Computer Vision Assignment 8: Shape Context

Charlotte Moraldo

1. Shape Matching

1.1 Shape Context Descriptors

For a given set of points, I compute the shape context descriptors by calling the function: `sc_compute(X,nbBins_theta,nbBins_r,smallest_r,biggest_r)`, where:

- `X` is the set of points
- `nbBins_theta` is the number of bins in the angular dimension
- `nbBins_r` is the number of bins in the radial dimension
- `smallest_r` and `biggest_r` are the length of the smallest and biggest radius, respectively

In order to compute the shape context descriptors, I followed the method described in the paper 'Shape Matching and Object Recognition Using Shape Contexts' [1]. The concept is that for each point of the shape, we must compute a histogram of the relative coordinates of the remaining points.
To do so, I begin by computing the vectors representing the edges of the histogram:

- `theta`, a vector containing `nbBins_theta` equally spaced numbers between 0 and 2π
- `r`, a vector containing `nbBins_r` equally spaced numbers between `log(smallest_r)` and `log(biggest_r)`

Then, for each point of the shape, I compute the relative polar coordinates `X_theta` and `X_r` of the $n-1$ remaining points. For increased robustness, I normalize the radial distance `X_r` by the mean distance of the distance between all points in the shape, and then take the log of this normalized radial distance. I finish by computing the histogram over [`X_theta`, `log(X_r/norm)`] by using the Matlab command `hist3` and by settting the edges to `theta` and `r`.

1.2 Cost Matrix

The function `chi2cost(s1,s2)` returns the cost matrix between two sets of shape context descriptors s_1 and s_2. As shape contexts are represented by histograms, it makes sense to compute the cost using the χ^2 test statistic. The ij-th element of the cost matrix gives the cost of matching a point p_i from the first shape with a point q_j from a second shape:

$$C_{ij} = C(p_i, q_j) = \frac{1}{2}\sum_{k=1}^{K} \frac{[s_{1,i}(k) - s_{2,j}(k)]^2}{s_{1,i}(k) + s_{2,j}(k)}$$

Where:

- $s_{1,i}(k)$ and $s_{2,j}(k)$ are the normalized histograms at points p_i and q_j, respectively
- K is the total number of bins, `nbBins_theta` $*$ `nbBins_r`

[1] Belongie, S., Malik, J. and Puzicha, J., 2002, Shape Matching and Object Recognition Using Shape Contexts, IEEE Trans. PAMI]

1.3 Hungarian Algorithm

The Hungarian Algorithm is already provided. It performs a one-to-one matching of the points based on the cost matrix, minimizing the total cost.

1.4 Thin Plate Splines

Now that I have computed the shape context descriptors, their cost matrix and performed the hungarian algorithm on it, I want to estimate a plane transformation that maps any point from one shape to another. In this case, the Thin Plate Splines (TPS) model is used. Given the points in the template shape `X`, the corresponding points in the target shape `Y`, and the regularizer parameter `lambda` equal to the square of the distance between two target points, I can call the function `tps_model` which computes the following:

- $P = (\mathbf{1}, X)$, meaning the i-th row of P is given by $(1, x_i, y_i)$
- $K = U($`eucl_dist`$)$, where and $U(t) = t^2 log(t^2)$ and `eucl_dist` is equal to the Euclidian distance between all the points in `X`, which can be computed in Matlab by doing: `sqrt(dist2(X,X))`

I can now solve the systems:

$$\begin{pmatrix} K+\lambda I & P \\ P' & 0 \end{pmatrix} \begin{pmatrix} \omega_x \\ a \end{pmatrix} = \begin{pmatrix} v_x \\ 0 \end{pmatrix} \Rightarrow \begin{pmatrix} \omega_x \\ a \end{pmatrix} = \begin{pmatrix} K+\lambda I & P \\ P' & 0 \end{pmatrix}^{-1} \begin{pmatrix} v_x \\ 0 \end{pmatrix}$$

$$\begin{pmatrix} K+\lambda I & P \\ P' & 0 \end{pmatrix} \begin{pmatrix} \omega_y \\ a \end{pmatrix} = \begin{pmatrix} v_y \\ 0 \end{pmatrix} \Rightarrow \begin{pmatrix} \omega_y \\ a \end{pmatrix} = \begin{pmatrix} K+\lambda I & P \\ P' & 0 \end{pmatrix}^{-1} \begin{pmatrix} v_y \\ 0 \end{pmatrix}$$

- the inverse operation is computed in Matlab using the `\` operator
- v_x and v_y are two vectors containing the x and y coordinates of the points on the target shape

After extracting ω_x and ω_y, I can finally compute the bending energy:

$$E = \omega_x' K \omega_x + \omega_y' K \omega_y$$

This whole process - shape descriptors, cost matrix, hungarian algorithm, then TPS - can be done several times. Below is an example of what was obtained when setting the maximum number of iteration to 6, using the provided function `get_samples_1.m`:

