Is Rotation a Nuisance in Shape Recognition?

Qiuhong Ke[1,2] Yi Li[2,3]

[1]Beijing Forestry University, [2]NICTA, [3]ANU
qiuhong.ke@nicta.com.au, yi.li@cecs.anu.edu.au

Abstract

Rotation in closed contour recognition is a puzzling nuisance in most algorithms. In this paper we address three fundamental issues brought by rotation in shapes: 1) is alignment among shapes necessary? If the answer is "no", 2) how to exploit information in different rotations? and 3) how to use rotation unaware local features for rotation aware shape recognition?

We argue that the origin of these issues is the use of hand crafted rotation-unfriendly features and measurements. Therefore our goal is to learn a set of hierarchical features that describe all rotated versions of a shape as a class, with the capability of distinguishing different such classes. We propose to rotate shapes as many times as possible as training samples, and learn the hierarchical feature representation by effectively adopting a convolutional neural network. We further show that our method is very efficient because the network responses of all possible shifted versions of the same shape can be computed effectively by re-using information in the overlapping areas. We tested the algorithm on three real datasets: Swedish Leaves dataset, ETH-80 Shape, and a subset of the recently collected Leafsnap dataset. Our approach used the curvature scale space and outperformed the state of the art.

1. Background

Matching closed contour shapes is an important problem in computer vision. Dated back to the early age, where applications are anthropology and biometric [15], it has novel applications in domains as diverse as computer graphics and mobile computing [9].

Typical issues in shape recognition include noise, scale, translation, and rotation. Most of them can be handled easily either by carefully chosen classifiers, different data representations, and simple normalizations. However, regardless various techniques are used, rotation invariance in shape recognition is particularly difficult to achieve [6]. Take the widely used "landmark point" representation for example, where a shape is uniformly sampled and represented by a series of high dimensional feature vectors. Rotation amounts to shifting these vectors in a circular order, which makes the comparison of two shifted series difficult, because this mis-correspondence phenomenon is not handled inherently by nearest neighbor methods and classifiers.

Documented solutions of rotation invariance include: 1) define domain specific global rotation angle (*a.k.a.* "major axis"), 2) adopt rotation invariant representation of local descriptors (*e.g.,* histogram), and 3) brute force search. Global rotation angle may be useful for the limited domains, however it is also believed that the major axis is not reliable [16]. Many solutions frame the shape matching problem as a histogram comparison problem [12]. Unfortunately, useful information such as structure has to be discarded during histogram computation in order to achieve rotation invariance in this process. Trying all possible rotations appears to be the last resort, but this is at the expense of efficiency [8].

With all the attempts to achieve rotation invariance, it is interesting to note that rotation invariance may even *not* be favorable in some domains. For example, rotation invariance techniques cannot differentiate between the shapes of the lowercase letters "d" and "p", since this pair of shapes only differs in their orientations. Therefore, "rotation limit" or "rotation awareness" may be more practical to improve recognition accuracy in these applications.

All these difficulties attest that rotation is a nuisance in shape recognition. Researchers frequently need to make series decisions of the following three puzzling questions: 1) shall we rotate shapes? 2) shall we use histogram? and 3) shall we use rotation invariant local feature? The answers appear elusive, but first of all, why we have these questions?

The origin that rotation becomes a bitter and resentful enemy is that we hand craft *orientation-unfriendly* representations and measurements. For instance, representing 2D circular points by 1D vectors and feeding them to Support Vector Machine already abandon any hope of handling the rotation, because kernels are hardly shift invariant. Even the widely used dynamic time warping is not rotation-friendly, because it neither encodes *global* mea-

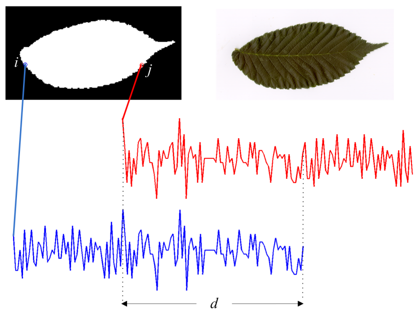

Figure 1. Two time series representations of the same shape, starting from two different contour points.

surement that describes *all* possible rotations between two shapes, nor exploits the rotation property.

Instead of defining the representations manually, can we learn a set of features for the same shape at all possible rotations and use them for distinguish different shapes at different rotations? In this paper we argue that we can tackle this challenging concerns drastically in *one shot*. We consider shapes along boundaries as 1D time series and use convolutional neural network [3] as a practical solver, because filtering is particularly suitable for time series analysis.

In this paper we further suggest that adding more rotated versions of shapes improves shape recognition by nature. In this case, the training will be time consuming with the brute force solution. While in contrast, we significantly improve the speed without losing accuracy. First, we take a slow but intuitive approach to explain the problem, and then we describe how we make the training and testing more efficient.

