Shape Matching using a Binary Search Tree Structure of Weak Classifiers

Nikolaos Tsapanos[a,b], Anastasios Tefas[a], Nikolaos Nikolaidis[a,b], Ioannis Pitas[a,b]

[a] *Department of Informatics, Aristotle University of Thessaloniki, Box 451, Thessaloniki 54124, Greece*
[b] *Informatics and Telematics Institute, CERTH, 6th km Charilaou-Thermi road, Thermi 57001, Greece*

Abstract

In this paper, a novel algorithm for shape matching based on the Hausdorff distance and a binary search tree data structure is proposed. The shapes are stored in a binary search tree that can be traversed according to a Hausdorff-like similarity measure that allows us to make routing decisions at any given internal node. Each node functions as a classifier that can be trained using supervised learning. These node classifiers are very similar to perceptrons, and can be trained by formulating a probabilistic criterion for the expected performance of the classifier, then maximizing that criterion. Methods for node insertion and deletion are also available, so that a tree can be dynamically updated. While offline training is time consuming, all online training and both online and offline testing operations can be performed in $O(\log n)$ time. Experimental results on pedestrian detection indicate the efficiency of the proposed method in shape matching.

1. Introduction

The ability to locate and identify an object in an image or a video segment is a core task in visual information understanding. Out of the different object features that can be used for detection, object shape provides robustness to texture and illumination. Given an input image and a collection of shapes in a database, the objective it to detect the presence and estimate the location of any shape stored in the database. In order to use shape matching for object recognition, one must consider the following factors:

- shape representation
- shape matching
- shape database organization.

In this paper, our main contribution concerns the last two factors, as we propose a novel way to efficiently manage large shape databases and train shape classifiers that can be used in binary search trees for shapes.

1.1. Shape representation

A 2D shape can be represented in various forms. For an overview, the reader may refer to [35]. One possible representation is contour based, in which a shape is described as closed, continuous, parametric curve [12]. The statistical shape analysis regarding this representation is related mostly to clustering and hierarchical organization [27]. The dissimilarity between two shapes is proportional to the amount of effort required by the geodesic of one shape in order to morph into the geodesic of the other one.

Another way to represent a shape is by describing the shape using elements from a set of features. The work presented in [11] syntactically describes a shape as an ordered list of line segments that are characterized by their length and orientation. Matching and clustering are achieved through techniques inspired by text string matching algorithms. In similar fashion, [28] uses Shock Trees, a variation of the Shock Graph [26], to represent shapes and clusters shapes by merging individual shapes' Shock Trees into the clusters' Shock Trees in a way that minimizes the minimum description length of the entire dataset.

The simplest and most abstract way to represent a 2D contour is as a list of points. There is no requirement for closure, nor any information regarding the interconnectivity of said points. Such a list of points can be easily obtained, for example, by applying an edge detector on the image.

1.2. Shape matching

Object recognition through shape matching involves comparing an input shape with various template shapes and reporting the template with the highest similarity (or lowest dissimilarity) to the input shape. There are various similarity measures to be considered for this task [31]. A very useful similarity measure that can be easily computed is the Hausdorff Distance [22, 23] and its many variants. The Hausdorff Distance was originally introduced

to measure similarity between two sets of points, so it can also be used to measure similarity between shapes, since we can consider them to be sets of points. In fact, Hausdorff distance based metrics have been widely used for locating objects in images [22, 4]. The directed Hausdorff distance formula from point set $\mathcal{X}$ to point set $\mathcal{Y}$ is given by:

$$D_{DHD}(\mathcal{X}, \mathcal{Y}) = \max_{\mathbf{x} \in \mathcal{X}} (\min_{\mathbf{y} \in \mathcal{Y}} (d(\mathbf{x}, \mathbf{y}))), \tag{1}$$

where $d(\mathbf{x}, \mathbf{y})$ is the scalar L_2 distance between point $\mathbf{x}$ of point set $\mathcal{X}$ to point $\mathbf{y}$ of point set $\mathcal{Y}$. The overall Hausdorff distance is defined as:

$$D_{HD} = \max(D_{DHD}(\mathcal{X}, \mathcal{Y}), D_{DHD}(\mathcal{Y}, \mathcal{X})). \tag{2}$$

This measure, however, is very sensitive to outliers. In practice, several other variants of the Hausdorff distance are preferred over the original definition. Such variants include the k-th quantile Hausdorff distance, the modified Hausdorff distance and the weighted Hausdorff distance, used in [17] to match letters in scanned text. The interested reader may refer to [6] for the description of each variant and their performance comparison between the various Hausdorff distance variants. An image window that is densely populated by edge pixels will yield small Hausdorff distances (thus high similarity), when any template is matched against it. This leads to many false positive decisions. In order to overcome this problem, taking the orientation of the edges into account has been proposed in [20].

Probabilistic formulations of Hausdorff distance based metrics revolve mostly around using previous measurements to affect the number of possible future searches. When searching for a certain template in an image, the matching results of that template in a specific location in the image can be used to infer whether it is necessary to search in neighbouring spots or not through maximum likelihood estimation [19]. A probabilistic model is used in [10] to determine whether a matching result given for a certain template can warrant the further matching of similar templates. In this paper, we attempt probabilistic formulations on pixel level, studying how to choose the right template in the presence of displaced pixels in the input shape, which is a more challenging task.

1.3. Shape template database organization

As the template database becomes larger, exhaustive template search becomes impractical and the need of a better organization the template

database aiming to minimize the cost of the search operations arises. Many applications, such as visual surveillance or pedestrian detection systems have real-time constraints, thus the template search speed is vital to the viability of such a system.

While binary decision trees have been employed for shape related tasks, these attempts mostly considered shape classification based on the presence or absence of features [25], or feature extraction for classification purposes [2]. A tree of weak classifiers has also been proposed in [8] for a sports pictures classification system. However, there is a key difference between the labelled classification task the above works handle and shape template matching. In the case of template matching, there is no correct answer concerning the decision of the classifier at each node. The objective is to find which template bears the closest resemblance to a target input shape. It is possible that every template of the training set is a sample from the same class and that every template belongs to its own subclass. Classical binary decision tree construction works by selecting a feature that minimizes the class entropy in the resulting split subsets. However, measuring entropy in this case is pointless, as the class entropy of the entire training set is 0, while the subclass entropy is impossible to reduce by splitting the training set. It is for this reason that such training methods are unsuitable for template matching.

An early attempt to take advantage of tree structures to match shapes can be found in [1], where a license plate reading system through template matching is presented. It uses 37 templates and employs a Coarse-to-Fine Strategy to first find rectangular areas in an input image then uses a data structure similar to a B-tree to gradually refine the matching. This tree's nodes represent clusters of templates, with the root containing every template and each child node representing a sub-cluster of its parent's cluster, until the leaf nodes, which only contain one template.

The tree construction issue is also addressed in [10], by organizing the templates in a data structure again similar to a B-tree. The templates are initially grouped in subsets in the tree leaves, based on their mutual similarity. Along the path from a subset of templates to the root, a shape exemplar is selected to represent the subset of the previous level tree nodes to the new level. These representatives (called exemplars) are again grouped by similarity and the process repeats, thus creating a hierarchical structure. A probabilistic model is used to determine whether to search further down a tree path or not, based on the matching result of the current node exemplar,

in order to reduce the total number of measurements in a set of multiple templates. The weak point of this approach is that it provides no theoretical sublinear upper bound in search complexity, as it allows multiple paths to be followed at each tree node, and it also does not allow new templates to be added in an already constructed tree.

A self-balancing binary search tree (BST) is a well known data structure for the quick search, insertion and deletion of data. It was originally proposed for the efficient storage of real and integer numbers and it has been successfully adapted for storing other information as well [13]. The many useful properties of binary search trees are a clear motivation for attempting an adaptation of this data structure for the handling of shapes. The challenge lies in the fact that Hausdorff distance relationships lack several useful properties. For example, they are not transitive, they are not symmetric and they do not have to satisfy the triangular inequality [31]. Thus, there is no way to order shape similarity for a standard binary search.

Such an attempt at adapting the binary search tree data structure to be used for shapes has been proposed in [29], where the traditional BST operations for the search, insertion and deletion of nodes are presented, along with a weak classifier inside the internal nodes that facilitates these operations. By using the described data structure it is possible to search for (and even insert or delete) shape templates in logarithmic worst case computational complexity. Furthermore, whenever the number of templates doubles, a balanced BST needs only one more tree level to handle the extra data, which means that the scalability of the structure is also very good.

1.4. Paper Outline

In this paper, we propose a novel offline node training algorithm and a novel offline tree construction method for a binary search tree data structure for shapes. Our work is also closely related to [10]. Our novelty consists of the definition of a novel weak classifier, a method to train it and the organization of said weak classifiers into a tree structure for improved shape matching results.

In order to obtain the offline node training formulae, we perform probabilistic formulations at pixel-level, thus examining the low-level elements of shape matching. So far, various probabilistic formulations of Hausdorff distance have been discussed. However, they are only based on distance measurements, a high-level element of the Hausdorff based metrics. The

usefulness of this approach is more limited, due to the poor properties of Hausdorff distance relationships.

This paper is organized as follows: In section 2 the proposed similarity measure is briefly discussed and the functionality of the proposed trees' nodes is defined. The offline construction of a binary search tree for shape templates is described in section 3. The implementation and computational complexity details are given in section 4. The various experiments on both artificially generated data and real data concerning pedestrian detection the were conducted to evaluate the performance of the proposed structure are detailed in section 5. Finally, the paper is concluded in section 6.

2. Tree Structure

The proposed data structure is a binary shape tree for shapes and will henceforth be referred to as a shape tree. For our purposes, we view a shape as a set of points with 2-dimensional integer (pixel) coordinates of the form $[i, j]^T$. A shape can either be represented as a set of points $\mathcal{X}$, or as a binary $N_X \times M_X$ matrix $\mathbf{X}$ such that:

$$X(i,j) = \begin{cases} 1, \text{ if } \mathbf{x} = [i,j]^T \in \mathcal{X} \\ 0, \text{ otherwise.} \end{cases}$$

Any shape stored in a shape tree is referred to as a template T_i, while $\mathbf{T_i}$ will be used to denote the template in matrix form. All the templates that are to be stored in a shape tree form the training set $\mathcal{T} = \{T_1, T_2, \dots\}$ of that tree.

