Deep Video Hashing

Venice Erin Liong, Student Member, IEEE, Jiwen Lu, Senior Member, IEEE, and Yap Peng Tan, Senior Member, IEEE

Abstract—In this work, we propose a new deep video hashing (DVH) method for scalable video search. Unlike most existing video hashing methods which first extract features for each single frame and then use conventional image hashing techniques, our DVH learns binary codes for the entire video with a deep learning framework so that both temporal and discriminative information can be well exploited. Specifically, we fuse the temporal information across different frames within each video to learn the feature representation under two criteria: 1) the distance between a feature pair obtained at the top layer is small if they are in the same class, and large if they are from different classes, and 2) the quantization loss between the real-valued features and the binary codes is minimized. We exploit different deep architectures to utilize spatial-temporal information in different manners and compare them with single frame based deep feature models and state-of-the-art image hashing algorithms. Experimental results on two video databases demonstrate the effectiveness of our proposed method.

Index Terms—Scalable video search, video hashing, deep learning.

I. INTRODUCTION

Fast and efficient visual search is a challenging task which has gained large interests in computer vision, especially with the increasing amount of multimedia content which are available over the internet in recent years. This task aims to retrieve the most relevant visual content from a database, given a query sample, in an accurate and efficient manner. In contrast to visual images, videos provide diverse and complex visual patterns consisting of low-level visual content in each frame as well as high-level structured content, such as actions or events, across frames [6], [20], [21], [30], [63]. This makes video search more challenging than image search. Moreover, each video may have a number of image frames which leads to exhaustive computation and comparisons between frames, which is impractical when we have a large video database. As can be seen in surveillance and social media (YouTube), videos are equally important as images. Hence, how to develop a framework for scalable video search where discriminative information from videos can be well exploited in the extracted features remains an important problem in visual search, especially considering that discriminative information is preserved as much as possible with minimal computation cost and memory storage.

A key topic in visual search is learning-based hashing, which transforms high-dimensional feature vectors to compact binary codes, by preserving visual content using statistical inference. However, most current works in scalable visual search focus on image-based retrieval [16], [17], [33], [35], [38], [41], [42], [69] or text-image/image-text retrieval [37], [44], [58], [59], [68]. To our best knowledge, there are only few works that present an efficient framework for video hashing. Hence, it is desirable to make better use of hashing methods to exploit the spatial-temporal information of videos.

Current video hashing methods are mainly applied in two multimedia computing applications. The first is the near-duplicate search where conventional hashing techniques [12], [25], [57] were used to identify duplicate videos efficiently [7], [8], [52]. The second is content-based video retrieval [2], [35], [61], [63] which retrieves the most semantically similar videos from a database for a given query video. In this work, we focus on the latter one. Video hashing for content-based retrieval can be divided into three categories. The first extracts a single representative feature vector, and then performs hashing. The second treats each frame as an image and performs image hashing first such that the resulting frame-frame hamming distances of two videos are then combined through averaging. The last selects several representative frames and employs image hashing on these selected frames. While these frameworks are straightforward, they cannot exploit the temporal information of videos in the learning stage. Furthermore, hashing image frames individually is computationally expensive considering that each video would usually have a minimum of 30-50 frames. Hence, it is desirable to present a video hashing framework which learns strong features by utilizing the temporal information, and in turn, also minimizes the hamming distance computation.

To address these challenges, we propose a Deep Video Hashing (DVH) method for scalable video search where we explore different CNN-based architectures. Since Convolutional Neural Networks (CNNs) have shown promising performance in several computer vision tasks due to their strong feature representation capability, we also employ CNN in our video hashing framework. The basic idea of our approach is shown in Fig. 1. Specifically, we build an end-to-end deep CNN learning framework which utilizes spatio-temporal information after the stacked convolutional-
pooling layers to extract representative video features, and obtain compact binary codes. We discriminatively train our model with a Siamese network, by maximizing the inter-class distance and minimizing the intra-class distance of the video feature pairs, as well as minimizing the quantization loss between real-value codes and binary codes. We exploit different spatial-temporal feature pooling architectures and compare them to single-frame CNN architectures as well as state-of-the-art image-based hashing methods. Experimental results on two video datasets show the effectiveness of our proposed approach.

