Similar.Pictures

Find near duplicates on your computer

fedulov.vitali@gmail.com, 24 January 2022

Algorithm for perceptual image comparison

The image comparison algorithm I developed for the service makes use of perceptual similarity by performing the following set of operations. To understand how the idea evolved, check previous version of the algorithm.

1. Image resizing

Source images are resized to square images ("icons") of 23x23 pixels. The resizing is done in such a way that color relationships between original image sub-regions are carefully preserved by using high precision values, unlike in common graphic editors. Specifically: float values are used instead of uint8 and pixel values are sampled with high density.

2. Box blur

Box blur filter of 3x3 is taken at every 2nd row/column. The final icon size is 11x11.

3. Color space modification

RGB color space of the icon is transformed into YCbCr to give more importance to luma vs chroma components. This step is optional, but works well for cases when a colored and grayscale versions of the same image are compared.

4. Histogram normalization

This operation stretches histograms for each color channel of the icon so that [min, max] value range becomes [0.0, 255.0]. This increases the range of pixel values (and increases icon contrast).

This step is optional, depending on similarity aspects: but usually same images of different contrast can be considered similar.

5. Image comparision (final step)

The input to this operation is a pair of icons from step (4) 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.

Then the two icons are compared by Euclidean distance. If the distance is larger than a certain threshold, images are considered non-similar.

Possible optimizations

Optimization 1: To increase precision of the algorithm, one may apply it on image sub-regions (instead of the whole image). If at least one region is not similar, images can be considered distinct.

Optimization 2: Resizing a rectangular image to a square icon causes proportion changes and allows to preserve information from every region of the input image. Such resizing is optional. Instead it is possible to use exclusively the central square area of the input image for comparison, thus discarding information outside the central square.

Optimization 3a: In addition to color values for icon generation it is also possible to use filter-based values, e.g. by passing an image through an edge detector. This allows to count additional visual signals within sub-regions. For example a photo of forest will have distinct edge values compared to some smooth background of the same color (because trees have leaves, thus many edges). Such approach allows to distinguish between textures. The challenge is finding an optimal coefficient for perceptual significance of edge information vs. color information during comparison.

Optimization 3b: Instead of summation of colors within icon sub-regions, it is possible to sum over features extracted with convolutional neural network filter layers.

Optimization 4: When the number of images is very large (more than millions), using Euclidean metric for search might be slow. It is possible to use hash tables as a preliminary step before checking Euclidean distance. For that a certain number of sample points from the icon can be taken and then a hashing algorithm for floating point vectors can be applied to the sample vector.

Algorithm implementation

Written in Golang image comparison (Github).

This is my original research. Please treat it as such for references, provided you do not find an earlier similar research by others, which is possible. Sometimes solutions have to be rediscovered, because either papers are hard to find, or they are difficult to understand.