Imagine we manually circular shift the time series and collect all shifted versions as training samples, this naive implementation is unsatisfactory for many scenarios. For example, it takes 200 times to train a network if a shape is sampled at 200 points.

We demonstrate that we can reduce the complexity drastically. The key ingredient is to reuse the redundant information between two shifted versions (Fig. 1). In the case where average pooling is used, this information reusing strategy forms a balanced binary tree. Thus, all responses of the same shape may be computed effectively. This also allows us to perform the "rotation limit" cases.

We adopt curvature scale space (CSS) as our local descriptor [13]. The curvature is a good representation of the shape, but numerical method of curvature calculation is sensitive to noise. Therefore, we applied integral measurements proposed by [9] as the curvature of contours.

Our contributions include

- We introduced an efficient algorithm for shape recognition, based on the recent progress of deep learning.
- The algorithm naturally handles rotation invariance and rotation limit cases.
- We outperformed state of the art methods in real datasets.

2. Related work

Shape is one of the important features in computer vision. Particularly we focus on the closed contour, where landmark points are obtained by uniformly sampling the contours. We review three components in shape recognition: features, distance measurements, and invariance.

Features Local descriptors can be rotation dependent or independent. In the early age, a large number of papers extract rotation invariant features such as curvature, entropy, and convex hull [5]. These rotation invariant features are less distinctive in literature.

Alternatively, Shape Context [2] and Inner Distance Shape Context [12] are two widely used descriptors that depend on rotation. In shape recognition, they are usually used in dynamic programing framework. Please note that such descriptors can be modified to be rotation independent by computing the "local" orientation, but it is not a common practice.

Distance There are many distance measures for shapes. When two point sets are used, Chamfer distance [4] is frequently used in the 2D space measures.

Experiments suggest that 1D representations (time series) achieve superior accuracy. For time series, typical distance measurements include l_2 norm and Dynamic Time Warping [12]. Typically there are implicit assumption to define starting point [4], thus one may need to check all possible circular shifts to find the optimal matching.

To avoid time consuming distance comparison, alignment or brute force search may be used to produce the best accuracy in different domains.

Rotation invariance With the elusive starting point, rotation invariance seems to be always hard to handle in the literature. Major axis appears to be the dominant choice in some areas, but even in Anthropology, a classical area of shape matching, [8] argued that the major axis is not useful. The points can be manually chosen for training samples, but it can be labor intensive for large dataset [7].

Histogram is an efficient approach to achieve rotation invariance. Kumar *et al.* [9] concatenated the histogram of the curvatures across multiple scales and achieved impressive results on a large dataset. The drawback is that structural information may be lost during this histogram computation.

While rotating all shapes is necessarily in some applications, it may be useful to express rotation-limited queries. For example, allowing a maximum rotation in a shape or shift in landmark points.

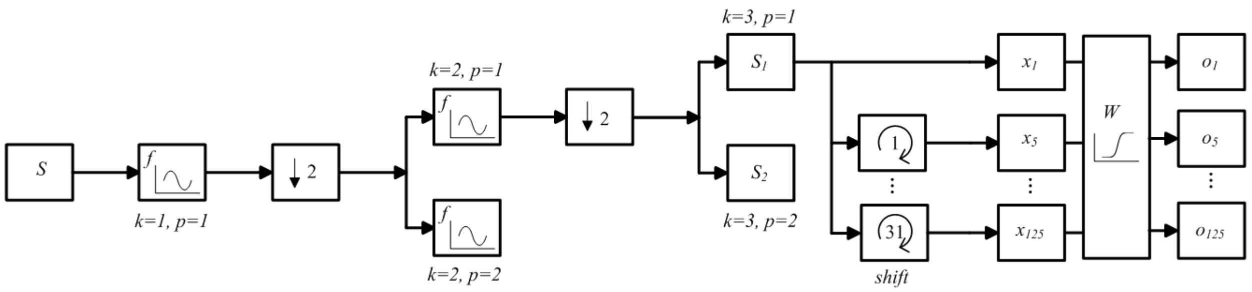

Figure 2. Shift-Delay Forward Operation illustrated.

3. Learning rotation robust features for shape recognition

We first briefly introduce basic concepts in the Convolutional Neural Network (CNN), and then present our efficient implementation of handling all possible shifts of shapes.

3.1. Convolutional Neural Network

Convolutional Neural Network (CNN) is a multilayer learning framework, which may consist of an input layer, a few convolutional layers and an output layer for logistic regression. The goal of CNN is to learn a hierarchy of feature representations. Signals in each layer are convolved with a number of filters and further downsampled by pooling operations, which aggregate values in a small region by functions including max, min, and average. In our work, we consider average pooling for simplicity. The learning of CNN is based on Stochastic Gradient Descent (SGD), which includes two main operations: Forward and Back-Propagation. Please refer to [10] for details.

