AU7005 Computer Vision (Spring 2024) **Homework 2**

Unit III: Representation

Lecture: Xu Zhao zhaoxu@sjtu.edu.cn

Student: Qingyue Tong(123032910106) tongqingyue@sjtu.edu.cn

Problem 1. Local feature detection

The **structure tensor** is a matrix derived from the gradient of an image, which is useful in local feature detection. It summarizes the predominant directions of the gradient in a specified neighborhood of a pixel, and the degree to which those directions are coherent. For simplification, we define the image gradient of each pixel as follows:

$$\frac{\partial}{\partial x}\boldsymbol{I}(x,y) = \boldsymbol{I}(x+1,y) - \boldsymbol{I}(x-1,y), \ \frac{\partial}{\partial y}\boldsymbol{I}(x,y) = \boldsymbol{I}(x,y+1) - \boldsymbol{I}(x,y-1),$$

where $\boldsymbol{I}(x,y)$ is the intensity value of image $\boldsymbol{I}$ in (x,y). Then the structure tensor M is defined as:

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} \boldsymbol{I}_x^2 & \boldsymbol{I}_x\boldsymbol{I}_y \\ \boldsymbol{I}_x\boldsymbol{I}_y & \boldsymbol{I}_y^2 \end{bmatrix},$$

where $\boldsymbol{I}_x = \partial\boldsymbol{I}(x,y)/\partial x$, $\boldsymbol{I}_y = \partial\boldsymbol{I}(x,y)/\partial y$, $w(x,y)$ is a window function, and $w(x,y) = 1$ only if (x,y) is in the area of interest. The eigenvalues of the structure tensor are often used to describe the local image properties.

Given two images, $\boldsymbol{I}_1^0$ and $\boldsymbol{I}_2^0$ shown in Figure 1 (a, c), can you analysis the local image properties of the highlighted regions by using the structure tensor M? Then if we want to use the Harris corner detector to extract local image features from $\boldsymbol{I}_1^0$ and $\boldsymbol{I}_2^0$, what is an appropriate threshold range on the *cornerness score* (refer to slides-10), where $k = 0.05$, to distinguish the two images? (20 points)

Grading standards. Totally 20 points, 10 for the calculation of two structure tensors, 5 for the analysis of local image properties, 5 for the threshold range of the Harris corner detector.

Answer here.

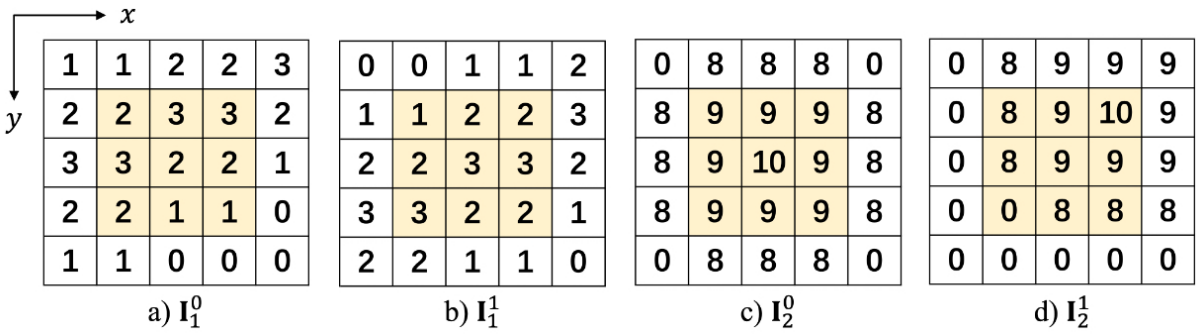

Figure 1: Here are four 5×5 images with different intensity values, where the highlighted regions within the 3×3 windows are the areas of interest. You will use these images in problem 1 and problem 2.

1. To get the structure tensors, I wrote the following function:

```
def structure_tensor(image):
M = np.zeros((2,2), dtype=np.float32)
for i in [1,2,3]:
    for j in [1,2,3]:
        # compute gradient
        grad_x = image[i+1,j]-image[i-1,j]
        grad_y = image[i,j+1]-image[i,j-1]
        M += np.array([[grad_x**2, grad_x*grad_y], [grad_x*grad_y, grad_y**2]])
return M
```

   The results are $M_1 = \begin{bmatrix} 9 & 12 \\ 12 & 24 \end{bmatrix}$ and $M_2 = \begin{bmatrix} 12 & 0 \\ 0 & 12 \end{bmatrix}$, where M_1 is structure tensor of I_1^0 and M_2 is structure tensor of I_2^0.