The contributions of this work are summarized as follows:

1) We propose a deep learning-based hashing method called deep video hashing (DVH) which learns a deep network to exploit the discriminative and temporal information of videos in order to represent each video with meaningful binary codes.

2) We conduct extensive scalable video search experiments on two video datasets to demonstrate the efficacy of our proposed method. We exploit different frame fusion architectures and compare them to several baseline network architectures, state-of-the-art single frame hashing methods, and existing video hashing methods.

The rest of the paper is organized as follows. Section II briefly reviews related work. Sections III presents the proposed deep video hashing methods with different temporal fusion architectures. Section IV presents the experimental results, and Section V concludes the paper.

II. RELATED WORK

In this section, we briefly review three related topics: 1) learning-based hashing, 2) video hashing, and 3) deep learning for video analysis.

A. Learning-based Hashing

Several learning-based hashing methods such as subspace models [16], [55], manifold models [22], [50], and kernel models [39], [45] have been exploited in the literature. These methods can be classified into two classes: unsupervised [9], [16], [17], [43], [57] and supervised [27], [36], [38], [39], [49], [50], [55]. The first category does not require label information. For example, Gong et al. [16] proposed a PCA-ITQ method which learns hashing functions by first performing PCA to maximize the variance of the hash bits and then learning a rotation matrix to minimize the quantization loss. Liu et al. [40] proposed an Anchor Graph Hashing (AGH) method by using the concept of anchors to identify the similarity between features. For the second category, class-wise label information is utilized. For example, Gong et al. [16] extended the PCA-ITQ to CCA-ITQ which first performs CCA to maximize the correlation between semantically similar features, and then minimizes the quantization loss. Liu et al. [39] employed the Kernel Supervised Hashing (KSH) method to perform kernel mapping and utilize supervised information through minimizing the distance of similar pairs and maximizing the distance of dissimilar pairs. Lin et al. [36] proposed a FastHash method to learn the binary codes through Graph Cuts and a greedy boosted decision tree framework. While these methods are mostly nonlinear hashing techniques, only a few works have used deep learning techniques to perform end-to-end nonlinear mapping [3], [10], [66], [67], [70]. Furthermore, these methods are specifically developed for large-scale image retrieval. In contrast, our work focuses on using statistical knowledge for scalable video search.

B. Video Hashing

Previous works on video hashing were mostly used for near-duplicate video retrieval tasks and several of them
focused on video feature representation rather than learning the hashing functions [7], [8], [52], [56], [63]. For example, Song et al. [52] introduced a multiple feature hashing method which utilizes multiple features and extracts different local structures to obtain efficient binary codes. Douze et al. [8] extracted representative spatial-temporal features from images and used conventional hashing methods to obtain binary codes. There are only a few video hashing methods proposed for content-based retrieval. For example, Cao et al. [2] proposed a submodular hashing framework which selects relevant frames from videos to learn the hashing functions for efficient video search. However, it did not really learn a video-based hashing function by using statistical knowledge. Ye et al. [61] proposed a supervised structural learning framework which exploits the temporal consistency to learn the linear hashing functions. However, it only learns a linear projection which may not truly capture the nonlinear nature of video data. Li et al. [34], [35] proposed a hashing model across the Euclidean space and the Riemannian manifold, which learns hashing functions based on the kernel max-margin framework for face video retrieval. However, their work represented videos with a single feature representation (covariance matrix), which may not fully exploit the spatio-temporal information in videos.

