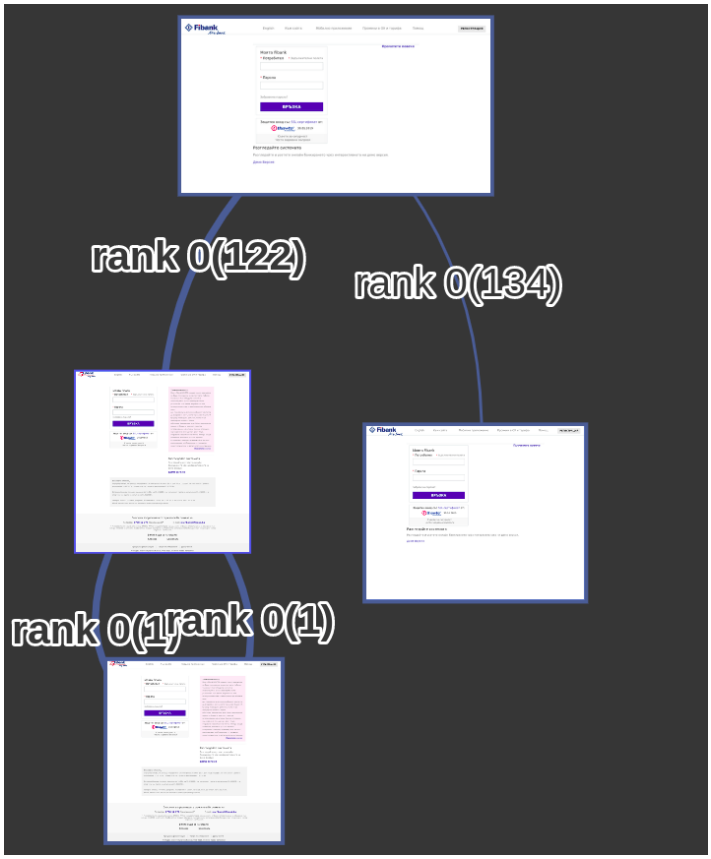
(a) Mismatch that would seems obvious. Upper header is probably the source of match.

(b) Clear mismatch, probably due to common layout.

Figure 3.19: Mismatches - TLSH with BMP

(a) Impressive good match on a logo brand (upper line), with a mismatch (bottom)

(b) Impressive match for a hash algorithm, on Microsoft login page



(c) Good matches even with slighty different page layout

Figure 3.20: Good matches - TLSH with BMP

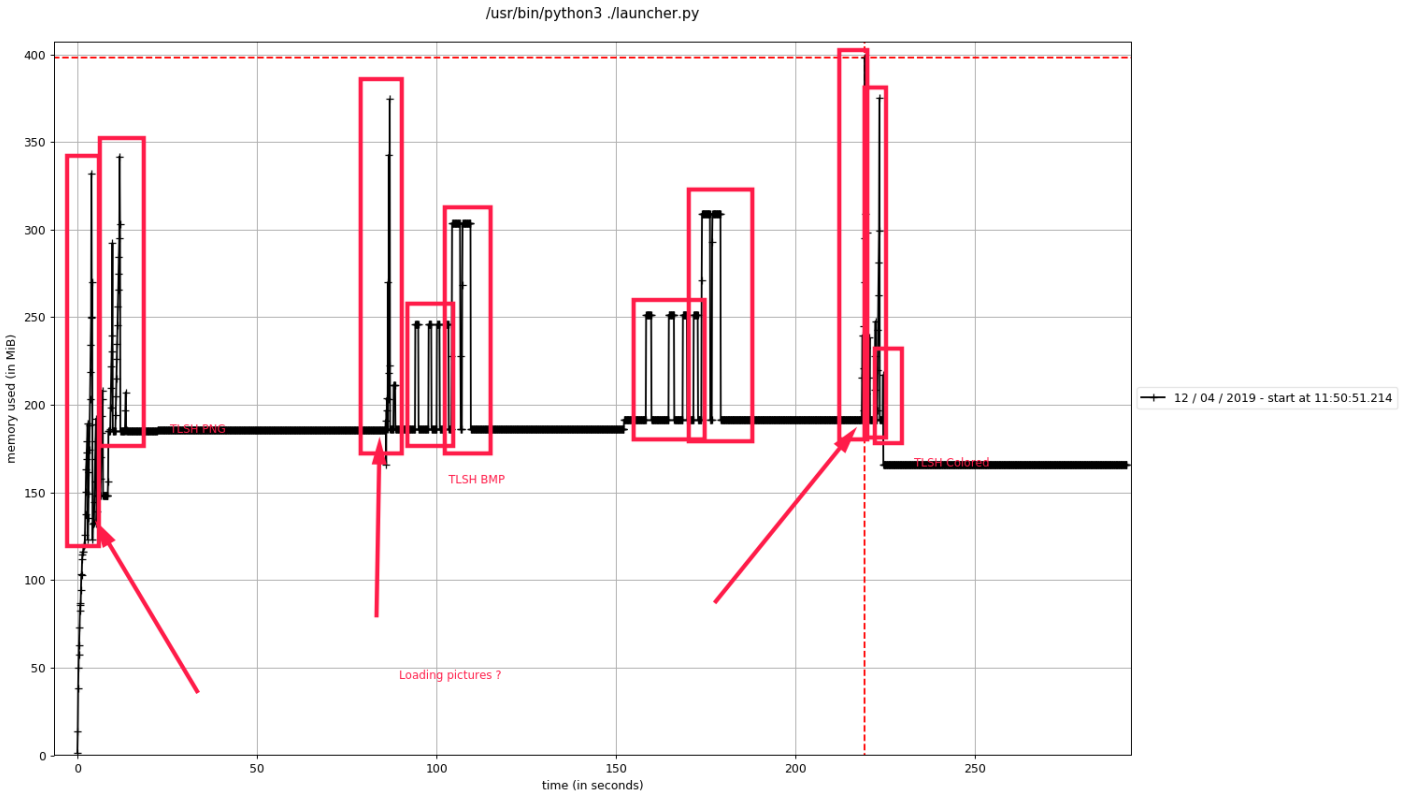


Figure 3.21: Memory consumption of TLSH. 84 Mo of pictures are loaded and kept in memory for debug and output purposes. 2 configuration (blocks) are tested, on 3 datasets (arrows). Few blocks are not identified. Overhead of the framework is displayed page 103 Figure 7.4

3.1.13 SSDeep - Similarity Digest

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

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 3.22: 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. To detail

3.2.2 Contrast context histogramm

From [Huang et al., 2008] Define a descriptor based on a contrast histogramm of keypoints. To detail

Chapter 4

Local Features Algorithms

Goal is to transform visual information into vector space

4.1 Comparison overview

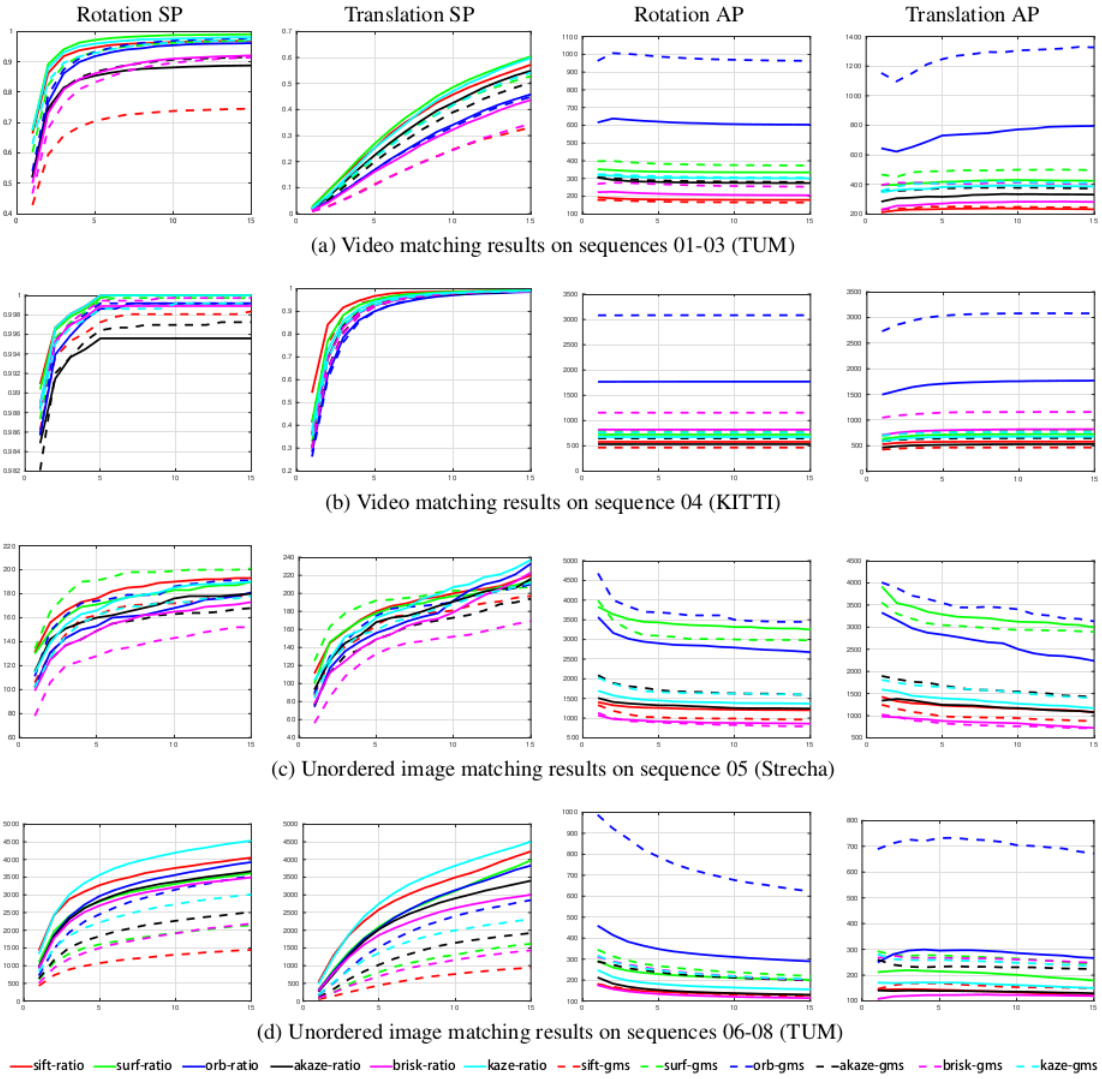


Figure 4.1: Benchmarking of SIFT, SURF, ORB, AKAZE with RATIO and GMS selection ; FLANN or Hamming for distance. SP curves show the success ratio or success number (number of correspondance for AP) with thresholds. X-Axis being the threeshold. AP curves illustrate the mean number of verified correspondences with thresholds. [Bian et al., 2017]

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 4.2: 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., 2017]

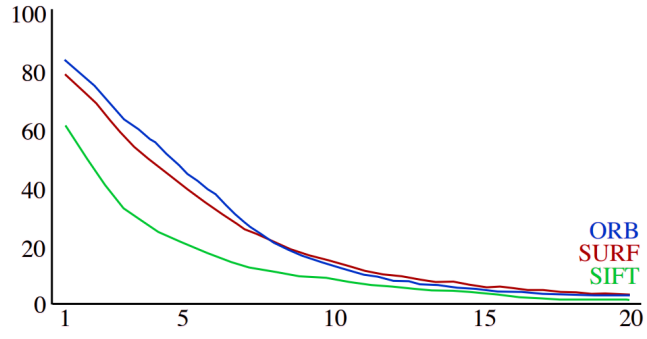
Table 2. The score of robustness, accuracy, and sufficiency exported from results on rotation cases. Best viewed in colors.

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

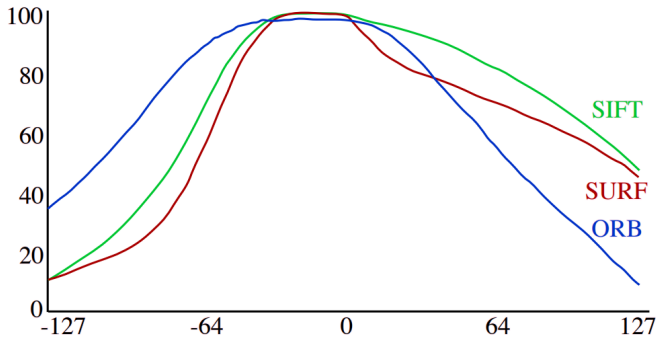
Figure 4.3: Benchmarking of SIFT, SURF, ORB, AKAZE, BRISK, KAZE on robustness (RS), accuracy (AS), sufficiency (SS). Ordered best time from best to worst : red, green, blue, black. [Bian et al., 2017]

- **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.

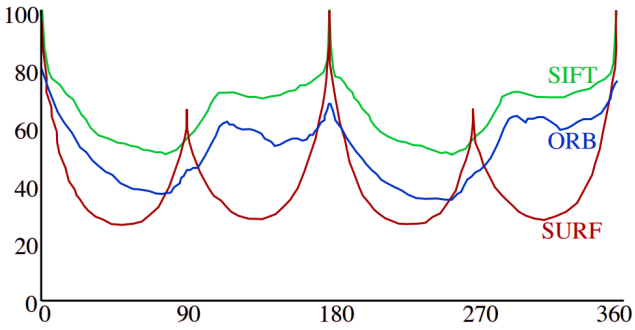
Some more comparisons are performed in [Key, 2014] about robustness of different algorithms.