2. To analyse the local image properties, I calculated the eigenvalues of two structure tensors. That is: 2.35, 30.65 for M_1 and 12, 12 for M_2. And we know that if

   - The eigenvalues are both large: there are corner points in the window;
   - One eigenvalue is large and the other is small: there are edges in the window;
   - The eigenvalues are both small:the window is in a flat area.

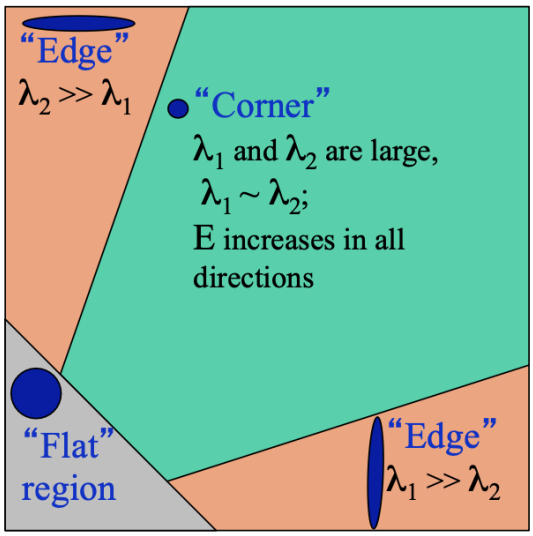

Figure 2: Relationship between local image properties and eigenvalues of structure tensor.

   So the local image of I_1^0 may be the edge and the I_2^0 may be the corner point.

3. To calculate the cornerness score, using the follwing formula:

$$C = det(M) - k \cdot trace(M)^2$$

   The results are $C_1 = 17.55$ and $C_2 = 115.2$. So if we want to distinguish the two images, we can set the threshold range as [17.6, 115.1]. Then the image I_2^0 will be judged as a corner point but I_1^0 won't.

Problem 2. Optical flow estimation

Optical flow is the apparent motion of brightness patterns in the image. Given two subsequent frames, the apparent motion field between them can be estimated if three assumptions are satisfied: brightness constancy, small motion, and spatial coherence. Figure 1 (a-b) and (c-d) are two pairs of images satisfying the above assumptions. For these images, the derivative of time is defined as:

$$\frac{\partial}{\partial t}\boldsymbol{I}(x,y) = \boldsymbol{I}^1(x,y) - \boldsymbol{I}^0(x,y)$$

We assume that the pixels of each image within the highlighted 3×3 window have the same motion, denoted as $(u,v)^T$, where u and v are the velocity of pixels along x-axis and y-axis correspondingly. Can you estimate the motions $(u_1,v_1)^T$ and $(u_2,v_2)^T$ for $\boldsymbol{I}_1$ and $\boldsymbol{I}_2$ using the Lucas-Kanade algorithm? Then discuss your results: Are they consistent with the actual motions of the objects? And why? (15 points)

Hint. Substitute the image gradient of $\boldsymbol{I}_1^0$ and $\boldsymbol{I}_2^0$ calculated in problem 1 into the brightness constancy equations.

Grading standards. Totally 15 points, 12 for the calculation of motion, 3 for the discussion of results.

Answer here.

The main step of Lucas-Kanade algorithm is solving the following equation:

$$\begin{bmatrix} I_{x1} & I_{y1} \\ I_{x2} & I_{y2} \\ \dots & \dots \\ I_{xN} & I_{yN} \end{bmatrix} * \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} I_{t1} \\ I_{t2} \\ \dots \\ I_{tN} \end{bmatrix}$$

And the code file can be found in the attachment. We got

$$(u_1,v_1)^T = (-0.33, 0.67)^T, (u_2,v_2)^T = (1,-1)^T$$

Discussion: The Lucas Kanade algorithm is a method for estimating optical flow based on the assumption of local brightness invariance. It is usually suitable for some simple motion situations, such as small-scale translational motion.

However, for some complex motion situations, such as fast, rotating movements, or situations with occlusion, the Lucas Kanade algorithm may not be suitable. Therefore, although the Lucas Kanade algorithm can provide relatively accurate motion estimation in certain situations, it does not guarantee that the estimated motion is completely consistent with the actual motion of the object.

