16-720 Computer Vision: Homework 2
Keypoints - Detectors, Descriptors and Matching

Instructor: Srinivasa Narasimhan
TAs: Arun Venkatraman, Aayush Bansal, Yiying Li

Due: Tuesday, February 24, 11:59:59 p.m.

If you have any questions, please contact TAs - Arun (`arunvenk@cs.cmu.edu`), Aayush (`aayushb@cs.cmu.edu`), and Yiying (`yiying@cmu.edu`). **Start early! Start early!**

Introduction

In this homework, you will implement an interest point (keypoint) detector and a feature descriptor. Interest point (keypoint) detectors find particularly salient points in an image. We then can then extract a feature descriptor that helps describe the region around each of the interest points. SIFT, SURF and BRIEF are all examples of commonly used feature descriptors. Once we have extracted the interest points, we can use feature descriptors to match them (find correspondences) between images to do neat things like panorama stitching or scene reconstruction.

We will be implementing the BRIEF descriptor in this homework. It has a very compact representation, is quick to compute, and has a discriminative yet easily computed distance metric, and is relatively simple to implements. This allows for real-time computation, as you have seen in class. Most importantly, as you will see, it is also just as powerful as more complex descriptors like SIFT, for many cases.

1 Keypoint Detector

For our implementation, we will be using the interest point detector similar to the one introduced in class. A good reference for its implementation can be found in [3]. Keypoints are found by using the Difference of Gaussian (DoG) detector. This detector finds points that are extrema in both scale and space of a DoG pyramid. This is described in [1], an important paper in our field. Here, we will be implementing a simplified version of the DoG detector described in Section 3 of [3].

NOTE: The parameters to use for the following sections are:
$\sigma_0 = 1$**,** $k = \sqrt{2}$**,** `levels` $= [-1, 0, 1, 2, 3, 4]$**,** $th_{contrast} = 0.03$ **and** $th_r = 12$.

1.1 Gaussian Pyramid

In order to create a DoG pyramid, we will first need to create a Gaussian pyramid. Gaussian pyramids are constructed by progressively applying a lowpass Gaussian filter to the input image. This function is already provided to your in `matlab/createGaussianPyramid.m`.

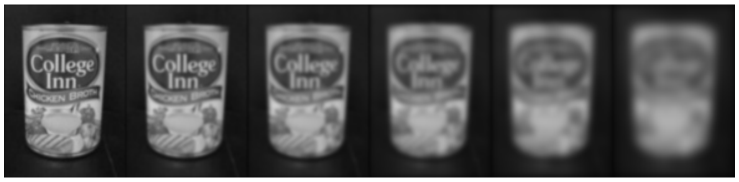

Figure 1: Example Gaussian pyramid for `model_chickenbroth.jpg`

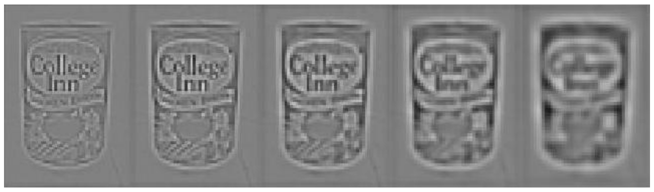

Figure 2: Example DoG pyramid for `model_chickenbroth.jpg`

```
GaussianPyramid = createGaussianPyramid(im, sigma0, k, levels)
```

The function takes as input a grayscale image im with values between 0 and 1 (hint: `im2double()`, `rgb2gray()`), the scale of the zeroth level of the pyramid `sigma0`, the pyramid factor `k`, and a vector `levels` specifying the levels of the pyramid to construct.

