Adaptive and Compact Shape Descriptor by Progressive Feature Combination and Selection with Boosting

Cheng Chen Yueting Zhuang Jun Xiao Fei Wu

Institution of Artificial Intelligence, Zhejiang University,
Hangzhou, 310027, China
cchen@zju.edu.cn

Abstract

Many types of shape descriptors have been proposed for 2D shape analysis, but most of them consist of component features that are not adapted to specific problems. This has two drawbacks. First, computation is wasted on the irrelevant components; second, the accuracy is impaired. This paper proposes an effective method that generates compact descriptors adapted to specific problems in hand, where each component of the new descriptor is a linear combination of the components in some classic descriptors. A progressive strategy is used to construct and select the most suitable linear combinations in successive rounds, where a variant of Adaboost is employed to ensure the optimum of the selected combinations in each round. Experiments show that our method effectively generates adaptive and compact descriptors for typical applications such as shape classification and retrieval.

1. Introduction

2D shape analysis is a classical and very important topic in computer vision and pattern recognition. Many applications rely on successful retrieval or classification of shapes. Examples are object recognition [13], human pose estimation [16], behavior recognition [2], and so on. A lot of shape descriptors have been proposed to that end [11] [20], such as Fourier descriptor [19] [8], wavelet descriptor [3], moment invariants [7] [9], shape context [1]. The idea is that the descriptor encodes the shape's property and can be used as the shape's vector representation in operations such as retrieval and classification.

However, most classical shape descriptors are not adaptive. Take Fourier descriptor for example. It is the collection of all Fourier coefficients regardless of the specific problems. It's well known that many coefficients are either irrelevant (a waste of dimension) or misleading (impairing the overall accuracy) or both. For typical applications, only a subset of the coefficients is useful. Some authors choose the coefficients by some general heuristics [10]. However, the question regarding which coefficients are useful depends on the specific problem. As a simple but convincing example, in a task of classifying by general shapes (e.g. discriminating rectangles from circles), Fourier coefficients in the low frequencies may play the role, while in another case of telling whether a shape is corrupted by noises, the coefficients in high frequencies speak.

In this paper we propose a method that generates new shape descriptor from classical ones, where the generated descriptor is well adapted to the problem in hand with reduced descriptor length. Each component of the generated descriptor is a linear combination of the components in the classical descriptor. Generating the new descriptor is therefore equivalent to constructing and searching for a small collection of such linear combinations that are best adapted for the problem. Figure 1 shows the algorithm overview. The search is conducted progressively in successive rounds. In each round, we first generate a pool of new candidate components by binary combinations of current components, and then the optimal combinations are selected and passed to the next round, where further combination takes place. The feature selection in each round is supervised by a variant of Adaboost algorithm to ensure that the selected components have best discriminative power when used jointly.

Compared to classical descriptors, the generated adaptive and compact descriptor has two advantages:

1. Higher accuracy. The irrelevant components in classical descriptors may dominate the overall judgment. By concentrating only on the components suitable for the problem, the accuracy can be improved.
2. Higher efficiency. Since irrelevant components are discarded, the descriptor length is reduced. This is particularly important in many applications where a large shape database must be processed online.

Note that there exists some work of combining shape descriptors that is related to ours. Terrades *et al.* [17]

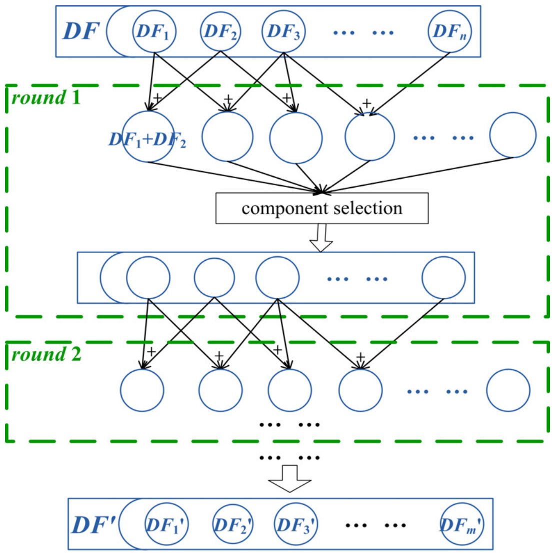


Figure 1: Algorithm overflow

combine several classifiers that operate on different descriptors (e.g. Fourier and wavelet descriptor) using a variation of Adaboost. Yin *et al*. [18] assign separate weights to each training sample on different descriptors and determine the contribution of each descriptor to the final classifier by its performance along the boosting iterations. This paper is different from theirs in two aspects:

1. Their work is essentially *feature combination*, while our method is a blending of *feature combination* and *feature selection*. In [17] and [18], each component of the involved descriptors is used for combination. For example, when combining Fourier descriptor and wavelet descriptor, all Fourier and wavelet coefficients are used and appear in the final classifier. Therefore, compact descriptor cannot be achieved. Our method, however, generates both adaptive and compact features.

2. They limit shape descriptor in shape classification [13]. Their training set is comprised of shapes with labeled classes, and the final output is *classifier* rather than *descriptor*. Therefore, they are not suited for applications such as shape retrieval or unsupervised clustering.