Problem 3. Convolutional neural networks

Convolutional neural network (CNN) is a popular deep-learning-based representation in computer vision. A CNN model is typically composed by convolutional layer, pooling layer, and fully-connected (FC) layer. Table 1 shows the architecture of a CNN model, which is used for image classification. The input size is $C \times H \times W$, where $C = 3$ is the channel size, $H = W = 256$ are the height and width of the input image. Please fill in the blanks in the table. Then choose appropriate activation functions for the output layer and the hidden layers. (15 points)

Grading standards. Totally 15 points, 2 for each blank, 3 for the activation functions.

Answer in Table 1.

For activation function, we usually choose ReLU after each convolutional layer since it is easy to implement and can quickly converge. And Softmax after the fully connected layer to do image recognition.

Layer name	Operations	Output size
Conv1	convolution 7×7, channel 32, stride 2, padding 3	$32 \times 128 \times 128$
Pool1	max pooling 3×3, stride 2, padding 3	$32 \times 66 \times 66$
Conv2	convolution 5×5, channel 64, stride 5, padding 3	$64 \times 14 \times 14$
Conv3	convolution 5×5, channel 128, stride 3, padding 3	$128 \times 6 \times 6$
Pool2	max pooling 6×6, stride 1, padding 0	$128 \times 1 \times 1$
FC1	weight matrix 128×10	10×1

Table 1: The architecture of a CNN model.

Problem 4. Image retrieval (Code & Report)

As shown in Figure 3, we have collected 25 photos of different buildings in SJTU as a database. Given a query image, please design an image retrieval algorithm to find the same buildings as the query image from the database. Your algorithm should use local image features. (50 points)

Figure 3: Image retrieval.

Hint.

- You can use any OpenCV function as long as you clearly explain how it works in your algorithm.
- Possibly useful tools: SIFT feature descriptor, bag-of-words model, k-nearest neighbors algorithm, ...

Grading standards.

- Local image feature detection, description and visualization. (20 points)
- Feature matching and visualization. (10 points)
- Scoring the matching results and retrieving the similar photos. (10 points)
- Evaluation and discussion of the results. (10 points)

Code in Jupyter Notebook, then report the algorithm and results here.

Figure 4: Visualization of SIFT features.

1. **SIFT Feature Point Extraction:** SIFT (Scale-Invariant Feature Transform) is a widely used algorithm in computer vision for extracting and describing local features from images. Developed by David Lowe in 1999, SIFT is robust to changes in scale and rotation, making it highly effective for object recognition, image stitching, and tracking applications.

   The main steps of the SIFT algorithm are as follows:

   (a) Scale-Space Extrema Detection:
   - Construct a scale space by applying Gaussian blurring at different scales and using the Difference of Gaussian (DoG) to detect potential keypoints.
   - Identify extrema (local maxima and minima) in the scale space as candidate keypoints.

   (b) Keypoint Localization:
   - Refine the detected keypoints by accurately locating their positions and scales.
   - Filter out keypoints with low contrast and those on edges using a quadratic fitting approach to find the exact location and scale of each keypoint.

   (c) Orientation Assignment:
   - Assign one or more orientations to each keypoint to achieve rotation invariance.
   - Compute the gradient magnitude and direction in the neighborhood of the keypoint, creating a histogram of gradient directions and selecting the most prominent directions.

(d) Keypoint Descriptor:

- Construct a descriptor for each keypoint based on the gradient orientations in its local neighborhood.
- The descriptor is typically a vector formed by concatenating histograms of gradient directions computed in sub-regions around the keypoint.

Using the OpenCV library, implementing the SIFT algorithm is straightforward. 2 examples are shown in Figure 4.

2. **K-means clustering:** Given a set of observations $(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n)$, where each observation is a d-dimensional real vector, k-means clustering aims to partition the n observations into k set $\mathbf{S} = \{S_1, S_2, ..., S_k\}$ so as to minimize the within-cluster sum of squares (WCSS) (i.e. variance). Formally, the objective is to find:

$$\arg\min_{\mathbf{S}} \sum_{i=1}^{k} \sum_{\mathbf{x} \in S_i} \|\mathbf{x} - \boldsymbol{\mu}_i\|^2 = \arg\min_{\mathbf{S}} \sum_{i=1}^{k} |S_i| \operatorname{Var} S_i$$

where $\boldsymbol{\mu}_i$ is the mean (also called centroid) of points in S_i. And steps of the K-means Algorithm are:

(a) Initialization: Choose the number of clusters k and Initialize k centroids randomly or using some heuristic method.

