Another Example: Circle Detection

Task: Detect any circular objects in a given image. Here, the output space is three-dimensional: center coordinates \((k, l)\) and radius \(r\).

The math is simpler here:

If there is an edge at position \((i, j)\) in the input image, add a vote for all positions \((k, l, r)\) in the Hough space for which we have:

\[(i-k)^2 + (j-l)^2 = r^2\]
Oriented FAST

\[ m_{pq} = \sum_{i,j} i^p j^q I(i,j) \]

Moments are useful for a variety of purposes in computer vision. For example, the centroid (center of gravity) \( G \) of an image patch is given by:

\[ G = \begin{pmatrix} m_{10} & m_{01} \\ m_{00} & m_{00} \end{pmatrix} \]

Oriented FAST

Now let us compute the centroid \( G \) for the square patch that embeds the FAST circle for a detected corner:

Then the orientation \( \theta \) of the vector from corner \( C \) to centroid \( G \) is given by:

\[ \theta = \text{atan2}(m_{01}, m_{10}), \]

where \( \text{atan2}(y, x) \) is the quadrant-aware variant of \( \text{arctan} \).

Oriented FAST

In order to make FAST less scale-dependent, compute an image pyramid and apply FAST at each level.

Compute a total of \( N \) keypoints for each image. To do that, set the FAST threshold low enough to generate at least \( N \) keypoints. Then pick the \( N \) keypoints with the strongest FAST responses. Each of the resulting keypoints now includes orientation information that can be used to make the BRIEF features rotation-invariant.

The basic idea is that we rotate the BRIEF features to match the orientation of keypoints.

Rotated BRIEF

Let us say that we have \( n \) BRIEF pairs \((a_1, b_1), \ldots, (a_n, b_n)\). Then we can write them as a matrix \( S \):

\[ S = \begin{bmatrix} a_1 & \cdots & a_n \\ b_1 & \cdots & b_n \end{bmatrix} \]

Using a rotation matrix \( R_\theta \) for the orientation \( \theta \) of a given keypoint, we compute a rotated version \( S_\theta \) of the BRIEF pairs as follows:

\[ S_\theta = R_\theta S \]

Rotated BRIEF

The resulting matching of BRIEF pairs is (ideally) rotation-invariant.

In order to avoid the matrix multiplication with every test, we can pre-compute rotated brief masks for 0°, 12°, 24°, ..., 348° and pick the one closest to the desired orientation.

We can also optimize the selection of BRIEF pairs instead of using random distributions, but that is beyond the scope of our discussion.
Rotated BRIEF

The resulting matching of BRIEF pairs is (ideally) rotation-invariant.

In order to avoid the matrix multiplication with every test, we can pre-compute rotated brief masks for 0°, 12°, 24°, ..., 348° and pick the one closest to the desired orientation.

We can also optimize the selection of BRIEF pairs instead of using random distributions, but that is beyond the scope of our discussion.

Bag-of-Keypoints Approach

We still have not answered the initial question: How can we use the computed image features to recognize objects (i.e., object classes)?

One possible approach is called "Bag of Keypoints."

We compute ORB feature descriptors for all test images.

Then we perform a cluster analysis on the set of all descriptors (for example, using k-means).

The centers of these clusters are our visual words, forming the vocabulary.

For each training image, determine the closest visual word for each keypoint.

Create a histogram of matched visual words.

These histograms form the feature vectors for the training images. They should be similar within object classes and distinct between classes. Then a classifier can be trained on these data and should be able to categorize the feature vectors of new (test) images.
In face images, we can expect certain features to always appear in roughly the same relative location and orientation.

Face Recognition

Therefore, when our goal is to recognize people from their face images, we do not have to determine the location of these features. Instead, we can normalize the face images as best as possible (for example, make sure that the eyes are always in the same two image positions) and then compute descriptors for fixed sections of the images.

The most popular and successful descriptor of this kind is referred to as Local Binary Patterns (LBPs).

Local Binary Patterns

For each pixel in a given section, we look at its 8 neighbors (we can also pick neighbors at radii greater than 1; radius 2 is most popular).

Then we form a binary string by looking at each neighbor in a specified order and adding a “1” to a binary string if it is brighter than the center pixel and “0” otherwise.

The binary string can be interpreted as a number.

Different patterns (numbers) describe different types of gradients, i.e., local changes in intensity.

A consistent gradient should result in exactly one uninterrupted sequence of 1s and the remaining bits should be 0s, as in the 5 examples below:

There are exactly 58 patterns of this kind.

For a given area of pixels, we can now compute how many fall into each of these 58 categories or the “inconsistent” category.

This results in a histogram with 59 bins.

We can split a given face image into, for example, 7×6 sections and obtain a histogram for each section.

Then we concatenate these histograms to obtain a feature vector that describes the given face image.

In our example, these feature vectors have 7×6×59 = 2478 elements.

Some of these sections are more important for recognition than others, so we may assign weights:
Local Binary Patterns

For LBP descriptors $A_i$ and $B_i$ and weights $w_j$ with bins $i$ and sections $j$, a good similarity measure is:

$$X_k^2 = \sum_{i,j} w_j \frac{(A_i - B_j)^2}{A_i + B_j}$$