SIFT features

Scale-Invariant Feature Transform (SIFT) is a method for detecting and describing local features in images. It transforms image data into scale-invariant coordinates relative to local features. That means, these features are invariant to scale, rotation, and partially invariant to changes in illumination and 3D camera viewpoint. These features can be described as aset of vectors [**Lowe.2004**]:

$$\{v_1, v_2, ..., v_n\} \tag{1}$$

where each v_i is a 128-dimensional vector.[1]

The pipeline for calculating SIFT features involves the following steps:

1. **Scale-space Extrema Detection:** This step computes the local extrema of the Difference of Gaussian function convolved with the image, which are keypoint candidates. See the next subsection for more details.

2. **Keypoint Localization:** The candidate keypoints resulting from the previous step will then go through a refinement process to eliminate low-contrast keypoints and poorly localized keypoints along edges, resulting in a set of stable keypoints.

3. **Orientation Assignment:** Each keypoint is assigned an orientation based on local image properties.

4. **Keypoint Descriptor Generation:** A descriptor is computed for each keypoint using highly distinctive yet as invariant as possible features such as local image intensities around the keypoint.

Calculation of Scale-Space Extrema

The first step in calculating SIFT features is to detect stable keypoints across all scales. These are points that are invariant to scale changes. To achieve this, the image is convolved with a scale-space kernel over a range of scales.

According to Koenderink (1984) and Lindeberg (1994), "the only possible scale-space kernel is the Gaussian function. Therefore, the scale space of an image is defined as a function $L(x, y, \sigma)$, that is produced from the convolution of a variable-scale Gaussian, $G(x, y, \sigma)$, with the image $I(x, y)$:

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y)$$

[1] For further details, see [**Lowe.2004**].

where * is the convolution operation in x and y, and

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2}.$$

"

The convolution operation can be implemented in Python using cv2.GaussianBlur() function (note that this is a discrete approximation of the continuous Gaussian function). Assume a symmetric kernel size of 3×3 and a standard deviation of σ in both x and y directions, the function can be called as follows:

```
kernel = (3, 3)
sigma = 1.0
cv2.GaussianBlur(src, kernel, sigma)
```

Stable keypoints in scale space can be detected using scale-space extrema in the Difference of Gaussian function convolved with the image($D(x, y, \sigma)$):

$$D(x, y, \sigma) = (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y) = L(x, y, k\sigma) - L(x, y, \sigma)$$

To find out the value of k, each octave of scale space is divided into s invervals, and the value of k is set to $2^{1/s}$. Then, we must produce s+3 blurred images for each octave, so that final extrema covers a complete octave. This can be implemented in Python using the following code:

```
s = 3
k = 2**(1/s)
octave_images = [cv2.GaussianBlur(src, kernel, sigma * (k**i)) for i in range(s+3)]
```

The result of this operation is a list of s+3 gaussian images, each with a different scale. Adjacent gaussian images are then subtracted to produce difference-of-gaussian (DoG) images. After each octave, the image is resized to half of its original size. This can be illustrated in the image below:

For further details, see [**Lowe.2004**].

Implementation in Python

The OpenCV library in Python already provides a function for this operation. Below is an example of how to calculate SIFT features using OpenCV:

```
sift = cv2.SIFT_create()
keypoints, descriptors = sift.detectAndCompute(image, None)
```

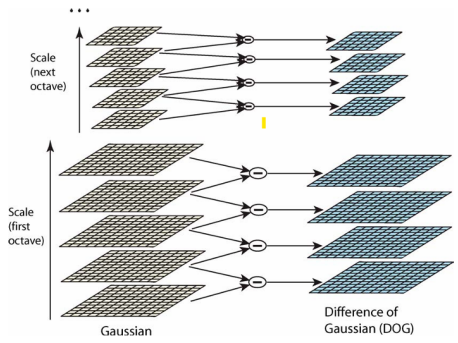

Figure 1: Difference of Gaussian images. It can be seen that the image size is reduced by half for each octave [**Lowe.2004**]

Vector of Locally Aggregated Descriptors (VLAD)

Overview

The Vector of Locally Aggregated Descriptors (VLAD) is a feature encoding and pooling method commonly used in the field of computer vision, particularly for tasks involving image retrieval and classification. VLAD is an approach that aggregates feature descriptors extracted from an image into a compact representation.

Mathematical Formulation of VLAD

Given an image, feature detection algorithms (e.g., SIFT or SURF) are first used to identify key points and compute corresponding descriptors. These descriptors are then assigned to the nearest cluster centers that have been precomputed using a clustering algorithm such as k-means on a training dataset.

Let x_i be a descriptor and c_k be the nearest cluster center. The aggregation for a cluster k is computed as follows:

$$V_k = \sum_{x_i \in k} (x_i - c_k) \tag{2}$$

Where V_k is the aggregated vector for cluster k, and the sum is over all descriptors x_i assigned to cluster k. The resulting VLAD vector is the concatenation of all V_k for each cluster center in the vocabulary. L2 Normalization is then applied to the VLAD vector.

