Fast Approximate Nearest Neighbor Methods for Non-Euclidean Manifolds with Applications to Human Activity Analysis in Videos

Rizwan Chaudhry[1,⋆] and Yuri Ivanov[2]

[1] Center for Imaging Science, Johns Hopkins University
3400 N Charles St, Baltimore, MD 21218, USA
rizwanch@cis.jhu.edu
[2] Mitsubishi Electric Research Laboratories
201 Broadway, Cambridge, MA 02139, USA
yivanov@merl.com

Abstract. Approximate Nearest Neighbor (ANN) methods such as Locality Sensitive Hashing, Semantic Hashing, and Spectral Hashing, provide computationally efficient procedures for finding objects similar to a query object in large datasets. These methods have been successfully applied to search web-scale datasets that can contain millions of images. Unfortunately, the key assumption in these procedures is that objects in the dataset lie in a Euclidean space. This assumption is not always valid and poses a challenge for several computer vision applications where data commonly lies in complex non-Euclidean manifolds. In particular, dynamic data such as human activities are commonly represented as distributions over bags of video words or as dynamical systems. In this paper, we propose two new algorithms that extend Spectral Hashing to non-Euclidean spaces. The first method considers the Riemannian geometry of the manifold and performs Spectral Hashing in the tangent space of the manifold at several points. The second method divides the data into subsets and takes advantage of the *kernel trick* to perform non-Euclidean Spectral Hashing. For a data set of N samples the proposed methods are able to retrieve similar objects in as low as $O(K)$ time complexity, where K is the number of clusters in the data. Since $K \ll N$, our methods are extremely efficient. We test and evaluate our methods on synthetic data generated from the Unit Hypersphere and the Grassmann manifold. Finally, we show promising results on a human action database.

Keywords: Approximate Nearest Neighbors, Hashing, Non-Euclidean Manifolds, Activity Analysis in Videos.

1 Introduction

Human action analysis is considered one of the most important problems in computer vision. It enables such applications as automatic surveillance, behavior

⋆ This work was done as part of a summer research internship at the Mitsubishi Electric Research Laboratories, Cambridge, MA, USA.

K. Daniilidis, P. Maragos, N. Paragios (Eds.): ECCV 2010, Part II, LNCS 6312, pp. 735–748, 2010.

analysis, elderly care, etc. There has been a tremendous amount of work towards automatic analysis of human motion in videos. However, due to extensive amount of computation required for video analysis, this work by necessity is often restricted to smaller models and datasets. In a real-life surveillance scenario video data is continuously recorded for a long period of time and saved for later analysis. Search in such extensive volumes of data remains a difficult task. Toward this goal, this paper proposes a major step in developing hashing techniques upon which sophisticated and efficient searches for a nearest neighbor in large corpora of video data can be built. It is often the case when classifying complex data, that using sophisticated features makes even a simple NN technique perform very well. Sampling from a neighborhood, commonly used in tracking applications, can also benefit from efficiency of hashing-based NN search. In this work we present two methods that have a goal of replicating the Nearest Neighbor search to make it applicable to very large datasets of complex features.

Prior work. Recently, there has been a surge in interest in fast content-based image retrieval from web-scale databases of tens of millions of images. However there has been little work in the same direction for videos. In this section we give a summary of notable work dominant in the field. Karpenko *et. al.* in [1] introduced a method where all the videos in a dataset were compressed to very small frame sizes and only a few key-frames. Using intensity statistics in the frames, the comparison of the new query is performed with the entire dataset. This technique leads to faster video comparison, but doesn't use semantically meaningful features and cannot be performed faster than $O(N)$. Biswas *et. al.* [2] provided a method that used two-level hash tables based on the invariant geometric properties of object shapes for efficient search and retrieval. Turaga *et. al.* [3] proposed a dynamical-systems based model for human activities that can be used for clustering different types of activities in a continuous video. Sidenbladh *et. al.* [4] used an approximate probabilistic tree search to find the closest match in the database for a query motion. Ben-Arie *et. al.* in [5] used a sparsely sampled sequence of body poses and velocity vectors of body parts as they move in a scene to construct multi-dimensional hash tables. For a test video, these features were extracted and the key was used to find the match in the hash-tables. Several other methods proposed in [6,7,8] cluster features derived from motion and appearance information for semantic retrieval.

All of these methods either use exact-match hashing, which generally has difficulties in performing a neighborhood search, or tree-based approaches, which often help increase performance, but are not as fast as hashing techniques.

Recently, new hashing algorithms that preserve neighborhood relationship between derived codes have been developed. Approximate Nearest-Neighbor methods such as the variants of Locality Sensitive Hashing (LSH), [9], Semantic Hashing, [10], and Spectral Hashing, [11], provide efficient algorithms for constructing binary codes for points in a high dimensional space. These methods have the property that codes for points that are nearby in the high-dimensional space are also close to each other in the binary code space under the Hamming distance. This provides an excellent method for creating hash tables because even if the

key for a query object is not in the table, the keys for neighbors in the Hamming space can then be checked by simply flipping a bit of the binary code.

