Computer Vision Exercise 6

Xingze Tian 17-941-527

December 8, 2017

1 Shape Matching

1.1 Shape Context Descriptors

1.1.1 Implementation

The implementation is in `sc_compute.m` with the following steps:

- **Compute distance and Angle between every two points**
  Input `X` is of dimension $n \times 2$. The results are stored in two $n \times n$ matrices. The distance matrix is then normalized by its mean distance.
- **Locate in the associated bin**
  For every point, I located every other point in the associated bin and updated the count of each bin accordingly. The radial bins used are *logr* and the localization method used is the matlab function `discretize()`.

1.1.2 Scale-invariance

The shape descriptor is scale invariant. If the image is scaled, the distance matrix would be scaled the same amount. As the distance matrix is normalized, the scale factor would be cancelled out, resulting the same shape descriptor.

1.2 Cost Matrix

The implementation is in `chi2_cost.m` that follows equation (1) in the specification, with $g(k)$ and $h(k)$ being the kth shape context descriptor in each set.

1.3 Hungarian Algorithm

1.4 Think Plate Splines

The implementation is in `tps_model.m` with the following steps:

- **Compute K and P**
  K and **P** are computed as in section 6.1 d) in the specification. The matlab method `isnan()` is used to get rid of the `nan` value caused by $log0$.
- **Solve system for each TPS model**
  I constructed matrix **A** based on equation (4) where **0** is a 3×3 zero matrix.

$$\mathbf{A} = \begin{pmatrix} \mathbf{K} + \lambda \mathbf{I} & \mathbf{P} \\ \mathbf{P}^T & \mathbf{0} \end{pmatrix}, \quad \mathbf{A} \begin{pmatrix} w \\ a \end{pmatrix} = \begin{pmatrix} v \\ 0 \end{pmatrix}$$

The weights `w_x` and `w_y` can thus be calculated using the following equation where v is the corresponding x values and y values in the target shape `Y`.

$$\begin{pmatrix} w \\ a \end{pmatrix} = \mathbf{A} \setminus \begin{pmatrix} v \\ 0 \end{pmatrix}$$

- **Compute bending energy**
  This is calculated using equation (5) as in the specification. The final output is the sum of the two bending energy calculated using `w_x` and `w_y`.

2 Shape Classification

2.1 Implementation

2.1.1 Shape Matching

The implementation is in `get_samples.m` and `compute_matching_costs.m`.

In `get_samples.m` I used the matlab function `randsample()` to randomly sample the indices of the points and output the sampled points according to the indices.

In `compute_matching_costs.m`, for every object I computed its matching cost with every other object using the function `shape_matching()`. For every object I sampled points as inputs to `shape_matching()`.

2.1.2 Nearest-Neighbour Classifier

The implementation is in `nn_classify.m`. I first computed the k smallest costs in `matchingCostVector`. Then I retrieved the matched classes using accordingly, `testClass` is thus the mode of the matched classes.

2.2 Evaluation

My classification results are as the following:

	k=1	k=3	k=5	k=10
get_samples	12/15	11/15	10/15	3/15
	13/15	12/15	10/15	2/15
	13/15	11/15	9/15	3/15
Average	12.7/15	11.3/15	9.7/15	2.7/15
get_samples_1	15/15	14/15	11/15	3/15
	14/15	12/15	12/15	3/15
	15/15	15/15	12/15	3/15
Average	14.7/15	13.7/15	11.7/15	3/15

Figure 1: Classification results

From the experiments we can see that:

- When increasing the size of k, the classification accuracy decreases. When k is bigger, more false points are accepted and thus decreases the accuracy.
- The provided `get_samples_1` produces better classification results. This is because the provided method ensures that the sample points have a certain minimum distance between each other to make the sampling along the contours somewhat uniform[1].

References

[1] S. Belongie, J. Malik, and J. Puzicha. Shape matching and object recognition using shape contexts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(4):509–522, Apr 2002.