DEPARTMENT OF COMPUTER SCIENCE, VISION AND IMAGE PROCESSING

Assignment 1: Feature Extraction

Alexander Wahl-Rasmussen, lgv740

December 2, 2013

1 INTRODUCTION

In this report I will first present a common approach to detect interest points and apply it to two different images. Thereafter, I will build upon that baseline by including blob detection and matching interest points between two images. Finally, I will comment on how to make the solution more robust.

2 DETECTING INTEREST POINTS

Detecting interest points or blobs can be done in many ways, where the two main methods are via differential calculations that relates to position or locating local extrema.
In this report I have chosen to locate extrema with a common measure, of detecting salient objects, by calculating the laplacian of gaussians and then use raw-patching-pixels to find local extrema.
Generally speaking, the locating an extrema is done by iterating through every pixel and comparing it to its' neighbours. If the pixel has the lowest or highest value it will be defined as a local extrema and thereby classified as salient pixel. This can be done on any picture, although the usability of the results will vary depending on what you combine it with. If you run it on the array of a greyscale picture, you will, generally speaking, find the "whitest" and "blackest" pixels in the desired area. However, that is necessarily not what you want, which where the Laplacian of Gaussians come into play.

When I speak of a Laplacian of Gaussians I speak of differentials of distributions. A Gaussian distribution is a distribution where most of the data are centred along the normal with fewer values as outliers, effectively making its' function a bellcurve. On the other hand, a Laplacian

is the divergence of the gradient of a function on Euclidean space. Applying the Laplacian on a Gaussian distribution in scale-space will then return a field of scalars where the highest numerical value will denote the extrema. An example of this can be seen below.

$$LoGfield = \begin{bmatrix} 0 & -1 & 0 \\ -1 & \mathbf{-4} & -1 \\ 0 & -1 & 0 \end{bmatrix} \tag{2.1}$$

Patching on these values can then be quite useful if we want to denote specific features in a picture. See figure 2.1 for an illustration on how detecting interest points can both be used for detecting features on one object and differentiating between objects. However, the results are not very good for these pictures, but I will get back to that.

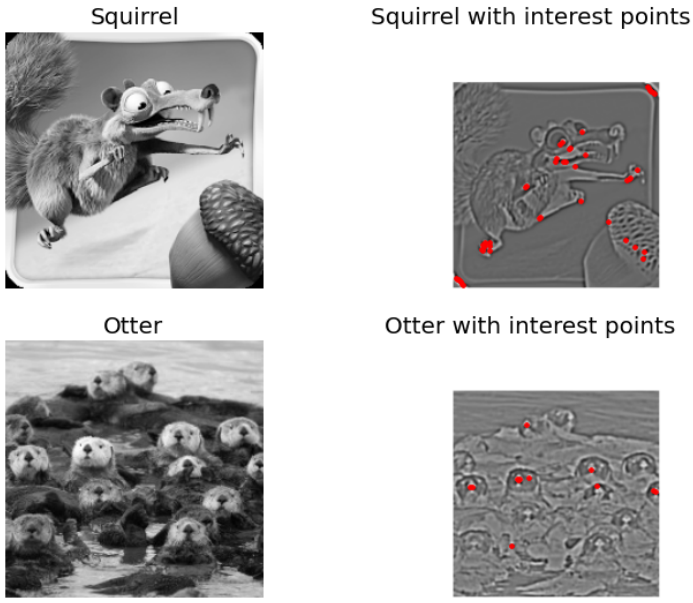

Figure 2.1: Detecting interest points

2.1 My implementation

As introduced above, my implementation utilizes the Laplacian of Gaussians to do interest point detection. To do this, I first open the images and convert them into arrays with numpy's array function. Each element inside these arrays are then floats denoting the grayscale colour. The floats range from 0 to 255 for the specific pixel they correspond to starting with the first element corresponding to the pixel in the uppermost left corner. I then apply SciPy's Laplacian of Gaussians to the image arrays as to get the scalar gradients. With this I can compute

the local extremas by iterating over each pixel and its four neighbours (top, down, left, and right) with an arbitrary threshold of 30/-30 . As the adjective suggest, the threshold has been set with the engineer's approach of trial and error to see what yielded acceptable detections (for the next part).
As you can see from Figure 2.1, there are many objects which you could describe as being salient that are not defined as as salient by the program. The first image is a snippet taken from Ice Age, and we can see that applying my solution enables us to identify the "squirrel"'s feet, eyes, claws amongst other things. Note that such features are found because their colour intensity is stronger than the surrounding pixels and their numerical value is higher than 30. The same goes for the Otter picture, where it locates some of the noses, but not all. As noted above, this is a very basic approach to detecting interest points. So let us move on and see what happens when we have two nearly identical images and we want to find the same features in both of the pictures.

3 SIMPLE MATCHING OF FEATURES

If we seek to expand upon the idea of detecting interest point, one way could be to match the features of one image to another. This is relevant in the case of nearly identical images, where the images depict the same thing albeit from different perspective as in the case of Figure 3.1 & Figure 3.2 below.

Figure 3.1: Image 1

Figure 3.2: Image 2

Although the pictures look quite alike, there's a small difference in perspective. Identifying matching points in these two images is not difficult when its' done by eyesight e.g. a matching point could be the uppermost left window on the left building. Computing it, however, can prove challenging. If we use the solution from the previous section we see that there's a dissimilarity in how many extrema are found in each picture[1], which is plotted below in Figure 3.2.