At level l in the pyramid, the image is smoothed by a Gaussian filter with $\sigma_l = \sigma_0 k^l$. The output `GaussianPyramid` is a $R \times C \times L$ matrix, where $R \times C$ is the size of the input image `im` and L is the number of levels. An example of a Gaussian pyramid can be seen in Figure 1. You can visualize this pyramid with the provided function

```
displayPyramid(pyramid)
```

1.2 The DoG Pyramid (5 pts)

The DoG pyramid is obtained by subtracting successive levels of the Gaussian pyramid. Specifically, we want to find:

$$D_l(x, y, \sigma_l) = (G(x, y, \sigma_{l-1}) - G(x, y, \sigma_l)) * I(x, y) \tag{1}$$

where $G(x, y, \sigma_l)$ is the Gaussian filter used at level l and $*$ is the convolution operator. Due to the distributive property of convolution, this simplifies to

$$\begin{aligned} D_l(x, y, \sigma_l) &= G(x, y, \sigma_{l-1}) * I(x, y) - G(x, y, \sigma_l) * I(x, y) && (2) \\ &= GP_l - GP_{l-1} && (3) \end{aligned}$$

where GP_l denotes the l level in the Gaussian pyramid.

Q 1.2: Write the following function to construct a Difference of Gaussian pyramid (we have provided a stub file in `matlab/createDoGPyramid.m`):

```
[DoGPyramid, DoGLevels] = createDoGPyramid(GaussianPyramid, levels)
```

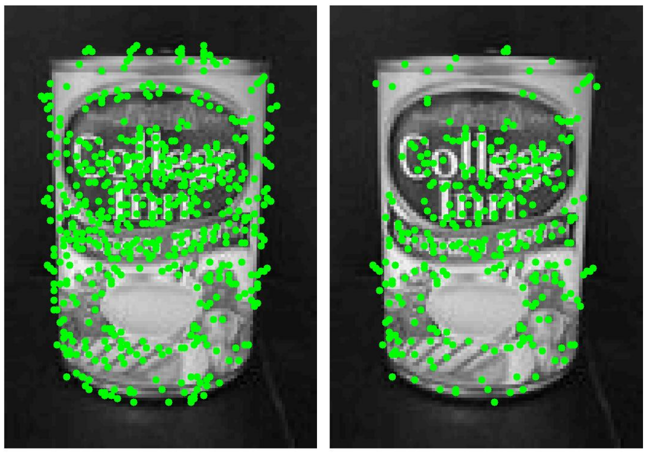
(a) Without edge suppression (b) With edge suppression

Figure 3: Interest Point (keypoint) Detection without and with edge suppression for `model_chickenbroth.jpg`

The function should return `DoGPyramid` an $R \times C \times (L-1)$ matrix of the DoG pyramid created by differencing the `GaussianPyramid` input. Note that you will have one level less than the Gaussian Pyramid. `DoGLevels` is an $(L-1)$ vector specifying the corresponding levels of the DoG Pyramid (should be the last $L-1$ elements of `levels`). An example of the DoG pyramid can be seen in Figure 2.

1.3 Edge Suppression (10 pts)

The Difference of Gaussian function responds strongly on corners and edges in addition to blob-like objects. However, edges are not desirable for feature extraction as they are not as distinctive and do not provide as stable of a localization for keypoints. Here, we will implement the edge removal method described in Section 4.1 of [3], which is based on the principal curvature ratio in a local neighborhood of a point. The paper makes the observation that edge points will have a "large principal curvature across the edge but a small one in the perpendicular direction."

Q 1.3: Implement the following function (stub provided in `matlab/computePrincipalCurvature.m`):

```
PrincipalCurvature = computePrincipalCurvature(DoGPyramid)
```

The function takes in `DoGPyramid` generated in the previous section and returns `PrincipalCurvature`, a matrix of the same size where each point contains the curvature ratio R for the corresponding point in the DoG pyramid:

$$R = \frac{\mathrm{Tr}(H)^2}{\mathrm{Det}(H)} = \frac{(\lambda_{min} + \lambda_{max})^2}{\lambda_{min}\lambda_{max}} \tag{4}$$

where H is the Hessian of the Difference of Gaussian function (i.e. one level of the DoG pyramid) computed by using pixel differences as mentioned in Section 4.1 of [3]. (hint:

Matlab function `gradient()`).

$$H = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{yx} & D_{yy} \end{bmatrix} \tag{5}$$

This is similar in spirit to but *different* than the Harris matrix you saw in class. Both methods examine the eigenvalues λ of a matrix, but the method in [3] performs a test **without** requiring the direct computation of the eigenvalues. Note that you need to compute each term of the Hessian before being able to take the trace and determinant.