We aim at learning representations for rotated shapes. A naive implementation is to manually rotate each signal, one at a time, to generate rotated versions at different starting contour points. However, this is time consuming, and as we see in this section, a lot of computation is redundant. In this paper we show how to reuse the information and our modification of the basic algorithm to speed up the computation.

3.2. Re-using information in Forward operation

Given a CNN, our goal is to generate the outputs for the shifted signals starting from possible contour points of the same shape in one shot.

Considering two time series from different starting point i and j of the same shape (Fig. 1), the Forward operation amounts to convolve the times series with CNN filters and average the outputs in each layer of the convolutional neural network. Obviously, the information in the overlapping area needs not to be computed twice.

Therefore, the idea of our efficient algorithm is to re-use information. This dramatically speeds up the computation, which reduces both the time for convolution and for shifting signals. In the following derivation, we ignore the bias term, but it can be added using the same idea.

Shift-delayed Forward Operation

Let us consider 1D time series representation for shapes. We first show a slow and naive approach, and then greatly speed it up. The higher dimensional time series can be handled using the same principle.

In order to generate all possible shifts of a time series, we have to shift one curve n times in a circular order, where n is the number of possible start points. Then, each shifted version must be convolved with many filters and downsampled. Considering $n = 200$ or 400 being the common choices, this brute force approach can be very slow.

Instead, we propose to postpone the shifting operation until necessary. Fig. 2 illustrates our idea using a signal processing diagram. In the downsampling process, we divide the signal into 2 groups, which contains points in odd and even locations. Notice that shifting the signal at the beginning of the system is equivalent to shifting its corresponding downsampled versions *properly* at any stage, we can perform shifting before the single layer perception.

Assuming we have K layers CNN and a signal x^0. The first layer is the input map, and the last layer is the single layer perception. Each layer has N_k ($k = 1, 2, ..., K$) output maps. For simplicity, we define the output map of the first layer is the same as input (e.g., $N_1 = 1$), and the filters are denoted by f_{ij}^k ($i = N_1, ..., N_{K-2}$, and $j = N_2, ..., N_{K-1}$). Denote the output of the 1st layer as column vector $x_1^1 = x^0$, convolution and average pooling from the second layer are

$$\bar{x}_j^k = \text{sigm}\left(\sum_{i=1}^{N_k} \text{conv}(x_i^k, f_{ij}^k)\right) \tag{1}$$

$$u_j^k = \text{conv}(\bar{x}_j^k, \left[\frac{1}{2}\ \frac{1}{2}\right]^T) \tag{2}$$

where sigm$(\cdot)$ denotes the sigmoid function. Since each downsampling will produce two subbands (Fig. 2), we add a subscript $x_{j,p}^k$ denotes the values of p^{th} subband at j^{th}

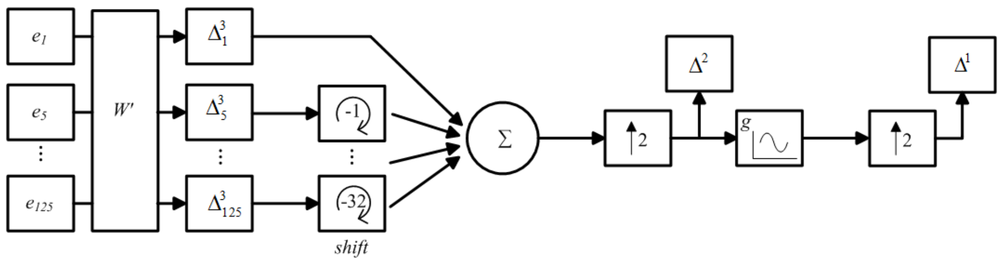

Figure 3. Average-Advanced BackPropagation illustrated.

output map at the k^{th} layer. Initiate $x_{1,1}^1 = x^0$, we downsample the signal by two from the second layer, which gives two subbands of output as

$$\begin{cases} x_{j,2p-1}^{k+1}(l) = u_{j,p}^k(2l-1) \\ x_{j,2p}^{k+1}(l) = u_{j,p}^k(2l) \end{cases} \tag{3}$$

for $l = 1, ... \frac{|\bar{x}_{j,p}^k|}{2}$, where $|\cdot|$ is the length of the signal. In each subband of the last layer, we only need to circular shift the elements to obtain the correct response for the single layer perception. Signal starting at r^{th} contour point can be obtained by shifting the original signal r times. If we know r corresponds to the q^{th} element of subband p at a layer, it is equivalent to take the subband p and shift it $q-1$ times. This can be done by index operation easily.