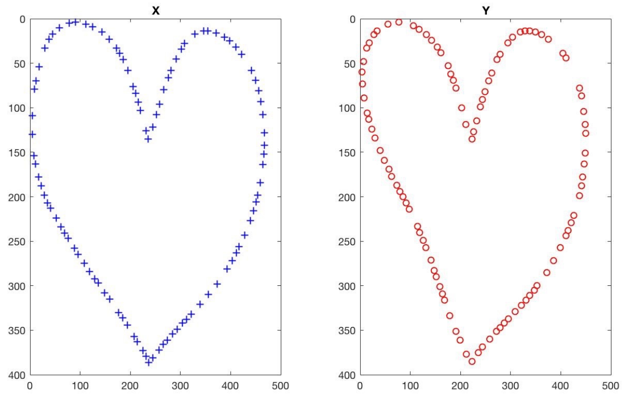

Figure 1: The two shapes being matched

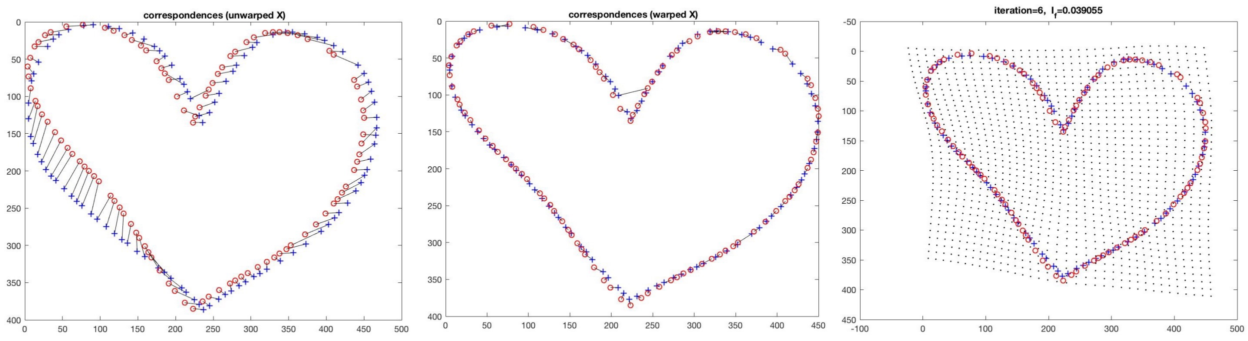

Figure 2

2. Shape Classification

2.1 Shape Matching

After writing a very simple function `get_samples(X)` that samples `nsamp` randomly sampled point in `X`, I can compute the cost of matching each test shape to all training shapes. This is done by calling the function `compute_matching_costs(objects,nsamp)`, which returns a cost matrix of size 15x15, where its ij-th element represents the cost of matching she test shape i to the training shape j. The cost of matching each pair of shapes is the bending energy of the final TPS model, refined after 6 iterations (results shown in section 1.4).
It is interesting to plot the cost matrix with a colorbar to have an idea of the cost value of matching each test shape to all training shapes. I did this using random sampling, and sampling with `get_samples_1`:

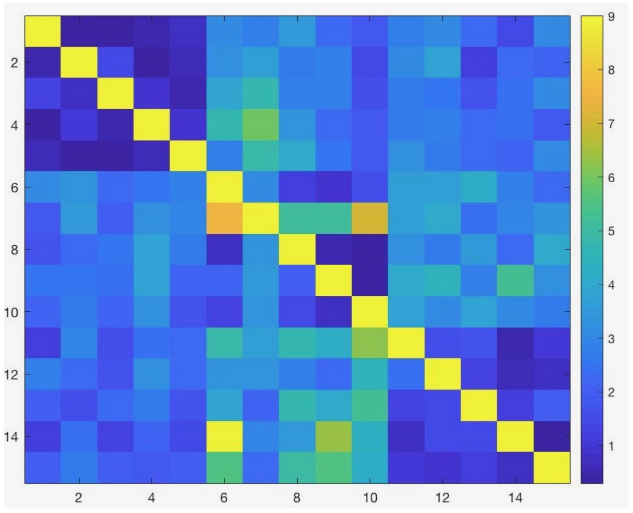

Figure 3: Random sampling

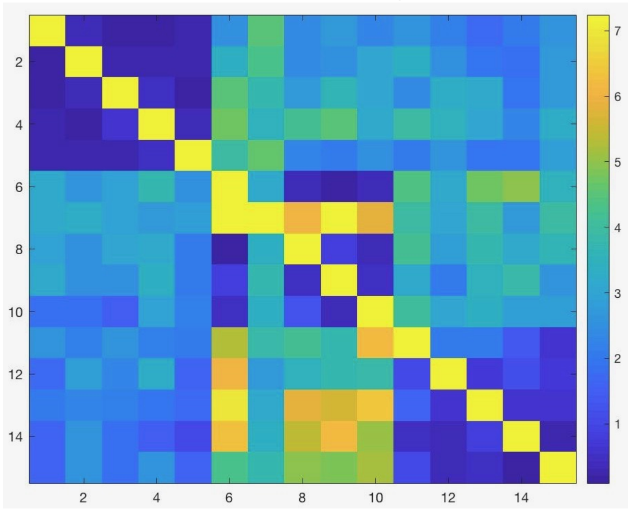

Figure 4: Sampling using the given function

The darker the the ij element, the smaller the matching cost is between shape i and j, and therefore the more chance there is that the two shapes are classified under the same class. The diagonal has elements of infinite energy. Knowing that we have 15 shapes classified in the order 5x heart, 5x fork, 5x watch, the three 5x5 darker areas around the diagonal make a lot of sense, since they correspond to shapes of the same class having a very low matching cost. In the case of the fork, we can see that the cost is in general higher, which