C. Deep Learning for Video Analysis

Deep learning techniques have shown great success in various computer vision tasks such as in image recognition [31], [53], scene labeling [11], [13], and pedestrian detection [65]. While a number of deep learning methods have also been proposed for video analysis, most of them focused on video action recognition [1], [26], [46], [51], video classification [29], [62], [64] and event detection [48], [60]. For example, Ji et al. [26] introduced a 3D Convolutional Neural Networks approach which considers spatial and temporal information for action recognition. Karpathy et al. [29] presented an extensive evaluation of different deep architectures for large-scale video classification where they introduced different spatio-temporal convolutions based on how the frames are fused in the network. Ng et al. [64] exploited different feature fusion methods after the stacked convolution and pooling layers and investigated the Long Short Term Memory (LSTM) networks for video classification. Xu et al. [60] explored pooling and encoding methods to combine frame-level features for event detection. Simonyan et al. [51] implemented a two stream convolution network for action recognition in videos to model the spatial and temporal data individually, and fuse the scores together. To our knowledge, nobody has investigated deep architectures for video hashing.

A. Learning-based Video Hashing

Let $X = \{X_1, y_1\}_{i=1}^M$ be a collection of $M$ videos where $X_i = [x_{i,1}, x_{i,2}, \ldots, x_{i,f_i}] \in \mathbb{R}^{d \times f_i}$ is the $i$th video with successive $f_i$ frames, $y_i$ is the label information, and $x_{i,j} \in \mathbb{R}^d$ is the $j$th image feature frame of the video $X_i$ with a feature length of $d$. The objective of learning-based video hashing is to learn $K$ hash functions to project each video into a single or multiple $K$-bit compact binary vectors as follows:

$$f_{X_i} : \mathbb{R}^{d \times f_i} \rightarrow \{-1, 1\}^{K \times g_i} \quad (1)$$

where $g_i \in \{1, f_i\}$.

To obtain hashing functions, we learn a linear projection matrix, $W = [w_1, w_2, \ldots, w_K] \in \mathbb{R}^{d \times K}$ to map the video features into compact binary codes. To obtain a single binary vector for the $i$th video, $b_i \in \{-1, 1\}^{K \times 1}$, we first extract a compact single feature representation for the $i$th video defined as $\tilde{x}_i$, and then project it linearly as follows:

$$b_i = \text{sgn}(W^T \tilde{x}_i) \quad (2)$$

However, it is difficult to represent a whole video as a single binary code without losing significant amount of information. Hence, a single video can be represented into multiple binary vectors by treating each frame as an image feature and performing image-based hashing. To obtain multiple binary vectors for the $i$th video, $B_i \in \{-1, 1\}^{K \times f_i}$, we compute the frame-wise feature representation as follows:

$$B_i = \text{sgn}(W^T X_i) \quad (3)$$

These conventional learning-based video hashing methods make use of hand-crafted single representation features [34], [35] and/or use a single linear projection [61], which may not effectively capture the nonlinear relationship of video representations and cannot exploit temporal information present in videos.

B. Deep Video Hashing

Our work employs a deep learning model to learn several nonlinear projections to obtain compact binary codes, where both the discriminative and temporal information of videos are exploited in an end-to-end learning framework. By doing so, we are able to learn powerful video representations in a spatial-temporal level. Unlike previous video hashing methods which either learn hashing functions from a single video feature representation or frame-by-frame, we process a set of successive frames to obtain a single binary vector which leads to a fewer number ($g_i < f_i$) of binary codes to represent the $i$th video. Therefore, we are able to minimize the number of binary codes to represent each video but still extract significant information as much as possible.

As shown in Fig. 1, given a fixed frame size $p$, we have a set of image frames $I_u \in \mathbb{R}^{p \times h \times w \times 3}$ that is passed through a series of convolution and pooling layers with $\mathbb{R}^{p \times h \times w \times 3}$ during retrieval.