One important limitation of all the above methods is that they are only applicable to data that resides in a Euclidean space. However, features frequently used for activity analysis in dynamic data have strong non-Euclidean character. For instance, histograms created as part of a bags of video words classification procedure on local features proposed by Laptev, [12] and Dollar et al. [13], or dynamical systems proposed in [14,15,16,17], naturally lie on a non-trivial manifold that has strong non-Euclidean properties. Hence the above methods are not directly applicable. The authors in [11] mentioned this limitation of Spectral Hashing and assumed that a suitable Euclidean embedding can be used. However, finding such an embedding is not always possible. A workaround for LSH that uses the *kernel trick* to implicitly embed the data in a high-dimensional Euclidean space is proposed by Kulis and Grauman [18]. Further, Kulis and Darrell in [19] use a similar kernel trick for Spectral Hashing. However as we will explain later, this method is no faster than performing exact nearest neighbor search.

As shown in [11], LSH usually gives very large codewords, whereas Semantic Hashing and Spectral Hashing give compact binary codewords and therefore are more useful for mapping objects directly to memory addresses in a computer. In this paper, we turn our attention to Spectral Hashing and propose two new fast approximate methods for performing Spectral Hashing on non-Euclidean data. In section 2 we summarize standard Euclidean spectral hashing and formulate the exact problem for non-Euclidean data. In section 3, we explain our proposed methods and their complexity. In section 4 we test our algorithm on both synthetic and real data sets; and give future directions of research in section 5.

2 Spectral Hashing

As presented by Weiss et al. in [11], given data points, $\{\mathbf{x}_i\}_{i=1}^N \in \mathbb{R}^d$, the goal is to find k-bit binary vectors, $\{y_i\}_{i=1}^N \in \{-1,1\}^k$ such that similar points in $\mathbb{R}^N$, under the similarity measure, $W_{ij} = \exp(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{\epsilon^2})$ map to binary vectors that are close to each other under the Hamming distance weighted by W.

If we assume that the data, $\mathbf{x}_i \in \mathbb{R}^d$, is sampled from a probability distribution $p(\mathbf{x})$, Spectral Hashing (SH) solves the following optimization problem:

$$\text{minimize} \quad \int \|y(\mathbf{x}_1) - y(\mathbf{x}_2)\|^2 W(\mathbf{x}_1, \mathbf{x}_2) p(\mathbf{x}_1) p(\mathbf{x}_2) d\mathbf{x}_1 d\mathbf{x}_2 \tag{1}$$

$$\text{s.t.} \quad y(\mathbf{x}) \in \{-1,1\}^k, \quad \int y(\mathbf{x}) p(\mathbf{x}) d\mathbf{x} = 0, \quad \text{and} \quad \int y(\mathbf{x}) y(\mathbf{x})^\top p(\mathbf{x}) d\mathbf{x} = I$$

Relaxing the first constraint gives the solution of the problem, y as the first k *eigenfunctions* of the weighted Laplace-Beltrami operator on the manifold. If p is the multi-dimensional uniform distribution on a subset of $\mathbb{R}^d$ and the weighting function, W, is defined as above, there exists a one-shot closed form solution for these eigenfunctions. However, in the case of a Gaussian distribution on $\mathbb{R}^d$, there exists an iterative solution.

Spectral hashing has a very appealing mathematical formulation. Ideally, one could take any probability distribution on a general manifold and a weighting function and analytically compute the eigenfunctions of the corresponding Laplace-Beltrami operator. However, even in the simpler case of Euclidean data, a closed form solution might not exist. Thus, analysis of non-Euclidean data may require solving this problem numerically. Furthermore, the weighting function, W, is computed from geodesic distances and thus, is no longer a simple exponential similarity. This makes the exact computation of the solution of the minimization problem in Eq. (1) computationally intractable.

As part of computing the eigenfunctions, the method in [11] employs PCA to compute a basis for the dataset, and then computes closed-form one-dimensional eigenfunctions of the weighted Laplace-Beltrami operator in each principal component direction using a rectangular approximation to the spread of the data. The method then combines these 1-D eigenfunctions to compute the eigenfunctions of the original dataset in $\mathbb{R}^d$. To deal with non-Euclidean data, Kulis *et. al.* [19] proposed using Kernel PCA instead of PCA in this step. We will refer to this method as *Kernel Spectral Hashing* (KSH). Even though their method is theoretically correct, as the kernels would embed the points in a high-dimensional Euclidean space, finding the value of the eigenfunction at each new test datapoint would involve computing the kernel of the test point with all the points in the training set used to compute the kernel PCA components. Because of this, even though a well-chosen kernel might give fine retrieval accuracy, the computational complexity of this method is at least $O(N)$.

3 Non-Euclidean Spectral Hashing

Noting the difficulty with applying Spectral Hashing techniques to non-Euclidean manifolds, we propose two new methods for finding compact binary codes for data lying on such manifolds with which this difficulty can be circumvented.

3.1 Riemannian Spectral Hashing

Since it is hard to compute closed form eigenfunctions in the SH algorithm for non-Euclidean data, we can embed the data in a Euclidean space. Then, under the assumption that it is drawn from a uniform distribution in that space, spectral hashing can be applied in this embedding space. Our first method, *Riemannian Spectral Hashing (RSH)*, follows this strategy.