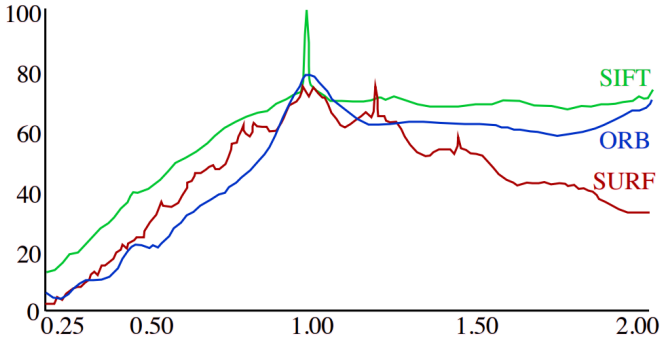
(a) Blurring repeatability (parameter)



(b) Brightness repeatability (offset)



(c) Rotation repeatability (angle)



(d) Scaling repeatability (factor)

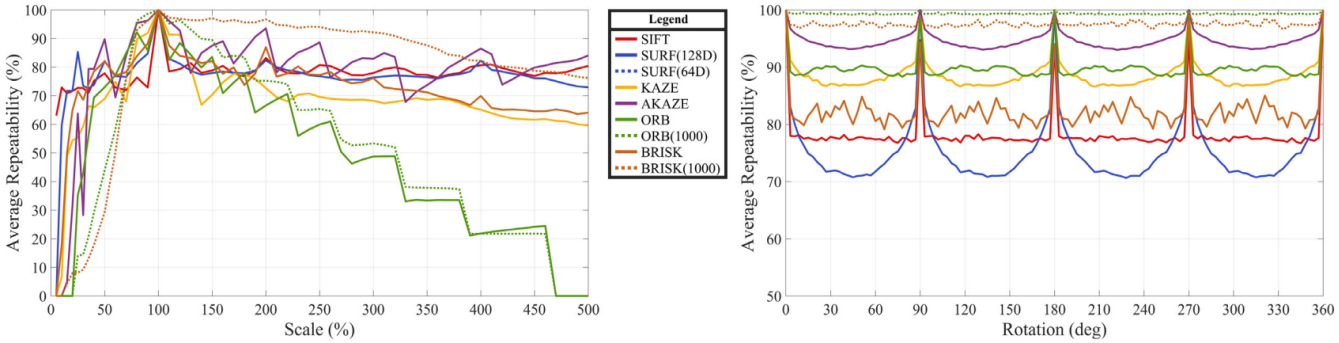
Figure 4.4: Robustness (nb of matches/nb points in original image in %) comparison from [Key, 2014]

A large performance overview has been operated by [Tareen and Saleem, 2018]. Many information are extracted from their tests and are showed on each algorithm description.The paper is ordering algorithms thanks to different objective.

Ability to detect high quantity of features :
$ORB > BRISK > SURF > SIFT > AKAZE > KAZE$
Computational efficiency of feature-detection-description per feature-point :
$ORB > ORB(1000) > BRISK > BRISK(1000) > SURF(64D) > SURF(128D) > AKAZE > SIFT > KAZE$
Efficient feature-matching per feature-point :
$ORB(1000) > BRISK(1000) > AKAZE > KAZE > SURF(64D) > ORB > BRISK > SIFT > SURF(128D)$
Speed of total image matching :
$ORB(1000) > BRISK(1000) > AKAZE > KAZE > SURF(64D) > SIFT > ORB > BRISK > SURF(128D)$

Algorithm	Mean Feature-Detection-Description Time per Point (µs)		Mean Feature Matching Time per Point (µs)
	1st Images	2nd Images	
SIFT	90.44	85.15	**142.02**
SURF(128D)	42.78	42.22	**168.55**
SURF(64D)	41.83	41.18	89.66
KAZE	**191.24**	**177.09**	60.58
AKAZE	60.93	57.04	24.61
ORB	**3.94**	**3.94**	97.25
ORB(1000)	13.51	13.92	**11.82**
BRISK	16.59	16.76	**124.64**
BRISK(1000)	20.70	21.49	**15.42**

Figure 4.5: Results from [Tareen and Saleem, 2018] - Time cost - mean value on original paper dataset



(a) Average repeatability of feature detectors for five images (*Dataset-B*) w.r.t synthetic scale changes.

(b) Average repeatability of feature detectors for five images (*Dataset-B*) w.r.t synthetic rotations.

Figure 4.6: Results from [Tareen and Saleem, 2018] - Resilience to changes graphs

4.2 Feature detector

"The choice of the convenient feature detector depends firstly on the nature of the problem." [Adel et al.,]
It mainly depends on the property to be verified : scale invariance, rotation invariance ...

Harris Detector

From the original paper [Harris and Stephens, 1988].
Based on the main principle that at a corner, the **image intensity** will change largely in multiple directions, with a windows shift. [Yu, 2011] says it is invariant to rotation, scale, illumination, noise .. Scale invariance is probably obtained by a variation of the original algorithm (increase gaussian blur many time, automatic scale selection ..)

"Harris is a corner detector based on Moravec algorithm, which is proposed by Harris and Stephens in 1988. A detecting window in the image is designed. The average variation in intensity is determined by shifting the window by a small amount in a different direction. The center point of the window is extracted as a corner point." from [Adel et al.,]

Pro and cons

- 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.

Kind of descriptor ?

GoodFeaturesToTrack

To detail
Kind of descriptor ?

"It expands the Harris detector to make its corners more uniformly distributed across the image. Shi and Tomasi showed how to monitor the quality of image features during tracking. They investigated a measure of feature dissimilarity that quantifies how much the appearance of a feature changes between the first and the current frame." from [Adel et al.,]

CSS - Curvature Space Scale

To detail
Kind of descriptor ?

Hit and Miss filter

To detail
Kind of descriptor ?

Shi/Tomasi

To detail
Kind of descriptor ?

MSER

To detail
Kind of descriptor ?

"MSER stands for Maximally Stable Extremal Regions Detector. It was generated by Matas et al. to find matching between image elements from two input images from different viewpoints. The 'maximally stable' in MSER describes the property optimized in the threshold selection process. The word 'extremal' refers to the property that all pixels inside the MSER may be either higher or lower intensity than all the pixels on its outer environment." from [Adel et al.,]

SUSAN

From ... a word in [Rosten and Drummond, 2006]
Less accuracy but faster. To detail
Kind of descriptor ?

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

"FAST is a high-speed feature detector that is much suitable for real-time applications. The algorithm considers a circle of 16 pixels around the candidate corner p. A feature is specified when a set of n contiguous pixels in the circle are all darker or brighter than the candidate pixel p plus a threshold t" from [Adel et al.,]

Pro

- High repeatability
- Scaling invariant To check

Con

- Not rotation-invariant (no orientation calculation)
- Not scale invariant
- Not robust to high levels noise
- can respond to 1 pixel wide lines
- dependent on a threshold

Kind of descriptor ?

Bag Of Feature

To detail + Not sure it's a feature detector !

4.3 Feature descriptor

4.4 Feature detector and descriptor

4.4.1 Unsorted

BRISK -

To detail
Kind of algo ?

AKASE -

To detail
Kind of algo ?

CenSurE

To detail
Kind of algo ?

KASE -

Shipped in OpenCV library. Example can be found at [Nikishaev, 2018] To detail
Kind of algo ?

4.4.2 Non-binary features / Vector descriptor

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, ...

"SIFT proposed by David Lowe and then improved in 2004. Currently, it is the most common known vector descriptor. It consists of four essential stages: scale-space extrema detection, key points localization, orientation assignment, and generating keypoint descriptor. In the first stage, the key points are extracted based on their strength that are invariant to orientation and scale using Difference of Gaussian. In the second stage, the wrong points are removed. Then in the following stage, one or more orientations are assigned to each keypoint. In the final stage, a vector descriptor is made for each keypoint." from [Adel et al.,]

Pro

- "SIFT is [...] most accurate feature-detector descriptor for scale, rotation and affine variations (overall)." [Tareen and Saleem, 2018]

- invariant to scale and rotational variation

Con

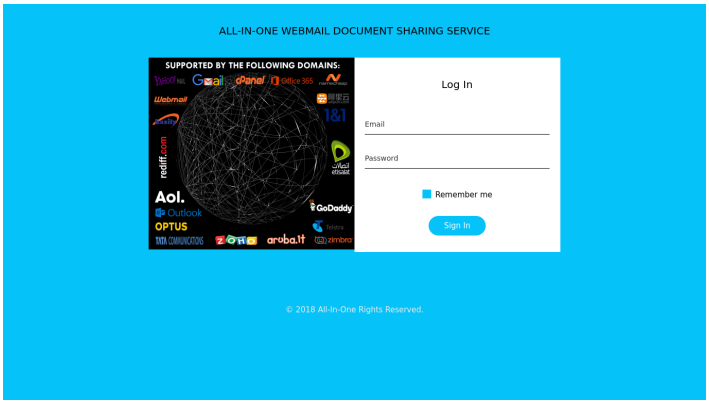
- Patented algorithm and not included in OpenCV (only non-free module)
- Slow : "a general perception that SURF is computationally efficient than SIFT, but the experiments revealed that SURF(128D) algorithm takes more time than SIFT(128D)" [Tareen and Saleem, 2018]

Steps of the algorithm

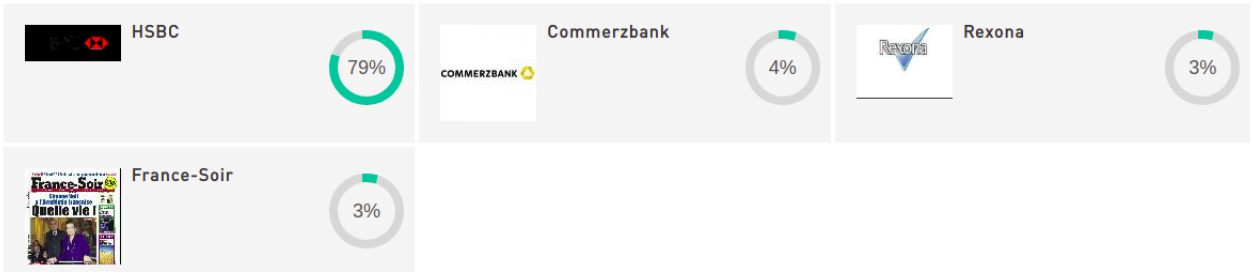
1. Extrema detection
   Uses an approximation of LoG (Laplacian of Gaussian), as a Difference of Gaussion, made from difference of Gaussian blurring of an image at different level of a Gaussian Pyramid of the image. Kept keypoints are local extrema in the 2D plan, as well as in the blurring-pyramid plan.
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)

DigInPix from INA does use SIFT along with indexing techniques. Few example of retrived pictures and logos can be seen at following links : `http://diginpix.ina.fr/en/analysis/9858-3499-9117-1290` and `http://diginpix.ina.fr/en/analysis/8640-0399-5437-8023`

(a) Source picture 1



(b) Source picture 2



(c) Result image 1

(d) Result image 2

Figure 4.7: Diginpix results on picture with lot of logos

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/`

Root-SIFT

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

SURF – Speeded-Up Robust Features

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

"The SURF algorithm was proposed by Bay et al. [7]. It is built upon the SIFT, but it works by a different way for extracting features. SURF is based on multi-scale space theory and speeds up its computations by fast approximation of Hessian matrix and descriptor using "integral images". Haar wavelets are used during the description stage" from [Adel et al.,]

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
- illumination invariance

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

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.

TOP-SURF

From [Thomee et al., 2010] and a word in [**?**]

Based on the BoW approach. Library provided with the paper under an open-source license. This seems a very promising tool to implement a BoW approach.

[**?**] describing it as : "[This method] was introduced as a method to combine key point features with visual word representation. Many features (over 33 million) were clustered in order to form up to 500,000 clusters (the lowest amount was 10,000). The sequence of such clusters is called a "visual word dictionary". Separate weight was assigned to each word to take into account its frequency. The recognition stage included the mapping of each key point feature of an input image to the closest visual word forming the weighted histogram of words. Only top N words were left as a final image descriptor; then, the distance between such descriptors could be computed to compare the images."

Pro

- Open-source
- Existing library

Con

- C++ API

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

LBP - Local Binary Pattern

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

4.4.3 Binary features / Binary descriptor

ORB – Oriented FAST and Rotated BRIEF

From [Rublee et al., 2011] which is rougly a fusion of FAST and BRIEF. See also [ORB, 2014] The threeshold of 0.75 (Lowe ratio) should be modified for ORB. 0.89 seems to be the equivalent to 0.75

"ORB technique developed by Rublee et al. It is a combination of features from FAST keypoint detection and Binary Robust Independent Elementary Features (BRIEF) descriptor. It describes the features of the input image in a binary string instead of the vector." from [Adel et al.,]

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.

Analysis Distance can be computed in many ways. Here is an analysis of each of the used distances.

- Min length : the number of matches divided by the minimal number of descriptors of both pictures. This doesn't work well, because if one of both pictures has a low number of descriptors, it will act as an "attractor" : this picture will be the closest one of many other one. It does not use the "internal distance of a match" : the distance of one descriptor to the other.
- Max length : the number of matches divided by the minimal number of descriptors of both pictures. This works pretty well, removing the issue encountered by the min length. However it does not use the "internal distance of a match" : the distance of one descriptor to the other.
- Mean of matches : it makes use of the "internal distance of a match" : the distance of one descriptor to the other. The distance between two pictures is computed as the mean of the matches distance. This doesn't work well, for the same reason as min-length : if one of both pictures has a low number of descriptors, it will act as an "attractor" : this picture will have very few matches with others, but this small set of matches will have very "good distances".

A question remains, about "How to select good and batch matches". Ratio approach (as in SIFT) are for example usable. A simple threeshold can be used :

I don`t know either. Distance is measure of the feature point similarity, less is better. The original ORB paper (fig. 5, below) shows distribution of the distances for good and bad matches. One can surely says that "good" distance threshold would be around 64.