In our method, we don't assume the shapes belong to different classes. The training set of our method is a collection of shape pairs. Each pair consists of two shapes (classical descriptors) labeled as *similar* or *dissimilar*, indicating whether the two shapes should be considered as similar according to the problem in hand. The output is a new descriptor that is adapted to the specific problem with compact length, which can be used in applications including classification, retrieval, clustering, and so on.

The paper is organized as follows. The proposed method is described in Section 2. Section 3 presents two experiments. Section 4 makes the conclusion.

2. The Proposed Method

2.1. Problem formalization

We write a *shape descriptor* as a function $\boldsymbol{DF}$:

$$\boldsymbol{DF}(\boldsymbol{S}) = [DF_1(\boldsymbol{S}), DF_2(\boldsymbol{S}), \ldots, DF_n(\boldsymbol{S})]. \quad (1)$$

Here, $\boldsymbol{S}$ is a digital shape (in some raw format, such as discrete pixel coordinates). $\boldsymbol{DF}()$ is an n-dimensional vector function that calculates descriptors from shapes. $DF_1()$, …, $DF_n()$ are *components*. For example, when $\boldsymbol{DF}$ is Fourier descriptor, the components DF_1, …, DF_n return the Fourier coefficients when applying DFT to the shape. Note that $\boldsymbol{DF}$ does not need to be one to one. That is, it's possible that different shapes have identical descriptor.

In this paper we are interested in generating a new descriptor function $\boldsymbol{DF'}()$, that is adaptive and compact, from a classical descriptor function $\boldsymbol{DF}()$. $\boldsymbol{DF'}$ is an m dimensional composite function:

$$\boldsymbol{DF'}(\boldsymbol{S}) = \boldsymbol{DF'}(\boldsymbol{DF}(\boldsymbol{S})) = [DF_1'(\boldsymbol{DF}(\boldsymbol{S})), \ldots, DF_m'(\boldsymbol{DF}(\boldsymbol{S}))], \quad (2)$$

where each component function $DF_1'()$,…, $DF_m'()$ is defined on $\boldsymbol{DF}()$. Thus, our task is to design the proper component functions DF_1',…, DF_m'.

2.2. Overview

The basic property of a shape descriptor is that it should encode shape similarity – similar shapes should have close descriptors and vice versa. This can be expressed as:

$$\left\|\boldsymbol{DF}(S^1) - \boldsymbol{DF}(S^2)\right\| \text{ is small if and only if } N(S^1, S^2) = 1, \quad (3)$$

where $N(,)$ labels a pair of shapes $\boldsymbol{S}^1$ and $\boldsymbol{S}^2$ as 1 if they are similar in ground truth and –1 otherwise, and $\|.\|$ is a distance metric in descriptor space (typically $L2$ distance).

Since the goal is to generate new descriptor that is adaptive to the specific problem, there need to be a way to feed the algorithm with the information about the problem in hand. Inspired by (3), we provide the algorithm with some positive and negative examples. Suppose we have T training shape pairs $<\boldsymbol{S}^{11}, \boldsymbol{S}^{12}>$, …, $<\boldsymbol{S}^{T1}, \boldsymbol{S}^{T2}>$. Then, for a classical descriptor function $\boldsymbol{DF}$ (from which new descriptor will be generated), a set of training samples can be defined. The tth sample is a triple $<\boldsymbol{d}^{t1}, \boldsymbol{d}^{t2}, y^t>$, where $\boldsymbol{d}^{t1}=\boldsymbol{DF}(\boldsymbol{S}^{t1})$, $\boldsymbol{d}^{t2}=\boldsymbol{DF}(\boldsymbol{S}^{t2})$, and $y^t = N(\boldsymbol{S}^{t1}, \boldsymbol{S}^{t2})$ [1]. All information about the current problem is conveyed by these training samples[2].

As in (3), the distance between the descriptors of similar

[1] In this paper, we use superscript to differentiate serialized instances in a series, and subscript to identify the component of a vector. For example, $\boldsymbol{d}^{t1}$ means descriptor #1 in the tth pair, and $\boldsymbol{DF}_i$ means the ith component of descriptor function $\boldsymbol{DF}$. As a more complex example, $\boldsymbol{d}_i^{t1}$ stands for the ith component of descriptor #1 in the tth pair.

[2] The labeling of training shapes (the y^t values) can be acquired manually (e.g. by human judgment), or automatically by some metric that encodes the ground-truth similarity. See the experiments for example.

shapes should be small and vice versa. In light of this, we can think of a threshold binary classifier that predicts whether two shapes are similar by their descriptors: if the distance between the descriptors is below some threshold δ, then the two shapes are predicted as similar, and vice versa. Obviously, the accuracy of this prediction is directly related to the discriminative power of the descriptor. Therefore, the "goodness" of $\boldsymbol{DF}$ on the training set can be numerically evaluated by the training error of this threshold binary classification:

$$e = \frac{1}{2T}\sum_{t=1}^{T}(y^t \cdot \mathrm{sign}(\|\boldsymbol{d}^{t1} - \boldsymbol{d}^{t2}\| - \delta) + 1), \tag{4}$$

where δ is a chosen threshold that minimizes e.