The tangent space, $\mathcal{T}_{\mathbf{y}}\mathcal{M}$, to a manifold, $\mathcal{M}$ at a point $\mathbf{y}$ is a Euclidean space. Therefore, assuming that the manifold is geodesically complete, the data, $\{\mathbf{x}_i\}_{i=1}^N$, can be projected onto the tangent space at $\mathbf{y}$ by using the *logarithm map*, $\Delta_i = \overrightarrow{\mathbf{yx}_i} = \log_{\mathbf{y}}(\mathbf{x}_i)$. This makes it possible to perform Spectral Hashing on the tangent space projections, $\{\Delta_i\}_{i=1}^N$ locally, around $\mathbf{y}$ without introducing significant projection error. In order to minimize projection errors, the RSH algrorithm approximates a manifold with a *set of tangent hyperplanes*, positioned on a set of representative points (poles) which follow the distribution of the data

on the manifold. The poles are found by clustering, for which we can use any extrinsic manifold clustering algorithm such as [20], [21] to cluster the data on the manifold into K clusters. We use the Riemannian k-means procedure:

1. Initialize cluster centers, $\{\mathbf{c}_j\}_{j=1}^{K}$ by randomly choosing K points from the training data.
2. For each point $\mathbf{x}_i$ in the data set, compute the geodesic distance to each cluster center, $d(\mathbf{c}_j, \mathbf{x}_i) = \| \log_{\mathbf{c}_j}(\mathbf{x}_i) \|$. Assign the cluster center that is the closest to the data point as the cluster membership, $w_i = \text{argmin}_j \| \log_{\mathbf{c}_j}(\mathbf{x}_i) \|$.
3. Recompute each cluster center as the Karcher mean of the points in each cluster, $\mathbf{c}_j = \text{mean}\{\mathbf{x}_l | w_l = j\}$. This requires repeated uses of the *exponential map* and the *logarithm map* on the manifold until convergence to a mean.
4. Repeat until convergence.

This clustering algorithm is a simple extension of the k-means algorithm to a single geodesically complete manifold under the assumption that no two points in the same cluster are antipodes. This method inherits the convergence properties of regular Euclidean k-means. Once the clusters, $\{\mathbf{c}_j\}_{j=1}^{K}$, and memberships, $\{w_i\}_{i=1}^{N}$, have been assigned, all the points in the same cluster are projected to the tangent space around the cluster center using the corresponding logarithm maps. A separate spectral hashing algorithm is then trained on each tangent space.

For computing the binary code of a new test point, $\mathbf{z}$, we first compute the geodesic distance of $\mathbf{z}$ with all the cluster centers and project it to the tangent space of the closest cluster center, $\mathbf{c}_k$, where $k = \arg\min_j \|\overrightarrow{\mathbf{c}_j \mathbf{z}}\|$ to get $\Delta_{\mathbf{z}} = \log_{\mathbf{c}_k}(\mathbf{z})$. We then use spectral hashing to find the binary code of $\Delta_{\mathbf{z}}$. Since finding the right cluster center, only requires K geodesic distance evaluations, this results in a computational cost of $O(K)$. Even though this is greater than $O(1)$ as in Spectral Hashing, it is much less than $O(N)$ as in Kernel Spectral Hashing, where $K \ll N$. Moreover, by clustering all the data, we better approximate the uniform distribution assumption in each cluster. We summarize RSH in Algorithm 1. Figure 1(b) provides an illustration of the method.

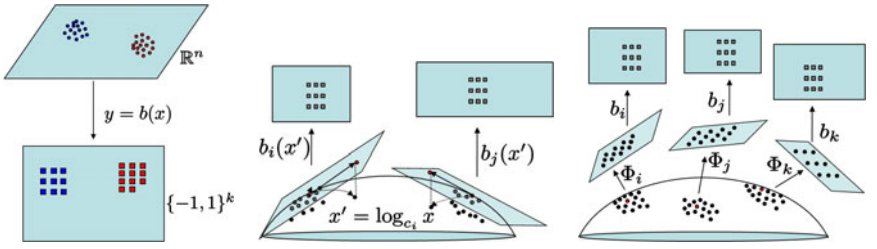
Algorithm 1: Riemannian Spectral Hashing

Training

1. Cluster training data using a manifold clustering algorithm, [20], [21].
2. Compute log-maps and project each cluster to the tangent space around center.
3. Train spectral hashing on points in each tangent space separately.

Testing

1. Find closest cluster center using geodesic distances on the manifold.
2. Project onto tangent space around closest cluster center.
3. Compute binary code of the projected point using Spectral Hashing.
4. Retrieve the nearest neighbor.