3
fully connected layers at the end. By letting $s(\cdot)$ be the output at the last fully connected layer where it contains $K$ nodes, the binary code for the set of image frames of the $i$th video is computed as follows:

$$b_i = \text{sgn}(s(I_i))$$

There are two important intuitions for our DVH model: (1) By performing several nonlinear transformations with a discriminative criterion, more robust visual representation can be obtained. While kernel-based models can provide explicit nonlinear mappings, pre-defined kernel functions cannot well capture the nonlinearity of samples; (2) By performing the temporal pooling, we can implicitly exploit the relevant frames and extract a balance of global and local information from video frames. By doing so, the noisy frames which may degrade the quality of the binary codes can be implicitly ignored. We discuss how to exploit both the temporal and discriminative information in our deep architecture as follows:

1) Temporal Information: Since videos typically represent motion-based features across time, it is necessary to consider temporal information to fully obtain video-level representation. In order to exploit temporal information in deep networks, we perform pooling operations across frames between the fully-connected layers. In our work, we perform fusion of temporal information through average
pooling. We describe three deep networks with various feature pooling architectures:

**Early Fusion:** The early fusion architecture first passes image frames through the convolution and pooling layers and then fuses the information at the first fully connected layer immediately, as shown in Fig. 2(a).

**Late Fusion:** The late fusion architecture first passes image frames through the convolution and pooling layers up to the other fully-connected layers and then fuses the information at the last fully connected layer, as shown in Fig. 2(b).

**Slow Fusion:** The slow fusion architecture is a balance of the early and late fusion. Image frames are passed through the convolution and pooling layers and then fused in a hierarchical manner such that smaller temporal windows are used as it approaches the top layer. In this work, a two-stage fusion strategy is implemented. Fig. 2(c) details the architecture of this fusion strategy.

2) Discriminative and Binary Information: To learn the parameters in the deep network discriminatively, we employ the Siamese network [5] with a large-margin learning framework rather than the conventional contrastive divergence criterion [18]. Specifically, we present a new formulation which consists of two new objective criteria for binary code learning. The first objective performs discriminative learning. Specifically, given two sets of image frames, \( I_u \) and \( I_v \), we minimize the intra-class variation and maximize the inter-class variation of the binary feature representation at the top layer of these two networks, simultaneously. Given their Hamming distance \( d_{u,v}(b_u, b_v) \) at the top layer, where \( b_u = \text{sign}(s(I_u)) \) and \( b_v = \text{sign}(s(I_v)) \), we expect that \( d_{u,v} \) is small if \( u \) and \( v \) are the same class, and large if they are from different classes, which is formulated as the following constraints:

\[
\begin{align*}
    d_{u,v}(b_u, b_v) &\leq \theta_1, \quad \text{if } y_u = y_v, \\
    d_{u,v}(b_u, b_v) &\geq \theta_2, \quad \text{if } y_u \neq y_v
\end{align*}
\]

where \( \theta_1 \) and \( \theta_2 \) are the small and large thresholds, respectively.

2We found that max pooling does not give very representative information compared to average pooling.

By combining (5) and (6), we have the following formulas:

\[
\delta_{u,v}(\theta - d_{u,v}(b_u, b_v)) > 1 \quad (7)
\]

where \( \theta_1 = \theta - 1 \) and \( \theta_2 = \theta + 1 \), and \( \delta_{u,v} = 1 \) means that \( u \) and \( v \) are from the same class, and \( \delta_{u,v} = -1 \) indicates that they are from different classes.

This leads to the following objective function:

\[
J_1 = f(1 - \delta_{u,v}(\theta - d_{u,v}(b_u, b_v))) \quad (8)
\]

where \( f(z) \) is a generalized logistic loss function which is a smooth approximation of the hinge loss function: \( z = \max(z, 0) \), and defined as follows:

\[
f(z) = \frac{1}{\rho} \log(1 + \exp(\rho z)) \quad (9)
\]

where \( \rho \) is the sharpness parameter set to 10 and \( \theta \) is the threshold parameter set to \( K/4 \).

The second objective is to ensure efficient binary codes by minimizing the quantization loss [16] between real-valued codes and binary codes as follows:

\[
J_2 = \|s(I_u) - b_u\|_F^2 + \|s(I_v) - b_v\|_F^2 \quad (10)
\]

Hence, the final objective function for our DVH is formulated as:

\[
\min_{b_u, b_v} J = J_1 + \lambda J_2 \\
= f(1 - \delta_{u,v}(\theta - d_{u,v}(b_u, b_v))) \\
+ \lambda(\|s(I_u) - b_u\|_F^2 + \|s(I_v) - b_v\|_F^2)(11)
\]

where \( J_1 \) exploits the discriminative information, \( J_2 \) minimizes the quantization loss, and \( \lambda \) is a constant parameter which balances the two criterions.

