Dynamic Shape Learning and Forgetting

Nikolaos Tsapanos[1,2], Anastasios Tefas[1] and Ioannis Pitas[1,2]
E-mail: {niktsap, tefas, pitas}@aiia.csd.auth.gr
⋆

[1] Department of Informatics, Aristotle University of Thessaloniki, Box 451, 54124, Greece
[2] Informatics and Telematics Institute, CERTH

Abstract. In this paper, we present a system capable of dynamically learning shapes in a way that also allows for the dynamic deletion of shapes already learned. It uses a self-balancing Binary Search Tree (BST) data structure in which we can insert shapes that we can later retrieve and also delete inserted shapes. The information concerning the inserted shapes is distributed on the tree's nodes in such a way that it is retained even after the structure of the tree changes due to insertions, deletions and rebalances these two operations can cause. Experiments show that the structure is robust enough to provide similar retrieval rates after many insertions and deletions.

1 Introduction

Object recognition by shape matching traditionally involves comparing an input test shape with various known shapes by measuring their similarity (usually by applying a Hausdorff based metric [3],[5]). The final matching returns the known shape that has yielded the maximum similarity with the input test shape. As the shape database becomes larger, however, exhaustive search becomes impractical. In order to overcome this problem, there have been a few tree structure based approaches proposed, such as the ones in [2] and [6]. An interesting property of the tree structure in [6] is that it provides means to not only learn new shapes, but to also forget previously learned shapes. In this paper, we perform various tests using this structure to determine its actual ability to learn the shape database, use the shapes it stores to classify unknown shapes and forget shapes without significant losses in classification performance for the remaining shapes. The paper is structured as follows: Section 2 briefly describes the structure, section 3 details the results of our experiments and section 4 concludes the paper.

2 Shape Trees

In this section we briefly describe the structure and the operations of the binary search tree originally presented in [6]. For our purposes, we view a shape as a set of points

⋆ The research leading to these results has received funding from the European Communitys Seventh Framework Programme (FP7/2007-2013) under Grant agreement No. 211471 (i3DPost).

with 2-dimensional integer coordinates. We do so because this is the simplest way to represent a shape, though there are several other, more complicated options [7].

The binary search tree consists of two types of nodes: leaf nodes and internal nodes. The shapes are stored in the leaf nodes. The internal nodes contain a template for each subtree, a matrix with the sum of the learned shapes in each subtree and they are used to traverse the tree. In order to search for a test shape in a shape tree, we must find a path of internal nodes from the root to the leaf node that corresponds to the matching shape. This can be achieved by using the internal nodes' parameters to measure the similarity of the test shape with two templates, one for each subtree. This similarity is based on the Hausdorff distance and is given by:

$$P(\mathcal{X}, \mathcal{Y}) = \frac{1}{|\mathcal{X}|} \sum_{\mathbf{x} \in \mathcal{X}} e^{-\alpha d(\mathbf{x}, \mathcal{Y})} \tag{1}$$

Where $\mathcal{X}$ is the set of the test shape points, $\mathcal{Y}$ is the set of the internal node template points, $d(\mathbf{x}, \mathcal{Y})$ is the distance from each point $\mathbf{x} \in \mathcal{X}$ to its closest point $\mathbf{y} \in \mathcal{Y}$ and α is a parameter determining the strictness of the similarity. Note that this similarity measure is directed and in the general case $P(\mathcal{X}, \mathcal{Y}) \neq P(\mathcal{Y}, \mathcal{X})$. The search is directed to the root of the subtree whose template yielded the highest similarity P. For practical purposes, a distance transform matrix [1] is used to calculate the measure instead of a set of points.

We will now describe how the shape tree finds the closest match of a test shape consisting of a set of points $\mathcal{X}$. Starting from the root of the tree we follow the path of nodes as dictated by comparing the similarities of $\mathcal{X}$ with each of the internal node's templates until we reach a leaf. That leaf node is reported as a possible result and the search backtrack and reverses the decision on the least confident node until another leaf node is reached, which is then reported as another possible result. This can be repeated $t - 1$ times so that a list of t possible shape matches is formed. The final result is found by exhaustively searching inside this list.

The insertion operation works almost identically with regular binary search trees. A new leaf node is constructed with the input shape. The input shape is then looked up in the tree. As the search progresses, the parameters of each internal node visited are updated by adding the input shape's points into the sum matrix of the appropriate subtree. The leaf node where the search ends is replaced by a new internal node, while itself and the input shape leaf node become the new internal node's children. Finally, the reverse path to the root is followed in order to rebalance and retrain nodes in case the tree has become unbalanced after the insertion.