Define $s_k^r = \lfloor \frac{\bmod (r+2^{k-1}-1, 2^{k-1})}{2^{k-2}} \rfloor$ and $q_k^r = \lfloor \frac{r+2^{k-1}-1}{2^{k-1}} \rfloor$, where $\lfloor \cdot \rfloor$ denotes the floor operation and $\bmod(\cdot)$ is the modular operation. s_k is a binary variable, so the subband index of r at level k is $p_k^r = (s_1^r..s_k^r)_{10} + 1$, where $(\cdot)_{10}$ denotes the binary to decimal conversion. p_k^r and q_k^r define the location of subband at any layer for a given starting point, thus the output is

$$f_v = \left[\text{cshift}_{q_{K-1}}(x_{1,p_{K-1}}^{K-1}); ...; \text{cshift}_{q_{K-1}}(x_{N_{K-1},p_{K-1}}^{K-1})\right] \tag{4}$$

$$o_r = \text{sigm } (W f_v) \tag{5}$$

where $\text{cshift}_q(\cdot)$ denotes the shift of the input $q-1$ units.

For a shape, there will be a set of o_r, each of which corresponds to the different starting points of the same sample. As a result, it generates the output of all rotation versions of a shape in one shot, which reduces the need to align shapes when conducting classification. The feature is divided into odd and even groups in each layer, which forms a balanced binary tree. Assuming the temporal cost of each layer in the network is c_k, the total cost of generating responses of all samples is $n \sum_{k=1}^{K} c_k$ for naive implementation. Instead, ours is $\sum_{k=1}^{K} 2^{k-1} c_k$, which is a significant reduction of the complexity. The complete algorithm is shown in Algo. 1.

Algorithm 1 Forward Operation in Circular Time Series

```
 1: procedure FastForward($x^0$)
 2:     $x_1^1 = x^0$; $id_1^1 = 1$;
 3:     for $k = 2$ to $K-1$ do
 4:         for $p = 1$ to $2^{k-2}$ do
 5:             for $j=1$ to $N_k$ do
 6:                 $\bar{x}_{j,p}^k = 0$;
 7:                 for $i=1$ to $N_{k-1}$ do
 8:                     $\bar{x}_{j,p}^k = \bar{x}_{j,p}^k$+conv($x_{j,p}^{k-1}$,$f_{i,j}^{k-1}$, 'circ');
 9:                 $\bar{x}_{j,p}^k$=sigm($\bar{x}_{j,p}^k$);                ▷ Sigmoid
10:                                                          ▷ Downsample
11:                 $u$ = conv($\bar{x}_{j,p}^k$,[$\frac{1}{2}$ $\frac{1}{2}$],'circ');
12:                 $x_{j,2p-1}^{k+1} = u(1:2:\text{end})$;
13:                 $x_{j,2p}^{k+1} = u(2:2:\text{end})$;
14:                 $id_{2p-1}^{k+1} = id_p^k$;
15:                 $id_{2p}^{k+1} = id_p^k + 2^{k-1}$;
16:     for $p = 1$ to $2^{K-2}$ do
17:         for $q = 1$ to length($x_0$)$/2^{K-2}$ do
18:             $x_p^{K-1}$=[cshift($x_{1,p}^{K-1}$, $q$); ...;cshift($x_{N_{K-1},p}^{K-1}$, $q$)];
19:             $r = id_p^K + q2^{K-2}$;
20:             $o_r$ = sigm($W x_p^{K-1}$);
```

3.3. Average in BP

Backpropagation is a time consuming operation. Every output from the Forward procedure needs to be compared to the ground truth, and the errors are propagated layer by layer. Since this error depends on each starting point, we cannot reuse the information in a similar way as we used the Forward operation.

First, we present our naive but slow implementation of BackPropagation. Then, we significantly speed it up by advancing the average operation in gradient calculation.

For the time series starting at the r^{th} contour point, we follow the standard procedure to compute error and the gradient of sigmoid function.

$$e_r = y_r - o_r \tag{6}$$

$$\Delta_{o_r} = e_r \circ o_r \circ (1 - o_r) \tag{7}$$

where y_r is the true label, o_r is the prediction output, and $\circ$ is the Hadamard product. Then, the error is back propagate to the convolutional layers as follows

$$\Delta_r^{K-1} = W^T \Delta_{o_r}. \tag{8}$$

Let $\Delta_r^{K-1} = [\Delta_{1,p_{K-1}}^{K-1}; ...; \Delta_{N_{K-1},p_{K-1}}^{K-1}]$, where p_{K-1} is the same as defined in Forward operation, $\Delta_{j,p_{K-1}}^{K-1}$ has the same size as $x_{j,p_{K-1}}^{K-1}$, we back propagate the gradient of the sigmoid function in each layer as follows. First, we upsample the signal after proper shifting:

$$v_{j,p_k}^k = \text{linspace}_2(\Delta_{j,p_{k+1}}^{k+1}) \tag{9}$$

where $\text{linspace}_2(\cdot)$ denote the linear upsampling by 2 operation. v_{j,p_k}^k is backpropagated as follows:

$$\bar{\Delta}_{j,p_k}^k = \text{cshift}_{q_k}(\bar{x}_{j,p_k}^k) \circ \left(1 - \text{cshift}_{q_k}\left(\bar{x}_{j,p_k}^k\right)\right) \circ v_{j,p_k}^k \tag{10}$$

$$\Delta_{i,p_k}^k = \sum_{j=1}^{N_{k-1}} \text{conv}(\bar{\Delta}_{j,p_k}^k, \bar{f}_{ij}^k) \tag{11}$$

where $\bar{f}_{ij}^k$ is a flipped version of f_{ij}^k. Since SDG operates in a batch mode, in every BackPropagation procedure we have M time series. Denote δ_{i,p_k}^k as the flipped version of $\bar{\Delta}_{i,p_k}^k$, calculating gradient of f_{ij}^k is simply the "valid" region of

$$\Delta f_{ij}^k = \frac{1}{M}\sum_M \text{conv}\left(\delta_{j,p_k}^k, \text{cshift}_{q_k}\left(x_{i,p_k}^k\right)\right) \tag{12}$$

Average-advanced BackPropagation

To speed up the back propagation, we first consider the mean gradient in a batch.

$$\Delta f_i^k = \frac{1}{M}\sum_M \text{conv}\left(\text{cshift}_{-q_k}(\delta_{j,p_k}^k), x_{i,p_k}^k\right) \tag{13}$$

$$= \text{conv}\left(\frac{1}{M}\sum_M \text{cshift}_{-q_k}(\delta_{j,p_k}^k), x_{i,p_k}^k\right) \tag{14}$$

This means the average of the filter gradient is equivalent to the filter gradient of the average input. Thus, we can average the results in each subband by properly shifting the output error. Note that Eq. 9-11 linear, it is possible to shift the gradient $\Delta_{j,p}^k$ in a reverse order, instead of shifting the $x_{j,p}^k$ every time. Therefore, all the shifted errors from the same subband share the same set of $x_{j,p}^k$. As a result, we advance the average the above mean operation at the early as possible in the BP (Fig. 3). In this process, the only thing needs to be changed is the addition of the following shift in Eq. 9:

$$v_{j,p_k}^k = \text{cshift}_{\text{mod}(p_k+1,2)}\left(\text{linspace}_2(\Delta_{j,p_{k+1}}^{k+1})\right) \tag{15}$$

This shift is because of the alignment for odd and even indexes during upsampling. Advancing the average operation is very useful, because we can treat the averaged results as new errors. This results in speeding up the BackPropagation significantly. We show the procedure in Algo. 2.

Algorithm 2 BackPropagation for Circular Time Series

```
1: procedure FastBackward($o_r, y_r, r = 1, ..., |x^0|$)
2:     for $p$ = 1 to $2^{K-2}$ do
3:         $d_p = 0$;
4:     for $r$ = 1 to $|x^0|$ do
5:         $\Delta_r = (o_r - y_r) \circ o_r \circ (1 - o_r)$;
6:         Compute $p_{K-1}$ and $q_{K-1}$ for $r$;
7:         $d_{p_{K-1}} = d_{p_{K-1}}$+cshift$(o_r, q_{K-1})$;
8:     for $p$ = 1 to $2^{K-2}$ do
9:         $d_p = \text{mean}(d_p)$;
10:    Backpropagate $d_p$ using Eq. 8,10,11,15.
11:                                         ▷ Calculate Gradient
12:    for $k$ = 1 to $K - 2$ do
13:        for $j$ = 1 to $N_k$ do
14:            for $i$ = 1 to $N_k - 1$ do
15:                $\Delta f = 0$;
16:                for $p$ = 1 to $2^{k-2}$ do
17:                    $\Delta f = \Delta f + \text{conv}\left(\delta_{j,p_k}^k, \text{cshift}_{q_k}\left(x_{i,p_k}^k\right)\right)$
                   $\Delta f_{ij}^k = \text{mean}\Delta f_{ij}^k$;
```

3.4. Rotation limit shape recognition

As discussed in the introduction, some applications may require rotation limited shape recognition. In our formulation, this is equivalent to selecting subsets of outputs in Forward and only computes the gradients in BackPropagation. To achieve this, one only needs to compute the average of the selected samples in each subband of the final layer, and the remaining procedure is the same.

This could be used to further reduce the computational cost. For instance, in some applications one wants to compute rotated versions of every two contour points. This can be done by selecting proper indexes as well. Another handy solution is to simply take the samples in the first subband. In our experiments, this practice speeds up the learning without sacrificing much accuracy.