We use the standard mini-batch gradient descent and back-propagation to solve the optimization problem. We first relax the binary constraints in the first objective and use the signed magnitude real codes during back-propagation. Given \( h_u = s(I_u) \) and \( h_v = s(I_v) \) are the real-value code values from the top layer, the back-propagation is implemented first by taking the derivative of \( J \) with respect
C. Extracting Binary Codes from One Video

Since the number of frames are usually different for different videos, we extract multiple binary codes for each video by using our DVH approach through a set of consecutive image frames in the video with a specific stride. Fig. 3 shows the hashing procedure for one video by using our model. The over-all hamming distance, $D_H$, for a pair of videos are then obtained by getting the average of the hamming distances, $d_H$, for each pair of binary codes as follows:

$$D_H(X_i, X_j) = \frac{1}{g_i g_j} \sum_{u=1}^{g_i} \sum_{v=1}^{g_j} d_H(b_u, b_v)$$

where $g_i$ and $g_j$ is the number of frame sets for videos $X_i$ and $X_j$. By doing so, we are able to compare the similarity of videos with less computation than using frame to frame comparisons, which are more representative than using single features for each video.

D. Implementation Details

Our deep network composes of a stacked convolutional and pooling layers with parameters obtained from pre-trained models, and connected to a series of fully connected layers such that the top-most layer contains $K$-dimensional features. The hidden fully connected layers use rectified linear unit (ReLU) as the activation function, while the top-most layer has a hyperbolic tangent activation to ensure centered feature and have balanced [-1,1] values. The parameters in the fully connected layers are initialized using the Xavier initialization [15]. To be consistent, all models have a fully-connected layers of dimensions $[4096 \rightarrow 500 \rightarrow 200 \rightarrow K]$. To avoid over-fitting, we enable training only in the fully connected layers. Our deep architecture and experiments and were implemented under the MatConvNet [54] framework. The learning rate, momentum, and weight decay were set to 0.002, 0.9, and 0.0001, respectively. Table I summarizes the implementations of different DVH methods after the stacked convolutional and pooling layers. At the training stage, we iteratively passed through all the training videos where we randomly chose video pairs. For each video pair, we randomly chose a set of $p$ successive frames and then packed them into batches to pass into the network. We ensure that the positive and negative pairs for each batch are in an approximate 1:2 ratio. The training procedure converged when the loss does not change within a certain threshold for an epoch. For all experiments, $\rho$ was set to 10 based on the empirical testing to obtain a smooth approximation for the Hinge Loss. We experimented with different values, and found that the results appear to be particularly insensitive to $\rho$.

<table>
<thead>
<tr>
<th>Early Fusion</th>
<th>Late Fusion</th>
<th>Slow Fusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>fc7 - 4096</td>
<td>fc7 - 4096</td>
<td>fc7 - 4096</td>
</tr>
<tr>
<td>pool at $p$ frames</td>
<td>-</td>
<td>pool at $p_1$ frames</td>
</tr>
<tr>
<td>fc8 - 500</td>
<td>fc8 - 500</td>
<td>fc8 - 500</td>
</tr>
<tr>
<td>-</td>
<td>pool at $p$ frames</td>
<td>-</td>
</tr>
<tr>
<td>fc9 - 200</td>
<td>fc9 - 200</td>
<td>fc9 - 200</td>
</tr>
<tr>
<td>$K$</td>
<td>$K$</td>
<td>$K$</td>
</tr>
</tbody>
</table>

A. Datasets and Experimental Settings

Columbia Consumer Video (CCV) dataset [28]: It consists of 9,317 videos with an average duration of 80 seconds extracted from YouTube. The videos were categorized to 20 different categories such as basketball, wedding and music performance. Similar to [61], we sampled frames every 2 seconds and ensured that each video had a minimum of 30 frames. Since most of the categories in this dataset are events, the videos contain large variations among frames making the task very challenging. In our experiments, we randomly selected 20 videos per category for training, 25 videos per category as the query data, and 100 videos per category as the gallery data. This results in 400, 500, and 2000 videos for the training, query, and gallery sets, respectively.