The deletion operation is more limited in shape trees, as only the deletion of leaf nodes is supported. Starting from the deleted node, the path to the root of the tree is followed, in order to properly update the internal nodes' parameters by subtracting the deleted shape's points from the appropriate sum matrix, and to rebalance and retrain any nodes as necessary. Since the deletion of a leaf node can leave an internal node childless and this is not allowed in shape trees, any node left childless is marked for deletion. The process is repeated until there are no more nodes marked for deletion.

Rebalancing the tree can be achieved by using the standard LL, LR, RL, RR rotations with one exception: if the top node during an LL or RR rotation has only one child.

Performing an LL or RR rotation in this case would make an internal node childless. In this case, the top node is simply removed and replaced with its child. When a node is affected by the rotation, it has to be retrained by extracting the new templates from the sum matrix of each subtree.

3 Experiments

3.1 The database

All the experiments were conducted using the MNIST handwritten digits database [4]. This database contains 70046 28×28 images of the numbers 0 through 9 as written by 250 writers. They are separated into a subset of 60027 training images and a subset of 10019 test images.

3.2 The system

We implemented a system that uses shape trees to classify handwritten digit images. We labeled every image with the digit it depicts. The shape of each image was extracted by thresholding the brightness of the pixels to obtain a binary matrix in which the 1s formed the shape of the digit. Every shape of the training set was inserted into the tree in random order. In order to classify an input shape $\mathcal{X}$, we used the tree to find a list of the t closest matches $\mathcal{Y}_1 \dots \mathcal{Y}_t$, as previously explained. The final decision was the label of the shape $\mathcal{Y}_f$ such that $f = \arg\max_i (\min (P(\mathcal{X}, \mathcal{Y}_i), P(\mathcal{Y}_i, \mathcal{X})))$. The classification was considered to be correct if the label of $\mathcal{X}$ matched the label of $\mathcal{Y}_f$.

All the tests run on an AMD Athlon X2 6400 processor (each core clocked at 3.2GHz). Every averaged number is presented as *mean* (*standard deviation*).

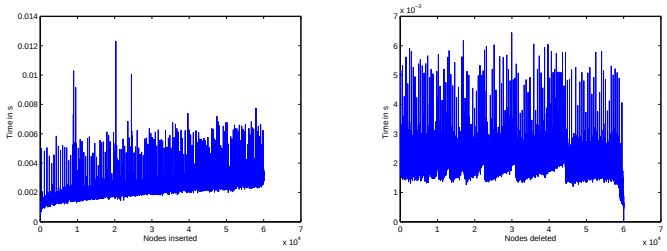
3.3 Insertion and Deletion

In this test we started with an empty tree and proceeded to insert all the 60027 shapes of the training set into it. We measured the time each insertion took. The graph of these values can be seen in Figure 1(a). By observing the graph, we can see that it reasonably follows a logarithmic curve, as theoretically expected. The various spikes are attributed to insertions that cause more rebalances than usual. Average insertion time was $0.0026(0.0006)$ ms, while total tree construction time was 154 seconds.

Starting from the tree constructed in the previous test, we deleted every shape that the tree contained and measure the time each deletion took. The graph of these values can be seen in Figure 1(b). Again, the spikes in the graph are attributed to some deletions causing a greater number of rebalances. Average deletion time was $0.00023(0.00029)$ ms, while the total time it took to empty the tree was 14 seconds.

3.4 Generalization Capabilities

Generalization capabilities were tested by first inserting the training set into the tree then using that tree to classify the test set. The number of tries was set at 64, as this number



(a) The times (in s) that a node insertion takes vs. number of nodes already in tree.

(b) The times (in s) that a node insertion takes vs. number of nodes already deleted from the tree.

Fig. 1. Insertion and deletion times.

seemed to yield improved results without slowing down the classification procedure considerably.

We used the 60027 shapes for training and tested the resulting trees in the 10019 digits that were designated as the test set. The shapes are inserted into the tree in random order. However, it is obvious that different insertion orders will produce different shape trees. Thus, in order to illustrate the effect of the insertion ordering has on the classification performance, we construct ten different trees that correspond to ten different random orderings of the inserted shapes.

The average classification rates and search times for this batch of trees are reported in Table 1. Total average classification rate was 0.9286 . Results also suggest that shape trees are consistent in their performance, as there are no wild variations in classification rates.

Digit	0	1	2	3	4	5	6	7	8	9
Rate	0.99	0.99	0.94	0.93	0.92	0.89	0.97	0.92	0.87	0.88

Table 1. Average classification rates for the trees trained with 60027 shapes and tested on 10019 shapes.

