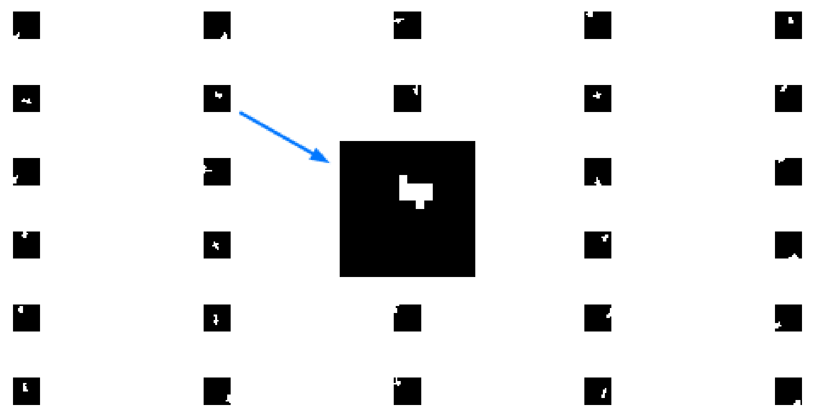
2018-10-29, by Vitali Fedulov (fedulov.vitali@gmail.com)

Algorithm for perceptual image comparison

The algorithm I developed for image search and comparison makes use of perceptual similarity by performing the following set of operations:

1. Mask generation

A set of square masks is generated. A mask is a black square image with several white pixels aggregating locally. Such pixel groups are located in distinct positions from mask to mask. The white pixels define image sub-regions to calculate average color at each of the regions. Such color values will be used during comparison stage. Depending on implementation the number of masks is 300-500, and the mask size from 8x8 to 24x24 pixels. A single mask size is used in specific implementation.



Example of 16x16 masks with one of them shown at larger scale

In the most recent implementation the masks have regular 3x3 shape (e.g. the median filter).

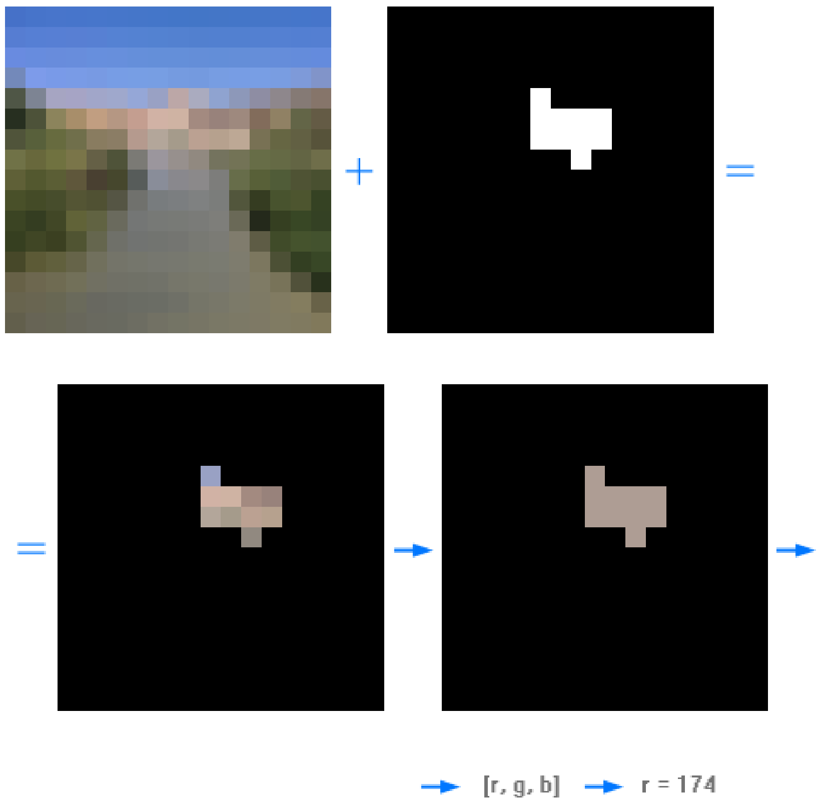
2. Image resizing

Input images are resized to the mask size. Resampling quality should be high enough in order to preserve near exact average colors in sub-regions of high-resolution input images. Low quality resampling would cause the algorithm to fail.

Input images are resized to the mask size

3. Generating image hashes

An image hash is an array of average color values defined by white pixels of separate masks superimposed over the resized image. Example hash: [23, 126, 77, ...], where the array length is equal to the number of masks. Each value in the array corresponds to an average color value corresponding to the location of white pixels in one mask. Black pixels of a mask are not used for calculations. Best results are achieved by mixing color channel values, so that each value corresponds to a different color channel or gray in the following order within a hash: [r, g, k, r, g, k, ...], where r is red, g is green, k is grayscale. Blue is not used as a separate value because it was found less relevant for perceptual similarity.



Calculating hash sub-value (average color in white-pixel area of the mask)

4. Image to image comparison

An input to this operation is a pair of image hashes and original image sizes. Image sizes are used as a first step to quickly eliminate possible mismatch. If image proportions are considerably different, images are considered non-similar, so no further checks are performed.

In the next step two image hashes are compared in a loop value-by-value (e.g. color by color). If a color difference for two images is larger than a threshold (~50), images are considered non-similar and the loop is broken. This allows to eliminate non-similar pictures before going to the more computationally intensive next step.

In the last step two image hashes are compared by their cosine similarity. If the similarity is smaller than threshold (~0.97), the images are considered different. This is the final step of the comparison procedure.

Image pairs that passed all the three filters above are similar (with high probability).

Possible optimizations

Optimization 1: The algorithm benefits from histogram normalization applied to resized images. This allows to better compare similar images containing line drawings, where white background occupies the majority of space. Since image similarity is a multidimensional problem, perceptual similarity is subjective. E.g. normalization approach will sometimes generate false positives, e.g. underexposed images normalized and found similar to well balanced images. Therefore in this version no normalization is applied.

Optimization 2: If searching for very similar images, e.g. strictly resized images, one can decrease color threshold and increase cosine similarity. Alternatively increase mask size. In general, assuming same size of white sub-regions of masks, smaller masks allow better generalization, e.g. comparing overall color/brightness distribution over the images. But such approach may also generate false positives for cases of equal average color values equal by accident.

Optimization 3: Resizing input images to a square mask causes proportion changes for non-square images and allows to preserve information from every region of an image. Such resizing is optional. Instead it is possible to use/resize only the central square area of the input image for image comparison, thus discarding information outside the central square. This should not cause considerable loss in comparison precision.

Optimization 4: In addition to color values for hash generation it is also possible to use filter-based values, e.g. by passing an image through an edge detector. This allows to integrate over additional visual signals within image/mask sub-regions. For example a forest photo will have distinct edge values vs. similar smooth background of similar average color, because trees have leaves. Such approach allows to distinguish textures. The challenge and difficulty is finding optimal coefficients between edge information and color information during the hash comparison step.

Optimization 5: In the algorithm above white pixels of the masks form small near circular regions. Instead it is possible to use more complex subject-specific region shapes. E.g. face-like masks to detect faces. Such masks can be potentially used for image clustering by subject and classification.

Resources

Algorithm in Github (Golang): image comparison.