# Play and Rewind: Optimizing Binary Representations of Videos by Self-Supervised Temporal Hashing

Hanwang Zhang[†] Meng Wang[‡] Richang Hong[‡] Tat-Seng Chua[†]
[†]National University of Singapore [‡]Hefei University of Technology
{hanwangzhang,eric.mengwang,hongrc.hfut}@gmail.com; dcscts@nus.edu.sg

## ABSTRACT

We focus on hashing videos into short binary codes for efficient Content-based Video Retrieval (CBVR), which is a fundamental technique that supports access to the ever-growing abundance of videos on the Web. Existing video hash functions are built on three isolated stages: frame pooling, relaxed learning, and binarization, which have not adequately explored the temporal order of video frames in a joint binary optimization model, resulting in severe information loss. In this paper, we propose a novel unsupervised video hashing framework called Self-Supervised Temporal Hashing (SSTH) that is able to capture the temporal nature of videos in an end-to-end learning-to-hash fashion. Specifically, the hash function of SSTH is an encoder RNN equipped with the proposed Binary LSTM (BLSTM) that generates binary codes for videos. The hash function is learned in a self-supervised fashion, where a decoder RNN is proposed to reconstruct the original video frames in both forward and reverse orders. For binary code optimization, we develop a backpropagation rule that tackles the non-differentiability of BLSTM. This rule allows efficient deep network training without suffering from the binarization loss. Through extensive CBVR experiments on two real-world consumer video datasets of Youtube and Flickr, we show that SSTH consistently outperforms state-of-the-art video hashing methods, *e.g.*, in terms of mAP@20, SSTH using only 128 bits can still outperform others using 256 bits by at least 9% to 15% on both datasets.

## Keywords

Temporal Hashing;Binary LSTM;Sequence Learning;Video Retrieval

## 1. INTRODUCTION

Content-based retrieval—a technique focusing on the indexing and querying of a large data collection based on visual content—is the key to many multimedia applications [17]. Unlike Content-based Image Retrieval (CBIR) that has been extensively studied in the past decades [24, 5, 42], Content-based Video Retrieval (CBVR) has not received sufficient attention in Multimedia community [25, 14]. However, due to the popularity of mobile video capturing devices and high-speed network transmission, we are witnessing the rapid growth of videos and video-related Web services such as Vine and Snapchat. Indeed, during the time of reading this paragraph, around 50 thousand video snippets are shared and 2.4 million videos are viewed on Snaptchat[1]. Without a doubt, the ever-growing abundance of videos on the Web has brought about an urgent need for more advanced CBVR technologies [18, 39, 40, 37].

Video is beyond a set of frames. However, most current works on video analytics generally resort to pooling frame-level features into a single video-level feature by discarding the temporal order of the frame sequence[2]. Such bag-of-frames degeneration works well when high-dimensional frame-level features such as CNN responses [35] and motion trajectories [31] are used, as certain temporal information encoded in a high dimension can be preserved after pooling. However, for large-scale CBVR, where hashing (or indexing) of these high-dimensional features as short binary codes is necessary, the temporal information loss caused by frame pooling will inevitably result in suboptimal binary codes of videos. The loss usually takes place in the process of hash function learning [33], which is a post-step after pooling; as compared to dominant video appearances (*e.g.*, objects, scenes and short-term motions), nuanced video dynamics (*e.g.*, long-term event evolution) are more likely to be discarded as noise in the drastic feature dimensionality reduction during hashing [7].

We argue that the key reason to the above defect is that both the temporal pooling and the hash code learning steps have not adequately addressed the temporal nature of videos. To this end, we propose a novel video hashing method for CBVR called **S**elf-**S**upervised **T**emporal **H**ashing (SSTH), which especially tackles this defect. In a nutshell, SSTH is an end-to-end system that encodes an $m$-frame video into a single $k$-bit binary code. We highlight three key characteristics that make SSTH effective and distinguishable from other state-of-the-art video hashing methods [26, 3, 38]:

**Self-Supervision**. Most existing unsupervised hashing methods are not temporal-aware and hence will lose long-

*MM '16, October 15-19, 2016, Amsterdam, Netherlands*
© 2016 ACM. ISBN 978-1-4503-3603-1/16/10...$15.00
DOI: http://dx.doi.org/10.1145/2964284.2964308