For our deep model, we used the pre-trained VGG-net [51] as our stacked convolution and pooling layers. We used a batch size of 200 and a frame size of $p = 10$. For our Slow Fusion architecture, the first pooling layer fuses the data of $p_1 = 5$ frames with a stride of 2, the second pooling layer fuses the data of the final $p_2 = 3$ frames. For testing samples, we obtained the binary codes for each video at a frame stride of 5.

Joint-annotated HMDB (JHMDB) dataset [24]: It consists of 928 action videos having 36 to 55 frames per video, which was taken from the HMDB dataset [32] for human motion recognition. The action videos are categorized into 21 human actions such as brushing hair, clapping, and climbing. Although action recognition makes use of flow and RGB information [4], we only used the full body optical flow representation for simplicity. In our experiments, we randomly selected 10 videos per category as training samples, 10 videos per category as query samples, and 20
Fig. 4. Two sample videos for the Columbia Consumer Video (CCV) database. The first video belongs to the music performance category, while the second video belongs to the baseball category.

Fig. 5. Two sample videos for the Joint-annotated HMDB (JHMDB) database. The first video represents a shooting action video, while the second video represents a catching action video.

videos per category as gallery samples. This results in 210, 210, and 420 videos for the training, query, and gallery sets, respectively.

For our deep model, we used the CNN motion network by [14] as our pre-trained stacked convolution and pooling layers. We used a batch size of 50 and a frame size of $p = 10$. For our Slow Fusion architecture, the first pooling layer fuses the data of $p_1 = 5$ frames with a stride of 2, the second pooling layer fuses the data of the final $p_2 = 3$ frames. For testing samples, we obtained the binary codes at a stride of 2.

B. Evaluation Metrics

To measure the performance of our DVH, we used the Hamming ranking and Hamming look-up as evaluation metrics to compare the performance of different methods. For Hamming ranking, the mean Average Precision (mAP) and Precision@N are evaluated. The mAP is defined as the mean of the average precision of the top retrieved samples across all queries, while Precision@N is defined as the percentage of true labels among the top N retrieved samples. For Hamming look-up, the precision when the hamming radius is set as $r = 2$ is evaluated where it measures the precision over all the samples that is within a hamming radius of $r = 2$. At $K = 64$, Hamming look-up precision is not evaluated because it will be impractical for longer bit lengths.

C. Experimental Results

Comparison with Different Deep Baselines: We first compared our DVH with three baseline deep architectures which do not use temporal fusion in the fully-connected layers. The baseline methods are described as follows:

**Single-Frame:** In the single-frame model, we considered each frame of the video as a single image with its own label information. Similar to DVH, we used the large-margin criterion for the Siamese network to learn the parameters. However, we only used single frames as the input and do not perform temporal fusion. The cost function is defined as:

$$J = f(1 - \delta_{u,v}(\theta - d_{u,v}(s(x_u), s(x_v))))$$

**Single-Frame + Temporal:** In the single-frame + temporal model, we exploited the temporal information with the same large-margin criterion so that the frames which are close to each other are similar as much as possible, defined as below:

$$J = f(1 - \delta_{u_1,v}(\theta - d_{u_1,v}(s(x_{u_1}), s(x_v)))) + \nu \|s(x_{u_1}) - s(x_{u_2})\|^2_2$$

where $\nu$ is the balancing term, and $u_1$ and $u_2$ are two randomly selected image frames from the same video which are apart by a minimum of five frames. In the experiments, we used $\nu = 0.1$. Fig. 6 shows the architectures of the first two baseline methods.

**Video-Level Feature:** In this model, we pooled all frame-level features from the single-frame deep model to compute video-level features to evaluate the large margin criterion. By doing so, we obtained a representative global binary vector for each video.