If, instead, we use single descriptor component rather than the whole descriptor in equation (4), we will get the training error for each single component:

$$e_i = \frac{1}{2T}\sum_{t=1}^{T}(y^t \cdot \mathrm{sign}(\|d_i^{t1} - d_i^{t2}\| - \delta) + 1), \tag{5}$$

where d_i^{t1} and d_i^{t2} denote the i^{th} component of descriptor $\boldsymbol{d}^{t1}$ and $\boldsymbol{d}^{t2}$, respectively. Equations (4) and (5) are basic guidance in the feature combination and selection later.

As stated above, different components are differently suited for the specific problem. We make two statements:

1. Many components are poorly suited for the problem in hand, i.e. have high training errors in equation (5). These components should be discarded because they increase the descriptor dimension while potentially misleading the overall judgment made by the descriptor as a whole.

2. To further reduce the descriptor dimension, it's possible to combine several components to act as one single component in the new descriptor, as long as the combined component has lower (or at least not higher) training errors than each constituting component.

Our method is inspired by the above two statements. Generally speaking, we construct new descriptor from classical descriptor by two actions:

1. Discard components not suited for the problem.
2. Combine several components into one, if preferable.

2.3. Progressive feature combination

Before discussing the feature combination algorithm, the combination model, i.e. how the components are combined, has to be determined. We use a linear combination model, where each component of the new descriptor is a linear combination of the components in the original descriptor. This can be expressed as follows:

$$DF_i' = \sum_{j=1}^{n} w_{ij} \cdot DF_j \tag{6}$$

where w_{ij} are the weights under constraints that each w_{ij} is a non-negative integer and that for any i value, at least one w_{ij} is nonzero. For example, DF_3, DF_1+DF_3, and $2DF_2+3DF_4+5DF_7$ are three possible combinations.

```
Procedure: progressive searching for combinations

Input:
    classical descriptor function: $\boldsymbol{DF} = [DF_1, DF_2, \ldots, DF_n]$,
    training set: $<\boldsymbol{d}^{11}, \boldsymbol{d}^{12}, y^1>, <\boldsymbol{d}^{21}, \boldsymbol{d}^{22}, y^2> \ldots <\boldsymbol{d}^{T1}, \boldsymbol{d}^{T2}, y^T>$
Output:
    new descriptor generation function $\boldsymbol{DF'}=[DF_1', DF_2', \ldots, DF_m']$

 1 begin initialize  single components set: $C_S = \{DF_1, DF_2, \ldots, DF_n\}$,
 2                   reserved components set: $C_r = \emptyset$
 3   for $k = 1$ to $K$
 4       candidate components set $C_C = \emptyset$
 5       $C_r = C_r \cup C_S$
 6       for each two components $c_i$ and $c_j$ in $C_r$
 7           $C_C = C_C \cup \{c_i + c_j\}$
 8       endfor
 9       select $M$ most suitable components in $C_C$ and store them in $C_r$
10   endfor
11   select $m$ best components $c_1, \ldots, c_m$ in $C_r$
12   $\boldsymbol{DF'} = [c_1, c_2, \ldots, c_m]$
13 end
```

Figure 2: Progressive feature combination.

The choice of linear combination model has two reasons:

1. It conforms to the problem's nature. It is possible that if several components are reasonably suited for the problem on their own, their linear combination is better than any single one. Moreover, linear combination has clear meaning for some descriptors. For example, it is reasonable that the accumulated energy over a range of Fourier frequencies matters in discriminating shapes.

2. Linear combination is ready for progressive construction. Even if we constrain the combination to be linear and constrain the weights, there are still a huge number of (probably indefinite) possible combinations, making exhaustive searching prohibitive. Therefore, we take a progressive way. Linear combination is ready to be constructed in this recursive way, as complex combinations can be constructed by linearly combining simpler ones.

Figure 2 shows our algorithm of progressive feature combination. The algorithm starts with the set of all single components in the classical descriptor (line 1), and finally outputs a set of best combinations as components of the new descriptor (line 12). In each round, we conduct exhaustive binary combinations on the reserved components (lines 6 and 7). These combinations, along with the original reserved components, form the candidate components from which M best ones are selected and reserved for the next round (line 9). The algorithm terminates if either of the two criteria is satisfied: (1) the output of a round is the same as its input, indicating that a set of stable combinations have been found; (2) a maximum of K rounds have reached. Note that the single components are passed down to each round since they are potential building blocks for combination, and excellent single components can be selected finally to become components in the new descriptor on their own.

Note that we do not permit negative combination weights in equation (6), because in practice it is unreasonable to have a descriptor component which has the reversed property – being quite different when two shapes are similar. Therefore, the combination in line 7 of Figure 2 is constrained to addition. We have done some informal experiments which permit negative weights, but no negative weights ever appeared in the intermediate or final results. Therefore, it's safe to allow only zero and positive weights.

M is a parameter controlling the number of combinations reserved in each round. If M is large, then more components are reserved in each round, indicating smaller chances that potential combinations are discarded. But larger M also means more computation. During the experiments we find that it is desirable to set M roughly to $2m$ (where m is the dimension of the output descriptor). Larger values do not perceptibly improve the performance of the final descriptor. In all our experiments in this paper, we set M=$2m$.

There remains one important issue in the algorithm: how to select the best combinations in each round (line 9 in Figure 2). This is discussed in the next subsection.

2.4. Feature selection using modified Adaboost

The remaining issue in the algorithm is the selection of combinations in each round (lines 9 and 11 in Figure 2).

