Deduplicating scraped data by using image deduplication

Maxime Nannan Eliott Rabin

Abstract

Deduplicating content from different websites can be a difficult problem to solve as the data has not the same structure in each websites and even for the same item you can have different types of data. However, often, you have images associated to those items and it can be a relevant attribute to match items from different websites. But deduplicating millions of items using images can quickly become computationally hard, nevertheless we can often reduce the number of candidates when looking for duplicates by filtering on other fields (eg if we compare real estate ads we can filter on the city). This papers aims to compare precision and computational cost of several techniques for image similarity.

1. Introduction

Our work will consist in giving a couple of images as input and then returning a matching score (similarity) as output that will tell us the how likely these 2 images are the same or not. This comparison makes sense as the 2 images won't be exactly the same as it would be 2 images of the same part of the flat, but posted on different websites, so small differences will appear.
To simulate the degradation of the images that will appear between the different websites on which the photos will be, we will use different trustworthy techniques and then check if the algorithms can recognize if the photos are the same. The transformations or pieces of degradation of the image we are going to do are cropping, blurring, resizing and adding a logo on the image. Then, we will also try to create different pieces of degradation, meaning different intensities, and measure how it will affect our matching score to assess the robustness of our algorithms through some similarity measures.

2. Related work

The idea here is to have a quick overview of each existing method, then in an other paragraph we're gonna describe methods that we will compare.

2.1. Local features

Local features are relevant as it is challenging to capture all the invariances of images in a global representation. That's why local representations seem well-adapted but require sophisticated matching.

Local descriptors will capture the local appearance at carefully chosen locations. To get what we need about the local features, we then need:
- Interest point detectors (Harris [4], Hessian-Affine [9], MSER [7], etc)
- Local descriptors (SIFT [8], SURF [5], ...)

We will now present briefly 2 of the main local descriptor methods: SIFT and SURF.

SIFT

The Scale Invariant Feature Transformation (SIFT) is a representation of the low level image content that is based on a transformation of the image data into scale-invariant coordinates relative to local features. Local feature are low level descriptions of keypoints in an image. Keypoints are interest points in an image that are invariant to scale and orientation. Keypoints are selected by choosing the most stable points from a set of candidate location. Each keypoint in an image is associated with one or more orientations, based on local image gradients. Image matching is performed by comparing the description of the keypoints in images.

SURF

The basic idea of Speeded Up Robust Features (SURF) is quite similar to SIFT. SURF detects some keypoints in an image and describes the keypoints using orientation information. However, the SURF denition uses a new method for both detection of keypoints and their description that is much faster still guaranteeing a performance comparable or even better than SIFT. Specifically, keypoint detection relies on a technique based on a approximation of the Hessian Matrix. The descriptor of a keypoint is built considering the distortion of Haarwavelet responses around the keypoint itself.

Alternative approaches are therefore interesting to consider. For example, global descriptors can aggregate individual local descriptors per image, or we can perform simple similarity measure between global descriptors. In any case, global features are to be considered.

2.2. Global features

We will describe 4 techniques that resort to global features.

Bag of features

A standard approach to describe an image for classication and retrieval purposes is to extract a set of local patch descriptors, encode them into a high dimensional vector and pool them into an image signature. The most common patch encoding strategy consists in quantizing the local descriptors into a finite set of prototypical elements, which leads to the most commonly-used bag of features.

The principle is the following:
- Extract local descriptors from the input image and selected locations
- Convert local descriptors into visual words, using a visual codebook. This codebook of k visual words is usually obtained by k-means clustering. The codebook can be defined like this: $C = \{c_1, ...c_k\}$.

- Represent images as a histogram of occurrences of visual words that will describe the image
- Finally, you can compare 2 images by comparing their histogram of occurrences of visual words

It's possible to refine the description offered by the bag of features method by adding higher order statistics such as the mean or the variance of the samples that belong to have been matched with one particular k "visual word", instead of counting the number of visual words that appear.

VLAD [6]

VLAD accounts for Vector of Locally Aggregated Descriptors and focuses on the mean of the sample. It aggregates all descriptors assigned to the same visual word that come once again from a codebook previously created.
Each local descriptor x is associated to its nearest visual word $c_i = NN(x)$. The idea of the VLAD descriptor is to accumulate, for each visual word c_i, the differences $x - c_i$ of the vectors x assigned to c_i. This characterizes the distribution of the vectors with respect to the center.

Fisher kernel [10]

The FV is an image representation obtained by pooling local image features. It will focus both on the mean and the variance of the sample.
A Gaussian Mixture Model (GMM) is used to model the distribution of features extracted all over the image. We therefore have a probabilistic codebook as a mixture of Gaussians. Descriptors are then assigned to words. The Fisher Vector (FV) encodes the gradients of the log-likelihood of the features under the GMM, with respect to the GMM parameters of the model.