2.1. Activated Hausdorff Proximity

In this section we describe the similarity measure between two shapes $\mathcal{X}$ and $\mathcal{Y}$ that the weak classifiers are based on. Among the many variants of the Hausdorff Distance, the one found to yield the best results [6] is known as the Modified Hausdorff Distance (MHD) [4]:

$$D_{MHD}(\mathcal{X}, \mathcal{Y}) = \frac{1}{|\mathcal{X}|} \sum_{\mathbf{x} \in \mathcal{X}} d(\mathbf{x}, \mathcal{Y})$$

Where $|\mathcal{X}|$ is the cardinality of the point set $\mathcal{X}$ and $d(\mathbf{x}, \mathcal{Y}) = \min_{\mathbf{y} \in \mathcal{Y}} ||\mathbf{x} - \mathbf{y}||_2$. We will be employing a similarity measure that is based on the Modified Hausdorff Distance, but uses the exponential kernel activation function

($e^{-\alpha x}$) on the individual distances $d(\mathbf{x}, \mathcal{Y})$. This similarity measure will be referred to as *Activated Hausdorff Proximity* (AHP):

$$P_{AHP}(\mathcal{X}, \mathcal{Y}) \triangleq \frac{1}{|\mathcal{X}|} \sum_{\mathbf{x} \in \mathcal{X}} e^{-\alpha d(\mathbf{x}, \mathcal{Y})} \tag{3}$$

where $\mathcal{X}$ is the input shape, $\mathcal{Y}$ is the target shape to be matched to $\mathcal{X}$ and α is a constant that determines how fast the activation function decreases with the distance $d(\mathbf{x}, \mathcal{Y})$.

We use this activation function in order to normalize the similarity measure to $(0, 1]$, 1 meaning that for every point $\mathbf{x}$ in $\mathcal{X}$ there is a point $\mathbf{y}$ in $\mathcal{Y}$ placed in the same relative coordinates. As dissimilarity increases, the AHP measure tends to 0. The proposed measure also provides robustness against outliers to some degree. Indeed, after a certain distance (determined by the parameter α), any point is considered as an outlier and its effect on the similarity measure is almost constant, regardless of the actual distance (due to the tail of the exponential function). This means that the measure is only affected by the number of outliers and not their position.

In practice, in order to calculate AHP, we use a distance transform (DT) [3] on the target set $\mathcal{Y}$ that outputs a matrix $\mathbf{\Phi}$, such that each element $\Phi(i, j)$ is the integer approximation of the distance of point $\mathbf{x} = [i, j]^T$ from the closest point in $\mathcal{Y}$, i.e. $\Phi(i, j) \approx d(\mathbf{x}, \mathcal{Y})$. We will henceforth use matrices instead of point sets in the similarity measure, as it is more convenient to do so in mathematical expressions. If $\mathbf{X}$ is a binary matrix obtained by a point set $\mathcal{X}$ and $\mathbf{\Phi}$ is the distance transform matrix of point set $\mathcal{Y}$, we define the similarity measure as follows:

$$g(\mathbf{X}, \mathbf{\Phi}) = \frac{1}{\sum_{i=1}^{M_X} \sum_{j=1}^{N_X} X(i, j)} \sum_{i=1}^{M_X} \sum_{j=1}^{N_X} X(i, j) e^{-\alpha \Phi(i, j)} \tag{4}$$

2.2. Tree Nodes

The templates are organized in a binary search tree (shape tree), in order to have fast and efficient search capabilities. There are two different types of nodes in such a binary search tree: leaf nodes and internal nodes.

2.2.1. Leaf Nodes

A leaf node p_i contains a single template T_i from the training set $\mathcal{T}$ along with its Distance Transform. The template is stored as a set of 2-dimensional

Figure 1: A leaf node containing a template and its distance transform.

points with integer coordinates. The template distance transform is a matrix whose dimension is $N_T \times M_T$, where N_T, M_T are the template height and width, respectively (in pixels). Only leaf nodes can contain templates. A sample leaf node can be seen in Figure 1.

2.2.2. Internal Nodes

The internal nodes are used to determine the search path to the leaf nodes, where the actual data is stored. An internal node q contains two matrices denoted as $\mathbf{S}_L = \sum_{\mathbf{T}_i \in \mathcal{L}_q} \mathbf{T}_i$ and $\mathbf{S}_R = \sum_{\mathbf{T}_i \in \mathcal{R}_q} \mathbf{T}_i$. These matrices contain the sum of every real template matrix $\mathbf{T}_i$ that belongs to the left subtree set $\mathcal{L}_q$ and the sum of every real template matrix that belongs to the right subtree set $\mathcal{R}_q$ of q, respectively. They are used by the online training procedure and we preserve them even in an offline trained tree, in case that tree will later have to be updated online. Moreover, node q also contains two matrices, $\mathbf{W}_L$ and $\mathbf{W}_R$, which are used to determine the search path an input shape will follow. Matrix $\mathbf{W}_L$ attempts to measure the similarity of an input shape to the templates on the left subtree, while $\mathbf{W}_R$ does the same for the right subtree. A key difference with the online approach in [29] is that they are not the Distance Transforms of $\mathbf{S}_L$ and $\mathbf{S}_R$, but instead they are weight matrices resulting from training that direct incoming shapes more accurately than the online approach in tree traversal. The training process is detailed in section 3.1. These matrices contain non-negative real numbers during the training procedure which are later rounded to integer matrices. They determine whether the node directs a search to its left or right subtree. The dimension of all these matrices is also $M_T \times N_T$. Figure 2 shows a sample internal node. Matrices $\mathbf{W}_L$ and $\mathbf{W}_R$ were obtained through the offline training process and thus have lower than average values near points that coincide with templates on their respective subtree, thus resulting in higher similarity measurements when presented with a shape that resembles

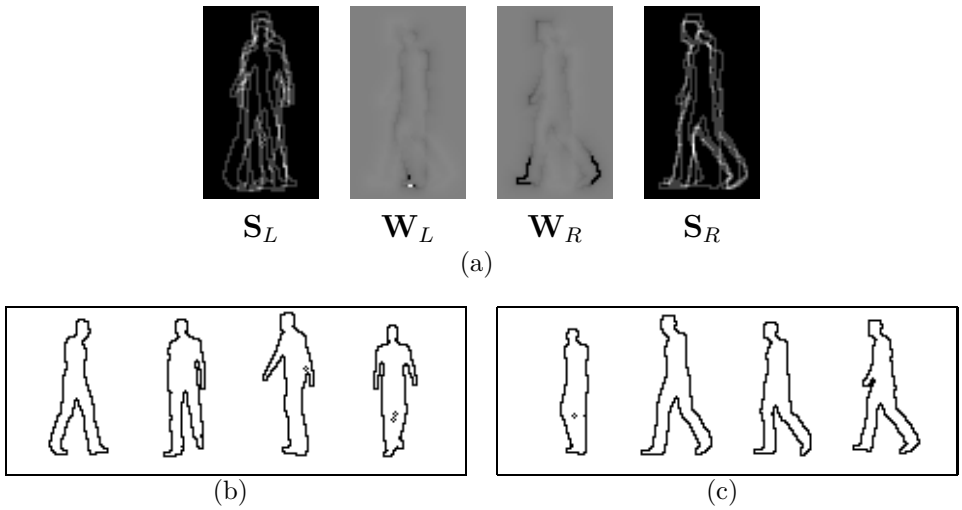


Figure 2: An internal node. (a) The matrices of the internal node. (b) The templates belonging the node's left subtree. (c) The templates belonging to the node's right subtree.

templates on their subtree, and higher than average values on points that coincide with templates in their opposite tree.

In order to search for an input shape $\mathcal{X}$ in a shape tree, we must follow a path of internal nodes starting from the tree root and ending in the leaf node that corresponds to the matching template. Each internal node on that path must decide on which subtree it must direct the search. The decision on which path to follow at a shape tree internal node for an input shape $\mathcal{X}$ corresponding to the binary matrix $\mathbf{X}$ is made according to the sign of the quantity

$$c = g(\mathbf{X}, \mathbf{W}_L) - g(\mathbf{X}, \mathbf{W}_R) \tag{5}$$

This type of weak classifier can be seen as equivalent to a linear perceptron. In order to replicate the internal node's classification of an input shape $\mathbf{X}$ with a perceptron, that perceptron would have to receive the elements of $\mathbf{X}$ as input, with $e^{-\alpha W_L(i,j)} - e^{-\alpha W_R(i,j)}$ being the synapse weight to the input $X(i,j)$. An illustration of this equivalence can be seen in Figure 3. The separating hyperplane of the perceptron is exactly the same as the internal node's separating hyperplane, though their difference is that the perceptron's output is not normalized by the sum of units in the input.

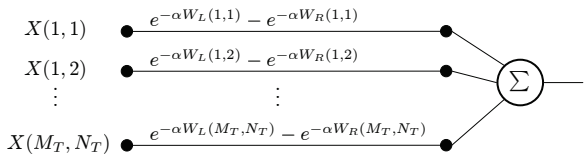


Figure 3: The equivalent perceptron of an internal node classifier.

2.3. Search

When we search for an input shape $\mathcal{X}$ inside a shape tree, we want to find the template that is most similar to the input shape. To do so, we start from the tree root and follow the path of nodes, as dictated by comparing the similarities of $\mathbf{X}$ with each node's $\mathbf{W}_L$ and $\mathbf{W}_R$ using (4), until we reach a leaf node. We then report the template of that leaf node as the most similar result.

As it stands, however, we cannot guarantee that the first search result (leaf node) is the best match as it takes only one wrong decision to miss the correct node. To overcome this problem, we define a measure called the confidence $|c|$ of each node in the path, where c is given by (5). We can use this confidence measure to backtrack through the path, reverse the traversal decision at the node with the lowest confidence and proceed to search the subtree we skipped in the previous search. Once a decision at a certain node has been reversed, we set that node's confidence to 1, so that its decision will not be switched back again, until the search is over. This way, if we allow t tries, we come up with t template candidates. We determine the best match by exhaustive search between these t candidates.