3.5. Integral measures for curvature scale space

Curvature is a fundamental property of natural shapes. Curvature scale space (CSS) has been proposed in the early age of computer vision. The theory of CSS is very attractive for continuous signals; however, when dealing with discrete pixelated images and imperfect segmentation results, the computation of curvature across different scales becomes

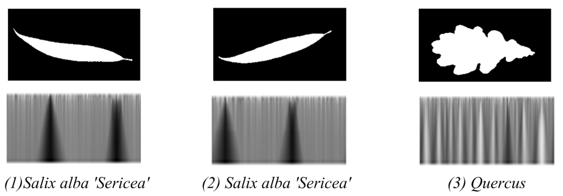

Figure 4. Curvature images. The first two are from the same species, but due to rotation they starts from different contour points.

very difficult. For example, standard differential techniques may be very sensitive to noise, which makes the CSS less effective in shape recognition.

Instead of computing curvature by the definition, a number of literature suggest to use integral measures to compute functions of the curvature [13]. The insight is that the curvature fundamentally reflects whether a curve is convex or concave, thus we can use other measurements to serve as a representation of curvature.

Following [9], we use the area of intersection of a disk centered at a landmark point and the inside of the silhouette as the measurement. Using filtering, this integral measure is fast, and robust to noise. With different radius of circles, we can derive the curvature of different scales and generate curvature images. Fig. 4 shows three examples. Each row is the curvature evolving along the boundary for each scale, and each column represent the change of curvature over scale for every contour points.

4. Experiments

4.1. Settings

In this section we show our experiments on three real datasets, namely, the Swedish Leaves [14], the ETH-80 Shape [11], and a subset of the recent Leafsnap dataset [9].

The shapes in each dataset were normalized with respect to the sizes of their convex hulls. Each shape was then represented by 200 uniformly sampled landmarks along boundary (clockwise). At each landmark, we computed the curvature scale space image with 25 scales.

The curvature image represents real datasets very well, but it is not a practical measurement for the artificial dataset such as the MPEG-7 dataset. This artificial dataset contains man-made and toy variations, such as inner holes and elongated branches. Thus integral measurements do not work properly on these artificial variations.

In all the experiments, we used four layers CNN (including the input layer and the single layer perception for regression output). The first convolution layer has 11 filters with the size 13×11, and the second convolution layer has 13 filters with size 13×9, respectively. Downsampling only applies to the curvature dimension.

We perform our method on three datasets, which are not aligned. The experiments were run on a standard PC with 3GHz Quad Core CPU and 32GB memory. The training time varies and the testing time is on average 0.0227 second per shape using Matlab.

Algo	IDSC	T-Dist	K-SVM	SM	Ours
%	91.20	93.8	91.47	94.40	**94.93/95.33**

Table 1. Performance comparison on Swedish leaf dataset. Please see [1] and [7] for the other four algorithms.

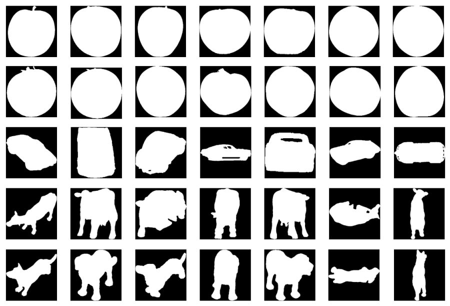
Figure 6. Examples for ETH-80 [11].

4.2. Swedish Leaves dataset

The Swedish Leaves dataset was developed at Linkoping University and the Swedish Museum of Natural History [14], which contains segmented leaves from 15 different Swedish tree species (75 leaves per species). We used 25 leaves per class for training and 75 for testing, which is a standard configuration of this dataset.

Table 1 shows our results. Particularly, we showed two configurations of our algorithms: 1) use all the contour points as starting points for training, and 2) only use the starting points that correspond to the first subband of the last convolutional layer (*i.e.*, 25% of the contour points).

Compared to the state of the art Shape Manifold [7], where starting points of training images were manually labeled and testing images were aligned with the training images, our performance using 25% contour points is already better. When more shifts (rotations) are used, the performance increases too. This suggests our method requires less labor in pre-processing shapes and supports our claim that rotations facilitate shape recognition.

4.3. ETH-80 dataset

We next demonstrate the benefits of our approach on shape categorization. The ETH-80 dataset [11] consists of 8 classes with 10 objects in each class. Each object has 41 images from different viewing angles. The contour images are used for many shape recognition algorithms (Fig. 6).

In existing matching approaches, this dataset is used in the Leave-One-Out-Cross-Validation (LOOCV) setup. Jayasumana *et al.* [7] proposed to use Leave-One-*Object*-

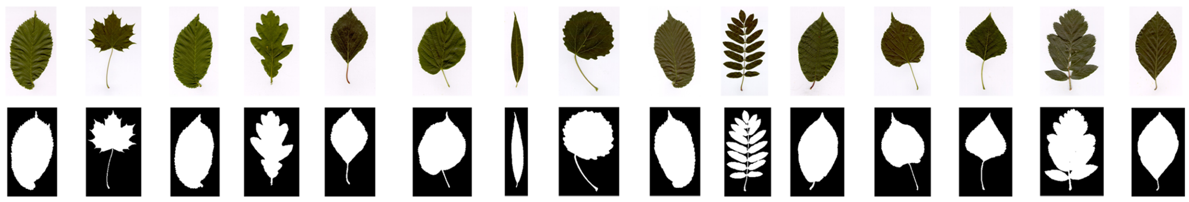
Figure 5. Examples of the Swedish leaf dataset ([14]). Each column shows two original images and their silhouettes for each class. The landmark points are then sampled from their boundaries.