So what can we do instead? Increasing our point detecting to a patch detection, with each blob having a size of N, leaves us with a bit more information to evaluate on, since we instead have a vector to compute with. It is then possible to do a normalized cross-correlation of

[1] 166 for Image 1 & 181 for Image 2

each patch to another. This is done by substracting the mean of the patch, for both patches, and then dividing by the product of their standard deviation. This will return a single scalar product ranging from -1 to 0 to 1. The closer the product is to 0 the different the patch distributions. This can then be combined with a distance measure, such as the euclidean distance, as to further distinguish a correct patch match to an incorrect.

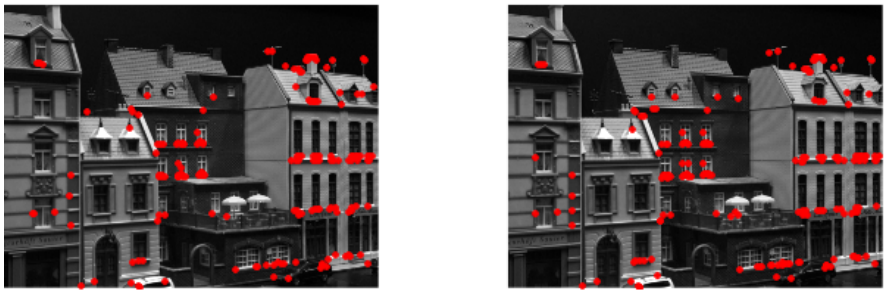

Figure 3.3: Simple detection

3.1 My implementation

My implementation utilizes raw pixel patching to localize patches. This means I, for every pixel extrema found, include the nearest pixels, which in my case is 7, on each side of the pixel. I then call the function `symmanalysis` on each images' list of patches, which is split into different parts, as to do a two-way analysis of matching pairs. Generally speaking, it matches the patches from Image 1 to the patches in Image 2 and vice versa and stores them in two different lists. Nevertheless, the actual matching is a bit more sophisticated than that. To match every patch correctly I first calculate the normalized cross correlation for every patch in the first image up against every patch in the second image[2], where its squared sum must be above 0.8 before it is considered a pair. Since the pictures are quite alike and without any noticeable shading difference, I have made the assumption that matching points will have somewhat the same distribution. If two patches are above the threshold they are treated as matching pairs. If there is more than one pair, the pairs are then evaluated upon by dividing taking the two patches with the shortest distance to the supposedly matching patch. Only the best match is returned if the ratio between the best and second best is below 0.1, i.e. the best is 90% closer than the second best. The implication is that the detection's precision rate will be extremely high, but it's recall will be severely reduced[3]. Had I been able to quantify the results with an ROC curve such a restriction would not be ideal. But since we are judging good pairs on eyesight, I have chosen to maximize the precision rate instead, as to get a high true positive rate and low false positive rate with few data points in general. Finally, I plot the matching pairs onto a merged image with a line denoting every match. See Figure 3.4.

[2] This is not to be confused with Image 1 and Image 2, since the function is called both ways

[3] By recall I mean the ratio between the number of points deemed pairs and the number of points that are actually pairs. Precision is here the ratio between the found pairs that are relevant and the total amount of found pairs

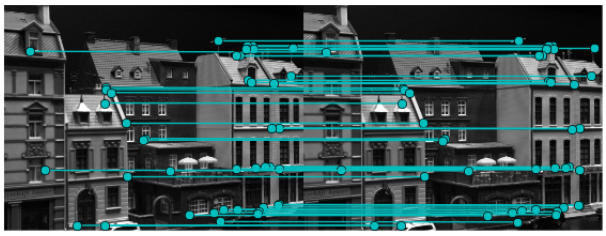

Figure 3.4: Symmetrical matching of Image 1 and 2

3.2 Results

As seen from Figure 3.4, and as stated in the previous section, the matches that are found are actual matches. But if we look at Figure 3.3 we see a loss of obvious pairs. Consider the right building's second floor, where we in Figure 3.2 see alot of interest points that suddenly do not appear in Figure 3.4. This is due to the restrictions I have set and thereby also a conscious choice. So if I had to improve on the result I would first increase the distance threshold, because there are many interest points to be potentially be gained due to being close to other extrema. Another threshold that can be changed is the NCC threshold, but I would be a bit more careful of changing that unless there is an obvious difference in shading. Another approach is to change the patch size until the patch size corresponds to the LoG field of the interest point. However, finding that optimal patch size might prove difficult, if not impossible, due to every interest point may vary in size (e.g. a car versus an eye seen at the same distance). If we do not change the thresholds and run it on a picture with a different perspective than the second picture, the result will look as the following.

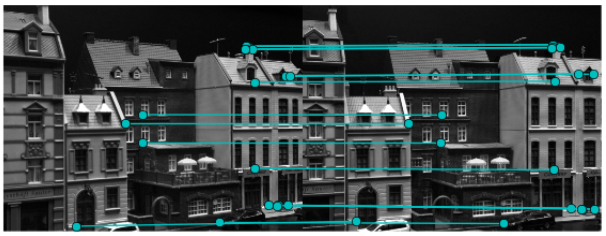

Figure 3.5: Symmetrical matching of Image 1 and 3

This is again partly due to the restrictions I have set with the same reasoning as above, but also due to the gradual change in perspective that makes for gradual changes in the surrounding pixel values. As it is, the best way of optimizing this code for a specific set of images is by trial and error, which is not very useful in the long run. An option would be to improve the Laplacian of Gaussians with scale normalization. This would result in local extrema that are extrema in relation to both scale and space if their numerical value is higher than their 26 neighbours.