Tables II and III show the performance of different methods on the CCV and JHMDB datasets, respectively. As can be seen, our DVH architectures outperform the other deep
Fig. 6. Deep baseline architectures based on frame-by-frame training for video hashing. The first baseline is similar to image hashing where each frame is a single image. The second baseline adds a temporal criterion during training such that given two frames that are temporally close should have a similar compact feature as much as possible.

TABLE II
RESULTS ON THE CCV DATASET IN COMPARISON WITH THE BASELINE DEEP ARCHITECTURE.

<table>
<thead>
<tr>
<th>Method</th>
<th>Hamming ranking (mAP, %)</th>
<th>precision (%) @ N = 100</th>
<th>precision (%) @ r=2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>16</td>
<td>32</td>
<td>64</td>
</tr>
<tr>
<td>Single</td>
<td>32.62</td>
<td>34.23</td>
<td>35.02</td>
</tr>
<tr>
<td>Single+Temporal</td>
<td>33.60</td>
<td>35.60</td>
<td>37.27</td>
</tr>
<tr>
<td>Video-Level</td>
<td>29.45</td>
<td>30.79</td>
<td>29.19</td>
</tr>
<tr>
<td>Early Fusion</td>
<td>37.18</td>
<td>40.86</td>
<td>41.54</td>
</tr>
<tr>
<td>Late Fusion</td>
<td>38.54</td>
<td>41.08</td>
<td>41.51</td>
</tr>
<tr>
<td>Slow Fusion</td>
<td>38.27</td>
<td>40.80</td>
<td>41.41</td>
</tr>
</tbody>
</table>

TABLE III
RESULTS ON THE JHMDB DATASET IN COMPARISON WITH THE BASELINE DEEP ARCHITECTURE.

<table>
<thead>
<tr>
<th>Method</th>
<th>Hamming ranking (mAP, %)</th>
<th>precision (%) @ N = 10</th>
<th>precision (%) @ r=2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>16</td>
<td>32</td>
<td>64</td>
</tr>
<tr>
<td>Single</td>
<td>32.73</td>
<td>33.89</td>
<td>31.74</td>
</tr>
<tr>
<td>Single+Temporal</td>
<td>33.19</td>
<td>34.85</td>
<td>35.58</td>
</tr>
<tr>
<td>Video-Level</td>
<td>30.07</td>
<td>30.95</td>
<td>32.53</td>
</tr>
<tr>
<td>Early Fusion</td>
<td>35.19</td>
<td>37.43</td>
<td>37.95</td>
</tr>
<tr>
<td>Late Fusion</td>
<td>34.93</td>
<td>36.78</td>
<td>37.53</td>
</tr>
<tr>
<td>Slow Fusion</td>
<td>34.86</td>
<td>36.59</td>
<td>36.63</td>
</tr>
</tbody>
</table>

learning based baseline architectures. It is interesting to see that the second baseline (Single+Temporal) beats the first baseline (Single), which shows that temporal information is important. The video-level features also yielded competitive representations but often achieved worse performance than the single-frame deep model. The late fusion and early fusion DVH architectures obtain the best performance on the CCV and JHMDB datasets, respectively. For CCV, a late fusion would mean it prefers to exploit high-level global information, while for JHMDB an Early Fusion shows it prefers to exploit local motion information. This is reasonable because JHMDB focuses on action information while CCV is more on events which are generally holistic.

We also examined the retrieval time of different deep baseline architectures, which is shown in Fig. 7. As can be seen, a longer retrieval time is necessary since the deep baseline architectures transform each image frame into a binary code. Differently, our DVH method performs temporal fusion so fewer binary code comparisons are implemented resulting in a faster retrieval time for the whole query-gallery set. This is most obvious on the CCV dataset since more frames are present in a single video, and larger gallery and query videos are used.