Equations (4) and (5) give the training errors of the whole descriptor $\boldsymbol{d}$ and of single components, respectively. When it comes to the combination of components, the training error can be calculated in a similar way. For a combination defined in equation (6), its training error is given by:

$$e = \frac{1}{2T}\sum_{t=1}^{T}\left(y^t \cdot \mathrm{sign}\left(\left\|\sum_{j=1}^{n} w_{ij} d_j^{t1} - \sum_{j=1}^{n} w_{ij} d_j^{t2}\right\| - \delta\right) + 1\right) \quad ,(7)$$

where w_{ij} is the weights in equation (6), and d_i^{t1} stands for the ith component of descriptor $\boldsymbol{d}^{t1}$.

The selected combinations in each round should be the ones that best suit the problem when used jointly. A simple way is by calculating the training errors of all the candidate combinations using equation (7) and selecting M ones with the lowest errors. However, this approach is not desirable. Although several combinations might each has low training error of their own, they might be improper to be used jointly. This will happen when their classification configurations on the training data are similar. For example, suppose there are two combinations c_1 and c_2, both with a training error of 20%. If c_1 and c_2 classify the training data in similar ways (i.e. the 80% of the training samples correctly classified by c_1 are also correctly classified by c_2, while the remaining 20% samples are correctly classified by neither), then it does no good to select both c_1 and c_2 to be the reserved components. Using both c_1 and c_2 jointly is a waste of dimension because the collaborative error is unlikely to be reduced. Moreover, because the combinations are used as components of the generated descriptor, there is a danger that two or more components with the same property (such as c_1 and c_2) may outweigh other components in the final descriptor, impairing the discriminative power.

Adaboost [4] [15] is an excellent choice in this case, where a final ensemble classifier is learned as a weighted linear combination of a set of weak classifiers. Adaboost operates in rounds. In each round, a best weak classifier is selected. The idea is that, after each selection, the algorithm forces subsequence classifier selections to pay more attention on the samples incorrectly classified. In this way Adaboost selects weak classifiers that performs well jointly.

In each round of the progressive searching in Figure 2, we employ Adaboost to conduct feature selection from candidate combinations. The feature selection algorithm is given in Figure 3. Here, candidate combinations are treated

```
Procedure: component selection by a variant of Adaboost

Input:
    candidate components set : $C_c$={$c_1, c_2, \ldots, c_N$}, where $c_i = \sum_{j=1}^{n} w_{ij} DF_j$
    training set: <$\boldsymbol{d}^{11}, \boldsymbol{d}^{12}, y^1$>, <$\boldsymbol{d}^{21}, \boldsymbol{d}^{22}, y^2$>… <$\boldsymbol{d}^{T1}, \boldsymbol{d}^{T2}, y^T$>
Output:
    reserved components set: $C_r$ with $M$ selected components

1 begin initialize    training sample weights: $\pi_1(i) = 1 / T\ (i = 1,\ldots,T)$
2     for $k$ = 1 to $M$
3         for each component $c_i$ in $C_c$
4             calculate training error $e_i$ using:
              $e_i = \frac{1}{2T}\sum_{t=1}^{T}\pi_k(t)(y^t \cdot \mathrm{sign}(\|\sum_{j=1}^{n} w_{ij} d_j^{t1} - \sum_{j=1}^{n} w_{ij} d_j^{t2}\| - \delta) + 1)$
5             penalized training error $e_i' = e_i \cdot \exp(\lambda \sum_{j=1}^{n} w_{ij} - 1)$
6         endfor
7         select the component $c(k)$ in $C_c$ with minimum $e_k'$
8         $\alpha_k = (1/2)\ln[(1 - e_k)/e_k]$
9         for $t$ = 1 to $T$
10            update weight $\pi_k(t)$ using:
              $\pi_{k+1}(t) = \frac{\pi_k(t)}{Z_k} \times \begin{cases} \exp(-\alpha_k), & \text{if the } t^{\text{th}} \text{ sample is correctly classified} \\ \exp(\alpha_k), & \text{if the } t^{\text{th}} \text{ sample is incorrectly classified} \end{cases}$
11        endfor
12    endfor
13    $C_r$ = { $c(1), c(2), \ldots, c(k)$}
14 end
```

Figure 3: Feature selection by modified Adaboost

as weak classifiers whose training errors are given by (7), and the algorithm operates M rounds (line 2), each round selecting one weak classifier. A list of weights is maintained for every training sample. When the algorithm starts, all these weights are initialized equally (line 1), indicating that no bias on the training samples is present. In each round, the training error for each weak classifier is calculated using the weights as distribution (line 4). After the weak classifier with lowest penalized training error is selected (line 7), the weight list is updated, where samples correctly classified by the selected weak classifier have their weights reduced and weights of incorrect samples are increased (line 10).

For each selected weak classifier, Adaboost computes its contribution α in the ensemble classifier (line 8). Because our focus is not classification, we discard these weights when assembling the final descriptor.

In order to prevent overfit, we make a modification to the original Adaboost by penalizing complex combinations (line 5). The complexity of a combination is defined to be the sum of its weights. The penalized training error is:

$$e' = e \cdot \exp(\lambda \sum_{j=1}^{n} w_{ij} - 1) \tag{8}$$

where w_{ij} is the combination weights as in equation (6), e is the error given by (7), and λ is the penalty factor and its value is set to 0.01 by inspection.