makes sense since it is a complicated shape and that a lot of wrong matches can be made.
We can also notice that in the case where the points are sampled randomly, the value of the cost is a little bit more spread out: the shapes of the same class have a slightly higher cost, and those of different classes have a lower cost. The difference between these two sampling methods will be further discussed in section 3.

2.2 Nearest-Neighbour Classifier

Classification is performed by using a k-nearest-neighbour classifier. For each test shape, I look at the matching cost obtained with the training shapes and only keep the k smallest ones. The most common class in these kept shapes is then returned and compared with the test shape.

3. Additional Questions

Is the shape context descriptor scale-invariant?

When computing the histogram in order to form the shape context descriptors, the radial distances are normalized using the average distance between all points. Scaling the shape by a factor m would result in both the distances and their mean being shaped by m. Therefore, the normalization step makes the descriptor scale-invariant.

What is the average accuracy of your classifier?

The average accuracy of my classifier varies with the value of k used for the k-nearest-neighbour algorithm. Further details follow in the next question.

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

The way the classification accuracy varies with k varies between executions due to the changes in the cost matrix due to points sampling (see Fig.5, where several executions are illustrated and then the average computed). With $k \leq 5$, the average accuracy is quite high (between 0.85 and 0.95), but as soon as k is bigger than five, it drops below 0.7. This is due to the fact that there are only 5 different shapes of each class, so finding 6 or more neighbours automatically increases the cost, as at least one of these nighbours will be from a different class.

Do you get better classification results with the provided sampling function rather than with the random sampling?

As seen previously in part 2.1, the results obtained with the provided sampling function are better than the ones obtained with random sampling. Indeed, the points sampled with `get_samples_1` give a better representation of the shape, therefore improving the TPS model and improving the classification average accuracy. With a basic random sampling function, it may happen that most samples belong to totally different parts of the compared shapes, resulting in a high number of wrong matches and therefore reducing the classification accuracy.
A solution would be to ignore matches with a cost above a predefined threshold, as explained in the paper 'Shape Matching and Object Recognition Using Shape Contexts' [2].

[2]Belongie, S., Malik, J. and Puzicha, J., 2002, Shape Matching and Object Recognition Using Shape Contexts, IEEE Trans. PAMI]

The figure below shows the classification accuracy for different values of k, and for different executions. The average of the values obtained with these different executions is then taking, resulting in the blue and red solid lines. It can be very well seen here how the random sampling deteriorates the accuracy of classification, in comparison to a more complete sampling.

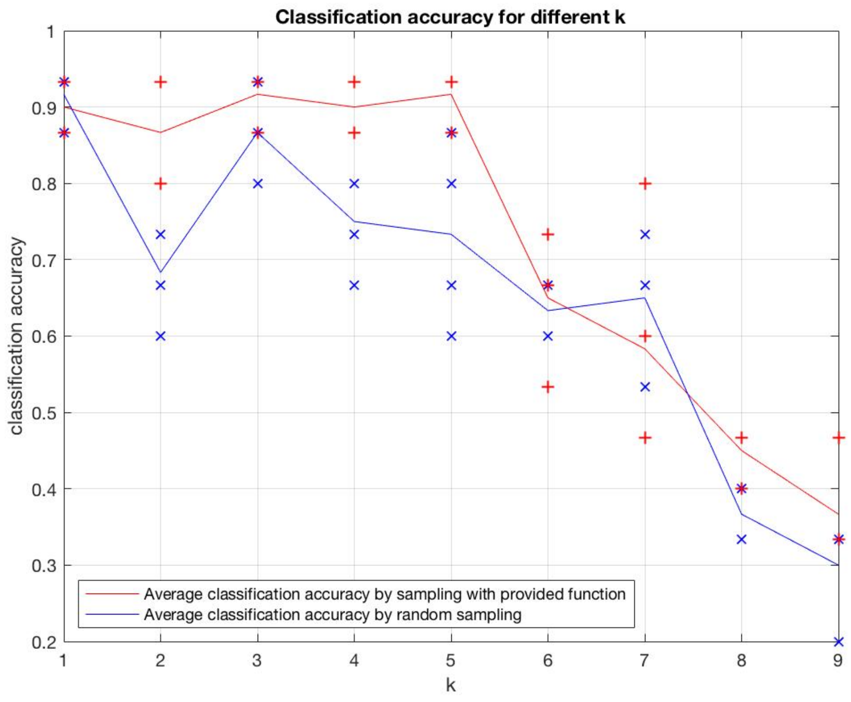

Figure 5