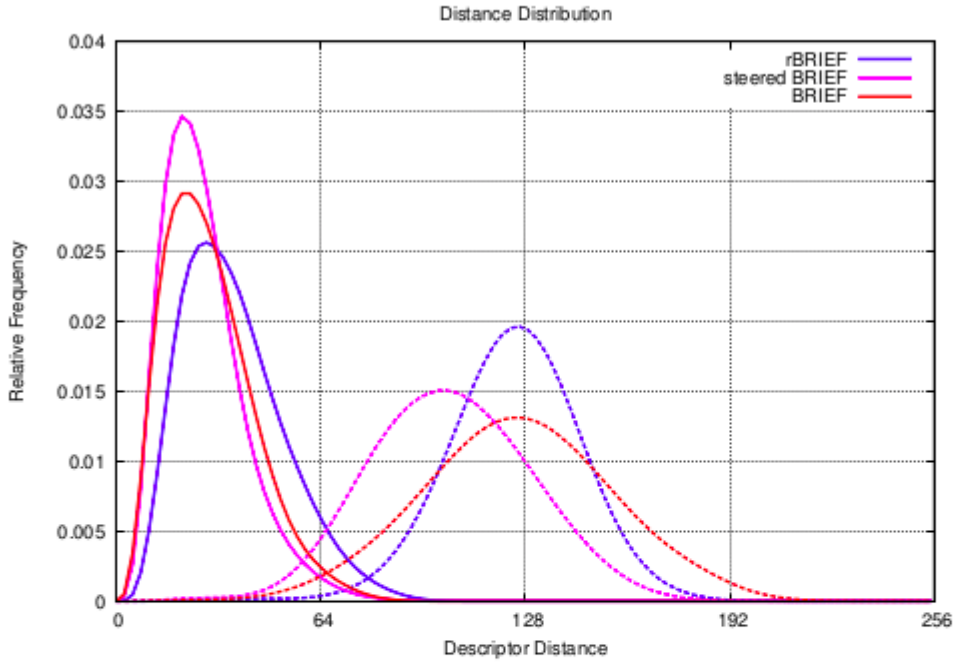
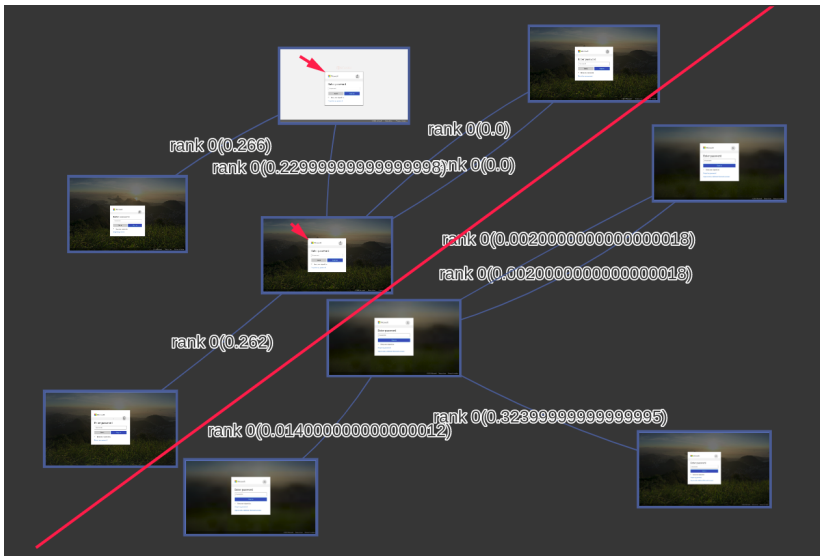


Figure 5. The dotted lines show the distances of a keypoint to outliers, while the solid lines denote the distances only between inlier matches for three feature vectors: BRIEF, steered BRIEF (Section 4.1), and rBRIEF (Section 4.3).

So more correct is :

```
double dist_th = 64;
for( int i = 0; i < descriptors_object.rows; i++ )
{ if( matches[i].distance < dist_th )
 { good_matches.push_back( matches[i]); }
}
```

Figure 4.8: Threeshold to use from [And,]



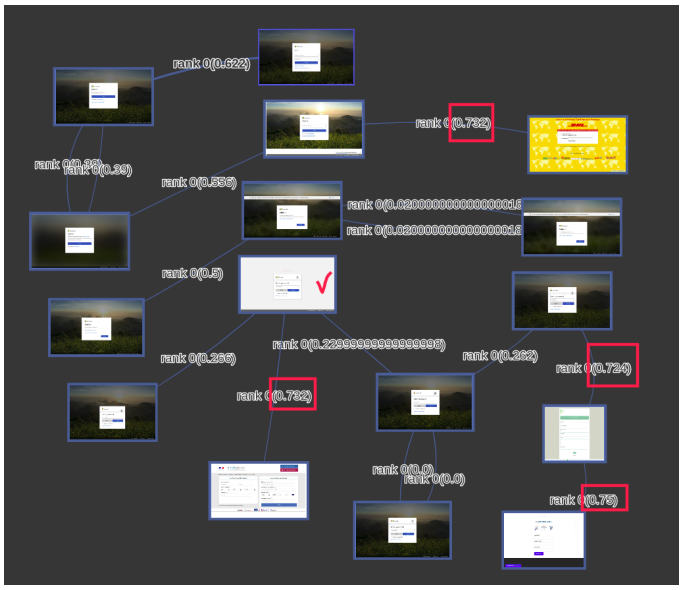
(a) Good Microsoft matching despite color change.

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

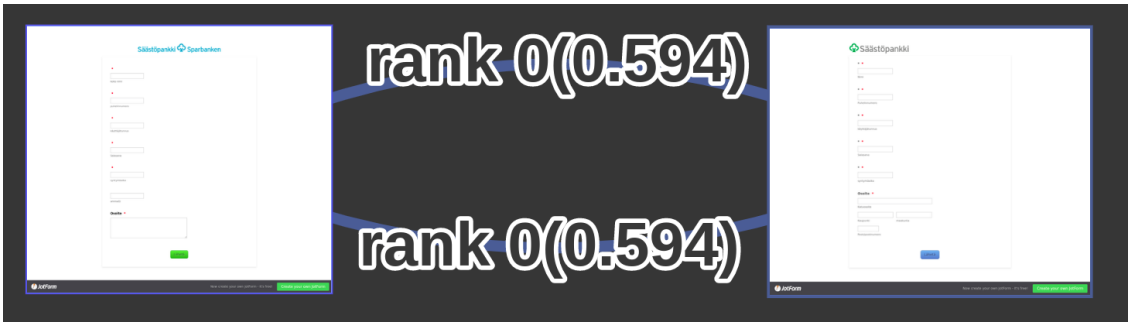
(c) No descriptor problem : pictures without descriptor should be handled differently (here : fixed distance, still matching to the last evaluated picture)

Figure 4.9: Results - ORB - min version

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



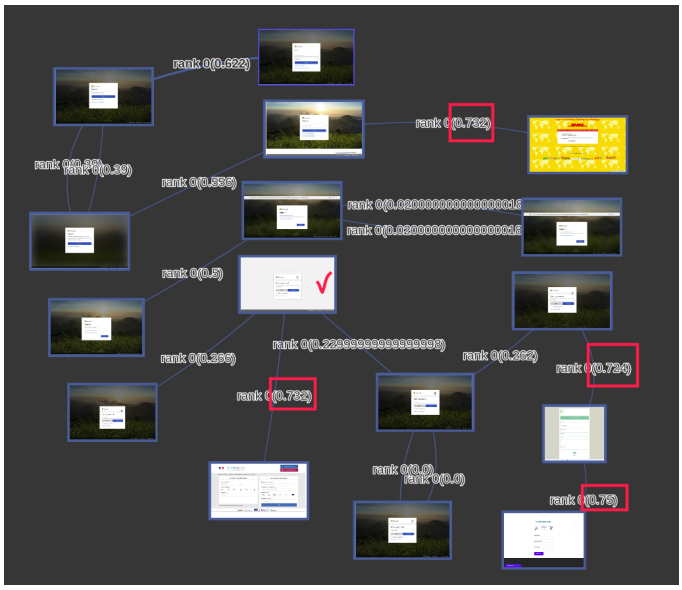
(b) Good matching on brand even with disturbed background



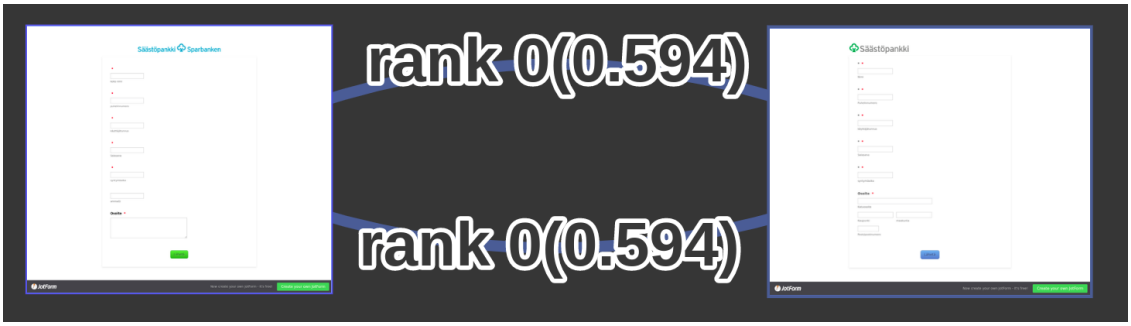
(c) Good matching on brand

Figure 4.10: Results - ORB - max version

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



(b) Good matching on brand even with disturbed background

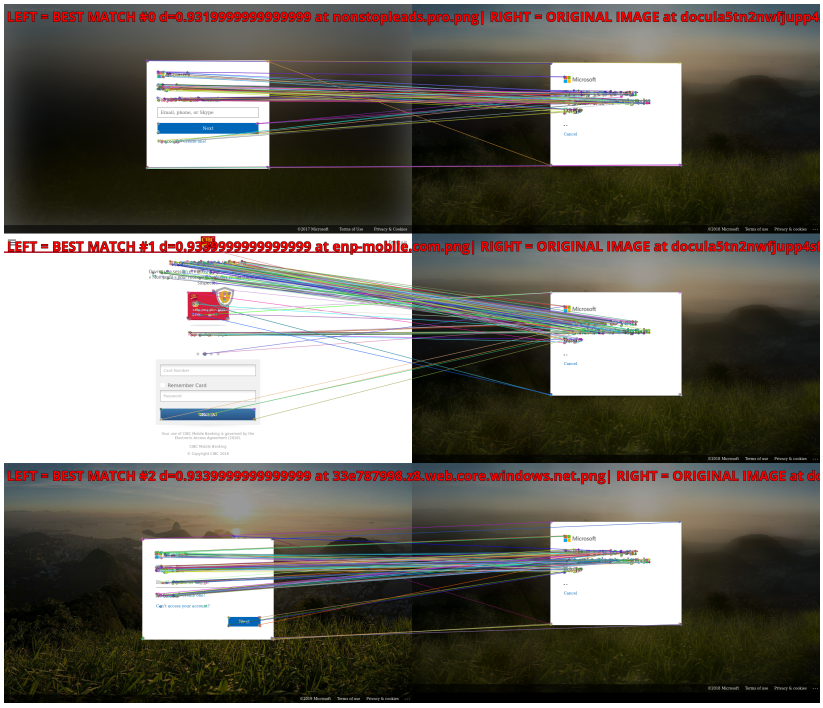


(c) Good matching on brand

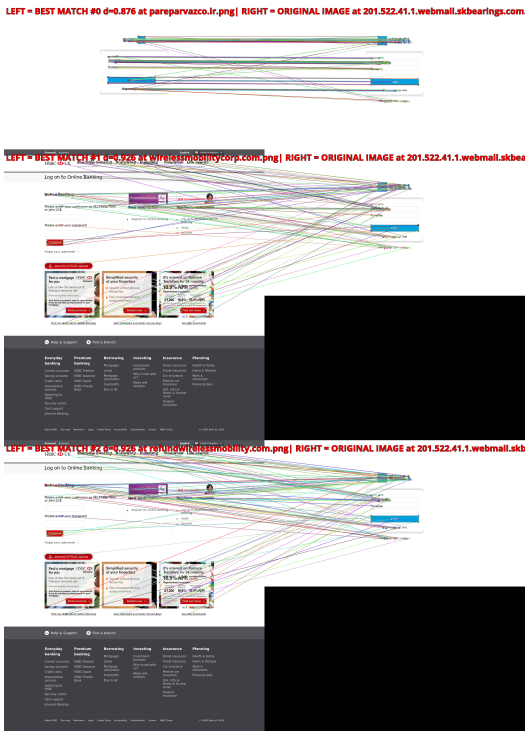
Figure 4.11: Results - ORB - max version

Following pictures are showing drawbacks of ORB algorithm. As a first "general overview" of some matching results, few examples with commentary are giv en below. Tests were performed in "ratio" configuration, however, drawbacks are generalized to other submethods.
Few tips to analyze following pictures :

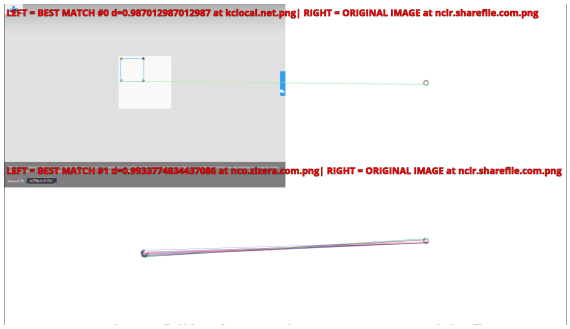
- **Parrallel lines** (if there is not rotation) are a indicator of quality matching. It keeps the spatial consistency between source and candidate pictures.
- **Text** seems to be a problem. Letters are matched to letters, generating false positive. It also "uses" descriptor space (number of descriptors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



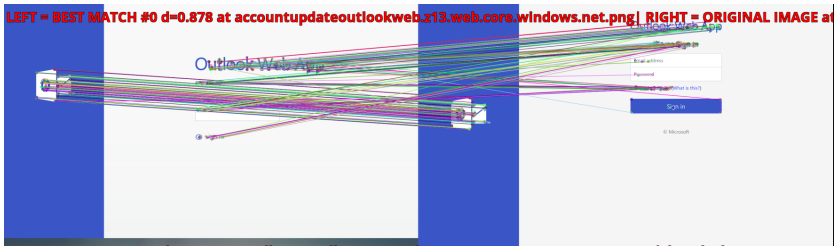
(a) Good first match, bad second match, good third match. Bad overall ranking. Text matching problem.