Processing Pipeline Using VLAD

The typical pipeline for processing an image using VLAD involves the following steps:

1. **Feature Detection and Description:** Extract descriptors from the image using a feature detector.
2. **Descriptor Assignment:** Assign each descriptor to the nearest cluster center from a predefined set of centers (vocabulary).
3. **Aggregation:** For each cluster center, aggregate the differences between the descriptors assigned to that cluster and the cluster center itself.
4. **Normalization:** Normalize the resulting VLAD vector to enhance its robustness and improve similarity measurement during retrieval. Normalization methods include L2 normalization and power normalization.
5. **Post-Processing:** Optionally, further process the VLAD vector using techniques such as dimensionality reduction (PCA) or whitening depending on the application.

The overall similary of two VLAD vectors can then be calculated as:

$$\frac{1}{C^{(1)}} \sum_i (x_{k,i}^{(1)} - \mu_k)^T \cdot \frac{1}{C^{(2)}} \sum_j (x_{k,j}^{(2)} - \mu_k) \tag{3}$$

0.1 Fisher Vector (FV)

The Fisher Vector (FV) extends the Bag-of-Words (BOW) model by encoding higher-order statistics, such as the first and second-order differences, instead of just counting the occurrences of visual words. This method is derived from the Fisher kernel framework, which describes a sample set's deviation from an average distribution. The distribution is typically modeled using a Gaussian Mixture Model (GMM).

Given a set of T local descriptors $X = \{x_t; t = 1, \ldots, T\}$ extracted from an image, we assume that the generation process of X can be modeled by an image-independent probability density function u_λ with parameters λ. The Fisher vector G_λ^X is obtained by computing the gradient of the log-likelihood of the sample set X with respect to the parameters λ:

$$G_\lambda^X = \frac{1}{T} \nabla_\lambda \log u_\lambda(X)$$

where G_λ^X describes how the set of descriptors X deviates from the generative model defined by u_λ.

Fisher Kernel Framework

Let $X = \{x_t; t = 1, \ldots, T\}$ be a set of T local descriptors extracted from an image. Assuming that these descriptors are generated independently according to a probability density function u_λ with parameters λ, the Fisher kernel is defined as:

$$K(X, Y) = (G_\lambda^X)^T F_\lambda^{-1} G_\lambda^Y$$

where: - F_λ is the Fisher information matrix, defined by:

$$F_\lambda = E_{x \sim u_\lambda} \left[\nabla_\lambda \log u_\lambda(x) \nabla_\lambda \log u_\lambda(x)^T \right]$$

- G_λ^X is the Fisher vector after applying the Cholesky decomposition on $F_\lambda^{-1} = L_\lambda^T L_\lambda$, and is computed as:

$$G_\lambda^X = L_\lambda \nabla_\lambda \log u_\lambda(X)$$

Fisher Vector Representation

Unlike the Vector of Locally Aggregated Descriptors (VLAD), which encodes each descriptor to a single cluster center, the Fisher vector encodes each descriptor to multiple Gaussian components (soft assignment). The probability of a descriptor x_t belonging to the i-th Gaussian is computed with the Gaussian Mixture Model (GMM). For more details, see [**viroli2017deep**].

The Gaussian Mixture Model (GMM) is chosen for $u_\lambda(x) = \sum_{i=1}^K w_i u_i(x)$, where w_i, μ_i, Σ_i are the mixture weights, mean vectors, and variance matrices (assumed diagonal) of the Gaussian u_i. The Fisher vector is computed by focusing on the gradient with respect to the mean:

$$\gamma_t(i) = \frac{w_i u_i(x_t)}{\sum_{j=1}^K w_j u_j(x_t)}$$

$$G_i^X = \frac{1}{T\sqrt{w_i}} \sum_{t=1}^T \gamma_t(i) \Sigma_i^{-1/2} (x_t - \mu_i)$$

where: - $\gamma_t(i)$ is the soft assignment of descriptor x_t to the i-th Gaussian. - w_i, μ_i, and Σ_i are the weight, mean vector, and covariance matrix of the i-th Gaussian component.

The final Fisher vector G_λ^X is the concatenation of the vectors G_i^X for $i = 1, \ldots, K$, resulting in a Kd-dimensional vector. This vector captures both the occurrence and distributional properties of the local descriptors.

Normalization

The Fisher vector undergoes two normalization steps:

1. **Power Normalization**: Apply a power normalization function independently to each component:

$$f(z) = \text{sign}(z)|z|^\alpha, \quad \text{where } 0 \leq \alpha \leq 1$$

2. **L2 Normalization**: Normalize the vector using the L2 norm to ensure unit length:

$$G_\lambda^X = \frac{G_\lambda^X}{\|G_\lambda^X\|}$$

These normalization steps help reduce the influence of bursty visual elements and improve the separability of the descriptors.