Node confidence is a measure of how easily separable the templates on each subtree are. The values of node confidence along the path from the root to a leaf are expected to be lower a) towards the root of the path, because there will be a lot of templates to separate in each subtree and, thus, more difficult to make a correct decision and b) towards the end of the path, because templates, whose lowest common ancestor is near the leaves of the tree will be similar and, thus, again, the corresponding node will be more difficult to make a correct decision. Node confidences are expected to be higher in nodes in the middle of the path, as the templates on each subtree are similar to each other, while being dissimilar to the templates in the other subtree. In practice, we artificially prohibit decision reversals at the higher levels of the tree during the first few tries to ensure that searches can spread

over a large tree section.

3. Offline Node Classifier Training and Tree Construction Algorithm

In this section, we describe the offline construction of a shape tree and the training of the classifiers (whose parameters are $\mathbf{W}_L$, $\mathbf{W}_R$, $\mathbf{S}_L$ and $\mathbf{S}_R$) in its internal nodes. We begin by deriving a training method that maximizes the a posteriori probability that every template will be correctly classified at a given node and a tree construction algorithm that determines how the training set of each node is partitioned.

3.1. Offline Node Classifier Training

In order to derive a criterion for the training of the tree node's classifier, we consider an internal tree node q that must separate the templates T_i ($\mathbf{T}_i$ in their binary matrix form) of its training set $\mathcal{T}_q \subseteq \mathcal{T}$ (a subset of the entire tree's training set $\mathcal{T}$) into a left subtree set $\mathcal{L}_q$ and a right subtree set $\mathcal{R}_q$ that have already been determined. In this section we describe the training algorithm that accomplishes this task. The choice of the sets $\mathcal{L}_q$ and $\mathcal{R}_q$ for every node will be dealt with in section 3.2, when we describe the construction procedure of the shape tree.

Let us assume that $\mathbf{X}$ is a random vector, which represents all possible inputs to a node q. When node q needs to classify an instance of $\mathbf{X}$, it computes its c_q measure using (5) and, depending on the sign of this measure, directs the input to either its left subtree or its right subtree. Since $\mathbf{X}$ is a random vector, whose PDF is not known, we must make a hypothesis at this node on which subtree is the correct one based on the observable evidence c_q. By invoking the Bayes rule, it is possible to link the a posteriori probability of the hypothesis being correct with the observable evidence and form a training criterion that includes the templates of the training set.

We begin to formulate the a posteriori criterion, by supposing that template T_i is node q's only left child in order to simplify the formulations. We assume that $\mathbf{T}_i$ (T_i in matrix form) is an instance of the random vector $\mathbf{A}_i$, which represents all possible appearances that the template T_i might take. Consider all the instances of $\mathbf{X}$ that produce a positive confidence ($c_q > 0$), when classified by node q. In order for the classification decision to be correct, the instances of $\mathbf{X}$ must also be instances of $\mathbf{A}_i$ (we use $\mathbf{X} = \mathbf{A}_i$ to denote

this). Therefore, the a posteriori probability that the decision of node q is correct is $p(\mathbf{X} = \mathbf{A}_i | c_q > 0)$. Using the Bayes rule we get that:

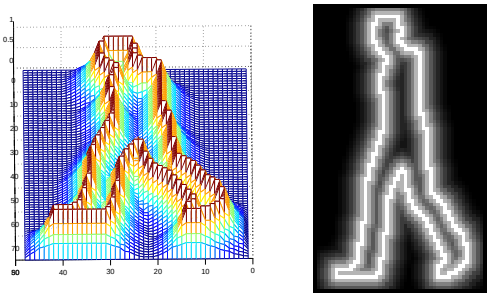
$$p(\mathbf{X} = \mathbf{A}_i | c_q > 0) = \frac{p(c_q > 0 | \mathbf{X} = \mathbf{A}_i) p(\mathbf{X} = \mathbf{A}_i)}{p(c_q > 0)},$$

where $p(\mathbf{X} = \mathbf{A}_i)$ is the prior of $\mathbf{A}_i$ occurring, $p(c_q > 0)$ is the bias of the classifier to direct shapes to it's left subtree and $p(c_q > 0 | \mathbf{X} = \mathbf{A}_i)$ is the probability that node q will correctly classify an instance known to belong to $\mathbf{A}_i$. If the priors of each $\mathbf{A}_i$ are unknown, we can assume that $p(\mathbf{X} = \mathbf{A}_i) = p(\mathbf{X} = \mathbf{A}_j) \, \forall i, j$. $P(c_q > 0)$ is trivial (yet costly) to compute and can be done by generating and classifying every possible binary shape of the dimensions of the classifier.

We now focus on the approximation of $p(c_q > 0 | \mathbf{X} = \mathbf{A}_i)$, the individual probability that the classifier will direct an input shape to the correct subtree, when we know which the correct subtree is. It is through this probability that we can use the templates of the training set in order to train the classifier. We assume that the correct subtree is the "left" subtree, as the "right" case is an equivalent one.

In order to approximate $p(c_q > 0 | \mathbf{X} = \mathbf{A}_i)$, we begin by assigning weights to each element of $\mathbf{A}_i$. The resulting weight matrix is denoted by $\tilde{\mathbf{A}}_i$. Let $\mathbf{\Psi}$ be the distance transform of the template $\mathbf{T}_i$. The elements of the weight matrix $\tilde{\mathbf{A}}_i$ are obtained by passing the elements of $\mathbf{\Psi}$ through the same activation function $\tilde{A}_i(j,k) = e^{-\alpha \Psi_i(j,k)}$ used in the AHP measure (3). These weights represent the probability that a contour point is present at each position in an instance of the random vector $\mathbf{A}_i$. They take the highest values directly on template T_i points. Their value drops exponentially as the distance of the contour position from the nearest point of this template increases. The form of the activation function applied to a template distance transform and the resulting weight matrix are illustrated in Figure 4.

The idea, at this point, is to generate a large number of $\mathbf{A}_i$ instances using $\tilde{\mathbf{A}}_i$, calculate the c_q measure of the classifier for node q and then compute the average c_q. The greater this average is, the more likely it will be that an instance of $\mathbf{A}_i$ will be correctly classified and, thus, the greater $p(c_q > 0 | \mathbf{X} = \mathbf{A}_i)$ will be. In order to generate the instances, we would also have to scale the weights of $\tilde{\mathbf{A}}_i$ such that $e^{-\alpha \Psi_i(j,k)}$ becomes a proper Probability Density Function (integrates to a unit). However, this procedure is not necessary, because, due to the simplicity of our classifier, we can equivalently use $\tilde{\mathbf{A}}_i$



(a) (b)

Figure 4: (a) The activation function on $\mathbf{T}_i$'s distance transform. (b) The weight matrix $\tilde{\mathbf{A}}_i$.

directly. Therefore, we approximate $p(c_q > 0|\mathbf{X} = \mathbf{A}_i)$ using the average $\tilde{c}_{(q,i)}$ as defined by:

$$\tilde{c}_{(q,i)} = g(\tilde{\mathbf{A}}_i, \mathbf{W}_L) - g(\tilde{\mathbf{A}}_i, \mathbf{W}_R) = \tag{6}$$
$$\sum_{x=1}^{M_T}\sum_{y=1}^{N_T} \tilde{A}_i(x,y)e^{-\alpha W_L(x,y)} - \sum_{x=1}^{M_T}\sum_{y=1}^{N_T} \tilde{A}_i(x,y)e^{-\alpha W_R(x,y)}.$$

The approximation of $p(c_q > 0|\mathbf{X} = \mathbf{A}_i)$ is therefore given by:

$$p(c_q > 0|\mathbf{X} = \mathbf{A}_i) \approx \frac{1 + \tilde{c}_{(q,i)}}{2} \tag{7}$$

Since both similarities lie in the range of $(0, 1]$, their difference $(\tilde{c}_{(q,i)})$ lies in $(-1, 1)$ and, therefore, the probability we approximate will never be greater than 1 or lower than 0. Note that this approximation is not strictly mathematically derived, but it is based on the intuition that, if we maximize the average node confidence on all inputs $\tilde{c}_{(q,i)}$, then we also maximize the true probability that a shape will be directed to the correct subtree $p(c_q > 0|\mathbf{X} = \mathbf{A}_i)$. If $\tilde{c}_{(q,i)} = 0$, then the probability approximation becomes $1/2$, equivalent to the random choice.

The probability that a single template T_i is correctly classified by the tree is given by the product of the corresponding probabilities for each node on the path $\mathcal{Q}_i$ from the root to T_i's leaf node.

$$P_{T_i} = \prod_{q \in \mathcal{Q}_i} p(\mathbf{X} = \mathbf{A}_i | c_q ? 0)$$

We use $c_q?0$ to denote the proper relation of the node's confidence to zero, when its input is $\mathbf{X} = \mathbf{A}_i$. If the correct path for input shape $\mathbf{X} = \mathbf{A}_i$ is the left subtree, then c_q should be greater than zero ($c_q > 0$) and it should be less or equal to zero ($c_q \leq 0$), if the correct path for $\mathbf{X} = \mathbf{A}_i$ is the right subtree. The question mark is disambiguated in equation (8). The probability that all templates are correctly classified by the tree is therefore

$$P_{total} = \prod_{T_i \in \mathcal{T}} \prod_{q \in \mathcal{Q}_i} p(\mathbf{X} = \mathbf{A}_i | c_q?0)$$

We can rewrite the above as

$$P_{total} = \prod_{q \in \mathcal{Q}_i} \prod_{T_i \in \mathcal{T}} p(\mathbf{X} = \mathbf{A}_i | c_q?0)$$

and easily see that the total probability that the tree will correctly classify every template is the product of the probabilities that each individual node will correctly classify every template it is responsible for.