(b) Good and clear first match

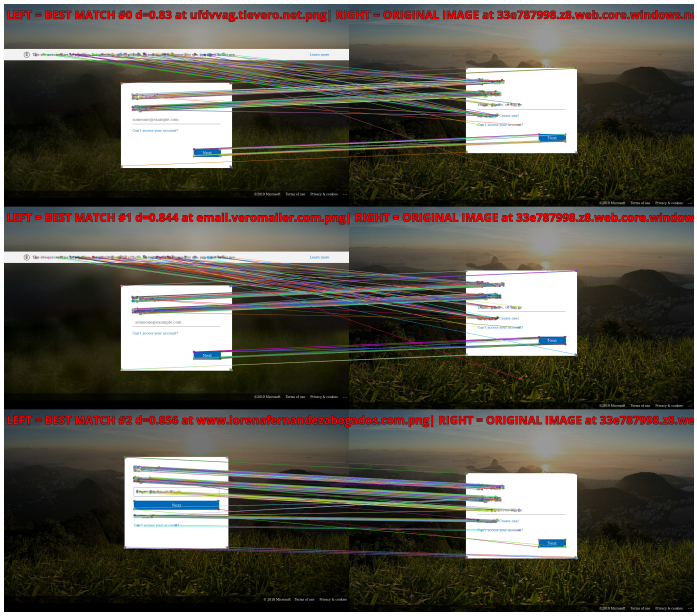


(c) True edge case : a header on a "almost-white" page does not match with the original "almost white-page"

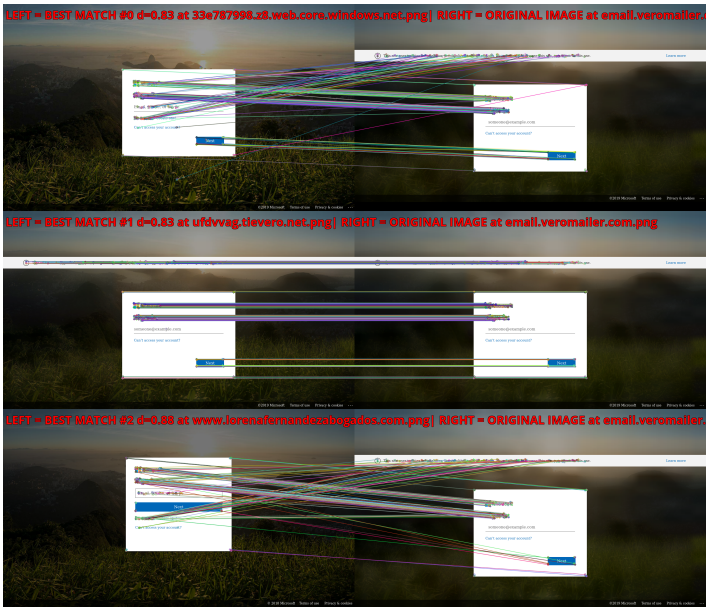


(d) Good matching on brand even with disturbed background

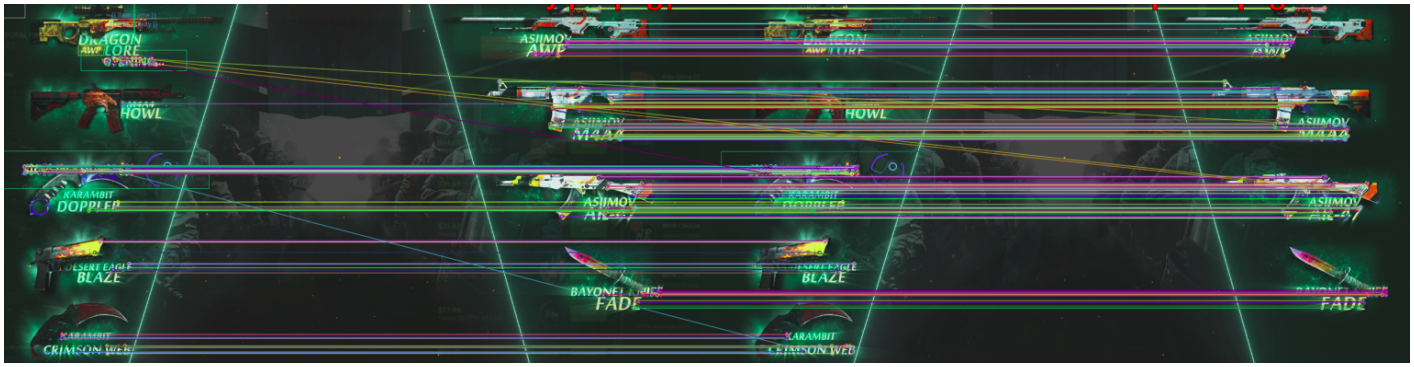
Figure 4.12: Results - ORB - drawbacks examples 1/3



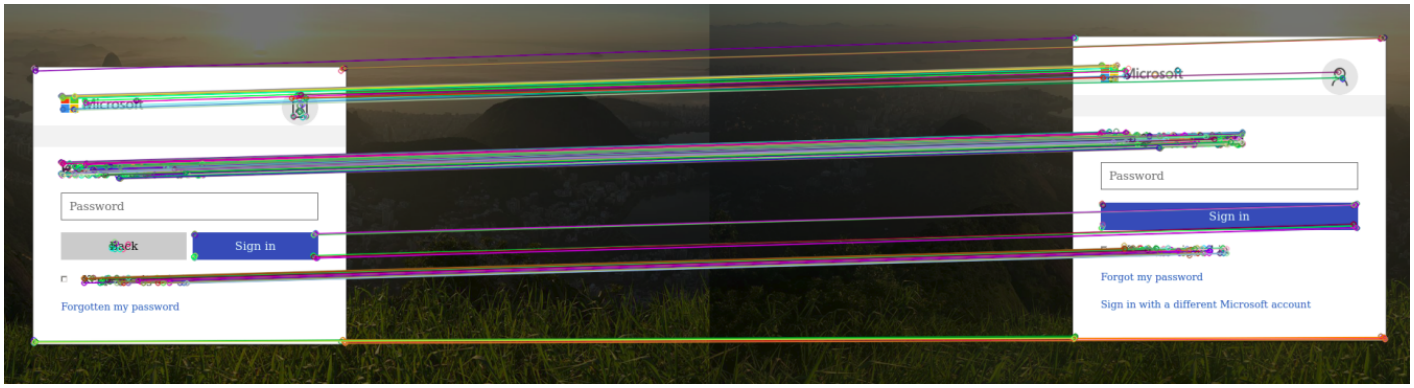
(a) Good parrallel matching on a microsoft form



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



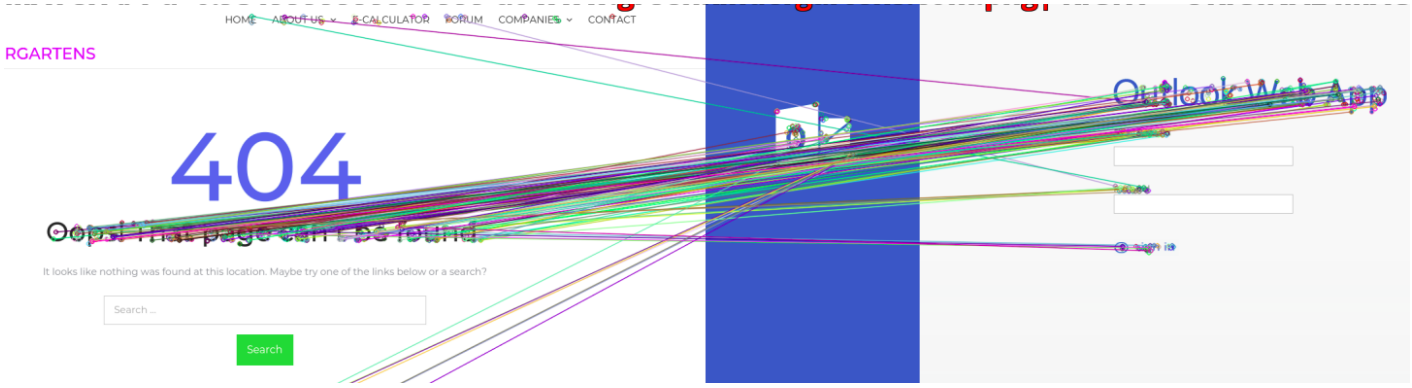
(c) Specific Typo and pictures allow a good matching



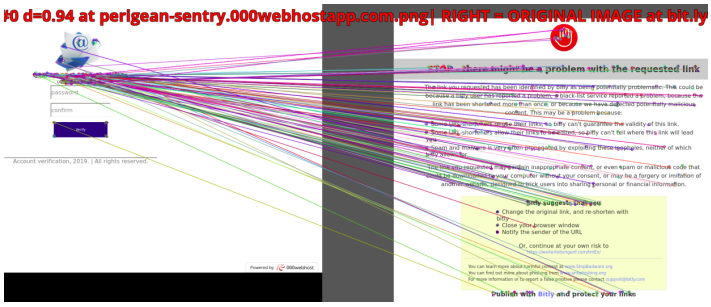
(d) Perfect match on Microsoft form

Figure 4.13: Results - ORB - drawbacks examples 2/3

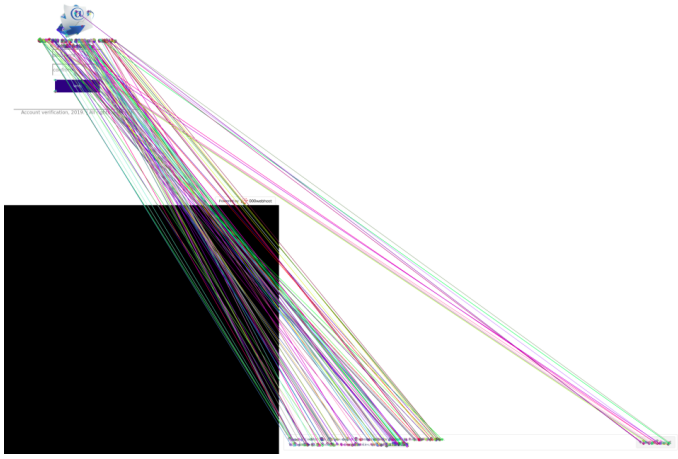
(a) Logo are matching, form text is mismatching with title text



(b) Mismatch due to text and font matching



(c) Glaring mismatch due to long text



(d) Glaring mismatch due to text only presence

Figure 4.14: Results - ORB - drawbacks examples 3/3

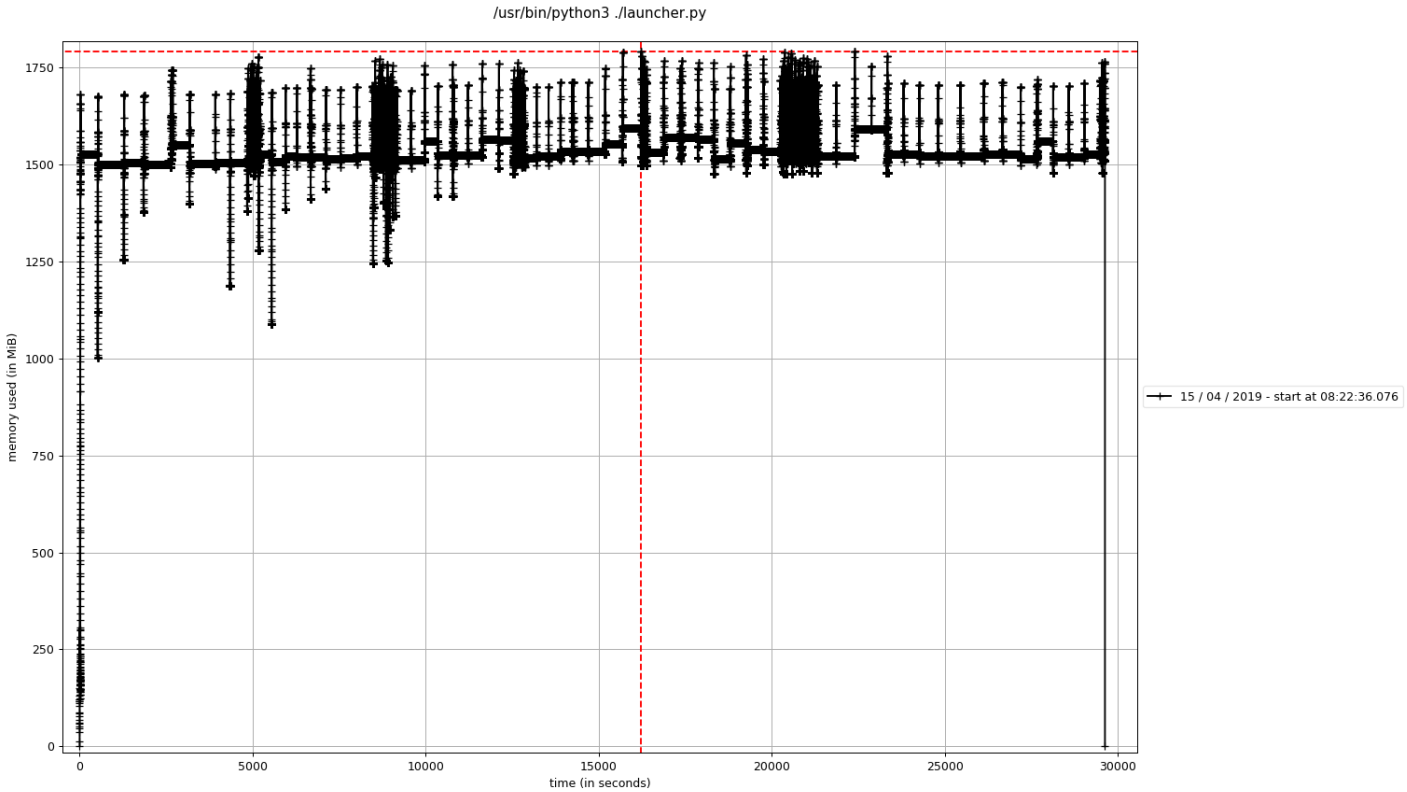


Figure 4.15: Memory consumption of ORB. 84 Mo of pictures are loaded and kept in memory for debug and output purposes. Each spike is a different ORB configuration trial. About 60 configurations are successfully tested. Overhead of the framework is displayed page 103 Figure 7.4

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.

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 detail are selected to ensure a high variance.

4.5 Feature matching

PSO

From ... few words in [Nurnajmin Qasrina et al., 2018] To detail
Kind of algo ?

SKF

From [Nurnajmin Qasrina et al., 2018]
Faster than PSO. To detail
Kind of algo ?

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.
To detail
Kind of algo ?

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

Fast Spectral Ranking

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

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.