3. Experiments

We conduct experiments on shape classification and retrieval.

3.1. Classification experiment – MINST database

Experiment setup

MNIST handwritten digit database is used for shape classification evaluation. 12,000 training samples and 2,000 testing samples are selected and used in our experiment. Noises are also added to the samples.

We use a composite classical descriptor consisting of Fourier descriptor (FD) and wavelet descriptor (WD). Typically, four low level shape signatures defined on shape border can be used to calculate FD and WD ([8] [21]):

1. *Complex coordinates*: the coordinates along border.
2. *Centroid distances*: the distances to shape centoid along border.
3. *Turning angles*: the tangent angles along border.
4. *Curvatures*: the curvatures along border.

For each shape, we uniformly sample 32 points along its border[3], and the above four shape signatures are generated using these 32 points. FD is comprised of Fourier coefficients calculated by applying DFT to one of the

[3] The sampling of 32 points along the border is due to the low resolution of MNIST images (28*28). We have done experiments using 32 and 64 points, and no perceptible accuracy difference is discovered.

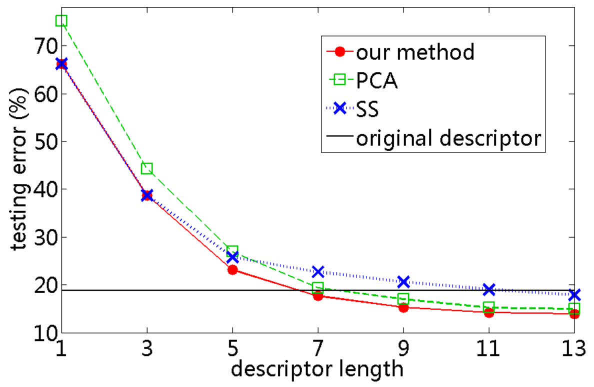


Figure 4: Average testing errors on MNIST database.

signature and WD is composed of wavelet coefficients computed by DPWT (Discrete Periodic wavelet Transform [5]). All levels of wavelet coefficients are reserved. We compute FD and WD for each of the four shape signatures and concatenate all the FDs and WDs to be a final classical descriptor which is 203 dimensional.

Some positive (similar) and negative (dissimilar) shape pairs are needed as training data. In the classification scenario, since the training shapes are classified, the training data can be easily constructed by considering shapes from the same class as similar and those from different classes as dissimilar. We randomly get 100,000 positive pairs and 100,000 negative pairs in this way.

Because the components in the composite classical descriptor are heterogeneous (Fourier and wavelet coefficients), only homogeneous combinations are allowed. That is, Fourier coefficients can only be combined with Fourier coefficients. So do wavelet coefficients.

Evaluation results

We compare the descriptor generated by our algorithm to two other methods: *PCA* and *SS*. *PCA* means reducing descriptor length using PCA, and *SS* stands for a simplification of our method by removing boosting selection in each round. That is, the reserved components in line 9 of Figure 2 are simply determined by selecting the components with the lowest training error.

*k*nn is used to classify the testing shapes. Figure 4 shows average testing errors for different methods at descriptor lengths 1, 3, 5, 7, 9, 11 and 13. The testing error when using the original classical descriptor is also shown for reference. Because it only makes sense to use this classical descriptor as a whole, its error is displayed by a horizontal line. The detailed numbers are shown in Table 1. It can be seen that our method gets lower errors with smaller descriptor lengths. When the descriptor length is very small (≤5), the error of *SS* is very near to our method. But as the length increases, the difference becomes substantially larger. This indicates that the components added later by *SS* are not as discriminative as ours. Also note that all three methods get a lower error than the original classical descriptor at some descriptor lengths. This indicates that the irrelevant components in the original descriptor do impair the discriminative power of the whole descriptor.

	testing error (%) when descriptor length *m* =						
	1	3	5	7	9	11	13
Ours	66.3	38.7	23.1	17.6	15.3	14.2	13.9
PCA	75.2	44.3	26.9	19.4	17.0	15.2	14.9
SS	66.3	38.7	25.8	22.7	20.6	19.0	17.9
orig	18.8						

Table 1: Testing errors. ("*orig*" stands for the original descriptor)

3.2. Retrieval experiment – CMU Mocap database

Experiment setup

This experiment involves shape retrieval in the context of example based human pose estimation from silhouettes. Suppose we have a database of 2D silhouettes labeled by their ground-truth 3D poses. In the running stage, when a new silhouette (typically extracted from video) arrives, its 3D pose can be inferred by searching for the best match(es) in the database using some shape descriptor calculated on the silhouette [6][14]. Because human poses usually lie in a large pose space, there are typically a large number of samples in the database, and a descriptor that is compact yet accurate plays an important role.