We can see that R reaches its minimum when the two eigenvalues λ_{min} and λ_{max} are equal, meaning that the curvature is the same in the two principal directions. Edge points, in general, will have a principal curvature significantly larger in one direction than the other. To remove edge points, we simply check against a threshold $R > th_r$. Fig. 3 shows the DoG detector with and without edge suppression.

1.4 Detecting Extrema (10 pts)

To detect corner-like, scale-invariant interest points, the DoG detector chooses points that are local extrema in both scale and space. Here, we will consider a point's eight neighbors in space and its two neighbors in scale (one in the scale above and one in the scale below).

Q 1.4: Write the function (stub provided in `matlab/getLocalExtrema.m`):

```
locs = getLocalExtrema(DoGPyramid, DoGLevels, PrincipalCurvature,
                       th_contrast, th_r)
```

This function takes as input `DoGPyramid` and `DoGLevels` from Section 1.2 and `PrincipalCurvature` from Section 1.3. It also takes two threshold values, `th_contrast` and `th_r`. The threshold $th_{contrast}$ should remove any point that is a local extremum but does not have a Difference of Gaussian (DoG) response magnitude above this threshold (i.e. $|D(x, y, \sigma)| > th_{contrast}$). The threshold th_r should remove any edge-like points that have too large a principal curvature ratio specified by `PrincipalCurvature`.

The function should return `locs`, an $N \times 3$ matrix where the DoG pyramid achieves a local extrema in both scale and space, and also satisfies the two thresholds. The first and second column of locs should be the (x, y) values of the local extremum and the third column should contain the corresponding level of the DoG pyramid where it was detected. (Try to eliminate loops in the function so that it runs efficiently.)

1.5 Putting it together (5 pts)

Q 1.5: Write the following function to combine the above parts into a DoG detector (we have provided a stub file in `matlab/DoGdetector.m`):

```
[locs, GaussianPyramid] = DoGdetector(im, sigma0, k, levels, th_contrast,
                                      th_r)
```

The function should take in a gray scale image, `im`, scaled between 0 and 1, and the parameters `sigma0`, `k`, `levels`, `th_contrast`, and `th_r`. It should call each of the above functions and return the keypoints in `locs` and the Gaussian pyramid in `GaussianPyramid`. Figure 3 shows the keypoints detected for an example image. Note that we are dealing with real images here, so your keypoint detector may find points with high scores that you do not perceive to be corners.

2 BRIEF Descriptor

Now that we have interest points that tell us where to find the most informative points in the image, we can compute descriptors that can be used to match to other views of the same point in different images. The BRIEF descriptor encodes information from a 9×9 patch p centered around the interest point at the *characteristic scale* of the interest point. See the lecture notes for Point Feature Detectors if you need to refresh your memory.

2.1 Creating a Set of BRIEF Tests (5 pts)

The descriptor itself is a vector that is n-bits long, where each bit is the result of the following simple test:

$$\tau(p; x, y) := \begin{cases} 1, & \text{if } p(x) < p(y). \\ 0, & \text{otherwise.} \end{cases} \tag{6}$$

Set n to 256 bits. There is no need to encode the test results as actual bits. It is fine to encode them as a 256 element vector.

There are many choices for the 256 test pairs (x, y) used to compute $\tau(p; x, y)$ (each of the n bits). The authors describe and test some of them in [2]. Read section 3.2 of that paper and implement one of these solutions. You should generate a static set of test pairs and save that data to a file. You will use these pairs for all subsequent computations of the BRIEF descriptor.