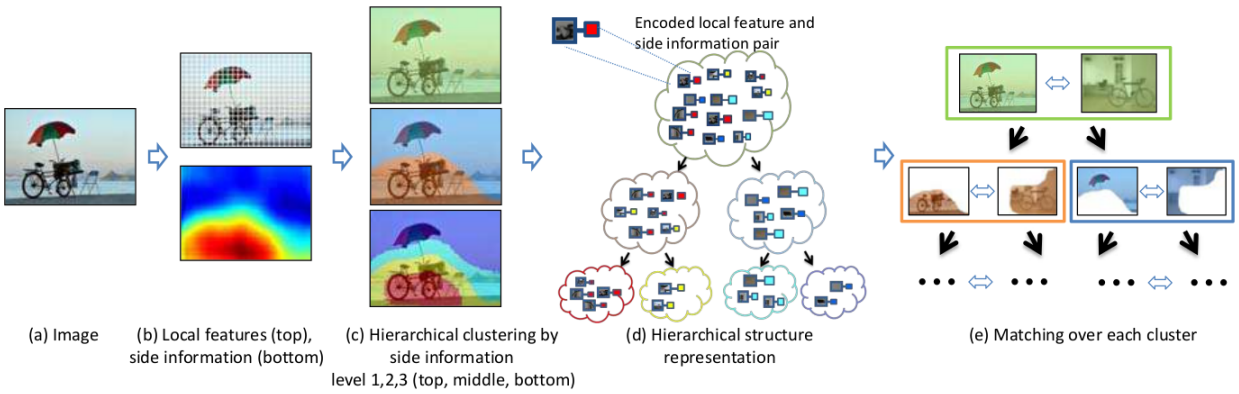


Figure 4.16: Hierarchical Hashing as showed in [Chen et al., 2012]

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.

4.6 Image matching

To detail
Kind of descriptor ?

Chapter 5

Algorithms combination

Few sources expose algorithms combination for specific goals. e.g. [Asudeh,], [Phi,], [Phi, 2016]

5.1 Phishing

Classification of approaches :

- Non-content based methods : URL, whois, ..
- Content-based methods : text, images, HTML, js, css, ...
  Sensitive to image injection instead of DOM, of invisible characters, ...
- Visual similarity and image processing methods
- Hybrid methods

"scale and rotation invariance [...] are rarely seen in phishing pages because the pages must be very similar to the corresponding authentic pages to deceive unsuspecting users. " [Chen et al., 2009]

5.1.1 Visual similarity - Histogramm

"Web pages [...] usually contain fewer colors than paintings or photographs [and] have similar color distributions" from [Chen et al., 2009]. They also state that add banners add a high amount of "color histogram" noise.

5.1.2 Visual similarity - Hash

5.1.3 Visual similarity - Keypoints

[Chen et al., 2009] is presenting keypoints based phshing website detection named L-CCH. L-CCH uses Harris-Laplacian corners detection on gray scale pictures, and store keypoint as Contrast Context Histograms. They also highlight "to judge two Web pages'similarity, we must consider [keypoints] spatial distribution, or locations.". Then, they also cluster keypoints in areas, and perform area matching. Each cluster of keypoints is evaluated to each candidate cluster of keypoints. Their solution seems not to handle rotation and scaling variation.

[Chen et al., 2010] is merging structures and layout to "higher level" structures before perform matching. They report %95 true positive rate with less than %1.7 false positive

Upon my request, I got the following information.

Chun-Rong Huang mainly contributed the L-CCH part in the paper. Thus, he can only show me what he knows for the implementation. The clustering is computed based on the spatial positions (x, y) of keypoints. They have done experiments on the phishing pages. The k-means method provides good results in this case and is easy to implement. Given the number of main clusters in this case, performance seems not to be a problem. If I'm interested in the assumption of unknown number of clusters, please refer to affinity propagation. To look forward to the implementation of the phishing page detection, Dr. Kuan-Ta Chen or Jau-Yuan Chen should be contacted.

[Grycuk, 2016] is presenting in the same way an idea to cluster keypoints with Kmeans.

[Leibe et al., 2006] describe a performance improvement uppon kmeans based feature clustering. It describes the structure of cluster to adopt to get better matching performances.

[Afroz and Greenstadt,] uses a two passes method, based on metadata (SSL, URL, .. 49% to 66.78% detection rate) and then SIFT keypoint matching. It reported %90 of accuracy with less than %0.5 of false positives.

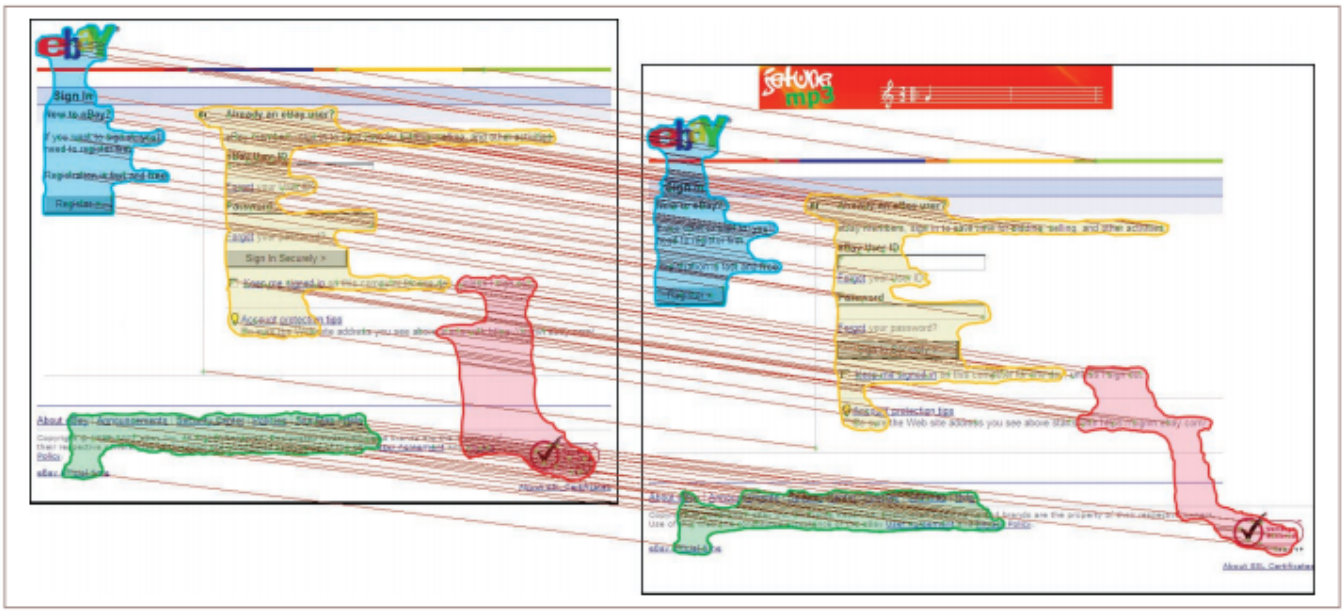


Figure 5. Clustering and matching of eBay's official page and a phishing page. Different clusters are circled in different colors.

Figure 5.1: L-CCH idea : Harris based + Clustering [Chen et al., 2009]

*Table I. The top five phishing target sites.**

Sites	Number of records	Correct rate (%)	False-negative rate (%)	False-positive rate (%)
eBay	701	96.8	0.0	0.1
PayPal	632	97.7	0.0	0.1
Marshall & Ilsley Bank	138	97.7	0.0	0.1
Charter One Bank	116	98.0	0.0	0.1
Bank of America	51	95.4	2.0	2.1

*Total number of phishing target pages: 300 pages in 74 sites

Figure 5.2: L-CCH results : Harris based + Clustering [Chen et al., 2009]

[Zhou et al., 2010] uses a spatial postprocessing algorithm upon SIFT, to remove matches that are not relevant, if not "spatially consistent" with source picture. [Zhou et al., 2013] seems to use a similar idea.

5.1.4 Visual similarity - Text handling

Text is an issue in screenshots matching. On one hand, it provides information that can be used to match pictures together, and on the other hand it creates keypoints that can be too easily matched, which create false positive.

Therefore, some solutions or their combinations are possible :

- Remove text / Black rectangle (or arbitrary color)
- Remove text / Background color rectangle
- Remove text / Blur
- Extract text and use it in the matching

Remove text : EAST - Efficient and Accurate Scene Text detection / deep learning text detector

[Rosebrock, 2018] provides some details about how to detect text at scale, as the presented solution handle text recognition on real-time video input.

Tests were conducted with this DeeplLearning text detector. The process is the following :

- Pictures are loaded
- Text location are detected with the provided model
- Text areas are filled (background color detected with different ways, fixed color, ...)
- Pictures are saved
- Standard algorithms are called on this new dataset

Results are interesting, as pictures were heavily modified. In few words, fuzzy hashing algorithms results are improved by approximately 5% and ORB results are decreased by 5%.

Remove text : Tesseract / Bundled text detector

Tesseract [Lee,] with its python wrapper PyTesseract is a Google OCR library. It allows to retrieve detected text as well as getting bounding boxes of detected text areas.

However, such solution provide false positive, some of which are easily prunnable and some of which are very hard to remove.

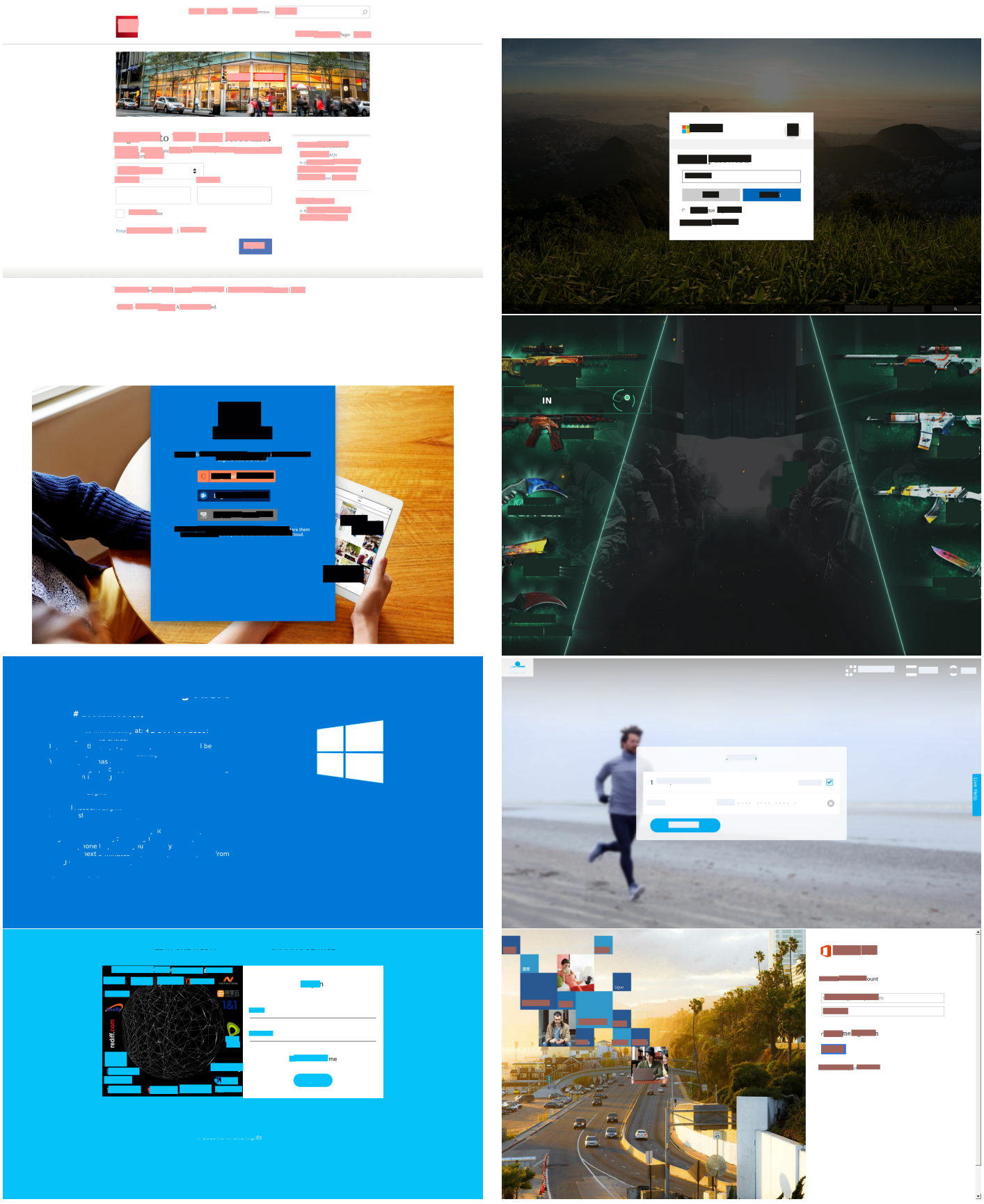
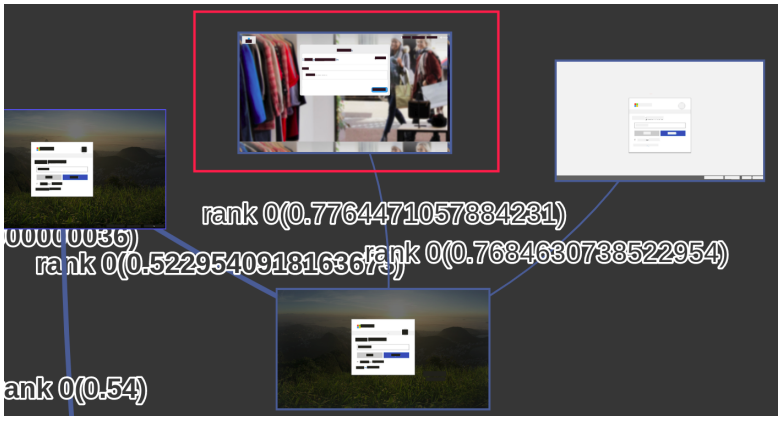


Figure 5.3: Pictures with painted text

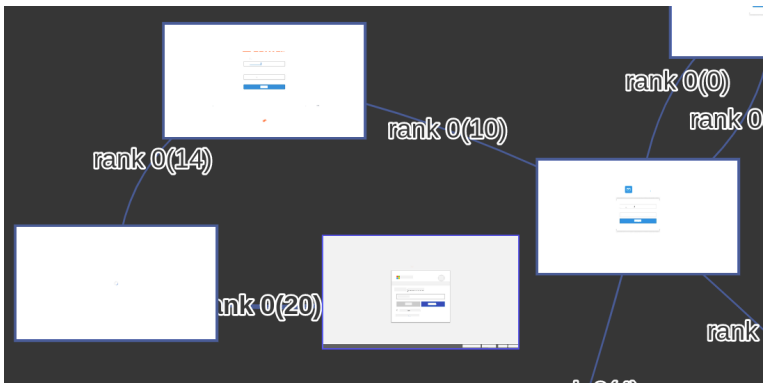


(a) Microsoft landing page linked to another form page

(b) Google landing page linked to another form page

Figure 5.4: ORB results on painted pictures (Bruteforce matcher)