(a) Spectral Hashing (SH)[11] (b) Riemannian Spectral Hashing (RSH) (c) Distributed Kernel Spectral Hashing (DKSH)

Fig. 1. Schematic diagram of state-of-the-art and proposed hashing methods

3.2 Distributed Kernel Spectral Hashing (DKSH)

In certain cases, closed form experssions for the logarithm and exponential maps for manifolds are not available. This limits the applicability of extrinsic manifold clustering algorithms as required in Alg. 1. If however, a kernel or other affinity measure, $W(.,.)$ is defined on the manifold, a non-linear dimensionality reduction method such as Multidimensional Scaling (MDS) [22] can be employed to project the data into a low-dimensional Euclidean space before performing k-means on this low-dimensional space. Alternatively, a non-linear clustering algorithm such as kernel k-means [23] or Spectral Clustering [24] can be used to compute cluster associations of the data. As a result, we would not have cluster centers but only cluster associations for the training data. After the clustering stage, one representative point is chosen in each cluster to represent all data within it. One method to choose this point is as follows [25]:

1. Compute the $N \times N$ affinitiy matrix, W, of the training data based on a kernel or affinity defined on the manifold.
2. Perform MDS using W to get a low-dimensional Euclidean representation $\{\mathbf{u_i}\}_{i=1}^{N}$ and perform k-means on these points to get K cluster centers $\{\mathbf{v}_j\}_{j=1}^{K}$ in the low-dimensional space.
3. Within each cluster center, choose the point $\mathbf{u}$ in the projected data that is closest to each cluster center $\mathbf{v}_j$.
4. Find the original points $\{\mathbf{x}_{p;j}\}_{j=1}^{K}$ on the manifold that mapped to the points $\{\mathbf{v}_j\}_{j=1}^{K}$ after MDS and use these points as cluster representatives (pivots).

Once a representative, or pivot, for each cluster has been computed, we train Kernel Spectral Hashing (KSH) separately for each cluster.

As in RSH, to find the binary code for a test point, $\mathbf{z}$, we first compute its affinity, $W(\mathbf{x}_{p;j}, \mathbf{z})$ with each pivot point and assign $\mathbf{z}$ to the j-th cluster if $\mathbf{x}_{p;j}$ has the highest affinity with $\mathbf{z}$. We then use Kernel Spectral Hashing trained for that specific cluster and compute the binary code for $\mathbf{z}$ to retrieve the nearest neighbors. Assuming that in the best case, all the points are equally divided between K clusters, the query time complexity of this method is $O(K + N/K)$ on average, which is more computationally expensive that RSH. However, it is

still significantly better than the complexity of KSH. In the worst case when only 1 cluster is chosen, the complexity is $O(N)$, the same as KSH. We summarize DKSH in Algorithm 2. Figure 1(c) provides an illustration of the method.

Algorithm 2: Distributed Kernel Spectral Hashing

Training

1. Cluster training data using Non-Linear clustering (MDS, Spectral clustering etc.) using kernel similarity
2. Pick a pivot point, representing each cluster.
3. Train Kernel Spectral Hashing on points in each cluster separately.

Testing

1. Use kernel similarity to compute pivot with the highest affinity to test point.
2. Compute binary code with kernel spectral hashing for that pivot.
3. Retrieve the nearest neighbor.

4 Experiments

In this section we compare the proposed methods, Riemannian Spectral Hashing (RSH) and Distributed Kernel Spectral Hashing (DKSH), against exact Nearest Neihbors (NN), and state-of-the-art Hashing methods: Kernel Locality Sensitive Hashing (KLSH) [18], Euclidean Spectral Hashing [11] (SH), and Kernel Spectral Hashing [19] (KSH).

4.1 Synthetic Data

We first test the proposed methods on synthetic datasets of points lying on two non-Euclidean manifolds: the 100 dimensional unit hypersphere, S^{99}, and the manifold of all 3-dimensional subspaces of $\mathbb{R}^{10}$, *i.e.*, the Grassmann manifold, $G_{10,3}$ or $G_{3,10-3}$. The evaluation is performed on an 8-core Intel Xeon 3.4 GHz machine with 32 GB of RAM. In each experiment, we restrict the number of processing cores to exactly one so that the run-times of various algorithms are comparable. When comparing our methods with Spectral Hashing, we treat the points on both the above mentioned manifolds as points in $\mathbb{R}^{100}$ and $\mathbb{R}^{30}$ respectively.

As a technical detail, it is noted that when the data size grows larger than 10^4 samples, the memory requirements of the PCA computation in SH and the kernel PCA in KSH become extremely large and can not be handled by our computational resources. As an example, consider computing the all pair kernel matrix for 10^5 points. Storing the result as a double precision matrix in memory requires a minimum of $(10^5)^2 \times 8 = 72$ GB of memory, which is not available in our system. Therefore for datasets larger than 10^4 points, we randomly sample 1000 points, equally sampling from each class, and pre-train all the hashing algorithms on this smaller set. We then compute the binary hash codes for all

the training points and store them for comparison against the test sets. Since the number of exponential and logarithm map evaluations as well as kernel evaluations will decrease, we will distinguish the training and testing times for the hashing methods where all the data was used for training ($10-10^4$ samples), and for methods where a pre-training approach was used, ($10^5, 10^6$ samples).