[1]https://www.snapchat.com/ads

[2]Though some approaches represent videos as a set of frame features (*e.g.*, TRECVID Instance Search Task [20]), they essentially fall into the pooling approach since the ultimate similarity calculation is temporal-independent

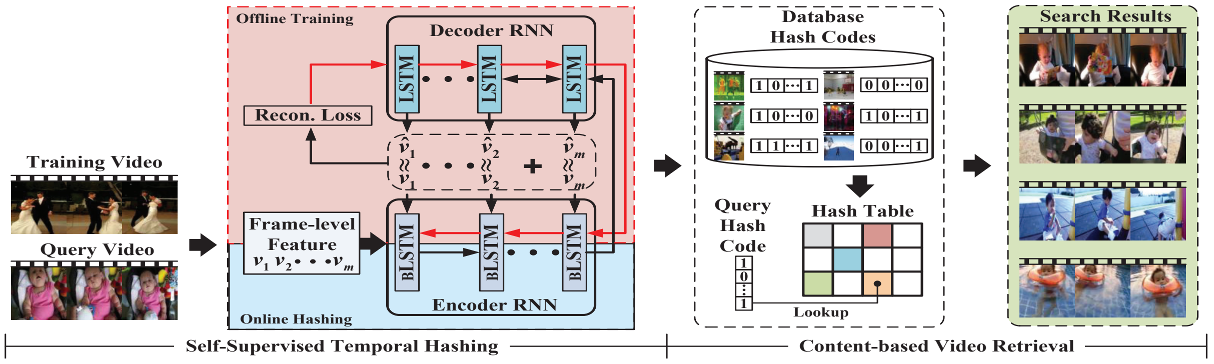


**Figure 1: The overview of the proposed Self-Supervised Temporal Hashing for Content-based Video Retrieval. In offline training, the black and red arrows denote forward and back propagation, respectively. In particular, the two-way black arrow in Decoder RNN denotes the forward and reverse reconstruction.**

term video dynamics. So, we explore an alternative unsupervised hashing: how can we exploit the frame order of a video to *self-supervise* the binary code learning? To this end, we propose a strategy dubbed **Play** and **Rewind** within an encoder-decoder Recurrent Neural Network (RNN) framework [27, 28]—after the RNN hash function encodes a video (*i.e.*, "play"), the output hash code should be able to decode the viewed frames in a certain order (*i.e.*, "rewind"), only if the code successfully encodes both video appearances and dynamics. In this way, SSTH is able to continuously refine itself by playing and rewinding the inexhaustible amount of videos on the Web.

**Temporal Awareness**. The hash function of SSTH explicitly encodes the temporal order of frames by using an RNN, which has shown to be especially effective in sequence modeling [27]. In particular, we propose a novel recurrent unit called **B**inary **L**ong-**S**hort **T**erm **M**emory (BLSTM), where the video binary code at time $t$ is a function of the code at time $t-1$. RNN equipped with BLSTM unifies temporal modeling and video hashing in a principled way, where the binary codes are expected to capture the long-term dynamics of the entire video.

**Optimized Video Binary Representation**. Since the problem of binary code learning is essentially NP-hard [13], existing video hashing methods generally follow a three-stage process: pooling, relaxation, and binarization. The isolated steps make the approach suboptimal. In the proposed SSTH framework, we cast the temporal modeling and binary code learning into a joint model. Specifically, we develop a binary backpropagation rule which tackles the binary nature of SSTH without any relaxation. In this way, SSTH can be considered as an end-to-end unsupervised learning framework for optimizing the transformation from videos to binary codes.

The overview of the proposed SSTH framework for CBVR is illustrated in Figure 1. In the offline training stage, a training video is represented by a sequence of frame-level features (*e.g.*, by deep CNN [23]). The encoder RNN with BLSTM runs through the sequence, generating a set of hash codes and then the decoder RNN decodes it to reconstruct the frame-level feature sequence in both forward and reverse orders. During optimization, the reconstruction error is back propagated through the entire encoder-decoder RNNs. In the online retrieval stage, the encoder RNN can be considered as a temporal-aware hash function, which generates binary hash codes for both database videos and query videos. Finally, the database hash codes are indexed into a hash table for practical retrieval. The effectiveness of SSTH is demonstrated on two real-world consumer video datasets from Youtube and Flickr.