(a) Mismatch between two different forms



(b) Mismatch between microsoft and another form

Figure 5.5: P-HASH results on painted pictures

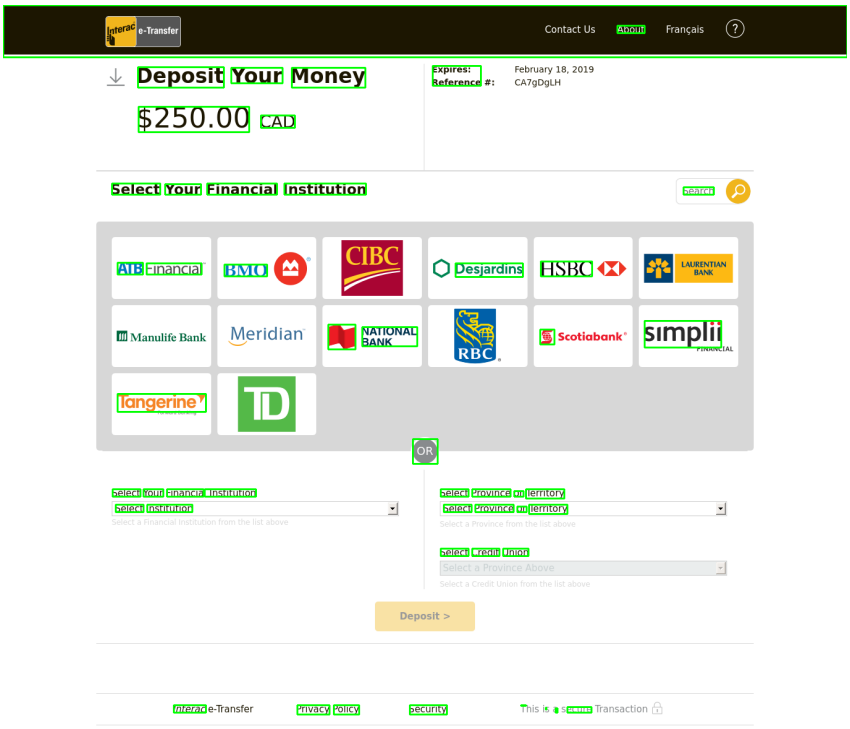


Figure 5.6: Logos have similar bounding boxes to text ones.

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.

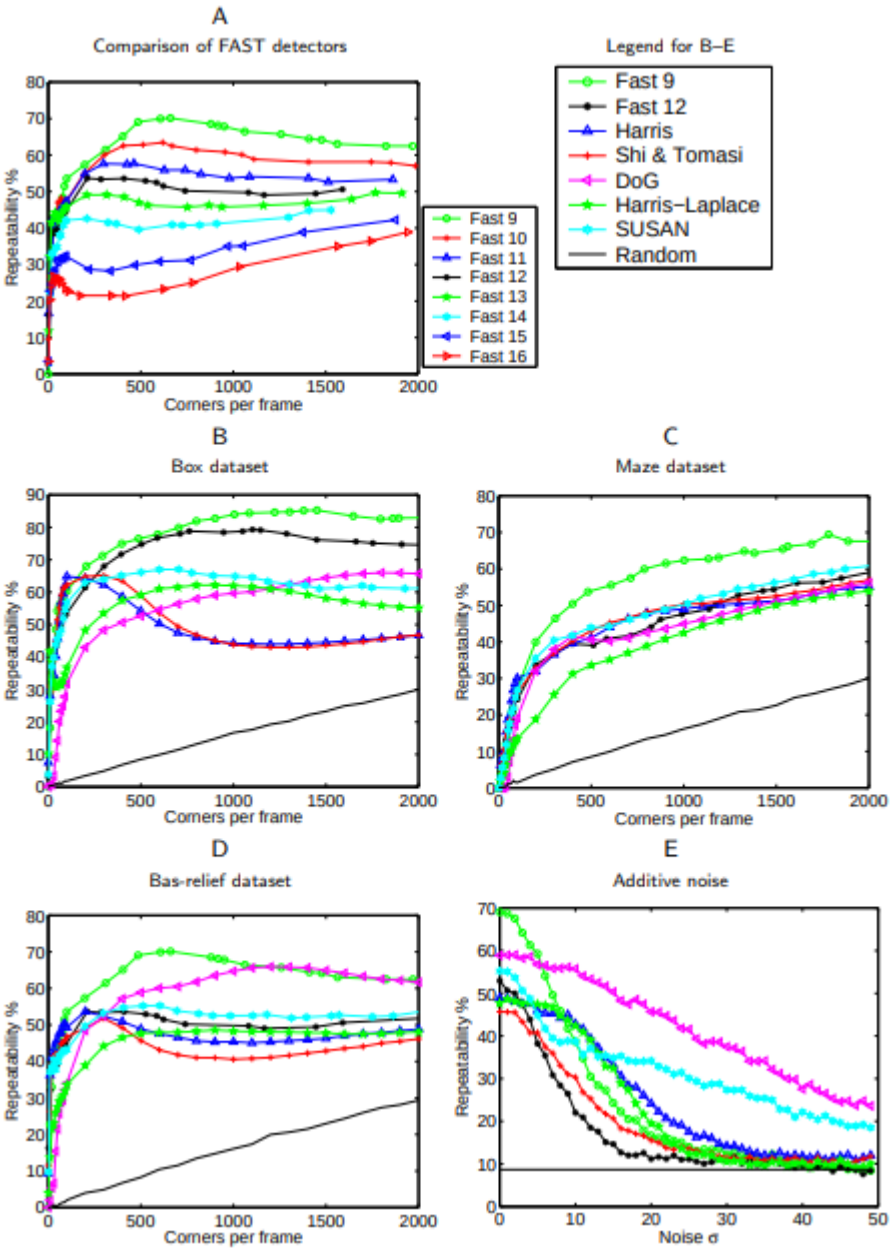


Figure 6.1: Corner detector from [Rosten and Drummond, 2006]

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.

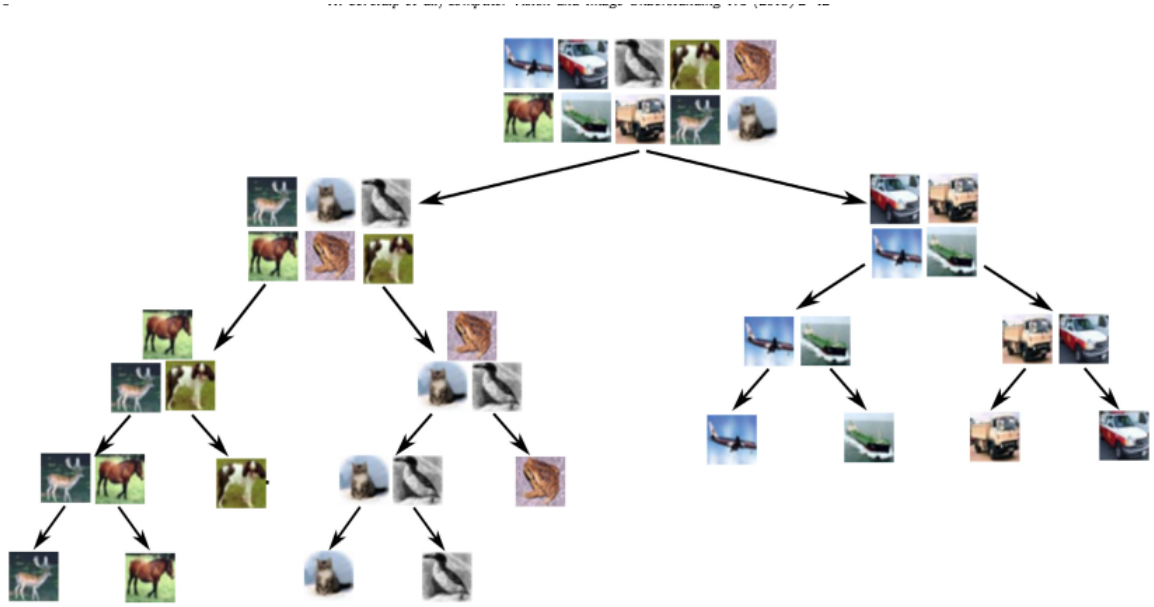


Fig. 2. Binary hierarchical tree obtained for CIFAR10 dataset using convex hull modeling of the classes. Each image represents an object class where it comes from.

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.

6.9 SVM - Support Vector Machine

Chapter 7

Utility algorithms

7.0.1 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.

7.0.2 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 $O(N)$)

7.1 Segmentation

7.1.1 SLICO - Simple Linear Iterative Clustering

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

[SLI,] describes a "cutting" method.

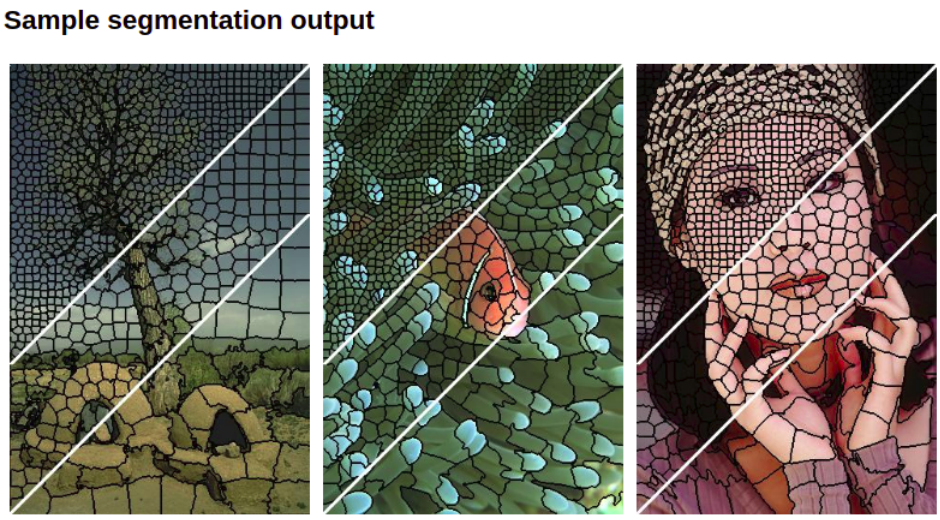


Figure 7.1: Sample Segmentation from [SLI,]

7.1.2 HSNW - ... indexing

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

7.2 Unsorted

7.2.1 Block-based approach + KeyPoint approach for Image manipulation

From [Raj and Joseph, 2016]

7.2.2 Scalable cluster discovery

A scalable approach of clustering is provided and described in [Chum and Matas, 2010]. This is very close to a graph-approach.

7.3 Raw results

Each algorithm produce a guess, which can be represented as a graphe between all input pictures. Each picture is linked to at least one other picture : its best match. This*result graph* is evaluated, thanks to the baseline graphe (ground truth graphe, hand made). The ratio of the intersection of the *result graph* thanks to the baseline graphe, divided by the original size of the *result graph* is equivalent to an "intersection ratio".

$$Intersection\_ratio = \frac{\sharp(result\_graphe\_edges \cap ground\_truth\_graphe\_edges)}{\sharp(result\_graphe\_edges)} \tag{7.1}$$

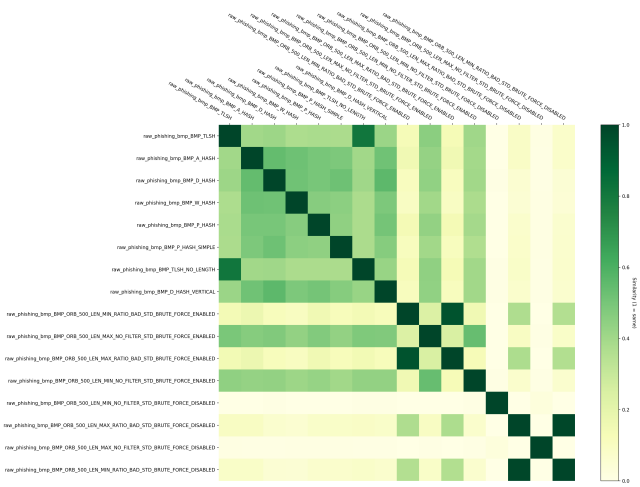
The same can be conducted between *result graph* themselves. An intersection ratio of each graphe thanks to each other graphe can be computed. It is an approximate measure of "similarity" between the algorithms outputs.

$$Output\_similarity \approx Intersection\_ratio = \frac{\sharp(result\_graphe\_edges_A \cap result\_graphe\_edges_B)}{\sharp(result\_graphe\_edges_A)} \tag{7.2}$$

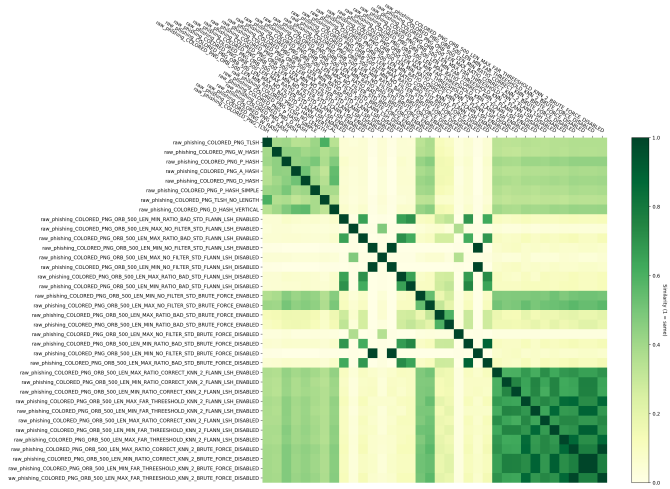
Results of this calculation are provided in Figure 7.2

Algoritms outputs can be paired. Their graphs can be merged into a uniq graph. This graph can me evaluated in the same way as any algorithm output.