Now we isolate the probability that node q's classification decisions will be correct for every input instance of $\mathbf{X}$. Considering that the node is responsible for classifying several templates in both its subtrees, we define the overall probability that the node will make correct decisions as:

$$P_{node} = \prod_{\mathbf{A}_i \in \mathcal{L}_q} P(\mathbf{X} = \mathbf{A}_i | c_q > 0) \prod_{\mathbf{A}_j \in \mathcal{R}_q} P(\mathbf{X} = \mathbf{A}_j | c_q \leq 0). \tag{8}$$

By using the Bayes rule on (8) we get that:

$$P_{node} = \frac{\prod_{\mathbf{A}_i \in \mathcal{L}_q} p(c_q > 0 | \mathbf{X} = \mathbf{A}_i) \prod_{\mathbf{A}_j \in \mathcal{R}_q} p(c_q \leq 0 | \mathbf{X} = \mathbf{A}_j) \prod_{\mathbf{A}_i \in \mathcal{L}_q} p(\mathbf{X} = \mathbf{A}_i) \prod_{\mathbf{A}_j \in \mathcal{R}_q} p(\mathbf{X} = \mathbf{A}_j)}{p(c_q > 0)^{|\mathcal{L}_q|} p(c_q \leq 0)^{|\mathcal{R}_q|}} \tag{9}$$

Regardless whether the priors are known or not, $\prod_{\mathbf{A}_i \in \mathcal{L}_q} p(\mathbf{A}_i) \prod_{\mathbf{A}_j \in \mathcal{R}_q} p(\mathbf{A}_j)$ can be considered to be constant. The value of $p(c_q > 0)^{|\mathcal{L}_q|} p(c_q \leq 0)^{|\mathcal{R}_q|}$ essentially represents how the entire template space is split between the two subtrees. Since we also want the tree to be balanced (and therefore we want $|\mathcal{L}_q| \approx |\mathcal{R}_q|$) and since $p(c_q \leq 0) = 1 - p(c_q > 0)$, then the quantity in question can be approximately written as $(p(c_q > 0)(1 - p(c_q > 0)))^{|\mathcal{L}_q|}$ whose maximum is achieved when $p(c_q > 0) = 1 - p(c_q > 0) = \frac{1}{2}$. This occurs when the template space is split in half by the classifier. Therefore, it should not

be taken into consideration when training the classifier, because it depends on the contents of $\mathcal{L}_q$ and $\mathcal{R}_q$ (which will be discussed later) and it also leads to favouring imbalanced trees. The probability $P^{(q)}_{training}$ we will use to train the classifier is therefore:

$$P^{(q)}_{training} = \prod_{\mathbf{A}_i \in \mathcal{L}_q} p(c_q > 0 | \mathbf{X} = \mathbf{A}_i) \prod_{\mathbf{A}_j \in \mathcal{R}_q} p(c_q \leq 0 | \mathbf{X} = \mathbf{A}_j) \tag{10}$$

We aim to maximize this probability through every template in the training set. We use (7) to substitute $p(c_{(q,i)} > 0 | \mathbf{X} = \mathbf{A}_i)$ with $\frac{1+\tilde{c}_{(q,i)}}{2}$ and apply the chain rule to calculate the partial derivative of $P^{(q)}_{training}$ for with respect to $W_L(x, y)$ for a given $\mathbf{A}_i$ as:

$$\frac{\partial P^{(q)}_{training}}{\partial \tilde{c}_{(q,i)}} \frac{\partial \tilde{c}_{(q,i)}}{\partial W_L(x,y)} = -\frac{\alpha}{2} \tilde{A}_i(x,y) e^{-\alpha W_L(x,y)} \prod_{\mathbf{A}_j \in \mathcal{L}_q}^{j \neq i} p(c_q > 0 | \mathbf{X} = \mathbf{A}_j) \prod_{\mathbf{A}_k \in \mathcal{R}_q} p(c_q \leq 0 | \mathbf{X} = \mathbf{A}_k) \tag{11}$$

(the $\mathbf{W}_R$ case is an equivalent one). Assuming that $p(c_{(q,i)} > 0 | \mathbf{X} = \mathbf{A}_i) \neq 0$, we can multiply the previous expression with and divide it by $p(c_{(q,i)} > 0 | \mathbf{X} = \mathbf{A}_i)$ to get:

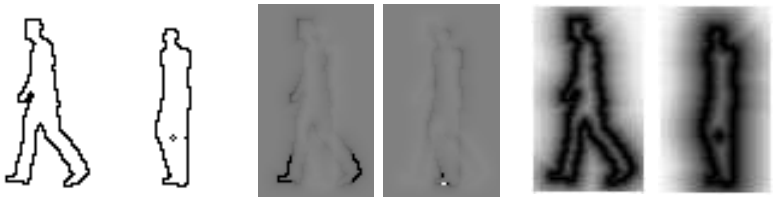
$$\frac{\partial P^{(q)}_{training}}{\partial \tilde{c}_{(q,i)}} \frac{\partial \tilde{c}_{(q,i)}}{\partial W_L(x,y)} = -\frac{\alpha}{2} \tilde{A}_i(x,y) e^{-\alpha W_L(x,y)} \frac{P^{(q)}_{training}}{p(c_{(q,i)} > 0 | \mathbf{X} = \mathbf{A}_i)}$$

Note that the term $P^{(q)}_{training}$ remains the same for every member of the gradient and can, therefore, be omitted from computation. According to (7), we can substitute $p(c_{(q,i)} > 0 | \mathbf{X} = \mathbf{A}_i)$ with $\frac{1+\tilde{c}_{(q,i)}}{2}$ and the direction of the partial derivative for the template T_i that belongs to the node's left subtree is, therefore, approximated by:

$$\frac{\partial P^{(q)}_{training}}{\partial \tilde{c}_{(q,i)}} \frac{\partial \tilde{c}_{(q,i)}}{\partial W_L(x,y)} = -\frac{\alpha}{1 + \tilde{c}_{(q,i)}} \tilde{A}_i(x,y) e^{-\alpha W_L(x,y)} \tag{12}$$

When the node has more than one template in its subtrees, the partial derivative is calculated by summing the partial derivatives from every template. Templates belonging to the right subtree are included in the sum for the derivative of W_L with the opposite sign and vice versa. The final formulae for the partial derivative with respect to the parameter matrices are:

$$\frac{\partial P^{(q)}_{training}}{\partial W_L(x,y)} = -\sum_{T_i \in \mathcal{L}_q} \frac{\alpha \tilde{A}_i(x,y) e^{-\alpha W_L(x,y)}}{1 + \tilde{c}_{(q,i)}} + \sum_{T_j \in \mathcal{R}_q} \frac{\alpha \tilde{A}_j(x,y) e^{-\alpha W_L(x,y)}}{1 + \tilde{c}_{(q,j)}}$$



(a) (b) (c)

Figure 5: (a) The two templates the node must separate. (b) The offline training result. (c) The online training result.

$$\frac{\partial P^{(q)}_{training}}{\partial W_R(x,y)} = \sum_{T_i \in \mathcal{L}_q} \frac{\alpha \tilde{A}_i(x,y) e^{-\alpha W_R(x,y)}}{1 + \tilde{c}_{(q,i)}} - \sum_{T_j \in \mathcal{R}_q} \frac{\alpha \tilde{A}_j(x,y) e^{-\alpha W_R(x,y)}}{1 + \tilde{c}_{(q,j)}}$$

Now that we have formulated the direction of the training criterion's partial derivative with respect to the node weight matrices $\mathbf{W}_L$ and $\mathbf{W}_R$, we can use gradient descent (or more sophisticated optimization methods) to maximize (10). We start by initializing the weight matrix elements at the middle value of their value's range (which is 128 for an unsigned character variable, so we set $W_L(i,j) = W_R(i,j) = 128$ in our experiments) and then we iteratively follow the gradient. Every time that we reach a new maximum, we store the $\mathbf{W}_L$ and $\mathbf{W}_R$ matrices that resulted in this new maximum. After the iteration limit is reached, we report the matrices of the greatest maximum encountered. The $\mathbf{S}_L$ and $\mathbf{S}_R$ matrices are trivial to compute. Results from the offline training can be seen in Figure 5.

3.2. Tree Construction

As discussed in the previous section, the probabilistic criterion for the training of a single node classifier is presented in (10). In order to construct a shape tree offline we need to maximize the product of that criterion for every internal node:

$$P_{tree} = \prod_q P^{(q)}_{training} \tag{13}$$

This can be accomplished in two ways: by applying optimization techniques at every node, so that it properly classifies its training set and, also, by structuring the tree in such a way that the classification is easier at every node. We have already covered the training process through which node

q separates $\mathcal{L}_q$ from $\mathcal{R}_q$ in the previous section. The remaining issue, i.e. choosing $\mathcal{L}_q$ and $\mathcal{R}_q$ so that they are easily separable, is not as straightforward.

Consider a full binary tree with O levels. This means that there are 2^O templates in the leaves. Therefore, every node q must be able to separate the templates in its left subtree $\mathcal{L}_q$ from the templates in its right subtree $\mathcal{R}_q$ and do so with a probability expressed in (10) that is maximized through training so as to reach the value of $p_q(\mathcal{L}_q, \mathcal{R}_q)$. This value depends on the separability of the training set of every node and is a function of $\mathcal{L}_q$ and $\mathcal{R}_q$. By changing $\mathcal{L}_q$ and $\mathcal{R}_q$ we also change the maximum probability $p_q(\mathcal{L}_q, \mathcal{R}_q)$ with which a node can be trained to properly classify its training set. In essence, we wish to find an ordered arrangement of the templates, so that the $\mathcal{L}_q$ and $\mathcal{R}_q$ sets of every node can lead to the global maximum of the criterion given in (13). The problem is particularly difficult when considering the global construction of the entire tree. In fact, this problem is NP-complete in its general form, as can be proven by reducing the *Hamiltonian Path* problem to this problem.