Image hashes [1]

Images hashes is a hash technique that resizes images to a 8x8 pixels and then turns it into a 64 bits hash by doing different techniques like averaging and thresholding.

The last type of method we will focus on are deep learning methods, as deep learning works quite well in a wide range of problems currently.

2.3. Deep learning

Deep learning methods are thriving in various fields, but they still lag behind in image retrievals or comparisons compared to the classical methods explained above. However, some recent deep learning methods have shown encouraging performances.
We will first describe a few deep learning methods that were used in the past but didn't provide satisfactory results for our problem compared to the state of art.

CNN-based retrieval [3]

These algorithms may be very good at image classifications, but they don't perform that well when considering our issue of image retrieval and comparisons. Indeed, they lack of robustness when cropping and blurring the images for example.

CNN-based retrieval methods often use local features extracted using networks pre-trained on ImageNet for a classication task. These features are learned to distinguish between different semantic categories, but the problem is that they are quite robust to intra-class variability. This is an undesirable property for instance retrieval, where we are interested in distinguishing between particular objects even if they belong to the same semantic category.

So in our problem, these algorithms won't help a lot to say if 2 similar images are the same or not. These types of

algorithms would be good at saying if both images belong to the same categories, but it doesn't make sense here as all images may belong to the same category since they will all represent flats.

Fine-tuning for retrieval [3]

The models that have been pre-trained on ImageNet for object classification could be improved by fine-tuning them on external sets of landmarks images, but one should pay attention to the image representations used, and a classification loss hasn't provided very good results so far.

Localization and region pooling [11]

Most of the retrieval methods consider random regions or a rigid grid of regions as regards their descriptors. It may be interesting to select the most relevant regions depending on the task we would like to perform in order for our algorithms to perform better.

Siamese networks and metric learning [2]

This last type of algorithms have been widely used for image descriptors for example. This class of neural network architecture contains 2 or more identical subnetworks (same configuration with the same parameters and weights). These algorithms are popular when finding similarity among comparable things, which is our issue here. However, their network architecture are often too simple, which involves pooling and aggregation of several regions, thus preventing from getting good results.

3. Methodology

3.1. Dataset

Our dataset is composed of images we get on https://www.leboncoin.fr/, we used 6 images that represents a building, a bathroom, a kitchen and 3 rooms in order to have as less variances as possible in our results.

We decided to apply several transformations to those images in order to create a bigger dataset and then apply similarity functions between images and their transformations. By doing this we will be able to measure for each similarity function and each transformation of close is an image and its transformation.

We applied the following transformations on the dataset:

- Cropping the image either vertically, or horizontally or in both direction by removing 1%, 5%, 10%, 15%, 20% and 25% of the image.
- Resizing the image either vertically, or horizontally or in both direction by a ratio of 200%, 150%, 125%, 75 %, 50%, 30%, 20%, 15%, 10%
- Adding a logo from two different websites https://www.seloger.com/ and https://www.leboncoin.fr/ and increasing its size from 1% of the width of the image to 20%.
- Blurring by using a kernel of different sizes from 2 by 2 to 10 by 10.

Those transformations has been carefully chosen in order to mimic transformations that we could find from a website to another.