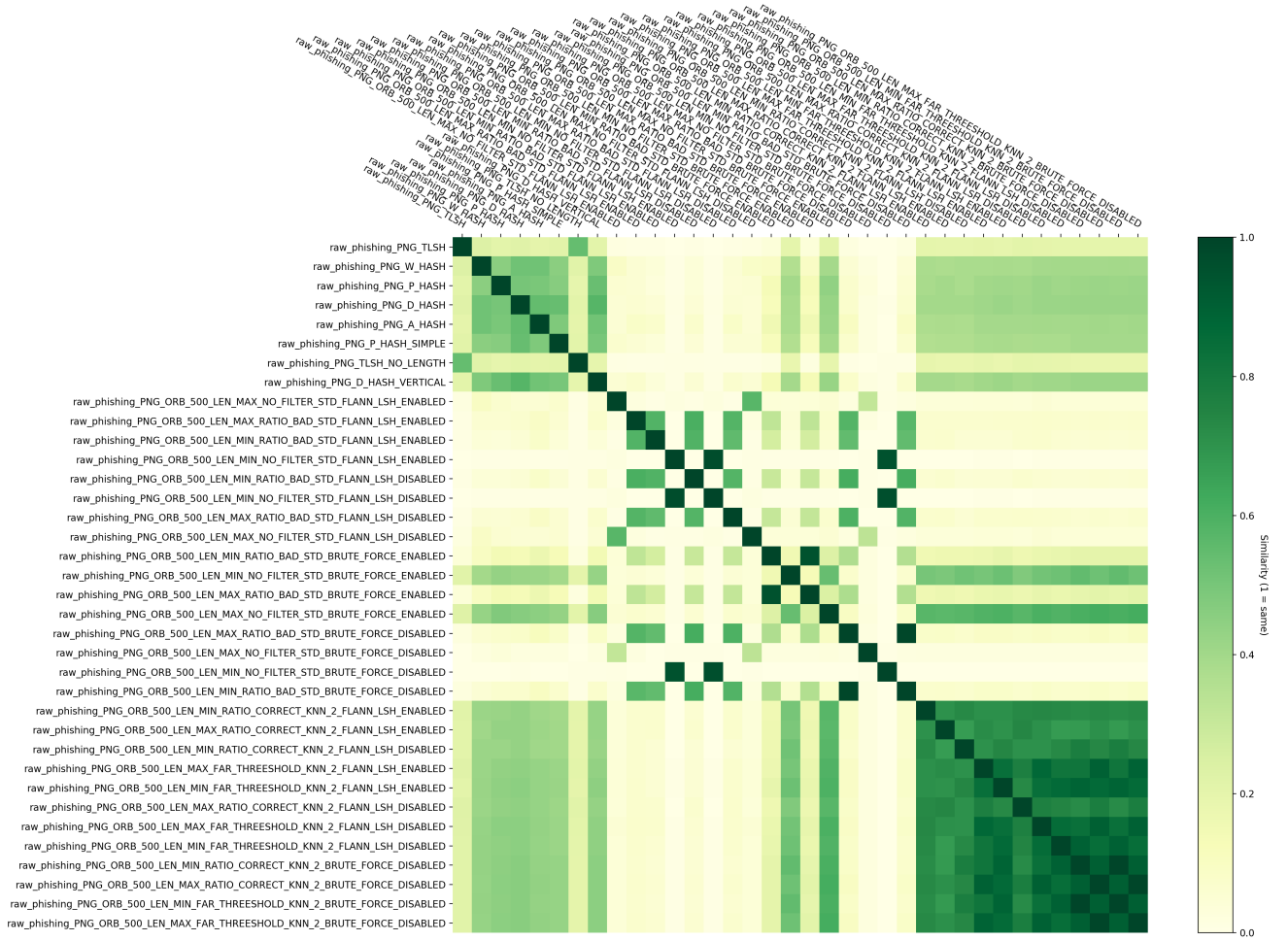
Results of this calculation are provided in Figure 7.3



(a) Algo intersection of BMP pictures

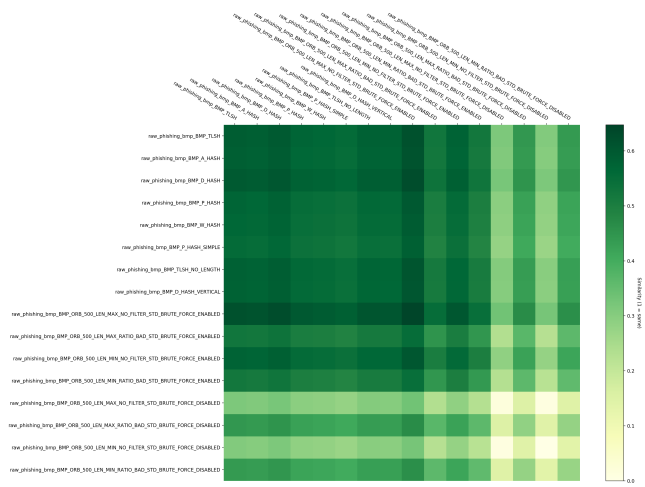


(b) Algo intersection of COLORED pictures

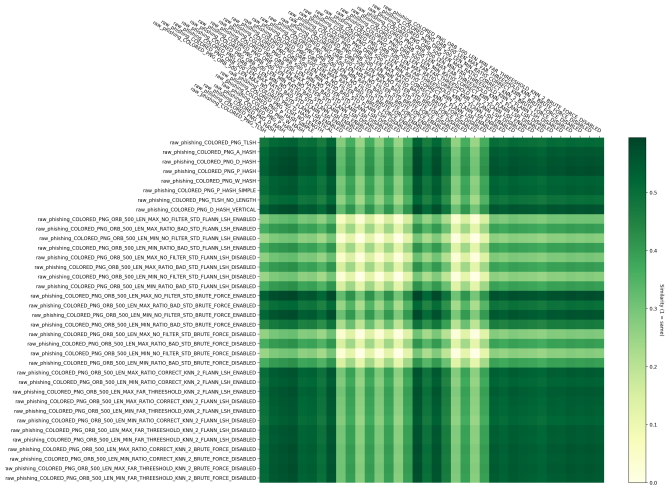


(c) Algo intersection of PNG pictures

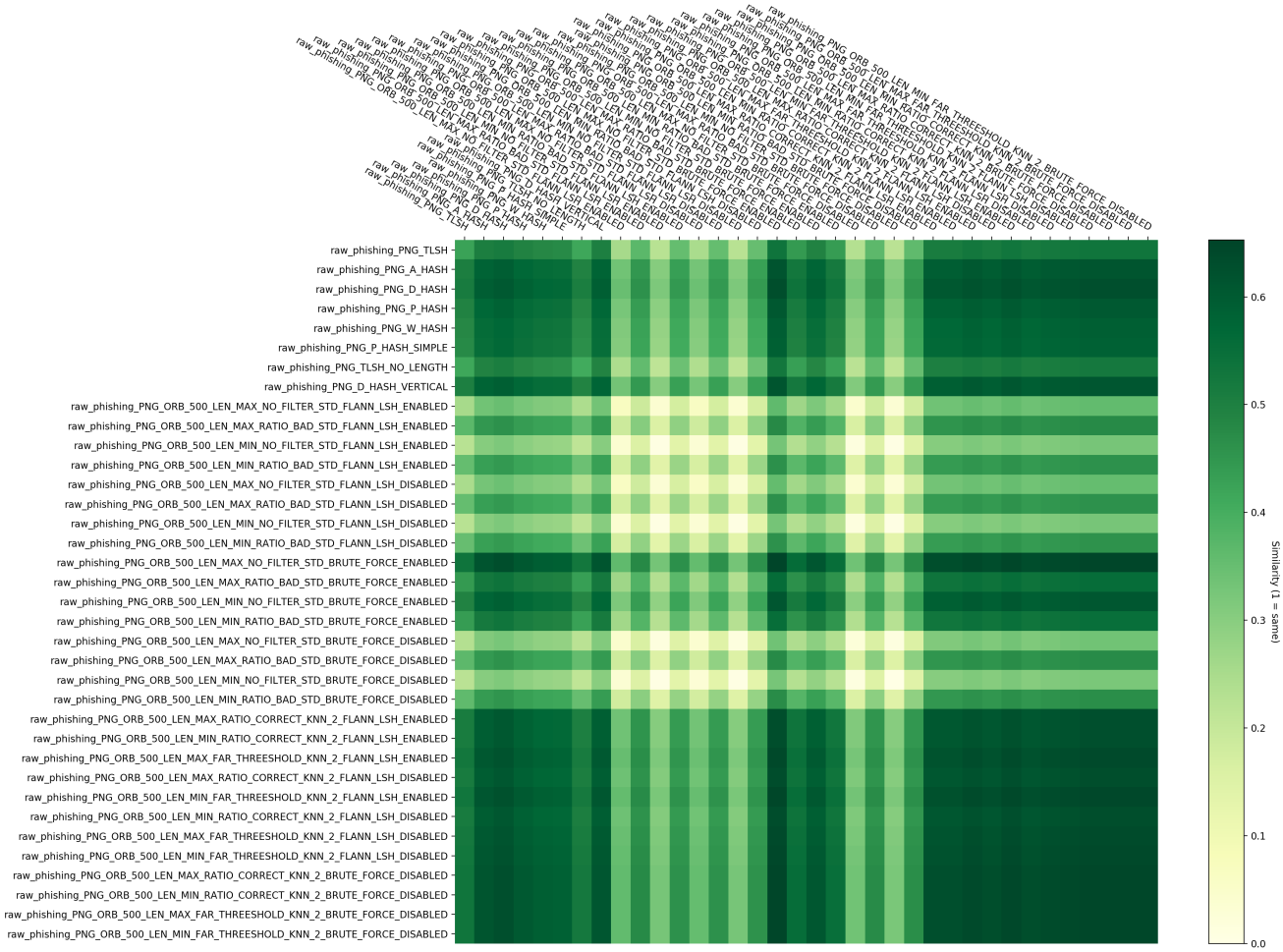
Figure 7.2: Intersection matrix - Guess of rank 1 for each image comparison



(a) Algorithms pairing - BMP pictures - true-positive rate



(b) Algorithms pairing - Colored pictures - true-positive rate



(c) Algorithms pairing - PNG pictures - true-positive rate

Figure 7.3: By pair true-positive matrix - Combination of algorithms per pair and their score (Please note that the legend saying it's "similarity" is wrong)

7.3.1 Phishing dataset / Hashing based

NAME	TRUE POSITIVE	PRE COMPUTING (s)	MATCHING (s)
raw phishing PNG ORB 500 LEN MAX NO FILTER STD BRUTE FORCE ENABLED	0.65263	0.04624	1.9087
raw phishing PNG ORB 500 LEN MIN FAR THREESHOLD KNN 2 BRUTE FORCE DISABLED	0.64737	0.06641	1.23615
raw phishing PNG ORB 500 LEN MIN RATIO CORRECT KNN 2 BRUTE FORCE DISABLED	0.64737	0.05256	1.17277
raw phishing PNG ORB 500 LEN MAX RATIO CORRECT KNN 2 BRUTE FORCE DISABLED	0.64737	0.04775	1.34641
raw phishing PNG ORB 500 LEN MAX FAR THREESHOLD KNN 2 BRUTE FORCE DISABLED	0.64737	0.06434	1.20571
raw phishing PNG ORB 500 LEN MIN FAR THREESHOLD KNN 2 FLANN LSH ENABLED	0.64211	0.04634	1.1203
raw phishing PNG ORB 500 LEN MIN FAR THREESHOLD KNN 2 FLANN LSH DISABLED	0.63684	0.04614	1.12314
raw phishing PNG ORB 500 LEN MAX FAR THREESHOLD KNN 2 FLANN LSH ENABLED	0.63158	0.05965	1.26475
raw phishing PNG ORB 500 LEN MAX FAR THREESHOLD KNN 2 FLANN LSH DISABLED	0.62632	0.04613	1.12695
raw phishing PNG ORB 500 LEN MIN RATIO CORRECT KNN 2 FLANN LSH DISABLED	0.61053	0.04764	1.19758
raw phishing PNG ORB 500 LEN MAX RATIO CORRECT KNN 2 FLANN LSH DISABLED	0.61053	0.04769	1.13552
raw phishing PNG ORB 500 LEN MIN RATIO CORRECT KNN 2 FLANN LSH ENABLED	0.60526	0.04787	1.14532
raw phishing PNG ORB 500 LEN MAX RATIO CORRECT KNN 2 FLANN LSH ENABLED	0.60526	0.04742	1.11723
raw phishing PNG D HASH	0.60386	0.06124	0.00214
raw phishing PNG A HASH	0.57971	0.06268	0.00218
raw phishing PNG D HASH VERTICAL	0.57488	0.05934	0.00216
raw phishing PNG ORB 500 LEN MIN NO FILTER STD BRUTE FORCE ENABLED	0.56842	0.0452	2.08476
raw phishing PNG P HASH	0.56039	0.06403	0.00244
raw phishing PNG W HASH	0.53623	0.1221	0.00205
raw phishing PNG P HASH SIMPLE	0.52657	0.06377	0.00222
raw phishing PNG ORB 500 LEN MAX RATIO BAD STD BRUTE FORCE ENABLED	0.46316	0.0445	1.68615
raw phishing PNG ORB 500 LEN MIN RATIO BAD STD BRUTE FORCE ENABLED	0.45263	0.0445	2.15439
raw phishing PNG TLSH	0.42512	0.00498	0.00096
raw phishing PNG TLSH NO LENGTH	0.4058	0.00494	0.0011
raw phishing PNG ORB 500 LEN MAX RATIO BAD STD FLANN LSH ENABLED	0.30526	0.05724	1.50447
raw phishing PNG ORB 500 LEN MAX RATIO BAD STD BRUTE FORCE DISABLED	0.3	0.07377	1.17801
raw phishing PNG ORB 500 LEN MIN RATIO BAD STD BRUTE FORCE DISABLED	0.28947	0.04555	0.99388
raw phishing PNG ORB 500 LEN MAX RATIO BAD STD FLANN LSH DISABLED	0.27368	0.14438	1.56765
raw phishing PNG ORB 500 LEN MIN RATIO BAD STD FLANN LSH ENABLED	0.27368	0.05961	1.47452
raw phishing PNG ORB 500 LEN MIN RATIO BAD STD FLANN LSH DISABLED	0.26842	0.04805	1.17483
raw phishing PNG ORB 500 LEN MAX NO FILTER STD FLANN LSH ENABLED	0.06842	0.05417	1.1589
raw phishing PNG ORB 500 LEN MAX NO FILTER STD FLANN LSH DISABLED	0.05789	0.04719	1.34306
raw phishing PNG ORB 500 LEN MAX NO FILTER STD BRUTE FORCE DISABLED	0.02105	0.0631	1.14467
raw phishing PNG ORB 500 LEN MIN NO FILTER STD FLANN LSH DISABLED	0.00526	0.05679	1.18021
raw phishing PNG ORB 500 LEN MIN NO FILTER STD FLANN LSH ENABLED	0.00526	0.04684	1.08095
raw phishing PNG ORB 500 LEN MIN NO FILTER STD BRUTE FORCE DISABLED	0.0	0.04518	0.95656