We use CMU motion capture library to construct the database by treating the captured 3D poses as ground truth and synthesizing corresponding silhouettes. We use 2400 training poses and 600 testing poses from walking, jumping, boxing and soccer shooting motions. The training poses are performed by seven subjects, and the testing poses are from four other subjects that the algorithm does not see during training. For each pose, its global translation is discarded, and 36 silhouettes are synthesized by cycling the yaw angle by 10° intervals. Thus, our dataset contains 86,400 training silhouettes and 21,600 testing silhouettes. The silhouettes are of size 256*256. [4]

The construction of classical descriptor is the same as the previous experiment, except that 128 instead of 32 points are sampled along the shape border due to higher image resolution. The classical descriptor is thus 827 dimensional. The generating of training data is different, because this is not a classification problem and the training silhouettes are not classified. In this case, the 3D pose vector can be used as a guide deciding whether two silhouettes are similar in ground-truth. If the distance between the corresponding 3D poses is below a threshold, then the two silhouettes are considered similar. The threshold is set by inspection. We randomly select 100,000 positive pairs and 100,000 negative pairs in this way as training samples.

[4] The dataset of this experiment is available as supplementary material.

Evaluation Criteria

The testing error of a shape descriptor in this pose retrieval experiment is evaluated by the distances between the ground-truth poses and the retrieved poses. Suppose the ground-truth pose for a testing silhouette is p, and let q_i (i = 1,…,k) denote the poses of the top k hits in the database (we set k=10 in this experiment). The following two metrics are used for evaluation:

$$MHD \text{ (mean hit distance)} = \frac{1}{k}\sum_{i=1}^{k}\|p-q_i\| , \quad (9)$$

$$BHD \text{ (best hit distance)} = \min_{i=1,\dots,k}\|p-q_i\| . \quad (10)$$

$\|.\|$ is the $L2$ distance between two pose vectors.

The inclusion of *BHD* as well as *MHD* for evaluation is due to the complex and one-to-many mapping from silhouette to pose. It is quite possible that significantly different poses have very similar silhouettes. This is especially true when symmetric ambiguity occurs, and the poses with such ambiguity will be drastically different from the ground truth, increasing the *MHD* values considerably.

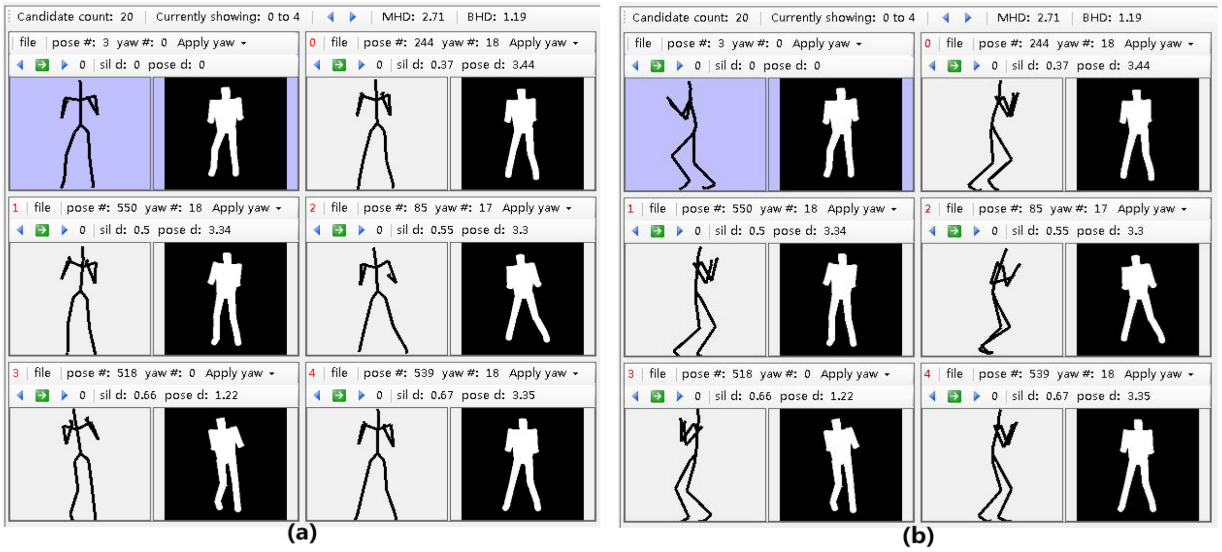
Figure 5 shows such an example. In Figure 5(a), the item shown in blue background is the query into the database (testing item), and the other five items are the top five hits in the database in sense of silhouette. For each hit, its silhouette distance and pose $L2$ distance to the query are displayed. Figure 5(b) makes the ambiguity clear by rotating all the 3D poses clockwise by 90°. It is clear that four out of the five hits suffer from symmetric ambiguity (opposite yaw angle). The pose distances for these ambiguous candidates are significantly larger, increasing *MHD* significantly. Most pose retrieval systems detect this kind of poses as outliers[5]. Therefore, we use both *BHD* and *MHD* for evaluation.

We also conduct evaluation in two different yaw constraints: *UY* (unconstrained yaw) and *CY* (constrained yaw). In *UY*, yaw angles of retrieved poses are not constrained, while in *CY* the yaw angle is constrained to have a maximum of 40° distance from the ground-truth of the previous frame. For example, if the yaw angle of the previous frame is 50°, then in *UY* all samples in the database are searched, and in *CY* we only search the samples whose yaw angles are 30°, 40°, 50°, 60° or 70°. Because human motion is continuous, many systems use yaw angle of the previous frame as an initial guess for the current frame. This helps to eliminate symmetric ambiguity substantially. Therefore, we include *CY* into evaluation.