Since the optimal shape tree construction problem in its general form is an NP-complete one, we will use a top-down shape tree construction algorithm. The construction begins from the root of the tree, where the templates are separated in two mostly equal halves, then recursively proceeds to construct the left and right subtrees of each constructed node, by further dividing the remaining templates in mostly equal halves. This approach, unfortunately, does not guarantee the global maximization of our probabilistic criterion. Each node construction process is iterative and consists of two phases: the training phase and the reorganization phase. During the construction of each node q, we aim to separate its training set $\mathcal{T}_q$ in two subsets, preferably having equal cardinality. We denote the left and right subtree subsets $\mathcal{L}_q^{(i)}$, $\mathcal{R}_q^{(i)}$ respectively, where i is the number of the training iteration. $\mathcal{L}_q^{(0)}$ and $\mathcal{R}_q^{(0)}$ are initialized by arbitrarily splitting $\mathcal{T}_q$ in half.

In the training phase for node q, we use a gradient descent according to the formulae in section 3.1 for the maximization of the a posteriori probability criterion in order to train the classifier of node q. After the ith training iteration, some templates in $\mathcal{L}_q^{(i)}$ will be correctly classified to the left subtree. We name this subset $\check{\mathcal{L}}_q^{(i)}$. The rest of the templates in $\mathcal{L}_q^{(i)}$ will be incorrectly classified to the right subtree. We call the latter subset $\hat{\mathcal{L}}_q^{(i)}$ (the errors of $\mathcal{L}_q^{(i)}$). Likewise for $\mathcal{R}_q^{(i)}$, we will have the subsets $\check{\mathcal{R}}_q^{(i)}$ and $\hat{\mathcal{R}}_q^{(i)}$.

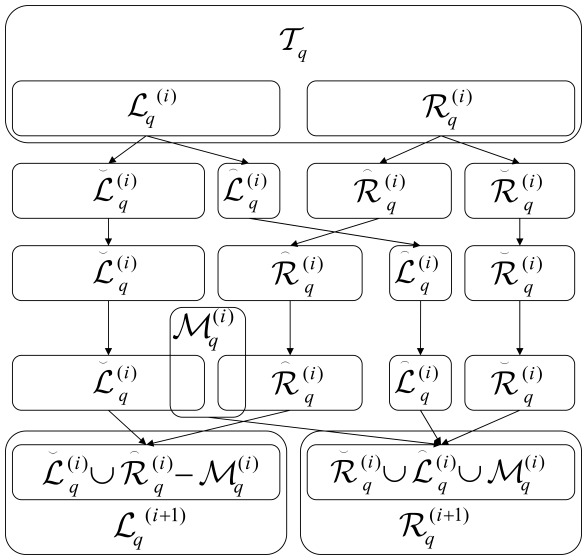


Figure 6: The shape tree reorganization phase.

In the reorganization phase, since our goal is to separate the original set in two subsets and we do not particularly care which subtree a certain template node is classified to. Therefore, we could set $\mathcal{L}_q^{(i+1)} = \check{\mathcal{L}}_q^{(i)} \cup \hat{\mathcal{R}}_q^{(i)}$ and $\mathcal{R}_q^{(i+1)} = \check{\mathcal{R}}_q^{(i)} \cup \hat{\mathcal{L}}_q^{(i)}$, so that the node now separates $\mathcal{L}_q^{(i+1)}$ from $\mathcal{R}_q^{(i+1)}$ perfectly. However, we want a balanced tree. Therefore, we want $|\mathcal{L}_q^{(i+1)}|$ to be as close to $|\mathcal{R}_q^{(i+1)}|$ as possible. Suppose that $\check{\mathcal{L}}_q^{(i)} \cup \hat{\mathcal{R}}_q^{(i)}$ is a larger set than $\check{\mathcal{R}}_q^{(i)} \cup \hat{\mathcal{L}}_q^{(i)}$ ($|\check{\mathcal{L}}_q^{(i)} \cup \hat{\mathcal{R}}_q^{(i)}| > |\check{\mathcal{R}}_q^{(i)} \cup \hat{\mathcal{L}}_q^{(i)}|$). We assign the $\frac{|\check{\mathcal{L}}_q^{(i)} \cup \hat{\mathcal{R}}_q^{(i)}| - |\check{\mathcal{R}}_q^{(i)} \cup \hat{\mathcal{L}}_q^{(i)}|}{2}$ template nodes in $\check{\mathcal{L}}_q^{(i)} \cup \hat{\mathcal{R}}_q^{(i)}$ that are classified to the left subtree with the minimum probability, as given in (6), to the set $\mathcal{M}_q^{(i)}$. We set $\mathcal{L}_q^{(i+1)} = \check{\mathcal{L}}_q^{(i)} \cup \hat{\mathcal{R}}_q^{(i)} - \mathcal{M}_q^{(i)}$ and $\mathcal{R}_q^{(i+1)} = \check{\mathcal{R}}_q^{(i)} \cup \hat{\mathcal{L}}_q^{(i)} \cup \mathcal{M}_q^{(i)}$ (the case is symmetrical, if the inequality is reversed) and move on to the next iteration. The reorganization process is illustrated in Figure 6.

An additional goal of the training is, therefore, to also minimize the cardinality of $\mathcal{M}_q^{(i)}$, essentially minimizing the difference between the error numbers on both sides, since we will latter swap them. The smaller $|\mathcal{M}_q^{(i)}|$ is, the more balanced the node will be after swapping each of its subsets' errors. If $|\mathcal{M}_q^{(i)}| \leq 1$, then we can achieve a perfect division of $\mathcal{T}_q$ as well as perfect training. In order to minimize $|\mathcal{M}_q^{(i)}|$, we modify the training phase

to give a bonus β to the contribution of every template in the gradient, according to the mistakes of each subset. Thus, the gradient (12) becomes $-\beta\frac{\alpha}{c_{(q,i)}}\tilde{A}_i(x,y)e^{-\alpha W_L(x,y)}$. Under normal circumstances, we assign $\beta = 1$. When one subset has more errors in it, the templates of that subset are assigned a higher value for β and have a bigger impact to the gradient than the templates of the other subset.

The construction starts from the root and proceeds by recursively repeating the process for the root's left and right children. We proceed to recursively train every node classifier to separate the templates directed to it by its parent in two roughly equal subsets, until every template finds itself in a leaf node.

4. Implementation and Complexity Issues

In this section we provide further details on the implementation of a real-time system that performs shape matching in images using the proposed data structure. We also provide estimates for the computational complexity of the basic tree operations.

4.1. Fast Search and Template Matching in Images

Here we describe the implementation of the proposed matching data structure into a general system for fast shape matching using a template database in greyscale images. A binary search tree is constructed using the input target templates. This can be either done offline, as described in section 3, or by incrementally inserting all the templates in an initially empty tree as described in [29]. Let the tree contain $N_T \times M_T$ pixels.

4.1.1. Initial Precomputations

Given a greyscale $N_F \times M_F$ image, we first perform edge detection using a standard Sobel mask on the image. We chose the Sobel edge detector due to its speed and the fact that the output is more simplified than other edge detectors (for example the Canny edge detector), which is better for our application, as edges that lie inside the detection objects have an adverse effect on the matching process. The edge pixel (x, y) coordinates are then inserted into a special data structure, which is a double binary search tree for integer numbers that first stores the pixels using the x coordinate as its key. Every node of the first coordinate tree contains another binary search tree that uses the y coordinate as its key. After this double binary search

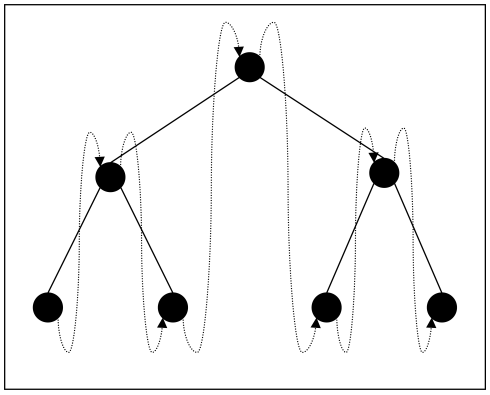


Figure 7: A second coordinate tree.

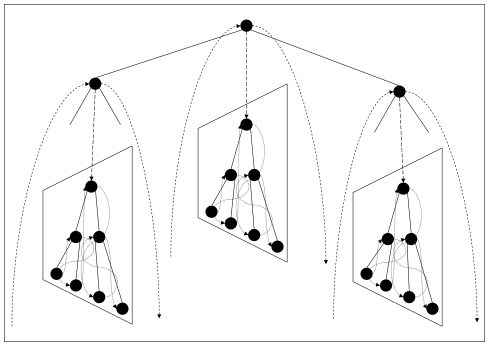


Figure 8: A first coordinate tree, containing multiple second coordinate trees.

tree has been constructed, the nodes of all the y coordinate trees and the x coordinate tree are threaded in their respective in-order traversal. The form of the y coordinate trees can be seen in Figure 7, while the overall data structure of the x coordinate tree is shown in Figure 8.

Then the distance transform matrix $\mathbf{\Phi}$ of the image edges is computed. Along with the distances, the edge gradients are stored in the gradient maps $\mathbf{G}_x$ and $\mathbf{G}_y$.

4.1.2. Planar search

The planar space of the search is handled as follows: A $M_T \times N_T$ window $\mathbf{Q}$ in the image, whose upper left corner has coordinates $(x_\mathbf{Q}, y_\mathbf{Q})$ is cropped and tested. If the test result yields a similarity that exceeds a threshold,

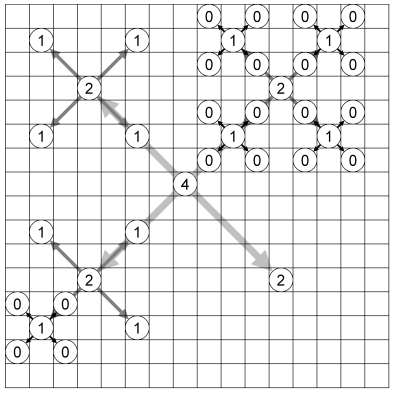


Figure 9: An example of the planar search strategy on a 16×16 pixel block. The circles represent the upper left corner of image windows that were tested. The number inside each circle denotes the displacement of the new searches it recursively spawns in both dimensions. The recursion begins at the circle with the number 4 inside it. Depending on the matching result, each circle may or may not expand the search area.

we recursively test the four windows whose upper left corners lie in an 'X' pattern centered on the original window's corner with a step of 4 pixels in both coordinate directions $(x \pm 4, y \pm 4)$. The recursion continues with the step halved and the threshold increased at every recursion level until the step reaches 0, in which case the recursion stops after similarity computation. This covers a 16×16 block of the image, since each dimension of the area covered is $1 + 8 + 4 + 2 = 15$ and we ignore the lower right corner of the block to increase speed. An illustration of this search strategy is provided in Figure 9. After the recursion is finished, we proceed to the next 16×16 block.