3.5 Robustness

We tested the proposed data structure's robustness to multiple insertions and deletions. We used the original training set to train the tree and the original test set to measure classification rates. Since the classes that are most likely to be confused with each other are those that correspond to the digits 3, 5 and 8, we begin by inserting these shapes into the initial tree. We denote the fact that the tree currently contains these digits as $\{3, 5, 8\}$. Thus, we consider the worst case scenario where the most difficult classes are

Digits in tree	0	1	2	3	4	5	6	7	8	9
$\{3, 5, 8\}$	-	-	-	0.92	-	0.93	-	-	0.96	-
$\{3, 5, 8\} \oplus \{0, 1\}$	0.99	0.99	-	0.91	-	0.92	-	-	0.94	-
$\{0, 1, 3, 5, 8\} \oplus \{7, 9\}$	0.99	0.99	-	0.90	-	0.91	-	0.95	0.92	0.93
$\{0, 1, 3, 5, 7, 8, 9\} \ominus \{0, 9\}$	-	0.99	-	0.91	-	0.93	-	0.98	0.95	-
$\{1, 3, 5, 7, 8\} \oplus \{4, 6\}$	-	0.99	-	0.90	0.98	0.91	0.98	0.96	0.91	-
$\{1, 3, 4, 5, 6, 7, 8\} \ominus \{1, 4, 6, 7\}$	-	-	-	0.92	-	0.92	-	-	0.96	-
$\{3, 5, 8\} \oplus \{0, 1, 2, 4, 6, 7, 9\}$	0.98	0.96	0.93	0.87	0.93	0.88	0.97	0.94	0.89	0.89

Table 2. Classification rates at the various stages of the robustness experiment.

always present inside the shape tree. We denote the insertion of additional digits with the symbol $\oplus$ and the deletion of digits with the symbol $\ominus$.

After starting with the initial tree, we proceed to insert and delete digits at consecutive stages. At each new stage (after the insertions or deletions are finished), we measure the classification rates for all the digits that are into the tree at the current stage. We also monitor the classification rates for the digits 3, 5 and 8 the are present at every stage. The graph that contains the change of classification rates of the three most difficult classes over the stages of this experiment can be seen in Figure 2, while the classification rates of all the digits involved in all the stages are presented in table 2 with a '-' denoting that the tree is not trained for that specific digit and that digit is therefore not tested.

By observing Figure 2 we can see that the classifier has an easier time classifying the three digits when there are fewer overall shapes stored in the tree. The classification rate decreases when more shapes are inserted and increases as shapes are removed. Note that when the only shapes inside the tree are those that correspond to the digits 3, 5 and 8 in stage 6 of the experiment their classification rate is the same as it was in the initial tree that also contained these three digits only. Furthermore, the final classification rates match the ones in Table 1, obtained from the generalization test. This observation highlights the online training capabilities of the proposed shape tree.

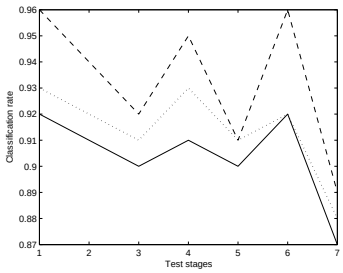


Fig. 2. The classification rate graphs for the digits 3 (solid line), 5 (dotted line) and 8 (dashed line).

3.6 Application to human detection

Since BSTs emphasize matching speed, shape trees can also be used in a fast human detection system. By training a tree with human silhouettes, we can scan the edge map of an image to quickly find potential matches. The resulting matches can either be thresholded, or passed through another, stronger classifier in order to be accepted as detections. Figure 3 shows a few sample detections using shape trees. These detections were carried out on multiview image data filmed at the University of Surrey under the i3dPost project. The tree was trained using silhouettes generated by the Poser software.

Fig. 3. Sample human detections using shape trees.

4 Conclusion

In this paper, the performance of a shape learning structure has been tested. Experiments performed on the MNIST handwritten digit database indicate that the structure is very efficient in terms of speed, performance and scalability. It can learn new shapes with reasonable loss in classification performance and even forget previously learned while retaining its classification abilities on the data that are still stored inside it.

References

1. Gunilla Borgefors. Distance transformations in digital images. In *Computer Vision, Graphics, and Image Processing, Volume 34 , Issue 3, pp. 344-371*, June 1986.
2. Dariu M. Gavrila. A bayesian, exemplar-based approach to hierarchical shape matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(8):1408–1421, August 2007.
3. G. Klanderman D. Huttenlocher and W.J. Rucklidge. Comparing images using the Hausdorff distance. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 15(9):850–863, September 1993.
4. Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(7):2278–2324, January 1998.
5. W.J. Rucklidge. Locating objects using the Hausdorff distance. *Proceedings of Fifth International Conference on Computer Vision*, 146(7):457–464, January 1995.
6. N. Tsapanos, Anastasios Tefas, and Ioannis Pitas. An online self-balancing binary search tree for hierarchical shape matching. In *VISAPP (1)*, pages 591–597, 2008.
7. Dengsheng Zhang and Guojun Lu. Review of shape representation and description techniques. *Pattern Recognition*, 37(1):1–19, 2004.