(b) Assignment Step: Assign each data point to the nearest centroid based on the Euclidean distance. This forms k clusters.

(c) Update Step: Calculate the new centroids by taking the mean of all data points assigned to each cluster.

(d) Repeat: Repeat the assignment and update steps until the centroids no longer change or change very little, indicating convergence.

Using the scikit-learn library to implement the algorithm. We clustered all features into 100 categories.

```
estimator = sklearn.cluster.KMeans(n_clusters=100, max_iter=300, n_init='auto')
estimator.fit(descriptors)
```

3. **Obtain the BOF vector:**

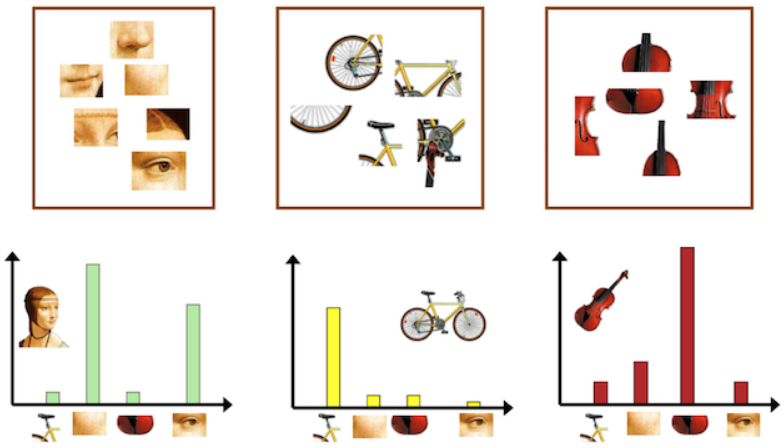

Figure 5: Bag of Features.

The BOF (Bag of Features) algorithm is actually the application of the BOW (Bag of Words) algorithm in the field of images. Based on the categories obtained in the previous step, determine which words the features of each image belong to, and obtain a word frequency vector of the same dimension as the number of words in the dictionary. This is the initial BOF vector. Multiply it by TF-IDF weights as the final BOF vector.

TF-IDF Weights for Visual Words:

- In an article, some words have a relatively high frequency of occurrence, such as: you, to, of, etc. These high-frequency words can greatly lead to text retrieval errors.
- IDF (Inverse Document Frequency): This is also true in BOF algorithms for image retrieval. Some similar features appear more frequently in many images, which can lead to incorrect image retrieval. The solution is to add a weight to the extracted feature. If this feature appears more frequently, its weight should be lower. Conversely, if some features appear less frequently, it indicates that they better represent the corresponding image, and the weight should be higher. The following figure shows the formula for calculating IDF weights, adding 1 to prevent the denominator from being 0.

$$IDF = \log(\frac{N}{n+1})$$

  where N is the total number of images and n_w is The number of images containing this word.

- TF (Word Frequency): The inverse document word frequency targets the extracted SIFT features, while word frequency targets visual words. Contrary to the reverse document word frequency mentioned above, if a visual word appears more frequently in an image, it is more representative of that image. The formula for calculating the weight of TF is as follows:

$$TF = \frac{f}{m}$$

  where f is the number of times words appear in the image and m is the total number of words in the image.

4. **Similarity sorting:** Extract feature descriptors and get BOF vectors from the test dataset, then sort the test machine images by cosine similarity. The cosine similarity between two BOF vectors is calculated as followings:

$$\text{Similarity (A, B)} = \frac{\mathrm{A} \cdot \mathrm{B}}{||\mathrm{A}|| \times ||\mathrm{B}||} = \frac{\sum_{\mathrm{i}=1}^{\mathrm{n}}(\mathrm{A_i} \times \mathrm{B_i})}{\sqrt{\sum_{\mathrm{i}=1}^{\mathrm{n}} \mathrm{A_i^2}} \times \sqrt{\sum_{\mathrm{i}=1}^{\mathrm{n}} \mathrm{B_i^2}}}$$

   For each test image, display the five images with the highest similarity. We can see that if the test image is classified as the image with the highest similarity, then all test images are successfully retrieved.

Test image: query.jpg

Test image: 26.jpg

Test image: 27.jpg

Test image: 28.jpg