Evaluation Results

According to the evaluation criteria, there are four groups of results: *UY+MHD*, *UY+BHD*, *CY+MHD*, and *CY+BHD*. Similar to the previous experiment, we compare

[5] While it is very difficult, if possible, to spot the outlier poses using a single silhouette, it is much easier if the information in previous frames is exploited because of motion continuity. Most systems take this approach.


(a)

Candidate count: 20 | Currently showing: 0 to 4 | MHD: 2.71 | BHD: 1.19

file | pose #: 3 yaw #: 0 Apply yaw
0 | sil d: 0 pose d: 0

0 | file | pose #: 244 yaw #: 18 Apply yaw
0 | sil d: 0.37 pose d: 3.44

1 | file | pose #: 550 yaw #: 18 Apply yaw
0 | sil d: 0.5 pose d: 3.34

2 | file | pose #: 85 yaw #: 17 Apply yaw
0 | sil d: 0.55 pose d: 3.3

3 | file | pose #: 518 yaw #: 0 Apply yaw
0 | sil d: 0.66 pose d: 1.22

4 | file | pose #: 539 yaw #: 18 Apply yaw
0 | sil d: 0.67 pose d: 3.35

(b)

Candidate count: 20 | Currently showing: 0 to 4 | MHD: 2.71 | BHD: 1.19

file | pose #: 3 yaw #: 0 Apply yaw
0 | sil d: 0 pose d: 0

0 | file | pose #: 244 yaw #: 18 Apply yaw
0 | sil d: 0.37 pose d: 3.44

1 | file | pose #: 550 yaw #: 18 Apply yaw
0 | sil d: 0.5 pose d: 3.34

2 | file | pose #: 85 yaw #: 17 Apply yaw
0 | sil d: 0.55 pose d: 3.3

3 | file | pose #: 518 yaw #: 0 Apply yaw
0 | sil d: 0.66 pose d: 1.22

4 | file | pose #: 539 yaw #: 18 Apply yaw
0 | sil d: 0.67 pose d: 3.35

Figure 5: Symmetric ambiguity in pose retrieval. (a): Top five hits in database for a query (in blue background). (b): All 3D poses are rotated clockwise by 90 degree for visualization of the ambiguity.

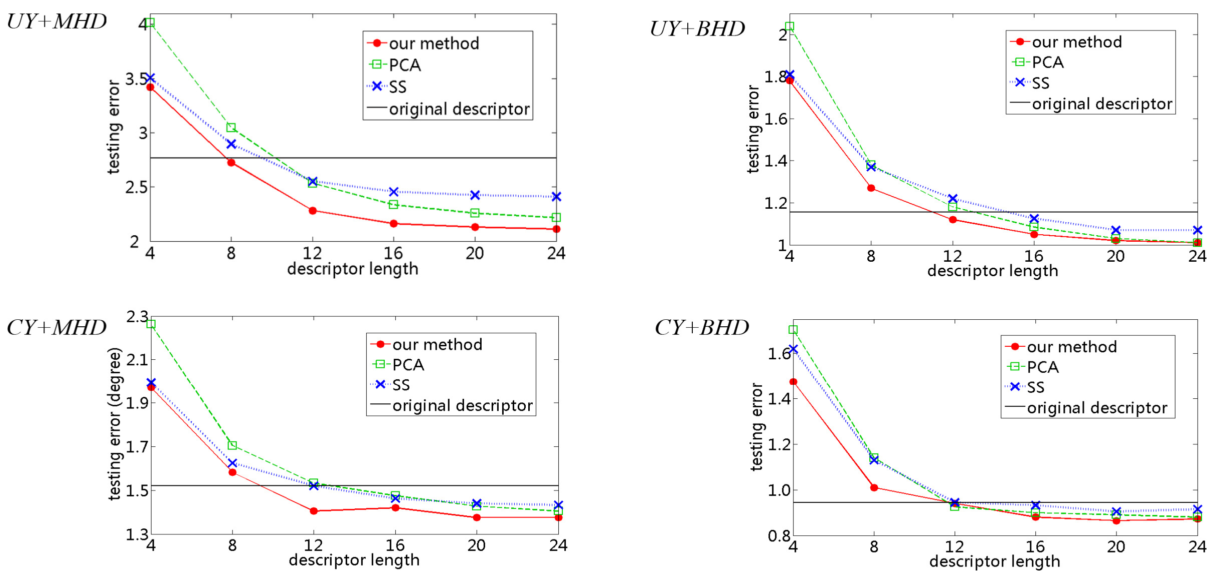


Figure 6: Evaluation results on pose retrieval.

our method to *PCA* and *SS*. The evaluation results are depicted in Figure 6, where the errors at descriptor lengths 4, 8, 12, 16, 20 and 24 are shown. The errors using the original descriptor are also displayed for reference.

In all cases, the compact descriptors of all three methods can achieve errors lower than the original descriptor at some lengths, implying that the irrelevant components in the original descriptor impairs the accuracy. When the descriptor length is small (e.g. 4~12), the errors drop quickly with increased length. As the length becomes larger, the errors tend to be more stable, and may eventually increase with the descriptor length. The compact descriptor generated by our method typically achieves a lower testing error compared with *PCA* or *SS*, especially when the descriptor length is small (≤20). As the descriptor length increases, the performance difference among the methods becomes subtler. As an extreme, at full descriptor length (827 dimensional), the performance difference vanishes and all methods involved produce virtually identical testing errors. (This is not reflected in Figure 6 because of the lack of space.)

