Computer Vision

Chennai Mathematical Institute

Assignment 3

Task

The task is to to create a local feature detection and description algorithm. We want to implement a simplified version of sift. Need to implement the two distinct steps of feature detection and description that are part of SIFT , (a) Interest Point Detection and (b) Local Feature Description

Submitted Files

1. **Harris.py**
2. **Sift.py**
3. **Notebook**
4. **Documentation**

Interest point detection

Harris Corner Detector is used to extract corners and infer features of an image. We want to identify new features by extracting corners. The code for this written in harris.py file. The idea is to consider a small window around each pixel in an image. We measure the amount of change that occurs in the pixel for small shift. We calculate this value by shifting the window by a small amount in a given direction. For corner there will be significant change in all direction. Our target is to maximize this change for corner detection. Our final problem boils down to compute the harris response. The steps and formulas are described below.

Steps:

1. First we convert our image to gray scale.

2. We then apply sobel filter on the grayscale image to find the horizontal and vertical derivatives of the image I_x, I_y

3. We compute three images corresponding to the outer products of these gradients. We get I_x^2, I_y^2, I_xI_y,

4. After convolving each of these images in step 3 with a larger gaussian we get, I_{xx}, I_{yy}, I_{xy} .

5. We then compute the harris response using the formula -

   Harris response, R = det(M) - k * $trace(M)^2$, (k is generally taken between 0.04 and 0.06)

$$M = \begin{pmatrix} I_{xx} & I_{xy} \\ I_{xy} & I_{yy} \end{pmatrix}$$

6 . We then search for local maxima.

We take a window,(say 3*3) around each pixel and compare that pixel with 8 neighbour pixels. We can just take the point or location if it has the maximum value(strictly greater than the others) in the neighbourhood. We can also take a thresold that if a point is greater than some fraction of maximum response value and is also a local maxima then it will be considered as interest point.

7. At last based on the feature width we remove some boundary points. We can avoid doing this step separately. But in case we want all the points including the boundary points, separating this step will help.

8. **ADAPTIVE NON-MAXIMAL SUPPRESSION:**

The same procedure followed as described in the problem.
We detect local maxima and whose response value is significantly (10%) greater than that of all of its neighbors within a radius r. The goal is to retain only those points that are a maximum in a neighborhood of radius r pixels. We do it by sorting all the points by the response strength from large to small response. The first entry in the list is the global maximum, which is not suppressed at any radius. Then, we iterate through the list and compute the distance to each interest point ahead of it in the list (these are pixels with even greater response strength). The minimum of distances to a keypoint's stronger neighbors (multiplying these neighbors by >=1.1 to add robustness) is the radius within which the current point is a local maximum. We call this the suppression radius of this interest point, and we save these suppression radii. Finally, we sort the suppression radii from large to small, and return the n keypoints associated with the top n suppression radii, in this sorted orderself. For different images n is taken differently based on the number of interest points.

First We get a list of tuples (x,y,R). (x,y) is location and R is response value. This values are sorted (in decresing order) based on the R value. After implementing ANMS we get a list of tuples of type(x,y,R, supression radius). Suppression radius for i th tuple(x,y,R) of the list is obtained by calculating minimum distance(Euclidean distance of location) of it with the tuples 0 to (i-1) having higher response values. Here we give one more condition. Minimum distance if response value is greater than 1.1 of it's own response value.

Functions written to implement Harris -

1. **compare** - This function compares the pixel value at a particular point with the neighbour pixel in the window considered. It returns true if the point is local maxima, otherwise false.

2. **keep_range_in_bound** - We need this function when we are considering window around a point. For points near boundary it may happen that the window is out of range. In that case we will consider only those part which is inside the image.

In comapre function we find the max and min co-ordinates of the window and apply this function. For e.g. - If we are considering a 5*5 window around point (1,1) then min location is (0,-1) and max location is (2,3). But we consider from (0,0) to (2,3). Same way if it exceeds the no of rows or columns.
We can write this function in several ways. Another algorithm is shown in the file in comments. Both works but complexity is slightly different.

3. **find_interest_points** - It uses the compare function and returns interest points as described in step 6.

4. **check_point** - It just deals the case of feature width. Based on the feature width it removes some boundary interest points.

5. **do_ANMS** - ANMS is implemented as described in step 5. If ANMS is set to be true then it implements ANMS.

Results

For the image Norte Dame -

Resize factor : 0.5

Shape of gray scale image: (1024, 768)
Feature width taken: 40
No of interest Points found : 1278
No of interest Points based on feature width: 1248
Window size - 5*5

All the points are plotted. Based on ANMS result We can choose first 1000 points.

For the image Episcopal Gaudi -

Resize factor : 0.5

Shape of gary scale image: (600, 800)
Feature width taken: 50
No of interest Points: 894
No of interest Points based on feature width: 837
Window size - 5*5
All the points are plotted. Based on ANMS result We can choose first desired no of points.

For the image Mount Rushmore -

Resize factor : 0.5
Shape of gary scale image: (972, 1296)
Feature width taken: 16
No of interest Points: 2601
No of interest Points based on feature width: 2517
Window size - 7*7
All the points are plotted. Based on ANMS result We can choose first desired no of points.