Unit hypersphere - S^{99}. The unit hypersphere, S^{99}, is the set of all points, $\mathbf{x} \in \mathbb{R}^{100}$ that satisfy the constraint, $\sum_{i=1}^{100} x_i^2 = 1$. The geodesic distance between two points, $\mathbf{x}$ and $\mathbf{y}$, on a hypersphere is defined as $d_G(\mathbf{x}, \mathbf{y}) = \cos^{-1}(\mathbf{x}^\top \mathbf{y})$. Moreover, the logarithm and exponential maps on the sphere are defined as,

$$\log_{\mathbf{x}}(\mathbf{y}) = \frac{\mathbf{y} - (\mathbf{x}^\top \mathbf{y})\mathbf{x}}{\|\mathbf{y} - (\mathbf{x}^\top \mathbf{y})\mathbf{x}\|} \cos^{-1}(\mathbf{x}^\top \mathbf{y}),$$

$$\exp_{\mathbf{x}}(\Delta) = \cos(\|\Delta\|)\mathbf{x} + \sin(\|\Delta\|)\frac{\Delta}{\|\Delta\|},$$

where Δ is a tangent vector at the pole $\mathbf{x}$. Finally, the standard inner product also defines a kernel on the sphere, i.e. $k(\mathbf{x}, \mathbf{y}) = \mathbf{x}^\top \mathbf{y}$.

We generate 5 sets of 5-class each training datasets containing 100, 1000, 10^4, 10^5 and 10^6 points on S^{99}. For testing, we generate 100 more points in each case. Figure 2 displays the difference between the recognition percentages of exact 1-NN and the state-of-the-art methods (Kernel LSH (KLSH), SH and KSH) and the proposed hashing algorithms (RSH and DKSH). We use 8 bits for all hashing algorithms and 5 clusters for the proposed methods. Both RSH and DKSH have the lowest percentage difference compared to the state-of-the-art methods for all training sizes. Moreover, the error percentages remain within 10-15% of the exact 1-NN method. This can clearly be attributed to the fact that the proposed methods specifically take into account the manifold structure of the space and thus result in better recognition performance.

Table 1 shows the training times required for each algorithm against the number of training samples. 1-NN does not require any training, whereas SH and

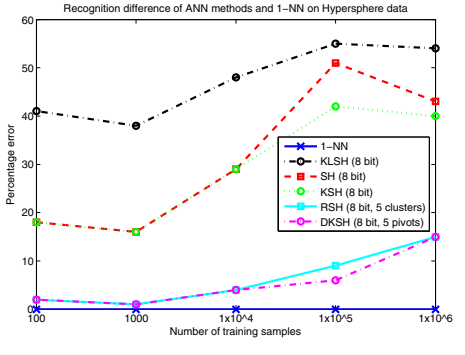


Fig. 2. S^{99} - Comparison of NN and ANN methods

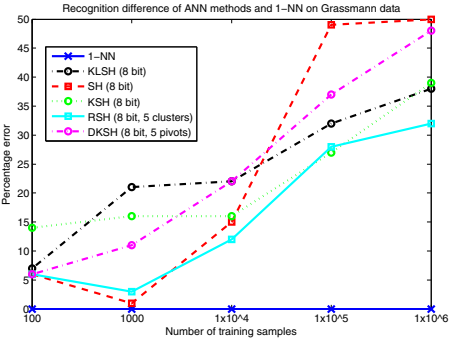


Fig. 3. $G_{10,3}$ - Comparison of NN and ANN methods

Table 1. S^{99} - Training times

Method	Training time (sec)				
# Training	100	1000	10^4	10^5	10^6
NN	0	0	0	0	0
KLSH	0.01	3.44	1.5h	85.0	11.7m
SH	0.01	0.40	56.02	46.11	6.4m
KSH	0.02	7.30	2.0h	50.7m	7.5h
RSH	0.35	1.90	33.8	70.17	8.8m
DKSH	0.08	7.62	2.0h	72.02	10.1m

Table 2. S^{99} - Testing times

Method	Testing time (sec)				
# Training	100	1000	10^4	10^5	10^6
NN	0.01	0.02	0.41	4.81	1.3m
KLSH	0.03	0.02	0.37	1.96	17.0
SH	0.04	0.04	0.04	0.09	1.06
KSH	0.06	3.13	4.1m	2.32	4.02
RSH	0.06	0.06	0.05	0.07	0.28
DKSH	0.06	0.07	10.09	0.07	0.25

Table 3. $G_{10,3}$ - Training times

Method	Training time (sec)				
# Training	100	1000	10^4	10^5	10^6
NN	0	0	0	0	0
KLSH	0.01	5.07	1.2h	34.0m	5.5h
SH	0.01	0.08	9.60	16.63	3.9m
KSH	0.12	17.2	2.5h	1.5h	13.2h
RSH	0.56	10.52	6.1m	10.2m	16.6m
DKSH	0.17	21.27	3.2h	36m	6.3h

Table 4. $G_{10,3}$ - Testing times

Method	Testing time (sec)				
# Training	100	1000	10^4	10^5	10^6
NN	2.06	21.2	3.3m	41.1m	5.7h
KLSH	0.20	2.29	22.5	4.15	23.1
SH	0.04	0.03	0.04	0.10	1.27
KSH	0.28	3.60	3.5m	4.83	5.47
RSH	0.12	0.11	0.10	0.13	1.21
DKSH	0.17	1.95	1.5m	1.16	1.88

RSH are the fastest to train. The training times for KLSH, KSH and DKSH increase greatly with the number of training samples. Table 2 provides the total test time for 100 samples. Coupled with higher accuracy, this is where we observe the real advantage of the proposed methods. As the size of the training data increases, not surprisingly, the time taken for 1-NN also increases. All test times for SH and KSH remain low but are still higher than the test times for RSH and DKSH. This again illustrates the superiority of the proposed methods.