This guarantees that the search can reach at least a neighbouring pixel for every pixel in the image, even though we completely ignore every 16th row and column. We can further prune the search space by stopping a recursive search whose matching results are sufficiently worse than the maximum result in the recursion so far. In our implementation this means that if the current matching result is less than 80% of the best matching result so far in the recursion, then this recursion branch is terminated early.

4.1.3. Shape Tree search

In order to test a $N_T \times M_T$ window $\mathbf{Q}$ whose upper left corner is the point $(x_\mathbf{Q}, y_\mathbf{Q})$, we first find the edges contained in that window of the image. We use the double binary search tree data structure described above to do that

efficiently using the following algorithm:

- Start with an empty table of edge pixel coordinates.
- Search inside the x coordinate tree for the node, whose key $u^{(0)}$ is the least possible key such that $x_{\mathbf{Q}} \leq u^{(0)} \leq x_{\mathbf{Q}} + N_T$.
  - Search inside the above y coordinate tree node (thick dashed arrow in Figure 8) for the node whose key $v^{(u,0)}$ is the least possible key such that $y_{\mathbf{Q}} \leq v^{(u,0)} \leq y_{\mathbf{Q}} + M_T$. Add the pixel $(u^{(0)} - x_{\mathbf{Q}}, v^{(u,0)} - y_{\mathbf{Q}})$ to the table.
  - Since all the trees are threaded, we can use the *next* pointer (dotted arrow in Figure 7) to move from y coordinate node $v^{(i)}$ to the next node $v^{(u,i+1)}$. Add the pixel $(u^{(0)} - x, v^{(u,i+1)} - y)$ to the table.
  - Continue until $v^{(u,i+1)} > y + N_T$.
- In a similar manner, we can move from the x coordinate tree node $u^{(u,i)}$ to the next node $u^{(i+1)}$ (dashed arrow in Figure 8) and repeat the process in the y coordinate tree of the new node.
- Continue until $u^{(i+1)} > x + M_T$.

When we finish, the table will contain all the edge pixels contained in the window $\mathbf{Q}$, translated into the relative coordinates of that window.

We input these edge pixels into our tree to find the template T_i that has the best Activated Hausdorff Proximity (3) from the window to the template. If $\mathbf{\Psi}$ is the distance transform of T_i and $\mathbf{\Phi}$ is the distance transform of the image window $\mathbf{Q}$, then the proximity of the image to the window is computed as:

$$\sigma = g(\mathbf{Q}, \mathbf{\Psi}) \tag{14}$$

according to (4). The reverse proximity from the template to the window is also measured as:

$$\rho = g(\mathbf{T}_i, \mathbf{\Phi}) \tag{15}$$

In order to get better matching results, oriented edge pixel information is also taken into consideration. We not only want edge pixels to be near the template contour points, but we also require that the gradient of the closest edge in the image forms an angle with the gradient of the template contour point that is either close to 0 or 180 (lighter object on a darker background or

vice versa). Using $\mathbf{T}_i$'s edge gradients (from the training stage) and the $\mathbf{G}_x$, $\mathbf{G}_y$ maps of the image, we refer to a pre-computed matrix with the cosine of every possible angle that two integer gradient vectors of a greyscale image can form. We give a bonus (for high values) or a penalty (for low values) to the contribution of the template contour points to the correlation of $\mathbf{T}_i$ with $\mathbf{Q}$.

4.1.4. Shape matching

Finally, we can use σ and ρ to determine if a match is made in various ways. Traditional Hausdorff metrics would suggest we use the $\min(\sigma, \rho)$ value to check for a match as a consequence of (2). A more straightforward approach is to demand that ρ exceeds a certain threshold, since there is very little noise in the template and the directed distance from the template to the image is more reliable than the other way around. It is also possible to combine both measurements in various ways (e.g. sum, logical *and*, logical *or* result) to come to a decision. The various stages of the search process are illustrated in Figure 10.

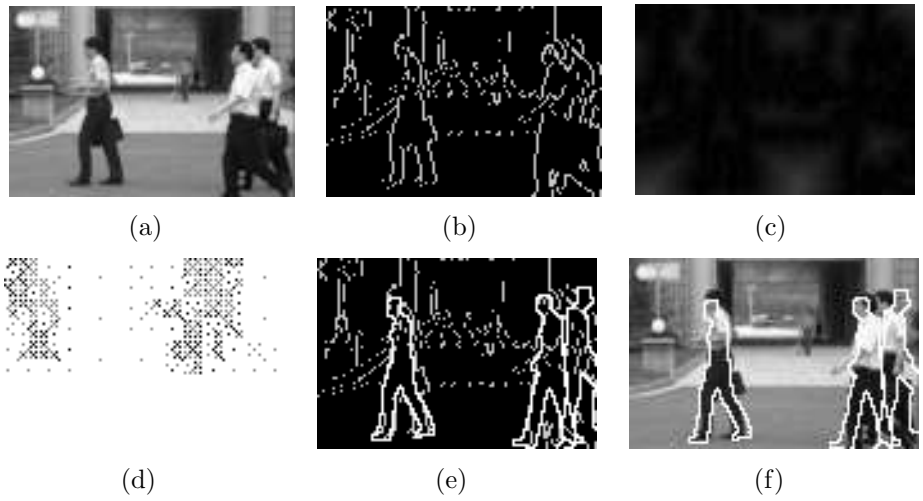


Figure 10: The search process. (a) Original image. (b) Edgemap of original image. (c) Distance transform of original image. (d) Spots where tree searches were performed (upper left corner, darker spots indicate both trees were used). (e) Shape matches presented on the edgemap. (f) Shapes matches presented in original image

4.2. Tree operations analysis

We study the complexity of the shape tree operations as a function of the number $n = |\mathcal{T}|$ of the $N_T \times M_T$ templates in the training set. This analysis only concerns the search operation of the shape trees that takes place in real time during the execution of the testing algorithm. The offline training process is only performed once in the beginning to create the offline shape tree. The similarity measure given in (4) uses matrices, because of their elegance in mathematical expressions. In practice, we use Activated Hausdorff Proximity as expressed in (3) and we view the tree's input as a point set $\mathcal{X}$. Note that the number of points in that set is $|\mathcal{X}| \leq M_T N_T$. For every point $\mathbf{x} \in \mathcal{X}$ we refer to the proper weight matrix ($\mathbf{W}_L$ or $\mathbf{W}_R$) to measure the "distance" $d(\mathbf{x}, \mathcal{Y})$. We then use a pre-computed array with the values of $e^{-\alpha k}$ for every integer k that we expect from the weight matrix. This way, for every point $\mathbf{x}$ we can look up $e^{-\alpha d(\mathbf{x},\mathcal{Y})}$ with only a total of 5 memory references. Since $|\mathcal{X}|$ is also bound by a constant, the computation of a single node matching decision takes constant time.

4.2.1. Tree search

Since a constant time element of regular binary search trees (number comparison) is replaced by with another constant time element (measuring similarity through Activated Hausdorff Proximity), it follows from binary search tree bibliography [13] that the search operation can be performed in $O(\log n)$ time. If we allow a total of t tries, it is easy to see that the complexity becomes $O(t \log n)$, but t is a user defined very small number ($t << n$) that usually does not exceed 10 in our experiments.

4.2.2. Coordinate trees

Collecting all the edge pixels in an image window using the coordinate trees takes $O(\log M_F + \sum_{x \leq i \leq x+M_T} (\log N_F + \sum_{y \leq j \leq y+N_T} p(i,j)))$ time, where $p(i,j) = 1$, if there is an edge pixel with the coordinates (i,j) and $p(i,j) = 0$ otherwise. This can be expanded to $O(\log M_F + \log N_F \sum_{x \leq i \leq x+M_T} p(i,*) + \sum_{x \leq i \leq x+M_T} \sum_{y \leq j \leq y+N_T} p(i,j))$, where $p(i,*) = 1$, if there is an edge pixel whose first coordinate is i regardless of its second coordinate and $p(i,*) = 0$ otherwise. Note that $\sum_{x \leq i \leq x+M_T} \sum_{y \leq j \leq y+N_T} p(i,j)$ is the total number of edge pixels in $\mathbf{Q}$, so the computational cost of collecting all the edge pixels in $\mathbf{Q}$ cannot exceed the number of edge pixels in the window.

5. Experimental Results

In this section the performance evaluation of the proposed data structure on a classification task is presented. We first tested the learning and generalization capabilities of both offline and online constructed trees. Finally, both types of shape trees have been tested using real data. We have chosen to evaluate the proposed shape matching system in the task of pedestrian detection [15], though the system is suitable for other applications as well, such as hand gesture recognition, surveillance and symbol recognition.

We used Poser's walking animation [18] to create the template database. The animation contains 30 frames of a person walking. Each of these 30 frames of the animation was captured from 16 view angles at 22.5^o intervals. Some additional poses were manually composed. Each silhouette was scaled to the height of 80, 75, 70, 65, 60 and 55 pixels and then morphologically eroded [24]. We erode our templates because we find it is better for the template silhouette to fall inside the real silhouette in the edge map rather than outside of it, due to the way gradient orientation is propagated. Finally, edge detection was performed on the silhouettes. The resulting 3660 templates were split into two subsets of 1830 templates each, grouped in descending order according to the templates' height in pixels. A separate tree was constructed for each subset.

Since we will be using our tree for the task of pedestrian detection to evaluate real data performance, we also used these same templates to test the performance of both online and offline trees at learning the template data set. All experiments were conducted on a 2.4GHz Intel Core 2 Quad processor. Taking advantage of the processor's SIMD hardware commands can speed up the distance transform and point set to matrix correlations by a factor of 4 according to [9].