Comparison with State-of-the-Art Learning-based...
Hashing methods: We also compared our DVH method with several popular hashing methods including PCA Hashing [55], PCA-ITQ [16], Anchor Graph Hashing (AGH) [40], Kernel Supervised Hashing (KSH) [39], CCA-ITQ [16], and FastHash [36]. Specifically, PCAH, PCA-ITQ and AGH are unsupervised hashing methods, and KSH, CCA-ITQ, and FastHash exploit the label information of samples to learn discriminative hash codes. The standard implementations of all methods are from the original authors and the default parameters were set based on their respective papers. For consistency, the experiments were carried out with the same selected training, gallery and query sets. For the different hashing methods being compared, we considered each frame as an image and encoded its respective binary code based on the 4096-dimension CNN feature obtained from the fully-connected layer of the pre-trained models used, and defined the hamming distance of two videos as the average of all hamming distances between images from each video.

Tables IV and V show the performance of different hashing methods on the CCV and JHMDB experiments, respectively. We found that the DVH architecture yielded the best performance, where the Late fusion was for the CCV dataset, and the Early fusion was for the JHMDB dataset, respectively. As can be seen, our method consistently outperforms the other existing hashing methods. Most surprising is the hamming look-up precision (HLP) evaluation results which show significant improvement consistently across varying bit lengths. This shows that representing the video in a deep nonlinear binary feature vector gives strong representation for retrieval. Figs. 8-11 show the recall and precision curve, and precision curves vs the retrieval number $N$ on the CCV and JHMDB datasets. We see that our method outperforms the compared methods in most scenarios.

**Comparison with Different Video Hashing Meth-**
Fig. 8. Precision-Recall (PR) curves on the CCV dataset versus varying code lengths.

Fig. 9. Precision-N curves on the CCV dataset versus varying code lengths.

Fig. 10. Precision-Recall (PR) curves on the JHMDB dataset versus varying code lengths.

Fig. 11. Precision-N curves on the JHMDB dataset versus varying code lengths.
Parameter Analysis: We also analyzed the varying values of $\lambda$ during training to see the contribution of the two criterions in the over-all performance of our DVH method. Fig. 12 shows the mAP performance of our DVH method at varying $\lambda$ for the 64-bit experiment on the (a) CCV and (b) JHMDB datasets. respectively.

**D. Discussion**

The above experimental results suggest the following three key observations:

1) Our deep video hashing method achieves very competitive performance compared to other deep baseline architectures which shows that performing temporal fusion during training contributes well to the over-all performance. In addition, retrieval time is also
reduced because of the temporal fusion.
2) Our DVH outperforms state-of-the-art image-based hashing methods which shows that the binary codes obtained from our hashing method are strong representations due to the discriminative training we employed. Furthermore, our DVH also outperforms other video hashing methods by a large margin.
3) The large-margin criterion yields the largest contribution in our DVH method. However, the binary quantization term also provides improvements in the over-all performance. For the parameter $p$, we see that the best performance can be obtained when the parameter of $p$ is set to 10 because it is a good balance of extracting global and local video features.

V. CONCLUSION

In this paper, we have proposed a deep video hashing approach with various frame pooling architectures to learn binary codes for each video in a deep framework such that both temporal and discriminative information are well exploited. Experimental results on two video databases clearly demonstrate that our method achieved better performance with the state-of-the-art hashing methods.

There are two interesting directions for future work:
1) Our DVH method composed of frame-level pooling layers to exploit temporal information. It is interesting to incorporate more complex networks such as recurrent neural networks (RNN) [47], long short term memory (LSTM) [19] and 3D-CNNs [26] to further improve the performance.
2) In this work, we learned our DVH network using supervised information. Hence, it is interesting to learn a deep network using quantization-based [16], [23] criterions, which does not exploit label information.

ACKNOWLEDGEMENT

This work is supported by the National Key Research and Development Program of China under Grant 2016YFB1001001, the National Natural Science Foundation of China under Grants 61672306, and the National 1000 Young Talents Plan Program. Partial of this work was carried out at the Rapid-Rich Object Search (ROSE) Lab at the Nanyang Technological University Singapore.

REFERENCES

[37] Z. Lin, G. Ding, M. Hu, and J. Wang. Semantics-preserving hashing


