Computer Vision
Homework Assignment 8 : Shape Context

Autumn 2018

Nicolas Marchal

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Computer Vision
and Geometry Lab

In this report, I will first go through my implementation in section 1 and 2. I will then answer the four questions at the end of the exercise sheet in section 3.

1 Shape Matching - 60 %

1.1 Shape Context Descriptors (25 %)

I first implemented the function `sc_compute.m` which computes the shape concept descriptor for a given set of points. I implemented the procedure described in [1]. I made sure to use a log scale for the radius of my bins. I used the function `dist2.m` that was provided to compute the average between all the points and normlize my distances.

1.2 Cost Matrix (10 %)

I implemented the function `chi2_cost.m` to compute the cost matrix between two shape context descriptors. To avoid a division by zero in the formula given in the exercise session I added $\epsilon = 10^{-10}$ to the denominator (see equation below).

$$C_{gh} = \frac{1}{2}\sum_{k=1}^{K}\frac{[g(k)-h(k)]^2}{g(k)+h(k)} \rightarrow \frac{1}{2}\sum_{k=1}^{K}\frac{[g(k)-h(k)]^2}{g(k)+h(k)+\epsilon} \tag{1}$$

1.3 Hungarian Algorithm

For this part I simply used `hungarian.m` to perform a one-to-one matching of the points based on the cost matrix, minimizing the total cost. Note that this function assumes a square cost matrix so the function `shape_matching(X,Y,display_flag)` must be called with an equal amount of points in X and Y. Rather than letting the function fail, I took a more robust approach and I select 100 points randomly in each if X and Y are not the same size.

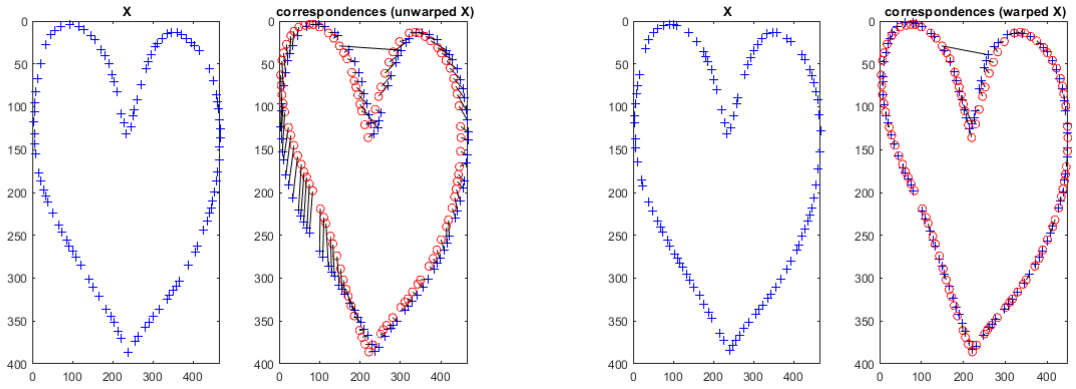
1.4 Thin Plate Splines (25 %)

I implemented the function `tps_model.m` to estimate a plane transformation that matches any point from one shape to the other. By carefully following the explanation in the exercise sheet and in [1] I construct the values of K, P and λ and use MATLAB solver for the linear system.

1.5 Results

I plotted the results, using the first and second image. I will show the correctness of the matching (fig 1) and the plane transformation that allows the hearts to overlap (fig 2).

You should note that I use the function `get_samples_1.m` to sample 100 points in each image. As this is a stochastic sampling, the results can vary. However, we tend to always observe 1-3 mismatch. If these mismatch are too important, they will strongly influence the plane transformation. We can iteratively match points and transform the images by adjusting the parameter `maxIterations`. The effect of this parameter will be discussed in more detail in section 3.



(a) unwraped (b) wraped

Figure 1: Correct Matching

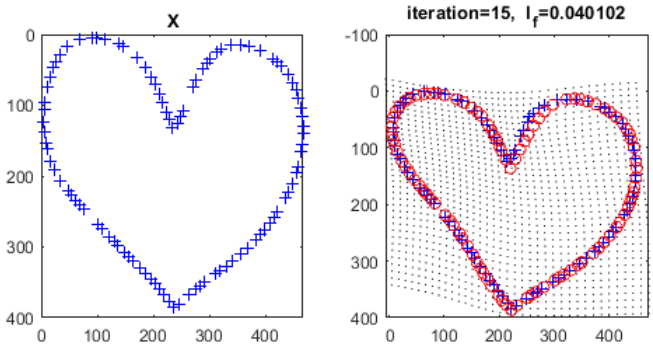


Figure 2: Plane Transformation

2 Shape Classification - 40%

2.1 Shape Matching (25 %)

For this task I implemented the function `compute_matching_costs.m`. This function is quite straightforward as it simply must calculate the cost between all pairs of images. Because the cost is given by the function `shape_matching` that we have implemented in the first part of this exercise, we simply have to call this function for all the pairs of images (total of $15 \cdot 15 = 225$ times). I deactivate the plottings for faster computation. I also added a variable `COMPUTE_MATRIX` in the function `shape_matching` that deactivates all the printing on the command window if it is set to true.