The contributions of this paper are as follows:

**1)** We propose a novel unsupervised video hashing framework called Self-Supervised Temporal Hashing (SSTH). To the best of our knowledge, SSTH is the first principled deep framework for video hashing. It is an optimized end-to-end approach that addresses a variety of weakness in conventional video hashing approaches, such as the ignorance of temporal nature and the isolation of pooling, relaxation and binarization.

**2)** We develop a novel LSTM variant dubbed Binary LSTM (BLSTM), which serves as the building block of the temporal-aware hash function. We also develop an efficient backpropagation rule that directly tackles the challenging problem of binary optimization for BLSTM without any relaxation.

**3)** We improve the conventional one-order encoder-decoder RNN [27, 28] by incorporating forward and reverse order frame reconstruction. Our strategy is a novel unsupervised learning objective that better models the temporal nature of data.

It is also worth mentioning that, although SSTH is developed for video hashing, it is actually a flexible and generic framework that can be easily extended to deal with the hashing of other signal sequences such as music and text.

## 2. RELATED WORK

By hashing data into short binary codes, efficient storage and search can be achieved due to the fast bit XOR operations in Hamming space. Today's hashing methods have evolved from hand-crafted hash functions [9] to data-driven learning-to-hash [33], which either requires pairwise data similarities for unsupervised learning [34, 11, 41] or data labels for supervised learning [32, 22]. However, due the challenging temporal nature of videos—to our best knowledge—limited video hashing studies have been carried out. For example, the hash functions proposed by Song *et al.* [26] and Cao *et al.* [3] are unaware of the temporal order of video frames. Ye *et al.* [38] exploited the pairwise frame order but their method requires video labels and only generates frame-level codes. Although Revaud *et al.* [21] exploited the short-term temporal order, their video quantization codes are not binary. In sharp contrast to the above methods, our SSTH is an unsupervised binary code learning framework that explicitly exploits the long-term video temporal information.

The encoder-decoder RNN framework used in SSTH is inspired by recent advances in sequence learning, where RNN has shown great success in machine translation [28] and image/video caption generation [30]. In particular, our architecture is most related to the unsupervised video representation learning work by Srivastava *et al.* [27]. However, we distant from them by developing Binary LSTM (BLSTM) in place of the original LSTM [12] for binary code learning and a forward/reverse reconstruction strategy for self-supervised learning. By viewing SSTH as a deep network for hashing, our work also relates to recent studies on deep learning-to-hash [8, 43, 6]. However, besides the fact that none of the existing methods considers the temporal nature of videos, they also resort to relax the binary optimization to real-valued optimization or merely append a quantization loss minimizer at the top layer [8]. In contrast, our deep hashing architecture is re-designed with BLSTM for direct binary optimization by efficient backpropagation without relaxation.

## 3. SELF-SUPERVISED TEMPORAL HASHING

In this section, we formulate the proposed Self-Supervised Temporal Hashing (SSTH) framework. First, we introduce the proposed temporal-aware hash function that yields a single compact binary code for a video sequence. Then, we present a novel recurrent unit called Binary LSTM (BLSTM), which is the building block for the hash function. Finally, we introduce the proposed self-supervised strategy for learning to hash and its deep architecture.

### 3.1 Temporal-aware Hash Function

Suppose a video sequence of $m$ frames is denoted as a matrix $\mathbf{V} = (\mathbf{v}_1, ..., \mathbf{v}_m) \in \mathbb{R}^{d\times m}$, where the $t$-th frame is represented by a feature vector $\mathbf{v}_t \in \mathbb{R}^d$. Our goal is to find a hash function $H : \mathbb{R}^{d\times m} \to \{\pm1\}^k$ that encodes $\mathbf{V}$ into a $k$-bit binary code[3] $\mathbf{b}_m \in \{\pm1\}^k$, where $k \ll d$. In order to capture the temporal nature of videos, we require the hash function to be temporal-aware, *i.e.*, if $\mathbf{V}'$ is a matrix that is column-permuted from $\mathbf{V}$, $\mathbf{b}'_m$ is not necessarily equal to $\mathbf{b}_m$. In other words, rather than assuming $\Pr(\mathbf{b}_m|\mathbf{V}) = \Pr(\mathbf{b}_m|\mathbf{V}')$ (*e.g.*, pooling-based hashing [3]) or $\Pr(\mathbf{b}_m|\mathbf{V}) = \prod_{t=1}^{m}\Pr(\mathbf{b}_t|\mathbf{v}_t)$ (*e.g.*, frame-level hashing [26, 38]), a temporal-aware hash function models $\Pr(\mathbf{b}_t|\mathbf{v}_1, ..., \mathbf{v}_t) = \Pr(\mathbf{b}_t|\mathbf{b}_{t-1}, \mathbf{v}_t)$, where we assume that the previous code $\mathbf{b}_{t-1}$ is statistically sufficient to represent the previous frames $\mathbf{V}_{t-1}$. So, the hash code $\mathbf{b}_t$ of a $t$-length frame sequence is dependent on the code $\mathbf{b}_{t-1}$ of the last viewed $(t-1)$ frames and the current frame $\mathbf{v}_t$.