NAME	TRUE POSITIVE	PRE COMPUTING (sec)	MATCHING (sec)
raw phishing COLORED PNG ORB 500 LEN MAX NO FILTER STD BRUTE FORCE ENABLED	0.59474	0.04536	1.5378
raw phishing COLORED PNG P HASH	0.58937	0.04627	0.00233
raw phishing COLORED PNG D HASH VERTICAL	0.57488	0.04983	0.00236
raw phishing COLORED PNG D HASH	0.57488	0.04925	0.00301
raw phishing COLORED PNG ORB 500 LEN MIN NO FILTER STD BRUTE FORCE ENABLED	0.56842	0.0739	2.11373
raw phishing COLORED PNG ORB 500 LEN MAX RATIO CORRECT KNN 2 BRUTE FORCE DISABLED	0.55789	0.0439	0.86356
raw phishing COLORED PNG ORB 500 LEN MAX FAR THREESHOLD KNN 2 BRUTE FORCE DISABLED	0.55789	0.0439	0.86493
raw phishing COLORED PNG A HASH	0.55072	0.04406	0.00229
raw phishing COLORED PNG ORB 500 LEN MIN RATIO CORRECT KNN 2 BRUTE FORCE DISABLED	0.54737	0.04379	0.85381
raw phishing COLORED PNG ORB 500 LEN MIN FAR THREESHOLD KNN 2 BRUTE FORCE DISABLED	0.54737	0.04405	0.85917
raw phishing COLORED PNG ORB 500 LEN MAX FAR THREESHOLD KNN 2 FLANN LSH ENABLED	0.54737	0.04403	1.08197
raw phishing COLORED PNG ORB 500 LEN MAX FAR THREESHOLD KNN 2 FLANN LSH DISABLED	0.54211	0.04388	1.084
raw phishing COLORED PNG ORB 500 LEN MAX RATIO CORRECT KNN 2 FLANN LSH ENABLED	0.53684	0.04413	1.08097
raw phishing COLORED PNG ORB 500 LEN MIN FAR THREESHOLD KNN 2 FLANN LSH DISABLED	0.53684	0.0441	1.07795
raw phishing COLORED PNG ORB 500 LEN MAX RATIO CORRECT KNN 2 FLANN LSH DISABLED	0.53158	0.0439	1.08247
raw phishing COLORED PNG P HASH SIMPLE	0.52657	0.04703	0.00298
raw phishing COLORED PNG ORB 500 LEN MIN FAR THREESHOLD KNN 2 FLANN LSH ENABLED	0.52105	0.04423	1.08245
raw phishing COLORED PNG W HASH	0.51208	0.11247	0.00229
raw phishing COLORED PNG ORB 500 LEN MIN RATIO CORRECT KNN 2 FLANN LSH ENABLED	0.51053	0.04389	1.0917
raw phishing COLORED PNG ORB 500 LEN MIN RATIO CORRECT KNN 2 FLANN LSH DISABLED	0.5	0.0438	1.08247
raw phishing COLORED PNG TLSH	0.47826	0.00513	0.00096
raw phishing COLORED PNG TLSH NO LENGTH	0.45411	0.00508	0.00111
raw phishing COLORED PNG ORB 500 LEN MAX RATIO BAD STD BRUTE FORCE ENABLED	0.44737	0.0706	2.30236
raw phishing COLORED PNG ORB 500 LEN MIN RATIO BAD STD BRUTE FORCE ENABLED	0.38947	0.0676	2.39523
raw phishing COLORED PNG ORB 500 LEN MIN RATIO BAD STD FLANN LSH ENABLED	0.24211	0.04551	1.05413
raw phishing COLORED PNG ORB 500 LEN MAX RATIO BAD STD FLANN LSH ENABLED	0.24211	0.04491	1.04322
raw phishing COLORED PNG ORB 500 LEN MAX RATIO BAD STD FLANN LSH DISABLED	0.24211	0.04502	1.05124
raw phishing COLORED PNG ORB 500 LEN MIN RATIO BAD STD FLANN LSH DISABLED	0.24211	0.04556	1.05038
raw phishing COLORED PNG ORB 500 LEN MIN RATIO BAD STD BRUTE FORCE DISABLED	0.23684	0.04937	1.31014
raw phishing COLORED PNG ORB 500 LEN MAX RATIO BAD STD BRUTE FORCE DISABLED	0.23684	0.05403	1.20556
raw phishing COLORED PNG ORB 500 LEN MAX NO FILTER STD FLANN LSH ENABLED	0.05789	0.04412	0.99475
raw phishing COLORED PNG ORB 500 LEN MAX NO FILTER STD FLANN LSH DISABLED	0.03684	0.04388	0.99439
raw phishing COLORED PNG ORB 500 LEN MAX NO FILTER STD BRUTE FORCE DISABLED	0.02105	0.05104	0.8506
raw phishing COLORED PNG ORB 500 LEN MIN NO FILTER STD FLANN LSH DISABLED	0.0	0.0444	0.99507
raw phishing COLORED PNG ORB 500 LEN MIN NO FILTER STD BRUTE FORCE DISABLED	0.0	0.08486	1.23023
raw phishing COLORED PNG ORB 500 LEN MIN NO FILTER STD FLANN LSH ENABLED	0.0	0.04395	0.99463

NAME	TRUE POSITIVE	PRE COMPUTING (sec)	MATCHING (sec)
raw phishing bmp BMP ORB 500 LEN MAX NO FILTER STD BRUTE FORCE ENABLED	0.64737	0.07296	2.52043
raw phishing bmp BMP D HASH	0.60386	0.02017	0.00224
raw phishing bmp BMP TLSH	0.58937	0.08356	0.00093
raw phishing bmp BMP A HASH	0.57971	0.02544	0.00245
raw phishing bmp BMP TLSH NO LENGTH	0.57005	0.0856	0.00114
raw phishing bmp BMP D HASH VERTICAL	0.56522	0.01986	0.00228
raw phishing bmp BMP ORB 500 LEN MIN NO FILTER STD BRUTE FORCE ENABLED	0.55789	0.04995	2.40043
raw phishing bmp BMP P HASH	0.55072	0.02438	0.00232
raw phishing bmp BMP W HASH	0.54106	0.09857	0.00221
raw phishing bmp BMP P HASH SIMPLE	0.52174	0.02224	0.00233
raw phishing bmp BMP ORB 500 LEN MAX RATIO BAD STD BRUTE FORCE ENABLED	0.45263	0.0637	2.45745
raw phishing bmp BMP ORB 500 LEN MIN RATIO BAD STD BRUTE FORCE ENABLED	0.44211	0.06207	2.39609
raw phishing bmp BMP ORB 500 LEN MAX RATIO BAD STD BRUTE FORCE DISABLED	0.28421	0.05223	1.45008
raw phishing bmp BMP ORB 500 LEN MIN RATIO BAD STD BRUTE FORCE DISABLED	0.27368	0.06659	1.3083
raw phishing bmp BMP ORB 500 LEN MAX NO FILTER STD BRUTE FORCE DISABLED	0.01579	0.0531	1.43634
raw phishing bmp BMP ORB 500 LEN MIN NO FILTER STD BRUTE FORCE DISABLED	0.0	0.05253	1.17698

The baseline memory usage of the framework is roughly 350 Mo. Note that this has been measured with a "void" algorithm, which doesn't compute hash, descriptors or any distance but affect hardcoded values to all pictures.

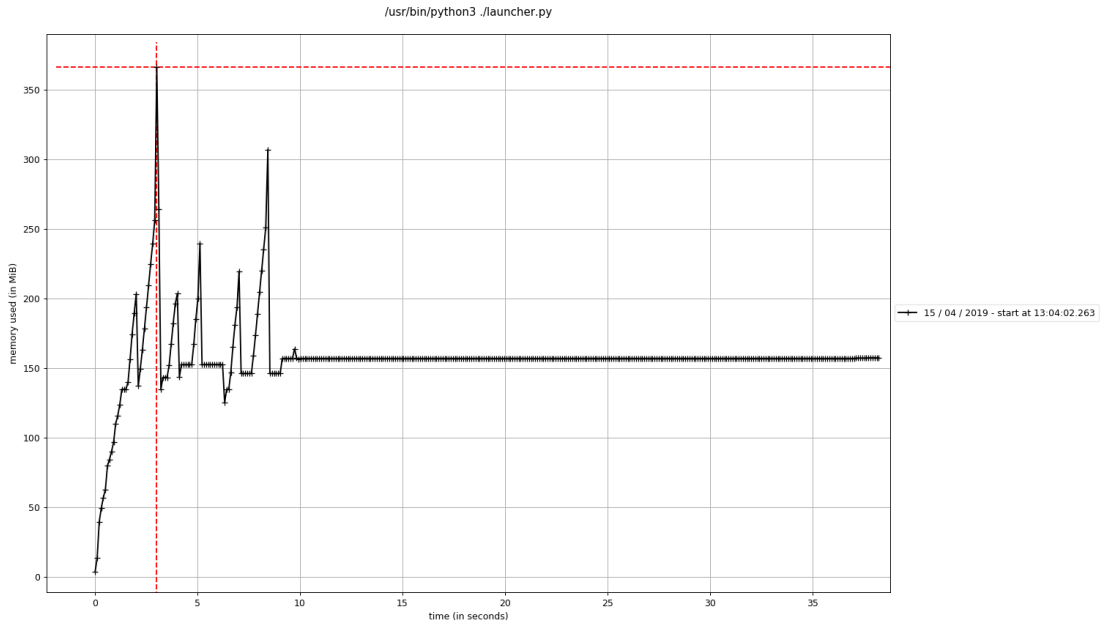


Figure 7.4: Baseline memory consumption of the test framework

Bibliography

[And,] Android - Matching ORB Features with a threshold. https://stackoverflow.com/questions/22857398/matchi orb-features-with-a-threshold.

[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/.

[SLI,] SLIC Superpixels – IVRL.

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

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

[Phi,] Toward a Phish Free World: A Cascaded Learning Framework for Phish Detection. https://pdfs.semanticscholar.org/8da7/08802d729ee110eabfb05302630a56f7c88a.pdf.

[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.

[Phi, 2016] (2016). Phishing Website Identification through Visual Clustering. https://pdfs.semanticscholar.org/1d56/fe566fff6fa4b1658cdbb0201cd4d912723e.pdf.

[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.

[Adel et al.,] Adel, E., Elmogy, M., and Elbakry, H. *Image Stitching System Based on ORB Feature-Based Technique and Compensation Blending.*

[Afroz and Greenstadt,] Afroz, S. and Greenstadt, R. *PhishZoo: An Automated Web Phishing Detection Approach Based on Profiling and Fuzzy Matching.*

[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.

[Asudeh,] Asudeh, O. A NEW REAL-TIME APPROACH FOR WEBSITE PHISHING DETECTION BASED ON VISUAL SIMILARITY. page 53.

[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., 2017] Bian, J., Zhang, L., Liu, Y., Lin, W.-Y., Cheng, M.-M., and Reid, I. D. (2017). Image Matching: An Application-oriented Benchmark. *arXiv:1709.03917 [cs].*

[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., 2009] Chen, K., Chen, J., Huang, C., and Chen, C. (2009). Fighting Phishing with Discriminative Keypoint Features. *IEEE Internet Computing*, 13(3):56–63.

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

[Chen et al., 2010] Chen, T.-C., Dick, S., and Miller, J. (2010). Detecting visually similar Web pages: Application to phishing detection. *ACM Transactions on Internet Technology*, 10(2):1–38.

[Chum and Matas, 2010] Chum, O. and Matas, J. (2010). Large-Scale Discovery of Spatially Related Images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(2):371–377.

[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.

[Gorokhovatskyi et al., 2018] Gorokhovatskyi, O., Gorokhovatskyi, V., and Peredrii, O. (2018). *Data*, 3(4):52.

[Grycuk, 2016] Grycuk, R. (2016). Novel visual object descriptor using SURF and clustering algorithms. *Journal of Applied Mathematics and Computational Mechanics*, 15(3):37–46.

[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.

[Huang et al., 2008] Huang, C.-R., Chen, C.-S., and Chung, P.-C. (2008). Contrast context histogram—An efficient discriminating local descriptor for object recognition and image matching. *Pattern Recognition*, 41(10):3071–3077.

[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.

[Lee,] Lee, M. Pytesseract: Python-tesseract is a python wrapper for Google's Tesseract-OCR.

[Leibe et al., 2006] Leibe, B., Mikolajczyk, K., and Schiele, B. (2006). Efficient Clustering and Matching for Object Class Recognition. In *Procedings of the British Machine Vision Conference 2006*, pages 81.1–81.10, Edinburgh. British Machine Vision Association.

[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.

[Rosebrock, 2018] Rosebrock, A. (2018). OpenCV Text Detection (EAST text detector).

[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.

[Tareen and Saleem, 2018] Tareen, S. A. K. and Saleem, Z. (2018). A comparative analysis of SIFT, SURF, KAZE, AKAZE, ORB, and BRISK. In *2018 International Conference on Computing, Mathematics and Engineering Technologies (iCoMET)*, pages 1–10, Sukkur. IEEE.

[Thomee et al., 2010] Thomee, B., Bakker, E. M., and Lew, M. S. (2010). TOP-SURF: A visual words toolkit. In *Proceedings of the International Conference on Multimedia - MM '10*, page 1473, Firenze, Italy. ACM Press.

[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.

[Zhou et al., 2013] Zhou, W., Li, H., Lu, Y., and Tian, Q. (2013). SIFT match verification by geometric coding for large-scale partial-duplicate web image search. *ACM Transactions on Multimedia Computing, Communications, and Applications*, 9(1):1–18.

[Zhou et al., 2010] Zhou, W., Lu, Y., Li, H., Song, Y., and Tian, Q. (2010). Spatial coding for large scale partial-duplicate web image search. In *Proceedings of the International Conference on Multimedia - MM '10*, page 511, Firenze, Italy. ACM Press.