Algo	Nearest Mean	Nearest Neighbor	Complex Bingham	Complex Normal	RIK	Trans. K-SVM	Shape Manifold	Ours
%	79.02	82.10	86.95	87.50	92.29	87.29	93.75	**95.80**

Table 2. Performance comparison on ETH-80 dataset. Please see [1] and [7] for the descriptions of other algorithms.

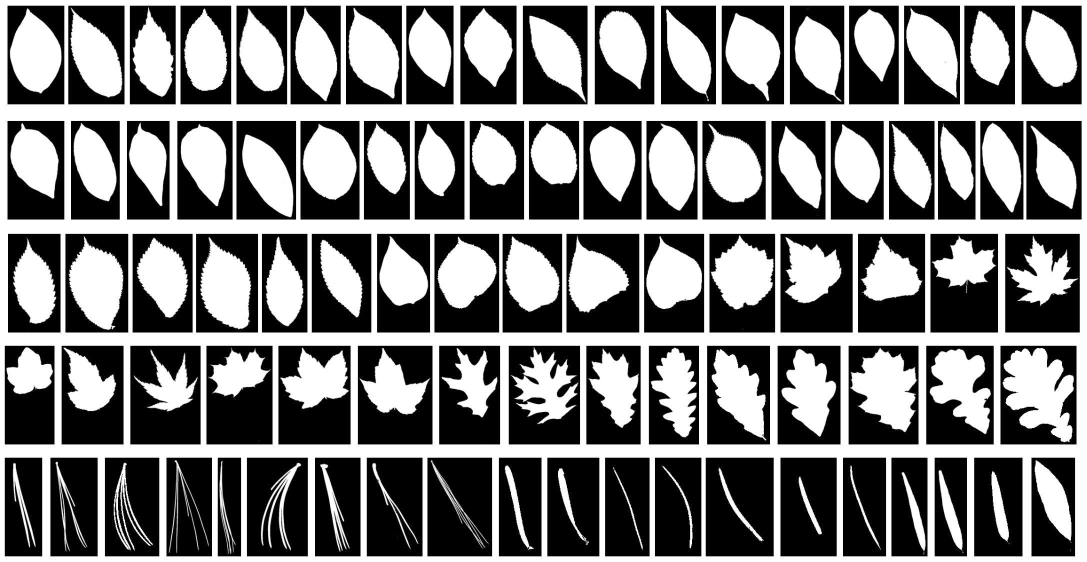
Figure 7. Examples of Leafsnap dataset. Each shape is from a different class, and some of them are very similar to each other.

Out-Cross-Validation for classification tasks: all images from one object are used as test images, and the images of other objects are used to train classifiers. If a test image is classified to the *same class* as the training images (*i.e.*, one of the remaining 9 objects in the same class for training), it is a correct classification.

In this setup, we reserved a validation set consisting of 10 images per object that are in the same class as the test object, and stop the training when the CNN converges. This process is repeated 80 times, and the average accuracy is reported in Table 3.

We compared the results of our approach to those reported in [7]. Our result outperformed the state of the art by 2%. This may be explained by the misaligned starting point issue, because there is no simple domain-specific approaches to label the starting points easily in ETH-80 dataset.

4.4. Leafsnap dataset (subset)

Finally, we verified our ideas on a subset of the largest reported shape dataset. Kumar *et al.* [9] collected the Leafsnap dataset. In total, this dataset contains 184 species in the northeastern United States, which consists of 23,915 "clean" lab images (by high quality camera) and 5,192 field images taken by mobile devices. The field images may contain varying amounts of blur and were taken with different viewpoints and illumination. The silhouettes were extracted and provided in the dataset.

Recently, the authors prepared a subset of this large dataset for academic purpose. This leaf subset includes over 5000 leaf images of the 143 trees (Fig. 7). Please note that while there may be a well defined starting point for most leaves (*e.g.,* apex), it is labor intensive to label all of them in this subset, and this labeling task may also be less practi-

Algorithm	75%	50%	25%	15%
Ours	**60.03**	**58.50**	**55.61**	48.12
HoCS	57.80	57.30	54.01	**48.55**
IDSC	54.14	56.41	46.09	42.66

Table 3. 1-NN results for Leafsnap dataset (subset).