Finally, Figure 4 displays the dependence of the *error rate* of RSH on the algorithm parameters, *i.e.*, the number of bits and the number of clusters. We can see that if the number of bits is kept constant, increasing the number of cluster centers decreases the testing error rate. Similarly, keeping the number of clusters constant, and increasing the number of bits also decreases the testing error rate. The first quality is highly desirable, since in a real scenario, the binary code will represent the memory location for a pointer to the data. Thus having more than 64 bits is not practical. In fact this shows that we can use relatively fewer number of bits and pack the data points in memory by using more clusters. Since the clusters can be located arbitrarily in memory, this reduces the need for large chunks of contiguous memory.

Grassmann manifold - $G_{10,3}$. In an analogous fashion to the previous section, we generate several training samples of different sizes on the Grassmann manifold, $G_{10,3}$, which is the manifold of all the 3-dimensional subspaces of $\mathbb{R}^{10}$. The data lies in 5 classes and is generated using the method in [26]. For non-linear clustering and tangent space to manifold projections and vice-versa, we use the expressions for the exponential and logarithm maps on the Grassmann manifold

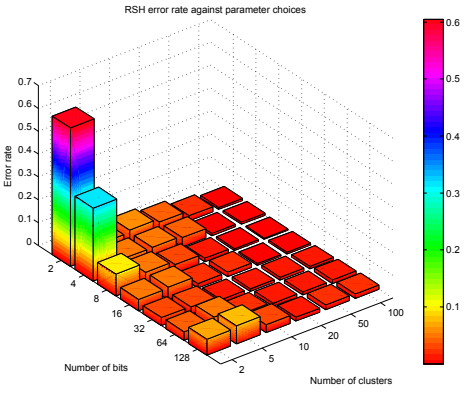


Fig. 4. S^{99} - RSH error rates against algorithm parameters

in [20]. For computing the kernel on the manifold we use the product of the cosines of the subspace angles between the subspaces [27]. Again, we use 8 bits for the binary codes for all hashing algorithms and 5 clusters for our proposed methods. Figure 3 displays the difference between the recognition percentage of 1-NN and all other methods. At first it might seem that SH performs better than the other methods for small data sizes, the trend is offset drastically with large training sizes where it performs the worst. Overall, RSH performs better than all state-of-the-art methods and the error stays within 30% of that of 1-NN.

Table 3 provides the training times for each of the training datasets. We notice that KLSH, KSH and DKSH require the largest training times, whereas RSH and SH require the least. For all the methods, the training time increases with the number of data points but due to the large number of kernel computations during the training stage, the increase in time is greatest for KLSH, KSH and DKSH. Table 4 gives the total test time for 100 test samples for each of the training sizes. We again see the computational advantage of the proposed methods against the exact method as well as the state of the art KLSH and KSH methods. The test time increases steeply with the size of the data for the kernel-based methods, whereas the corresponding increase in test time stays low for RSH.

From the above set of experiments, we have shown that the proposed approximate nearest-neighbor methods, RSH and DKSH, by explicitly considering the manifold structure of the space of data, provide great computational advantage against exact Nearest Neighbors while having very low to modest decrease in accuracy. Moreover, our methods always perform better than KLSH and KSH, the state-of-the-art non-Euclidean Hashing methods.

4.2 Human Action Dataset

Recent approaches in human activity recognition use features such as (1) distributions over a bag of spatial-temporal keypoints to represent the activity in a scene, or (2) dynamical systems learnt from a time-series of features extracted from the frames of the video. Both these features lie in non-Euclidean spaces

and therefore the proposed approach is directly applicable for the purpose of retrieving activities from a large dataset of activity videos. Even though, human activity analysis has been a vibrant field in computer vision, to the best of our knowledge, no datasets are available that contain more than a few thousand instances of human actions. Videos of unstructured scenes with multiple activities and events are available, however, the ground-truth activity segmentation and tracking is not provided and automatic extraction of these remains an open problem in computer vision. One of the most popular and largest datasets available is the KTH human action dataset [28]. This dataset contains six actions: walking, running, jogging, boxing, handwaving and handclapping. There are 25 persons performing these actions under four different scenarios: outdoors, outdoors across different scales, outdoors with bulky clothes on and indoors. There are a total of 2391 sequences in the dataset.