It is also worth noticing that the errors in *CY* conditions

(*CY+MHD* and *CY+BHD*) are notably lower than in the *UY* counterparts (*UY+MHD* and *UY+BHD*). This indicates that symmetric ambiguity is indeed a very tricky problem in 3D pose estimation from silhouettes if no temporal information is exploited.

4. Conclusion

In this paper we develop a method that generates new descriptor from classical descriptors. The inspiration is that many components in the classical descriptors are irrelevant to the specific problem. By discarding "bad" components and combining "good" ones, we can simultaneously reduce the descriptor length and increase the discriminative power. We generate favorable combinations from the components of classical descriptors in a progressive way, where a variation of Adaboost is employed in each round to ensure that the selected combinations have optimal discriminating power when used jointly. Experiments show that the generated descriptor can achieve a better accuracy with a reduced feature length.

We emphasize again that our algorithm is not inherently a classification method. We don't assume the shapes belong to different classes. The training data is composed of binary classified shape pairs instead of classified shapes, where the pairs are classified to feed the algorithm with the information of the specific problem by examples. The output of our method is a new, adaptive and compact descriptor that can be used in many applications including classification, retrieval, clustering and so on.

In this paper we conduct experiments using Fourier and wavelet descriptor as classical descriptors. Our method is not inherently constrained to any specific classical descriptor types. In the future we'd like to test on more kinds of classical descriptors. Depending on particular descriptor properties, combination models other than the linear one might be more preferable. Also, we plan to study more closely on the issue of overfiting.

Acknowledgement

This work is supported by National Natural Science Foundation of China (No.60525108, No.60533090), Program for Changjiang Scholars and Innovative Research Team in University (IRT0652), 973 Program (No. 2002CB312101), Science and Technology Project of Zhejiang Province (2006C13097, 2005C13032).

References

[1] S. Belongie, J. Malik, and J. Puzicha. Shape matching and object recognition using shape contexts. *PAMI*, 24(24): 509-522, 2002.

[2] A. F. Bobick and J. W. Davis. The recognition of human movement using temporal templates. *PAMI*, 23(3):257. 267, 2001.

[3] G. C. -H. Chuang and C. -C. Jay Kuo. Wavelet descriptor of planar curves: theory and applications. *IEEE Transactions on Image Processing*, 5(1): 56-70, 1996.

[4] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119-139, 1997.

[5] N. H. Getz. A Fast Discrete Periodic Wavelet Transform. Memorandum UCB/ERL M92-138, Electronic Research Laboratory Berkeley, California, 22 December 1992.

[6] N. Howe. Silhouette lookup for monocular 3D pose tracking. *Image and Vision Computing*, 25(3): 331-341, 2007.

[7] M. K. Hu. Visual pattern recognition by moment invariants. *IEEE Transactions on Information Theory*, 8: 179–187, 1962.

[8] H. Kauppinen, T. Seppanen, and M Pietikainen. An Experimental Comparison of Autoregressive and Fourier-Based Descriptors in 2D Shape Classification. *PAMI*, 17(2): 201-207, 1995.

[9] A. Khotanzan and Y. H. Hong. Invariant image recognition by zernike moments. *PAMI*, 12(5):489–497, 1990.

[10] R. D. Leon and L. E. Sucar. Human silhouette recognition with Fourier descriptors. In *ICPR*, 2000.

[11] S. Loncaric. A survey of shape analysis techniques. *Pattern Recognition*, 31(8): 983-1001, 1998.

[12] A. Opelt, A. Pinz, and A. Zisserman. A boundary-fragment-model for object detection. In *ECCV*, 2006.

[13] M. A. Ranzato, F. J. Huang, Y-L. Boureau, and Y. LeCun: Unsupervised Learning of Invariant Feature Hierarchies with Applications to Object Recognition. In *CVPR*, 2007.

[14] L. Ren, G. Sharknarovich, J. Hodgins, H. Pfister, and P. Viola. Learning Silhouette Features for Control of Human Motion. *ACM Transactions on Graphics*, 24(4): 1303-1331, 2005.

[15] R. E. Schapire. The strength of weak learnability. *Machine Learning*, 5 (2), 197–227, 1990.

[16] G. Shakhnarovich, P. Viola, and T. Darrell. Fast pose estimation with parameter sensitive hashing. In *ICCV*, 2003.

[17] O. R. Terrades, S. Tabbone, and E. Valveny. Combination of shape descriptors using an adaptation of boosting. In *ICPR*, 2006.

[18] X. Yin, C. Liu, and Z. Han. Feature combination using boosting. *Pattern Recognition Letters*, 26(14): 2195-2205, 2005.

[19] C.T. Zahn and R.Z. Roskies. Fourier descriptors for plane closed curves. *IEEE Trans. Computers*, 21(3): 269-281, 1972.

[20] D. Zhang and G. Lu. Review of shape representation and description techniques. *Pattern Recognition*, 37(1):1-19, 2004.

[21] D. Zhang and G. Lu. A Comparative Study on Shape Retrieval Using Fourier Descriptors with Different Shape Signatures. In Proc. International Conference on Multimedia and Distance Education, pp. 1-9, 2001.