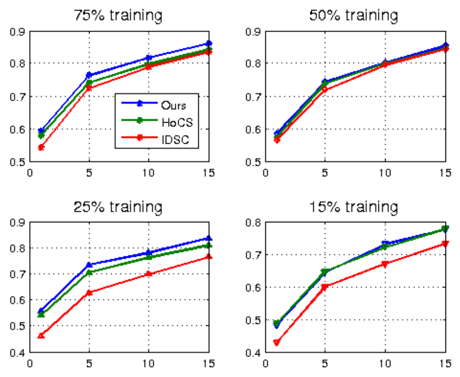

Figure 8. Accuracy comparison on Leafsnap dataset (subset).

cal in the original dataset.

Kumar *et al.* [9] proposed to use the histogram of curvature (HoCS) in a LOOCV matching framework. This is not applicable to our classification method. Therefore, we turned to a repeated random sub-sampling validation procedure. We divided the dataset and used 15%, 25%, 50%, 75% for training, respectively. We also reserved 5% for validation, and the rest for testing in each case. We repeated this procedure and reported the average values.

We compared our method to the IDSC and the HoCS on the Top 1, 5, 10, 15, 20 accuracies, respectively. Table. 3 shows the Top 1 accuracy. Our method outperformed the state of the art methods, and is inferior only when training samples are small. Note that 15% of the dataset contains only 700 images, the training may be less effective.

Fig. 8 further shows the top 5, 10, 15, 20 results, respectively. Our method consistently achieved higher accuracy for most of the test cases, and tied with the HoCS when training samples are small.

5. Conclusion

In this paper we propose a method to handle rotation in shape recognition. We effectively adopted a deep learning framework on the curvature scale space, which computes the forward and backpropation operations efficiently. We tested the algorithm on three real datasets, and the performance on these challenging dataset concludes that modeling rotation-friendly features facilitates shape recognition.

Acknowledgement: we would like to thank Prof. David Jacobs and Arijit Biswas from the University of Maryland at College Park for generously providing the Leafsnap dataset.

References

[1] X. Bai, X. Yang, L. J. Latecki, W. Liu, and Z. Tu. Learning context-sensitive shape similarity by graph transduction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(5):861–874, 2010.

[2] S. Belongie, J. Malik, and J. Puzicha. Shape matching and object recognition using shape contexts. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(4):509–522, Apr. 2002.

[3] Y. Bengio, A. C. Courville, and P. Vincent. Unsupervised feature learning and deep learning: A review and new perspectives. *CoRR*, abs/1206.5538, 2012.

[4] G. Borgefors. Hierarchical chamfer matching: A parametric edge matching algorithm. *IEEE Trans. Pattern Anal. Mach. Intell.*, 10(6):849–865, Nov. 1988.

[5] A. Cardone, R. K. Gupta, and M. Karnik. A survey of shape similarity assessment algorithms for product design and manufacturing applications. *Journal of Computing and Information Science in Engineering*, 3:109–118, 2003.

[6] Y. Gdalyahu and D. Weinshall. 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.

[7] S. Jayasumana, M. Salzmann, H. Li, and M. Harandi. A framework for shape analysis via hilbert space embedding. In *ICCV*, 2013.

[8] E. Keogh, L. Wei, X. Xi, M. Vlachos, S.-H. Lee, and P. Protopapas. Supporting exact indexing of arbitrarily rotated shapes and periodic time series under euclidean and warping distance measures. *The VLDB Journal*, 18(3):611–630, June 2009.

[9] N. Kumar, P. N. Belhumeur, A. Biswas, D. W. Jacobs, W. J. Kress, I. Lopez, and J. V. B. Soares. Leafsnap: A computer vision system for automatic plant species identification. In *The 12th European Conference on Computer Vision (ECCV)*, October 2012.

[10] Y. LeCun and Y. Bengio. The handbook of brain theory and neural networks. chapter Convolutional networks for images, speech, and time series, pages 255–258. MIT Press, Cambridge, MA, USA, 1998.

[11] B. Leibe and B. Schiele. Analyzing appearance and contour based methods for object categorization. In *In IEEE Conference on Computer Vision and Pattern Recognition (CVPR'03*, page 409–415, 2003.

[12] H. Ling and D. W. Jacobs. Shape classification using the inner-distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(2):286–299, 2007.

[13] S. Manay, D. Cremers, B. woo Hong, A. J. Yezzi, and S. Soatto. Integral invariants for shape matching. *IEEE PAMI*, 28:1602–1618, 2006.

[14] O. J. O. Söderkvist. Computer vision classification of leaves from swedish trees. Master's thesis, Linköping University, SE-581 83 Linköping, Sweden, September 2001. LiTH-ISY-EX-3132.

[15] D. W. Thompson. *On Growth and Form*. Cambridge University Press, 1983.

[16] D. Zhang and G. Lu. Review of shape representation and description techniques. *Pattern Recognition*, 37, 2004.