For our first experiment, we use the approach of [13] and extract several spatio-temporal keypoints and their corresponding descriptors in all the videos. We divide the data as follows: All the videos of the first 16 subjects are used for training whereas the videos of the remaining 9 subjects are used for testing. A k-means procedure is used to cluster the descriptors in the training data to form a dictionary of 100 keypoints. We then learn feature distributions for each action video around these keypoints. This provides a 100 dimensional histogram per video that represents the action in that video. These histograms are used for training and testing the proposed ANN methods. Note that for a fair comparison to nearest-neighbor algorithms, we will test our method against the simple nearest-neighbor algorithm and not against the state-of-the-art methods for human activity recognition that use sophisticated classification algorithms to achieve superior performance. The error rates reported below are not state-of-the-art on the KTH human action database; instead, they are the error rates achieved when using exact NN and state-of-the-art ANN methods and our proposed methods on the dataset. We emphasize that our goal here is not to find the best classification algorithm on the KTH database, but to compare the performance of the proposed ANN methods against state-of-the-art ANN and exact NN methods. Table 5 compares the performance of the proposed methods with Nearest Neighbors and state-of-the-art hashing methods. All the hashing methods use 8-bits for the binary codes. The proposed methods, RSH and DKSH divide the training data into 3 clusters. The results show that RSH has the highest recognition percentage other than exact NN, whereas the state-of-the-art KLSH has the worst recognition percentage. Moreover, RSH is also the most efficient method in terms of retrieval time, even though it requires the largest training time. Furthermore, the best recognition rate achievable using RSH was 69% with 64 bit code-words and 2 clusters, which is only 7% below the error rate achieved by exact 1-NN.

For our second experiment, we use the approach in [17] and compute the Histogram of Oriented Optical Flow (HOOF) features at each frame to get a normalized histogram time-series for each video. We then learn a linear-state non-linear dynamical system (NLDS) using the approach in [17] with the Geodesic

Table 5. BOW histograms

Method	Correct %	Train t	Test t
NN	76	0	11.5
KLSH	24	38.4	1.04
SH	51	3.4	0.45
KSH	51	41.1	81.1
RSH	62	58.6	0.39
DKSH	51	31.1	3.34

Table 6. Observability matrices

Method	Correct %	Train t	Test t
NN	72	0	149.3
KLSH	64	0.547	35.0
SH	22	5.262	1.67
KSH	17	31.24	38.7
RSH	65	321.7	10.3
DKSH	58	266.5	15.8

(Bhattacharya) kernel on histograms. Hence each activity video is now represented as a non-linear dynamical system. There are several methods for comparing dynamical systems, *e.g.* those proposed in [17] and the references therein. We represent the dynamics and output transformation functions using the observability matrix for each dynamical system. Since we are using the inner-product on the sphere as the kernel, we can simply use PCA to learn the approximate dynamical system parameters and thus get the parameter matrices, $A \in \mathbb{R}^{n \times n}$ and $C \in \mathbb{R}^{p \times n}$. Here n is the system order, and p is the size of the output, 10 and 64, respectively, in our case. The observability matrix can then be computed as $\mathcal{O} = [C^\top, (AC)^\top (A^2C)^\top, \ldots, (A^{n-1}C)^\top]^\top \in \mathbb{R}^{pn \times n}$. See [17] and the references therein for more details. Notice that the columns of $\mathcal{O}$ span an n-dimensional subspace of $\mathbb{R}^{np}$ and thus $\mathcal{O}$ lies on the Grassmann manifold, $G_{np,n}$. We can therefore follow the experiments in section 4.1 using these observability matrices as the data points.

Since the approach in [17] is directly applicable only for stationary cameras, we choose sequences from the first scenario, i.e. outdoors with stationary camera (around 600 sequences) to test our algorithms. We use 64% of the data for training and the remaining 36% for testing. Moreover, we use 64 bits for the binary codes and 15 clusters/pivots for the proposed methods.

Table 6 shows the recognition percentages and training and testing times for exact KNN using the Martin distance for dynamical systems, and the proposed and state-of-the-art hashing methods. We can see that our method, RSH, has the best recognition rate, slightly above KLSH. Notice that even though exact NN does not require any training, which could be as high as 321.7 seconds for RSH, the speed up in terms of test times is significant. Exact KNN requires 149.3 seconds for testing whereas RSH requires only 10.3 seconds and DKSH requires 15.8 seconds. Even though KLSH performs well in this scenario, due to the many kernel computations required, its testing time is at least 3 times greater than RSH, limiting the former's advantage.

5 Conclusion and Future Work

We have proposed two new methods, Riemannian Spectral Hashing (RSH), and Distributed Kernel Spectral Hashing (DKSH), for performing fast approximate

nearest-neighbor matching on non-Euclidean data. We have shown that state-of-the-art methods either do not take into account the manifold structure of the data, or are computationally inefficient and can in fact be slower in performance than exact nearest neighbors. Moreover, experiments on synthetic and real data have shown that our methods are applicable to points that lie on simple manifolds such as the unit hypersphere as well as to points that lie on highly complicated manifolds such as the space of dynamical systems. The proposed methods provide immense computational savings at the cost of a small decrease in accuracy and hence are ideal for approximate nearest neighbor matching in large datasets. We have provided average-case time complexity for our proposed methods and are looking into how the parameters such as the number of bits and number of clusters/pivots, can be set so as to achieve user-defined precision/recall tolerances. Finally we are working on collecting a very large human action dataset to further validate the benefits of our proposed methods.