Q 2.1: Write the function to create the x and y pairs that we will use for comparison to compute τ (we have provided a stub file in `matlab/makeTestPattern.m`):

```
[compareX, compareY] = makeTestPattern(patchWidth, nbits)
```

`patchWidth` is the width of the image patch (usually 9) and `nbits` is the number of tests n in the BRIEF descriptor. `compareX` and `compareY` are linear indices into the `patchWidth` $\times$ `patchWidth` image patch and are each $nbits \times 1$ vectors. Run this routine for the given parameters `patchWidth` $= 9$ and $n = 256$ and save the results in `testPattern.mat`. **Include this file** in your submission.

2.2 Compute the BRIEF Descriptor (10 pts)

It is now time to compute the BRIEF descriptor for the detected keypoints.

Q 2.2: Write the function (we have provided a stub file in `matlab/computeBrief.m`):

```
[locs,desc] = computeBrief(im, locs, levels, compareX, compareY)
```

Where `im` is a grayscale image with values from 0 to 1, `locs` are the keypoint locations returned by the DoG detector from Section 1.5, `levels` are the Gaussian scale levels that were given in Section 1, and `compareX` and `compareY` are the test patterns computed in Section 2.1 and were saved into `testPattern.mat`.

The function returns `locs`, an $m \times 3$ vector, where the first two columns are the image coordinates of keypoints and the third column is the pyramid level of the keypoints, and `desc`, an $m \times nbits$ matrix of stacked BRIEF descriptors. m is the number of valid descriptors in the image and will vary. You may have to be careful about the input DoG detector locations since they may be at the edge of an image where we cannot extract a full patch of width `patchWidth`. Thus, the number of output `locs` may be less than the input.

2.3 Putting it all Together (5 pts)

Q 2.3: Write a function (we have provided a stub file in `matlab/brief.m`):

```
[locs, desc] = brief(im)
```

Which accepts a grayscale image `im` with values between zero and one and returns `locs`, an $m \times 3$ vector, where the first two columns are the image coordinates of keypoints and the third column is the pyramid level of the keypoints, and `desc`, an $m \times nbits$ matrix of stacked BRIEF descriptors. m is the number of valid descriptors in the image and will vary.

This function should perform all the necessary steps to extract the descriptors from the image, including

- Load parameters and test patterns
- Get keypoint locations
- Compute a set of valid BRIEF descriptors

2.4 Descriptor Matching (15 pts)

A descriptor's strength is in its ability to match to other descriptors generated by the same world point, despite change of view, lighting, etc. The distance metric used to compute the similarity between two descriptors is critical. For BRIEF, this distance metric is the Hamming distance. The Hamming distance is simply the number of bits in two descriptors that differ. (Note that the position of the bits matters.) Hint: Matlab provides this in `knnsearch()` or in `pdist2()` – check the Matlab help!

Q 2.4.1: Write the function (we have provided a stub file in `matlab/briefMatch.m`):

```
[matches] = briefMatch(desc1, desc2, ratio)
```

Which accepts an $m \times nbits$ stack of BRIEF descriptors from a first image and a $n \times nbits$ stack of BRIEF descriptors from a second image and returns a $p \times 2$ matrix of matches, where the first column are indices into `desc1` and the second column are indices into `desc2`. Note that m, n, and p may be different sizes.

Section 7.2 of [3] introduces a ratio test to suppress matches between descriptors that are not particularly discriminative. **Implement this test** and select a value for ratio. In your PDF, mention how you designed this test and how you selected ratio.

Q 2.4.2: Write a test script `testMatch` to load two of the **chickenbroth** images, compute feature matches. Use the provided `plotMatches` function to visualize the result.

```
plotMatches(im1, im2, matches, locs1, locs2)
```

Where `im1` and `im2` are grayscale images from 0 to 1, `matches` is the list of matches returned by `briefMatch` and `locs1` and `locs2` are the locations of keypoints from brief.