2.2 Nearest-Neighbour Classifier (15 %)

I implemented this final task in the function `nn_classify.m`. I simply sort the cost vector in increasing values and look at the first k labels in the sorted vector. The label will simply be the one that appeared the most in this vector. If the neirest neighbour gives two solutions, I then select one randomly. Note that by using 100 points, the entire classification does not tak 30min as expressed in the project description but around 20 seconds.

3 Questions

Is the shape context descriptor scale-invariant? Explain why or why not.

The shape context descriptor is scale invariant as we normalize all radial distances by the mean distance between all the point pairs in a shape. This was implemented in the function `sc_compute.m` where I changed the radial bin limits according to a normalization factor (as seen in fig 3).

```
normalization = mean2(sqrt(dist2(X,X))) ;
smallest_r = smallest_r*normalization ;
biggest_r = biggest_r*normalization ;
```

Figure 3: Normalization code in \verb |sc_compute.m|

What is the average accuracy of your classifier?

I must first stress that the accuracy of my classifier depends a lot on the number of iteration done in `sc_compute.m` as well as the number of nearest neighbours k. I created a new function called `shape_classification_testiter2(testiter, testk)` where the argument *testiter* is the range on iterations (in the function `shape_matching`) and the argument *testk* is the range of k to test. I ran it with $testiter = 1:35$ and $testk = 1:14$, which allowed me to plot figures 4 and 5. I stored all my results in the file "result compare (xls)". The effect of the number of iterations and numbers of iterations is now discussed :

1. As we always have outliers, doing multiple iterations does not seem to influence the accuracy. I therefore decided to do only 1 iteration in the function `sc_compute.m`, which is convenient as it is the fastest to compute and gives good results.

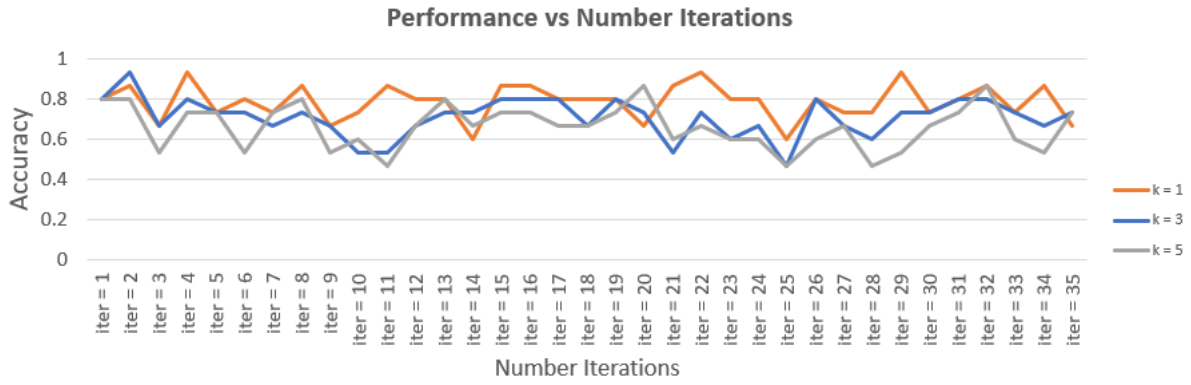


Figure 4: The accuracy does not significantly vary with the number of iterations.

2. the effect of k is discussed in the next question and the final choice is $k = 3$.

→ The best accuracy I can get with my random sampling is $\frac{14}{15}$.

How does your classification accuracy vary with the number of neighbours k?

To evaluate the effect of the nearest neighbours k, I used like before the results that I computed using `shape_classification_testiter2(testiter, testk)`. From figure 5, we can see that the the accuracy decreases as k increases. We notice that until $k = 5$ the accuracy seems more or less constant but suddenly decreases for $k > 5$. This makes sense as there is only 5 examples per class and for all $k' > 5$ there will be at least $k' - 5$ neighbours incorrect.

By looking at figue 5, I could chose more or less any value of k between 1 and 5. I want to avoid $k = 1$ as if the data is noisy or contains wrongly classified image this algorithm will not be robust. I also wanted to avoid choosing $k = 5$ as we have seen that in general the performance decrease if k gets too big. I believe that the best trade-off between robustness and performance is obtained for $k = 3$.