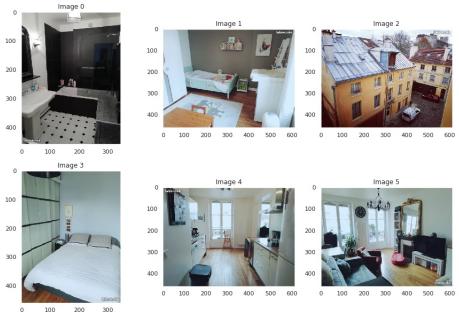

Figure 1. Images from leboncoin

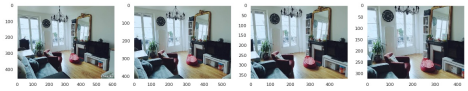

Figure 2. Cropping in both directions

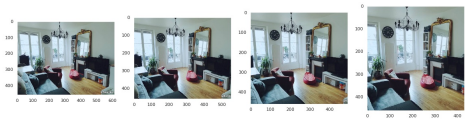

Figure 3. Horizontal cropping

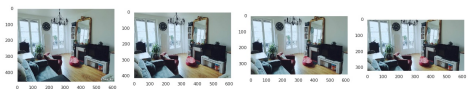

Figure 4. Vertical cropping

Figure 5. Resizing in both directions

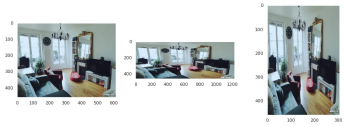

Figure 6. Horizontal resizing

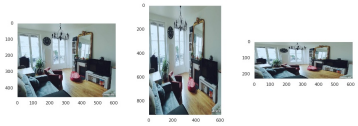

Figure 7. Vertical resizing

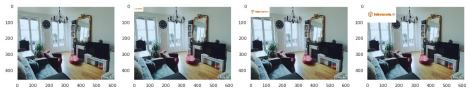

Figure 8. Adding leboncoin logo

Figure 9. Adding seloger logo

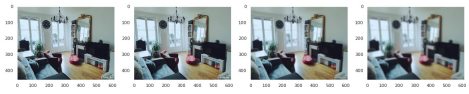

Figure 10. Blurring

3.2. Similarity measures

Our goal is to create functions f that given two images I_1 and I_2 returns a similarity between 0 and 1 based on local features, global features and deep learning features.

3.2.1 Local features similarities

To build our local similarity score between two images we first computed 50 SIFT [8] features for each images. Then given SIFT features we tried to match them together based on Lowe ratio that we set to 0.8 and then find the homography between the two group of features and only kept features that were respecting the homography. Finally with those matched features we computed 2 similarity scores:

Percentage of matches

This score was just the ratio of the number of matched features over the number of extracted features (50 in our experiments). This metric is not really relevant but it was interesting to have it to compare with other metrics.

Percentage of matches = $\frac{number\ of\ matched features}{number\ of\ features}$

Mean similarity

Let's denote X_1 (resp X_2) the matched features from the first (resp second) image. This score was the mean cosine similarity between matched features. One of the downside of such metric is that it does not really take into consideration how many features have been matched.

Mean similarity = $\frac{1}{|X_1|} \sum_{X_1,X_2} cosine\_similarity(x_1, x_2)$

3.2.2 Global features

Our similarity scores based on global features were only based on hashes.
We used 4 different methods:

- **ahash** also called "Average hash" which consists in converts the image into a grayscale 8x8 pixels and sets the 64 bits in the hash based on whether the pixel's value is greater than the average color for the image.
- **phash** also called "Perceptive Hash". This hash is similar to **ahash** but instead of averaging pixel color it uses discrete cosine transform and compare frequencies.
- **dhash** also converts images to 9x8 grayscale image then it computes the difference between adjacent pixels to get the gradient and then set 0 or 1 whether the gradient in a cell is larger that the gradient on the left cell.
- **whash** is a hash based on wavelet transform.

Given those hashes we used the following function to compute a similarity between two images:

sim(hash1,hash2) = $\frac{len(hash) - hamming(hash_1, hash_2)}{len(hash)}$

3.2.3 Deep learning

Deep learning network for image retrieval, RMAC [2]
For our similarity based on deep-learning features we were using a CNN-network trained specifically to do image retrieval. This network has been trained with a triplet loss in order to output a vector of size 2048 that can be simply compared by using a cosine similarity and that should be the same for two images representing the same thing.
So we just applied this network to our images and then simply compute cosine similarity between obtained vectors.

4. Evaluation

4.1. Robustness

We worked on 6 images. For each of them, we created 9 transformations of them, and then computed 7 similarities in comparison to the original image. The broken lines correspond to a threshold: if the similarity we get is higher than this threshold, we can suppose that the 2 images are the same. This threshold is achieved by comparing all pairs of image, measuring the similarity between these 2 images, and then taking the maximum of all the pairs. Below are a few results we obtained.

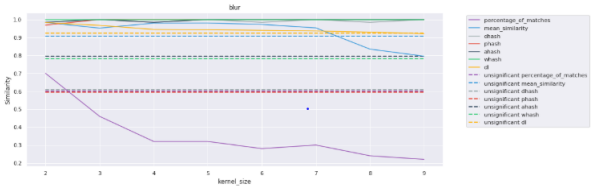

Figure 11. Blurring

For example regarding blurring, we observe that hashes methods perform pretty well. It makes sense as hashes function perform a resizing by taking the mean of some pixels, which corresponds to what blurring does. The deep learning method works not so well as the threshold is always close to the curve, and the percentage of matches method performs poorly.

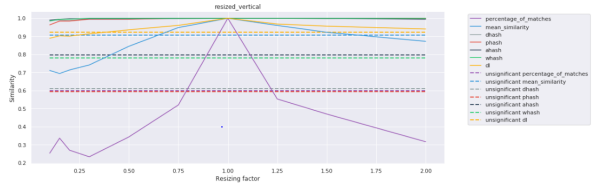

Figure 12. Resizing

This resizing is vertical. Resizing is the most important transformation in our problem since this is the most common one in the real world. Websites indeed often do this kind of image degradation. We observe once again that hashes methods perform pretty well, which makes sense again as hash functions do resizing. The mean similarity method is not that good here.