5.1. Classification Capabilities

In order to test the learning capabilities of the proposed shape trees, we considered the two datasets described above, one including 1830 templates whose heights are 70-80 pixels and another including 1830 templates whose heights are 55-65. We trained an online tree and an offline tree for each of the two datasets. Our first experiment aims to evaluate the self-relational and generalization capabilities of both offline and online trees. Each test result is labelled as either i-offline(j) or i-online(j) (depending on the type of tree), where i is the maximum height (in pixels) of the tree's templates and j is

Table 1: 80 pixel height tree learning capabilities

Tree	Learning ability (original data)	Time (ms)	Tree to best match ratio (noisy data)	Best matches found (noisy data)
80-offline(1)	1830/1830	0.1239 (0.000376)	0.9275 (0.0041)	202/1830
80-offline(2)	1830/1830	0.1622 (0.000963)	0.9506 (0.0020)	273/1830
80-offline(4)	1830/1830	0.2620 (0.002669)	0.9605 (0.0014)	374/1830
80-online(1)	1140/1830	0.1552 (0.007890)	0.9047 (0.0034)	82/1830
80-online(2)	1567/1830	0.1787 (0.001565)	0.9294 (0.0022)	177/1830
80-online(4)	1791/1830	0.2923 (0.009989)	0.9484 (0.0015)	235/1830
80-online(6)	1821/1830	0.3597 (0.006780)	0.9571 (0.0012)	282/1830
80-online(8)	1822/1830	0.4573 (0.016678)	0.9632 (0.0010)	321/1830
80-online(10)	1824/1830	0.5268 (0.012101)	0.9645 (0.0010)	326/1830
80-online(12)	1826/1830	0.6020 (0.015737)	0.9683 (0.0009)	384/1830
80-online(14)	1827/1830	0.7455 (0.037167)	0.9692 (0.0009)	407/1830
80-online(16)	1829/1830	0.8136 (0.049964)	0.9700 (0.0008)	401/1830
80-online(20)	1830/1830	0.8684 (0.026441)	0.9714 (0.0008)	398/1830

the total number of tries that are allowed. The results of this experiment are presented in Tables 1 and 2 for each tree respectively. Non constant numbers are presented as *mean (standard deviation)*.

5.1.1. Learning ability

Learning performance was measured by searching for every template in the training set using both trees. A search is considered successful if the output template is exactly the same as the input template. The trees were evaluated by the ratio of successful searches to the total number of searches. As we can see in the second column of Tables 1 and 2, the offline tree can indeed guarantee perfect training and it will always find a template it has stored in the first try for both datasets. The online tree needed at least 20 tries on the first dataset and 14 tries on the second dataset to learn every template, though both performed quite well with 4-6 tries.

5.1.2. Noise Robustness

Noise Robustness was tested by adding noise to a template and then searching for it in the tree. Noise was added by moving each shape point in a random location inside a 5×5 square centered on it's original location. In order to evaluate the generalization ability we compare the performance of the proposed tree structure against exhaustive search. The matching robustness was measured by the ratio calculated by dividing the proximity of each tree match with the maximum proximity of the best match found through exhaustive search. The results of these tests for offline and online trees for both datasets are reported in the third column of Tables 1 and 2 respectively, as the average ratio of each tree's result to the best possible match. Results the

Table 2: 65 pixel height tree learning capabilities

Tree	Learning ability (original data)	Time (ms)	Tree to best match ratio (noisy data)	Best matches found (noisy data)
65-offline(1)	1830/1830	0.1022 (0.000322)	0.9318 (0.0028)	110/1830
65-offline(2)	1830/1830	0.1384 (0.000780)	0.9533 (0.0014)	147/1830
65-offline(4)	1830/1830	0.2046 (0.001858)	0.9607 (0.0010)	184/1830
65-online(1)	969/1830	0.1147 (0.000830)	0.9324 (0.0034)	81/1830
65-online(2)	1443/1830	0.1655 (0.003459)	0.9567 (0.0010)	144/1830
65-online(4)	1762/1830	0.2086 (0.002868)	0.9693 (0.0007)	252/1830
65-online(6)	1814/1830	0.2707 (0.003261)	0.9737 (0.0005)	262/1830
65-online(8)	1822/1830	0.3486 (0.061034)	0.9772 (0.0004)	319/1830
65-online(10)	1827/1830	0.4282 (0.009168)	0.9782 (0.0004)	350/1830
65-online(12)	1829/1830	0.4898 (0.015015)	0.9788 (0.0004)	336/1830
65-online(14)	1830/1830	0.5225 (0.011551)	0.9796 (0.0004)	373/1830

first dataset (Table 1) indicate that offline trees have better robustness than the online trees for the same number of tries. As observed in these Tables, an online tree requires about double the number of tries that an offline tree needs in order to yield a similar robustness ratio. Online trees, however, are not far behind and seem to catch up if given plenty of tries. On the second dataset, however, the difference in the performance of both trees seems to be within the statistical error margin.

We also investigated whether the trees can find the best possible match that an exhaustive search would come up with. The last column of Tables 1 and 2 shows the number of searches whose result coincided with the exhaustive algorithm's output. The trees do not always return the best result and, as expected, the number of times they do find this best match increases along with the number of tries. The trees do, however, find very close matches most of the time, as indicated by the fourth column of Tables 1 and 2. Figure 11 shows the results of some sample template searches that have varying similarity ratios to the similarity of the corresponding best match found exhaustively. Some results are perfect, as in Fig 11(a), the quality of the matching in the average case is best illustrated by 11(b) and there are some results that are not very good, as seen in 11(d), but they are uncommon.

5.2. Real Data

Finally, we have tested the performance of both offline and online shape trees in a pedestrian detection system using a real pedestrian image database. The pedestrian detection task involves feeding an input image or video sequence to the detection system which must then locate all pedestrians in the

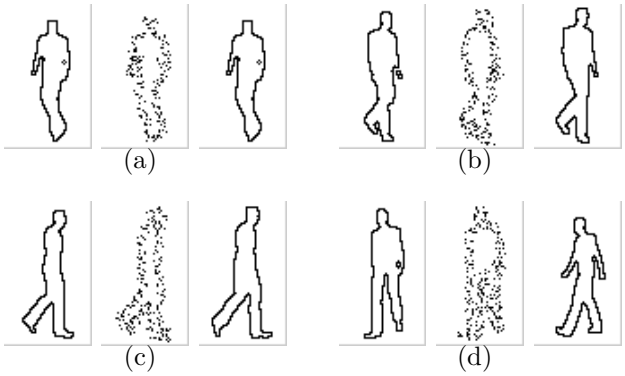


Figure 11: The original template, the noise-affected input and the final result for different values of the tree/best match ratio. (a) 1.000000 (b) 0.965899 (c) 0.897539 (d) 0.856618

image or video frame. In "smart" vehicles, such a system must work in real time.

In the field of pedestrian detection, a multitude of methods have been proposed in recent years. Histograms of Oriented image Gradients (HOG) are used in [5]. An SVM approach using wavelet features is described in [21], while statistical learning classifiers and SVMs are combined into a cascaded classifier using Haar-like features in [34]. A scale-invariant implicit shape model to initially detect and then top-down segmentation to confirm a possible result is used in [14] and [32]. A recent advancement in [30], selects several image features for each pixel, then computes the covariance matrices of these features over various image regions to use as object identifiers. These covariance matrices are symmetric positive definite and therefore lie on a Riemannian manifold. The classification occurs on the tangent space of the weighted Karcher mean point that also lies on that manifold. A rejection cascade is constructed through the use of boosting that determines the weighted mean point and the decision boundary for each classifier. Detailed surveys on pedestrian detection are presented in [15] and [7].

The idea of using shape matching for the task of real-time pedestrian detection has been explored in [9]. Recently, this method was improved in [10], which proposes a B-tree like data structure and a probabilistic framework for traversing it. Since we also propose a tree data structure for the same task, we will use his work as our benchmark. As noted in the same paper, pedestrian detection proves to be one of the most challenging tasks

to perform using shape matching techniques.

The state of the art approach, as described in [16], uses a tree of part-template silhouettes to make an initial possible human segmentation in a detection window. A feature vector similar to the HOG features in [5] is formed out of selected blocks around the initial human silhouette estimation. That feature vector is then fed through an appropriately trained SVM in order to make the final decision of whether the detection window contains a human or not. Note that the tree in this approach is static and a greedy search algorithm is used, so this approach could potentially benefit from our data structure in order to greatly expand the available part-template silhouettes.

We used the pedestrian detection database containing images from the University of Pennsylvania and Fudan University, which was also used in [33]. The image database contains 421 pedestrians in 170 images. Sample images can be seen in Figure 12. All the images were scaled so that the pedestrians' height in pixels falls into the 55-80 pixel range. Some pedestrians, whose height after the scaling was too small for our system to detect were ignored (21 pedestrians in total). We used the strategy described in section 4.1 to search for possible matches in the database images. The point sets for each tree search were undersampled by considering every third edge point in a set. Note that since both the training and test sets are completely different from the ones used in [10], the results are not immediately comparable. This is, however, the closest comparative evaluation we could manage, since the contour templates and image data set that were used in [10] are publicly unavailable. Also note that the comparison is focused on the difference between our system with the state of the art in fast, tree-based shape matching, not the state of the art pedestrian detection systems, which involve stronger classifiers.

We built one offline and one online tree containing the templates, whose heights range from 70-80 pixels. Likewise, we built another tree for the templates, whose heights are in the 55-65 pixel range. We implemented one system for each type of trees. The number of tries was set to 4 for the offline trees and 8 for the online trees. We first used the tree which stored the templates whose height is 55-65 pixels and, if the resulting match was good enough, the tree with the templates whose height is 70-80 pixels was used as well. Matches that were too close together spatially were merged to the match having the highest matching criterion. A match was considered successful if the corners of the detection bounding box were within 20 pixels

Figure 12: Some output examples. White bounding boxes indicate detections, grey boxes indicate a person successfully detected, black boxes indicate false negatives.

of the respective corners of the ground truth bounding box (same as in [10]). Multiple detections of the same ground truth pedestrian were only counted once. Sample results can be seen in Figure 12.