It is natural to use a Recurrent Neural Network (RNN) [12] to fit the temporal-aware requirement. In particular, the output $\mathbf{b}_t$ of a recurrent unit (or layer) at $t$-th time is expressed by a non-linear function $f$, whose input includes the last output $\mathbf{b}_{t-1}$ and the current frame $\mathbf{v}_t$:

$$\mathbf{b}_t = f(\mathbf{b}_{t-1}, \mathbf{v}_t). \tag{1}$$

[3]It is trivial to transform binary code $b \in \{\pm1\}$ to $b \in \{0, 1\}$ by $b \leftarrow (1 + b)/2$.

Therefore, the resultant binary code $\mathbf{b}_m$ for video $\mathbf{V}$ can be generated recurrently by:

$$\mathbf{b}_m = H(\mathbf{V}) = f(\mathbf{b}_{m-1}, \mathbf{v}_m). \tag{2}$$

The design of $f$ is crucial to the hash function in an RNN style. Recently, Long-Short Term Memory (LSTM) has shown state-of-the-art performance on sequence learning tasks, due to its capability in dealing with the vanishing and exploding gradient issues of deep RNN (*i.e.*, long sequence) [12]. However, original LSTM can only generate a real-valued hidden variable rather than a binary code, *e.g.*, $\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{v}_t)$, where $\mathbf{h}_t \in \mathbb{R}^k$. In order to modify LSTM to generate binary codes, a straightforward approach, which is similar to some hashing deep networks [8, 43], is to use sgn function to binarize the resultant hidden variable, *e.g.*, $\mathbf{b}_m = \mathrm{sgn}(\mathbf{h}_m)$, where $\mathrm{sgn}(x) = 1$, if $x \geq 0$; and $\mathrm{sgn}(x) = -1$ otherwise. However, we argue that it is essentially based on frame pooling, where the pooling function is an RNN: even though the pooling is temporal-aware, the hash codes *per se* do not directly capture the temporal nature of videos.

### 3.2 Binary LSTM

In order to design a hash function that not only inherits the numerical stability of LSTM but also generates binary codes, we propose a novel variant of LSTM named Binary LSTM (BLSTM). As illustrated in Figure 2(a), BLSTM follows a similar data flow as LSTM. First, the input variable $\mathbf{z}_t$ is calculated by tanh-squashing the linear combination of the current feature $\mathbf{v}_t$ and the last binary code $\mathbf{b}_{t-1}$. Then, the memory variable $\mathbf{c}_t$ is updated by adding how much the "old knowledge" $\mathbf{c}_{t-1}$ should forget and how much the "new knowledge" $\mathbf{z}_t$ should memorize. Note that this additive memory update design enables the derivatives distribute over sums and hence the error does not vanish quickly when performing backpropagation over time [12]. Then, the memory is batch normalized (BN). Finally, the hash code $\mathbf{b}_t$ is a binarization (sgn) of how much the "current knowledge" $\mathbf{h}_t$ should be output. The behaviors of "forget", "memorize", and "output" are respectively controlled by three gate variables: forget gate $\mathbf{F}$, input gate $\mathbf{I}$, and output gate $\mathbf{O}$. They are applied element-wise multiplicatively and thus can either keep a value from the gated variable if the gate is 1 or discard a value if the gate is 0. In particular, all of the gate variables: $\mathbf{i}_t$, $\mathbf{f}_t$, and $\mathbf{o}_t$, use sigmoid function $\sigma$ as an estimation of the pass-through probability. The detailed implementation of BLSTM in Eq. (1) is given as follows:

$$\mathbf{z}_t \leftarrow \tanh\left(\mathbf{W}_{vz}\mathbf{v}_t + \mathbf{w}_{bz} \circ \mathbf{b}_{t-1} + \mathbf{a}_z\right) \tag{3a}$$