For the image Chess board -

Resize factor : 1

Shape of gary scale image: (700, 700)
Feature width taken: 16
No of interest Points: 638
No of interest Points based on feature width: 638
Window size - 3*3
All the points are plotted. Based on ANMS result We can choose first desired no of points.

The results are satisfactory. The algorithm detected the corners very well. In chessboard image the results are easily verified. The results are clear for other images also. The results are shown in notebook.

Local feature description

Scale Invariant Feature Transform:

Here we have the keypoints(interest points) which we got using harris.py.

We now want to create the local descriptor.

1.We first apply sobel filter on the grayscale image to find the x and y gradient values for every pixel.

2. We then compute two matrices. One contains magnitude of gradients and other contains the orientation for each pixel. We store another matrix which is nothing but gaussian applied on gradient magnitude matrix.

Let say we are considering feature width 16

3. Now we consider a 16*16 grid around the interest point.

4. Now we find the orientation of the key point based on the the neighbourhood considered in step 3.

We create a histogram with 36 bins(as angles are from 0 to 360 deg. Each bin width is 10) for the 16*16 grid extracted from

the orientation matrix. We have the corresponding weights extracted from the gaussian applied gradient magnitude matrix.

The above procedure of creating the weighted histogram can be done in two ways -

(a) Just add the corresponding weights to the bin in which one orientation value falls.

(b) **Interpolation:** A orientation value actually contributes to multiple bins. In which it falls and also the two adjacent bins. So based on the distance from mid point we assign some part of the weight to adjacent bins.
If orientation value is less than the midpoint of a bin then -
Distance measure, d = (Mid point of the bin - orientation value) / bin width. We assign (1-d) to the it falls and d to the previous bin. The algorithm works in this way. We can check whether the orientation value is greater or not and then assign parts to the bin. It means If a orientation value is less than a mid point of bin then it contributes to this bin and also the previous bin based on the distance of mid point of previous bin also.
Here we store the weights corresponding to each bin.

Now we want to assign the orientation to the key point based on the histogram weights obtained. This again can be done in two ways.
(a) We can just assign the midpoint of the bin having maximum weight as the orientation of the key point.(i.e. the dominant orientation)

(b) **Fit Parabola:** We can check for multiple peaks in the histogram. If we don't have any bin with weight higher than 0.8*weight of bin corresponding to dominant orientation then we just return the dominant orientation. Otherwise if we have two more peaks then **we fit a parabola** to get the orientation of the key point. We just need to solve a linear system of equations for 3 parameters with 3 equations.

We fit a parabola - ax^2 + bx + c = y and we return - b/2a as the orientation of key point. x values are midpoints of those bins and y value correspond to weights of those bins.

5. We now replace the orientation of the key point with the one obtained in step 4.

6. Now for each point in that 16*16 grid except the keypoint, we subtract the keypoint orientation from each point's own orientation value and store that in place of previous ones. This make it rotation invariant.

7. Now we consider each non-overlapping 4*4 grid of this big 16*16 grid starting from the (0,0) th position. We will have total 16 4*4 grids.

8. We now compute histogram with 8 bins for each of these grids. Here bin width is (360/8) = 45. We follow the same process described in step 4 to compute the histogram weights but with 8 bins. Here we don't need to fit any parabola as we are only interested in 8 weights obtained. So for each of the bins we will a (8*1) vector. We normalize the values. For total 16 bins we have a vector of length 128. This vector is the sift vector for the interest point we considered.

9. We do all these steps for all the interest points and get a vector of length 128 for each of these interest points.

Functions written to implement sift -

1. **box_around_point** - It takes a matrix, location and feature_width returns a sub matrix which is around the "location" of the matrix

2. **extract_small_boxes** - Takes any matrix(here sub matrix of step 1) and size returns all the non-overlapping small grids possible from the starting position (0,0) for that matrix with their actual location in the main matrix of step 1

3. **adjust_angle** - We adjust the angle. As the angle ranges from -180 to 180 degree we convert it to 0 to 360 degree. We add 360 when the angle is within -180 to 0 and keep it same when it is within 0 to 180.

4. **interpolated_hist** - It does the same as mentioned in step 4. Returns the weights corresponding to the bins.(Fitting parabola part is within the function fit_parabola.

Optional: **Histogram**- It doesn't give weight to multiple bins. Works the way normal histogram occurs.

5. **fit_parabola** - We fit parabola the way it mentioned in step 4.

6. **rotation_invariant** - Based on the neighbourhood orientations we got the orientation of key point. Now we subtract it from each value of the 16*16 grid except the key point. It makes sift rotation invariant.

7.**return_sift_vector** - It gives the final sift vector for an interest point identified using harris.py . It implements the same as described in step 8.

Results

One sift vector for one interest point for the images are shown in the notebook. The shapes are shown there.

Extra Credit

1. Scaling factor - Scaling factor is considered. Resizing the scale and implementing the same will help.
2. Rotation Invariant - It is done in sift.py. We estimated the orientation of keypoint based on the neighbour orientations. We fit parabola to estimate the orientation. Then we make local features rotation invariant as described in step 6 of sift.
3. Interpolation - Explained in step 4.
4. Fit parabola for orientation of key point - Explained in step 4.

These extra credit things are also in the places it is used.
All the results are shown in the notebook.