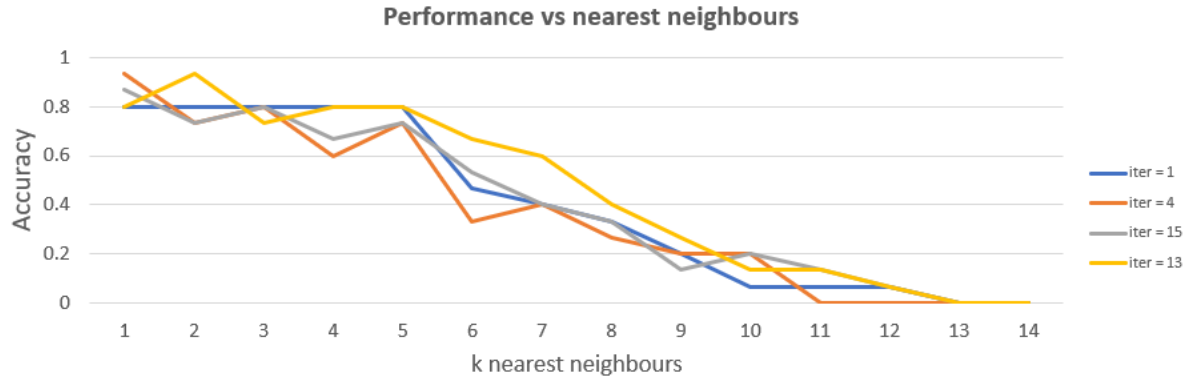


Figure 5: The accuracy decreases after 5 nearest neighbours.

If instead of your own sampling function, you use the one that we provide (get samples 1.m), do you get better classification results? Why or why not?

We get much better classification with the function `get_samples_1` provided. My sampling function chooses random samples, and therefore it is not guaranteed that every sampled shape context descriptor will actually have a match in the second image.

Lets take for example the matching of two hearts, A and B. I might sample 40 points on the right side of A and 60 on its left side. If the random sampling then selects 60 points on the right side of B and 40 on its right, there will be a probem : we have no choice but to match 10 points from the right of A to the left of B since we do a one-to-one matching. This will create a lot of error.

On the other hand, the provided sampling function does not sample the points completely randomly, and it assures that all the points should have a good match. If we compare with the previous examples, `get_samples_1` should sample the same amount of point on the right and left of both hearts, making the one-to-one matching much more precise.

This increases the performance as shown in 6, where we can even get 100% accuracy with the current choice of parameter.

Finally, I would like to point out that my code does not take 30 minutes to run as said in the project description as my function `shape_matching` takes a fourth of a second to run and my function `shape_classification(3)` takes about 20 seconds to run. I use 100 samples in `shape_classification(3)`, and do 1 iteration in `shape_matching`.

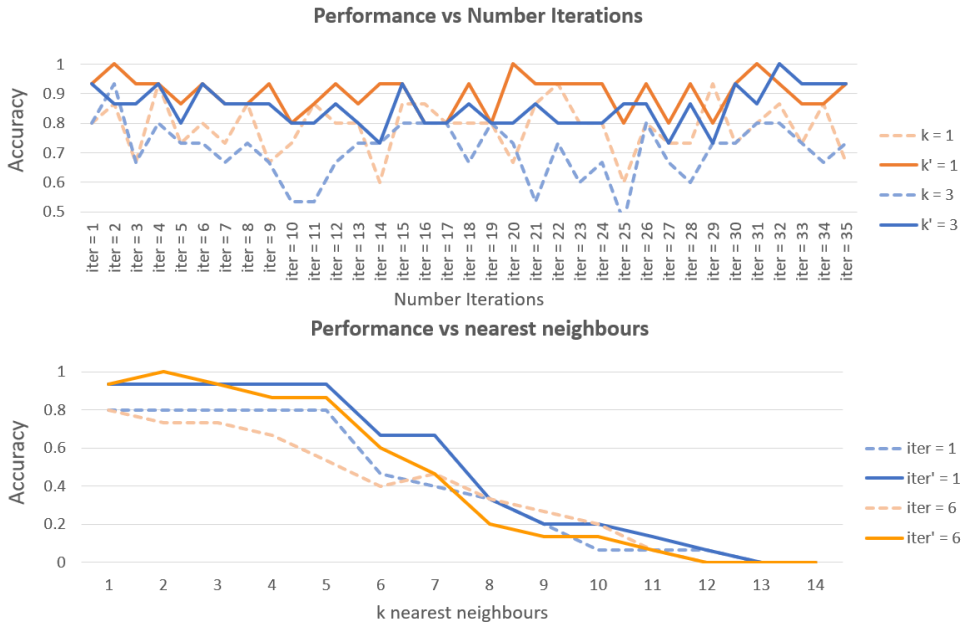


Figure 6: Overall better accuracy for the provided sampling (full line).

References

[1] S. Belongie, J. Malik, and J. Puzicha. Shape matching and object recognition using shape contexts. Technical report, CALIFORNIA UNIV SAN DIEGO LA JOLLA DEPT OF COMPUTER SCIENCE AND ENGINEERING, 2002.