Acknowledgments. The authors would like to thank Kinh Tieu, Ashok Veeraraghavan and Oncel Tuzel for their comments and discussions that helped improve the presentation of this work.

References

1. Karpenko, A., Aarabi, P.: Tiny videos: Non-parametric content-based video retrieval and recognition. In: IEEE International Symposium on Multimedia (2008)
2. Biswas, S., Aggarwal, G., Chellappa, R.: Efficient indexing for articulation invariant shape matching and retrieval. In: IEEE Conference on Computer Vision and Pattern Recognition (2007)
3. Turaga, P., Veeraraghavan, A., Chellappa, R.: From videos to verbs: Mining videos for events using a cascade of dynamical systems. In: IEEE Conference on Computer Vision and Pattern Recognition (2007)
4. Sidenbladh, H., Black, M.J., Sigal, L.: Implicit probabilistic models of human motion for synthesis and tracking. In: Heyden, A., Sparr, G., Nielsen, M., Johansen, P. (eds.) ECCV 2002. LNCS, vol. 2350, pp. 784–800. Springer, Heidelberg (2002)
5. Ben-Arie, J., Wang, Z., Pandit, P., Rajaram, S.: Human activity recognition using multidimensional indexing. IEEE Transactions on Pattern Analysis and Machine Intelligence 24, 1091–1104 (2002)
6. Chen, D.Y., Lee, S.Y., Chen, H.T.: Motion activity based semantic video similarity retrieval. In: Advances in Multimedia Information Processing (2002)
7. Chen, X., Zhang, C.: Semantic event retrieval from surveillance video databases. In: IEEE International Symposium on Multimedia (2008)
8. Kashino, K., Kimura, A., Kurozumi, T.: A quick video search method based on local and global feature clustering. In: International Conference on Pattern Recognition (2004)
9. Andoni, A., Indyk, P.: Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In: Symposium on Foundations of Computer Science (2006)
10. Salakhutdinov, R., Hinton, G.: Semantic hashing. International Journal of Approximate Reasoning 50, 969–978 (2009)

11. Weiss, Y., Torralba, A., Fergus, R.: Spectral hashing. In: Neural Information Processing Systems Conference (2008)
12. Laptev, I.: On space-time interest points. International Journal of Computer Vision 64, 107–123 (2005)
13. Dollar, P., Reabaud, V., Cottrell, G., Belongie, S.: Behavior recognition via sparse spatio-temporal features. In: VS-PETS (2005)
14. Bissacco, A., Chiuso, A., Ma, Y., Soatto, S.: Recognition of human gaits. In: IEEE Conference on Computer Vision and Pattern Recognition (2001)
15. Bissacco, A., Chiuso, A., Soatto, S.: Classification and recognition of dynamical models: The role of phase, independent components, kernels and optimal transport. IEEE Transactions on Pattern Analysis and Machine Intelligence 29(11), 1958–1972 (2007)
16. Basharat, A., Shah, M.: Time series prediction by chaotic modeling of nonlinear dynamical systems. In: International Conference on Computer Vision (2009)
17. Chaudhry, R., Ravichandran, A., Hager, G., Vidal, R.: Histograms of oriented optical flow and binet-cauchy kernels on nonlinear dynamical systems for the recognition of human actions. In: IEEE Conference on Computer Vision and Pattern Recognition (2009)
18. Kulis, B., Grauman, K.: Kernelized locality-sensitive hashing for scalable image search. In: International Conference on Computer Vision (2009)
19. Kulis, B., Darrel, T.: Learning to hash with binary reconstructive embeddings. Technical Report UCB/EECS-2009-101, Electrical Engineering and Computer Sciences, University of California at Berkeley (2009)
20. Subbarao, R., Meer, P.: Nonlinear mean shift over riemannian manifolds. International Journal of Computer Vision 84, 1–20 (2009)
21. Goh, A., Vidal, R.: Clustering and dimensionality reduction on Riemannian manifolds. In: IEEE Conference on Computer Vision and Pattern Recognition (2008)
22. Hastie, T., Tibshirani, R., Friedman, J.H.: The Elements of Statistical Learning. Springer, Heidelberg (2003)
23. Schölkopf, B., Smola, A.: Learning with Kernels. MIT Press, Cambridge (2002)
24. Shi, J., Malik, J.: Normalized cuts and image segmentation. IEEE Transactions on Pattern Analysis and Machine Intelligence 22, 888–905 (2000)
25. Ravichandran, A., Chaudhry, R., Vidal, R.: View-invariant dynamic texture recognition using a bag of dynamical systems. In: IEEE Conference on Computer Vision and Pattern Recognition (2009)
26. Çetingül, H.E., Vidal, R.: Intrinsic mean shift for clustering on stiefel and grassmann manifolds. In: IEEE Conference on Computer Vision and Pattern Recognition (2009)
27. Cock, K.D., Moor, B.D.: Subspace angles and distances between ARMA models. System and Control Letters 46, 265–270 (2002)
28. Schuldt, C., Laptev, I., Caputo, B.: Recognizing human actions: A local svm approach. In: International Conference on Pattern Recognition (2004)