Let N_{TP} be the number of true positives, N_{FP} the number of number of false positives and N_{FN} the number of false negatives. Recall and precision were defined as $N_{TP}/(N_{TP}+N_{FN})$ and $N_{TP}/(N_{TP}+N_{FP})$ respectively. The performance results for both trees, the performance of the system reported in [10], as well as the performance of our system in which the tree search was replaced with an exhaustive template search are presented in Table 3. The computational time figures for the system in [10] were inferred from the frames per second rates reported. The computational time figures for our systems were averaged. In our time measurements, we excluded the time it takes for the image to be loaded from the disk but included the time it takes to scale the image and compute its gradients, edges and distance transform, as well as the actual search time. Note that it is possible for our systems' computational times to more closely match those of [10] through the use of multiple threads and hardware optimization.

By overviewing the results, we should once again note that direct comparison with [10] is impossible due to inevitable differences in training and test sets. We attribute the apparent superior performance of our method to two factors:

a) The ability of the proposed trees to manage more templates, due to the low theoretical upper bounds for binary search tree operations.

b) The fact that we use the edge pixels themselves to guide us to the best result and do not rely on measurements with shape exemplars, as such features are less reliable in Hausdorff based metrics.

It is also evident that offline trees provide an improvement in performance over the online trees (offline trees have slightly higher recall and precision) in addition to being measurably faster. However, in contrast to what was expected, the exhaustive algorithm found fewer template matches, since it has worse recall and better precision than the trees. Since it is impossible for the exhaustive search to find template matches with less σ, as defined in (14), than the trees, the only remaining explanation for the fact that exhaustive search finds less overall matches is that some template matches do not satisfy the threshold for ρ, as defined in (15), and are, therefore, rejected. The behaviour of the exhaustive search algorithm indicates overtraining and leads us to believe that finding the best one way match is not the optimal approach to shape matching. The decision making nature of the trees also provides

Table 3: Performance comparison

System	Recall	Precision	Time (seconds)
[10]	0.80	0.26	0.06 - 0.14
Offline(4)	0.885	0.69	0.15
Online(8)	0.8825	0.67	0.21
Exhaustive	0.8075	0.72	4.0833

matches that yield higher reverse similarity, thus performing better than the exhaustive approach. Finally, we do not claim to have devised an extremely high performance pedestrian detection classifier. We have provided a real time system for the quick detection and localization of potential pedestrian matches. A more powerful but slower classifier can be used to verify the matches produced by our system for further improved results.

6. Conclusion

In this paper, a novel algorithm for shape matching based on the Hausdorff distance and a binary search tree data structure have been proposed. By using a variation of a well known similarity measure in a manner that allows us to make a decision on which subtree to direct a search at each node of the proposed tree, we can store and search for templates in logarithmic time. Through the use of probabilistic formulations, we came up with a way to train a weak classifier inside each node so that it can correctly direct a search to its proper subtree, thus ensuring that the output template is very close to the best result we would find through exhaustive search.

Experimental results indicate that the proposed structure is capable of storing and later finding the templates it must learn. The performance under noise interference is also very good. While offline trained trees slightly outperform them, online constructed trees are still a viable solution for shape matching tasks. Experiments on real data also suggest an improvement over the previous state of the art real-time pedestrian detection system.

References

[1] Amit, Y., Geman, D., Fan, X., 2004. A coarse-to-fine strategy for multiclass shape detection. IEEE Transactions on Pattern Analysis and Ma-

chine Intelligence 26 (12), 1606–1621.

[2] Amit, Y., Geman, D., Wilder, K., 1997. Joint induction of shape features and tree classifiers. IEEE Trans. Pattern Anal. Mach. Intell. 19 (11), 1300–1305.

[3] Borgefors, G., June 1986. Distance transformations in digital images. In: Computer Vision, Graphics, and Image Processing, Volume 34 , Issue 3, pp. 344-371.

[4] D. Huttenlocher, G. K., Rucklidge, W., Sep. 1993. Comparing images using the Hausdorff distance. In: IEEE Trans. Pattern Analysis and Machine Intelligence, vol. 15, no. 9, pp. 850-863.

[5] Dalal, N., Triggs, B., 2005. Histograms of oriented gradients for human detection. In: Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 1. pp. 886–893.

[6] Dubuisson, M.-P., Jain, A., 1994. A modified Hausdorff distance for object matching. In: Pattern Recognition, 1994. Vol. 1 - Conference A: Computer Vision & Image Processing., Proceedings of the 12th IAPR International Conference on. pp. 1: 566–568.

[7] Enzweiler, M., Gavrila, D. M., 2009. Monocular pedestrian detection: Survey and experiments. IEEE Trans. Pattern Anal. Mach. Intell. 31 (12), 2179–2195.

[8] Foo, B., Turaga, D., Verscheure, O., van der Schaar, M., Amini, L., 2008. Resource constrained stream mining with classifier tree topologies. Signal Processing Letters, IEEE 15, 761–764.

[9] Gavrila, D., Philomin, V., 1999. Real-time object detection for "smart" vehicles. IEEE International Conference on Computer Vision 01.

[10] Gavrila, D. M., Aug. 2007. A bayesian, exemplar-based approach to hierarchical shape matching. In: IEEE Transactions on Pattern Analysis and Machine Intelligence, Volume 29, Issue 8, pp. 1408-1421.

[11] Gdalyahu, Y., Weinshall, D., 1999. Flexible syntactic matching of curves and its application to automatic hierarchical classification of silhouettes.

IEEE Transactions on Pattern Analysis and Machine Intelligence 21, 1312–1328.

[12] Klassen, E., Srivastava, A., Mio, W., Joshi, S. H., 2004. Analysis of planar shapes using geodesic paths on shape spaces. IEEE Transactions on Pattern Analysis and Machine Intelligence 26 (3), 372–383.

[13] Knuth, D. E., April 1998. Art of Computer Programming, Volume 3: Sorting and Searching (2nd Edition).

[14] Leibe, B., Seemann, E., Schiele, B., 20-25 June 2005. Pedestrian detection in crowded scenes. Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on 1, 878–885 vol. 1.

[15] Li, Z., Wang, K., Li, L., Wang, F.-Y., 13-15 Dec. 2006. A review on vision-based pedestrian detection for intelligent vehicles. Vehicular Electronics and Safety, 2006. ICVES 2006. IEEE International Conference on, 57–62.

[16] Lin, Z., Davis, L. S., 2009. Shape-Based Human Detection and Segmentation via Hierarchical Part-Template Matching. IEEE Transactions on Pattern Analysis and Machine Intelligence (1).

[17] Lu, Y., Tam, C. L., Huang, W., Fan, L., 2001. An approach to word image matching based on weighted Hausdorff distance. In: ICDAR '01: Proceedings of the Sixth International Conference on Document Analysis and Recognition. IEEE Computer Society, Washington, DC, USA, pp. 921–925.

[18] Curious Labs, 2006. Poser.

[19] Olson, C., 1998. A probabilistic formulation for Hausdorff matching. Computer Vision and Pattern Recognition, 1998. Proceedings. 1998 IEEE Computer Society Conference on.

[20] Olson, C. F., Huttenlocher, D. P., Jan. 1997. Automatic target recognition by matching oriented edge pixels. IEEE Transactions on Image Processing 6 (1), 103–113.

[21] Papageorgiou, C., Evgeniou, T., Poggio, T., 1998. A trainable pedestrian detection system.

[22] Rucklidge, W., June 1995. Locating objects using the Hausdorff distance. In: Proceedings of Fifth International Conference on Computer Vision, pp. 457-464.

[23] S. Belongie, J. M., Puzicha, J., Apr. 2002. Shape matching and object recognition using shape contexts. In: IEEE Transactions on Pattern Analysis and Machine Intelligence Volume 24 , Issue 4, pp. 509 - 522.

[24] Serra, J., 1983. Image Analysis and Mathematical Morphology. Academic Press, Inc., Orlando, FL, USA.

[25] Shlien, S., 1990. Multiple binary decision tree classifiers. Pattern Recogn. 23 (7), 757–763.

[26] Siddiqi, K., Shokoufandeh, A., Dickinson, S. J., Zucker, S. W., 1998. Shock graphs and shape matching. In: ICCV '98: Proceedings of the Sixth International Conference on Computer Vision. IEEE Computer Society, Washington, DC, USA, p. 222.

[27] Srivastava, A., Joshi, S. H., Mio, W., Liu, X., 2005. Statistical shape analysis: Clustering, learning, and testing. IEEE Transactions on Pattern Analysis and Machine Intelligence 27 (4), 590–602.

[28] Torsello, A., Hancock, E. R., 2006. Learning shape-classes using a mixture of tree-unions. IEEE Trans. Pattern Anal. Mach. Intell. 28 (6), 954–967.

[29] Tsapanos, N., Tefas, A., Pitas, I., 2010. Online shape learning using binary search trees. Image and Vision Computing 28 (7), 1146 – 1154, online pattern recognition and machine learning techniques for computer-vision: Theory and applications.

[30] Tuzel, O., Porikli, F., Meer, P., 2008. Pedestrian detection via classification on riemannian manifolds. IEEE Trans. Pattern Anal. Mach. Intell. 30 (10), 1713–1727.

[31] Veltkamp, R. C., Hagedoorn, M., 2000. Shape similarity measures, properties and constructions. In: VISUAL '00: Proceedings of the 4th International Conference on Advances in Visual Information Systems. Springer-Verlag, London, UK, pp. 467–476.

[32] Wang, L., Shi, J., Song, G., fan Shen, I., 2007. Object detection combining recognition and segmentation. In: ACCV (1). pp. 189–199.

[33] Wang, L., Shi, J., Song, G., Shen, I., 2007. Object detection combining recognition and segmentation. In: 8th Asian Conference on Computer Vision. pp. I: 189–199.

[34] Xu, Y., Cao, X., Qiao, H., Wang, F., 13-15 June 2006. A cascaded classifier for pedestrian detection. IEEE Intelligent Vehicles Symposium,, 336–343.

[35] Zhang, D., Lu, G., 2004. Review of shape representation and description techniques. Pattern Recognition 37 (1), 1–19.