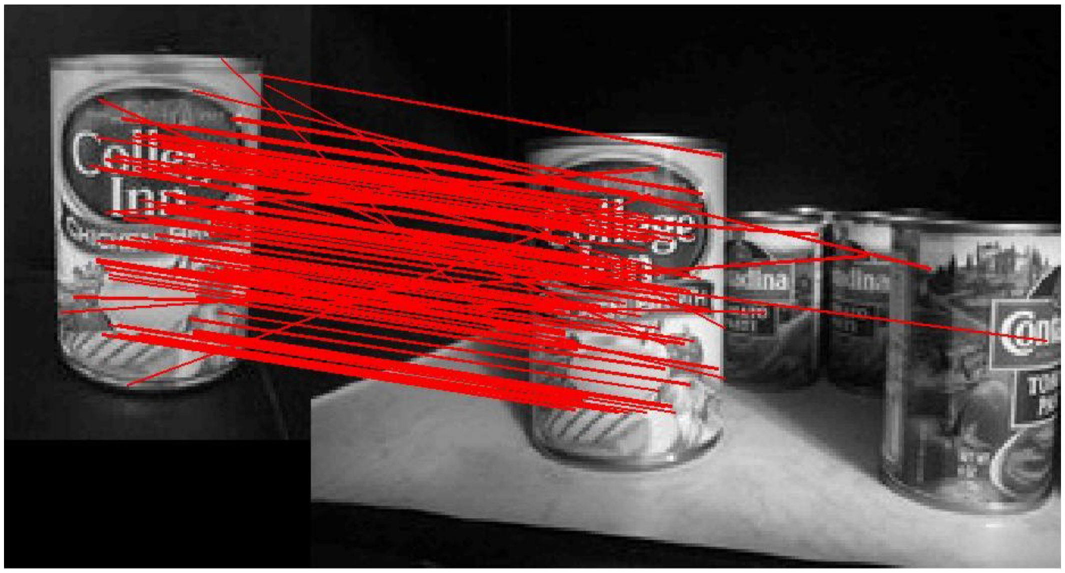

Figure 4: Example of BRIEF matches for `model_chickenbroth.jpg` and `chickenbroth_01.jpg`.

Save the resulting figure and submit it in your PDF. Also, present results with the two `taj` images and with the computer vision textbook cover page (template is in file `pf_scan_scaled.jpg`) against the other `pf_*` images. Briefly discuss any cases that perform worse or better.

Figure 4 is an example result. A good test of your code is to check that you can match an image to itself.

2.5 BRIEF and rotations (10 pts)

You may have noticed worse performance under rotations. Let's investigate this!

Q 2.5: Take the `model_chickenbroth.jpg` test image and match it to itself while rotating the second image (hint: `imrotate`) in increments of 10 degrees. Count the number of correct matches at each rotation and construct a bar graph showing rotation angle vs the number of correct matches. Include this in your PDF and explain why you think the descriptor behaves this way. **Create a script** `briefRotTest.m` that performs this task.

3 Extra Credit

The extra credit opportunities described below are optional and provide an avenue to explore computer vision and improve the performance of the techniques developed above.

1. **(10 pts)** As we have seen, BRIEF is not rotationally invariant. Design a simple fix to solve this problem using the tools you have developed so far. Explain in your PDF

your design decisions and how you selected any parameters that you use. Demonstrate the effectiveness of your algorithm on image pairs related by large rotation.

2. **(10 pts)** This implementation of BRIEF has some scale invariance, but there are limits. What happens when you match a picture to the same picture at half the size? Look to section 3 of [3] for a technique that will make your detector more robust to changes in scale. Implement it and demonstrate it in action with several test images. You may simply rescale some of the test images we have given you.

4 What to Submit

Your submission should consist of only 1 file, a zip file named `<AndrewId>.zip`. This zip file should contain

- a folder `matlab` containing all the `.m` and `.mat` files you were asked to write and save
- a pdf named `writeup.pdf` containing the results, explanations, graphs, and images asked for in the assignment

You may leave the `data` folder and the provided `.m` files in your submission, but they are not needed. Please submit you homework using blackboard as usual.

References

[1] P. Burt and E. Adelson. The Laplacian Pyramid as a Compact Image Code. *IEEE Transactions on Communications*, 31(4):532–540, April 1983.

[2] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. BRIEF : Binary Robust Independent Elementary Features .

[3] David G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60(2):91–110, November 2004.