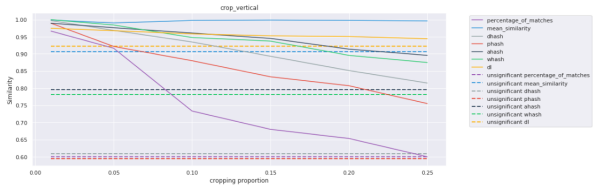

Figure 13. Cropping

This cropping is vertical. We can observe that the deep learning and mean similarity methods perform quite well since there are always above their threshold for all different intensities of cropping. For hashes we can observe a degradation of the similarity score but they remain above their threshold. We can't really conclude about their efficiency after 15 % of cropping because our thresholds have been computed only on 50 pairs of images and it should be higher.

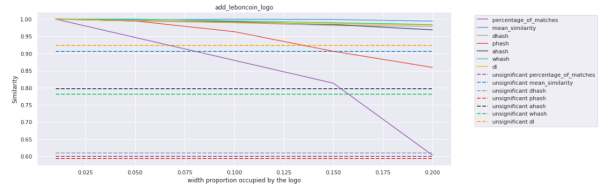

Figure 14. Add logo

When adding the "Leboncoin" logo at the top left corner, we can observe that all the methods work as their are all above their respective threshold, even the percentage of matches method that always perform badly. This can be explained as the logo doesn't modify a lot the image compared to the other methods.

4.2. Speed

There are 2 types of computation time to take into consideration because if we wanted to implement this in production we would have to first compute features for each image and then compare features of images using similarity functions presented above.

method	time (ms)
SIFT	34
DHASH	1.38
PHASH	1.75
AHASH	1.36
WHASH	7.5
DL	2640

Figure 15. Feature computation

About computation time, deep learning features are the longest but this number should be 10 times smaller according to [2] if we were using a GPU. As we can see there is a factor 10 between global features and local features and another factor 10 between local features and deep learning features.

method	time (ms)
Percentage of matches	19.16
mean similirity	19.20
DHASH	0.09
PHASH	0.09
AHASH	0.09
WHASH	0.09
DL	0.490

Figure 16. Similarity computation

We then should pay attention to the similarity computation time as we will do these operations a great number of time. So it is important that it won't last too long. We can observe that hashes are still the quicker to compute then there is the deep learning features and finally local features.

If we compare both computation times we can claim that hashes are the fastest to compute and that deep learning features would be quicker if we are comparing one image more that 15 times (because SIFT features are taking 19 ms to be compared against 0.5 ms for deep learning features.

5. Conclusion

Local features were quite robust to cropping (as it does not deform images) but was lacking of accuracy for resizing which is the main transformation observed on websites. Moreover it was the slowest approach if we consider that we will compare images a lot of times (because comparing SIFT features requires to match features).

Deep learning did not perform as expected, results pointed out that it was not discriminant for our task which could be explained by the fact that the network has been trained to output an embedding invariant to what is on the picture and not about the picture it self. Besides it requires a GPU to compute features.

Finally hashes performed quite well and mainly for resizing but it is not so surprising as they have been designed to be robust against those transformations. Moreover they are the quickest to be computed and each hash is only 64 bits which is quite interesting knowing that we would probably need to store millions of hashes.

5.1.

References

[1] 2013. http://www.hackerfactor.com/blog/index.php?/archives/529-Kind-of-Like-That.html.

[2] J. R. Albert Gordo, Jon Almaz an and D. Larlu. Deep image retrieval: Learning global representations for image search. 2016.

[3] A. C. V. L. Artem Babenko, Anton Slesarev. Neural codes for image retrieval. 2014.

[4] C. Harris and M. Stephens. A combined corner and edge detector, 1988. 4th ALVEY Vision Confer-ence.

[5] T. T. Herbert Bay and L. V. Goo. Surf: Speeded up robust features. 2016.

[6] C. S. P. P. Herv Jgou, Matthijs Douze. Aggregating local descriptors into a compact image representation. 2010.

[7] M. U. J. Matas, O. Chum and T. Pajdla. Robust wide baseline stereo from maximally stable extremal regions. 2004. InImage and Vision Computing, 22(10):761767.

[8] D. G. Lowe. Distinctive image features from scale-invariant keypoints. 2004.

[9] K. Mikolajczyk and C. Schmid. An affine invariant interest point detector, 2002. An affine invariant interest point detector. In Proceedings of the 7th European Conference on Computer Vision, Copenhagen, Denmark.

[10] F. Perronnin and C. R. Dance. Fisher kernels on visual vocabularies for image categorization. 2007.

[11] H. K. G. R. S. J. Ren, S. Faster r-cnn: Towards real-time object detection with region proposal networks. 2015.