$$\mathbf{f}_t \leftarrow \sigma\left(\mathbf{W}_{vf}\mathbf{v}_t + \mathbf{W}_{bf}\mathbf{b}_{t-1} + \mathbf{w}_{cf} \circ \mathbf{c}_{t-1} + \mathbf{a}_f\right) \tag{3b}$$

$$\mathbf{i}_t \leftarrow \sigma\left(\mathbf{W}_{vi}\mathbf{v}_t + \mathbf{w}_{bi}\mathbf{b}_{t-1} + \mathbf{w}_{ci} \circ \mathbf{c}_{t-1} + \mathbf{a}_i\right) \tag{3c}$$

$$\mathbf{c}_t \leftarrow \mathbf{f}_t \circ \mathbf{c}_{t-1} + \mathbf{i}_t \circ \mathbf{z}_t \tag{3d}$$

$$\mathbf{c}_t \leftarrow \textsc{BatchNorm}(\mathbf{c}_t) \tag{3e}$$

$$\mathbf{o}_t \leftarrow \sigma\left(\mathbf{W}_{vo}\mathbf{v}_t + \mathbf{W}_{bo}\mathbf{b}_{t-1} + \mathbf{w}_{co} \circ \mathbf{c}_t + \mathbf{a}_o\right) \tag{3f}$$

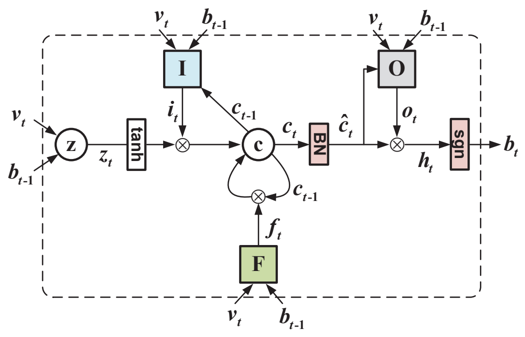
$$\mathbf{h}_t \leftarrow \mathbf{o}_t \circ \mathbf{c}_t \tag{3g}$$

$$\mathbf{b}_t \leftarrow \mathrm{sgn}\left(\mathbf{h}_t\right) \tag{3h}$$

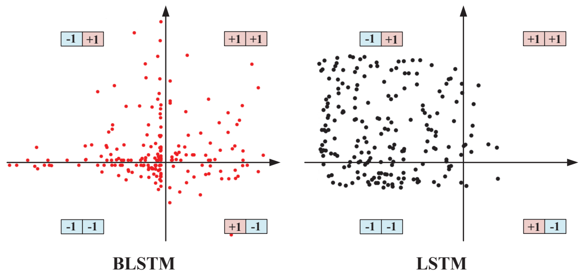
where $\circ$ denotes the element-wise multiplication.

Now we discuss the motivation behind the two novel designs of the proposed BLSTM as compared to LSTM:

**Binarization (sgn).** This allows BLSTM yielding $\{\pm1\}$ binary output. Different from the popular vanilla LSTM [12],



(a)



(b)

**Figure 2: (a) The data flow of the proposed Binary LSTM (bLSTM) at the $t$-th time. (b) Two dimensions of $\mathbf{h}_t$ of the proposed BLSTM (left) and LSTM (right). We can see that BLSTM yields more decorrelated and balanced binary codes as compared to LSTM.**

where $\mathbf{h}_t \leftarrow \mathbf{o}_t \circ \tanh(\mathbf{c}_t)$, we remove the tanh squash $\mathbf{c}_t$ before entering the sgn function as in Eq. (3g). Intuitively, as sgn only cares for the sign of $\mathbf{h}_t$, squashing the input to $[-1, +1]$ is unnecessary. In fact, since the gradient of tanh is always nonzero, any small change in the input modifies most of the entries in memory $\mathbf{c}_t$. Therefore, as we will discuss later in Section 4.1, when the tanh before $\mathbf{c}_t$ is removed, small input changes will not modify the memory values if they are already definitive on its sign, *e.g.*, $|\mathbf{c}_t| > \mathbf{1}$. In this way, BLSTM can be considered as a sparse model that contributes better disentangling ability for data variance and efficient training [10].

**Batch Normalization (BN).** It has shown to be effective in removing layer-wise covariance shift in deep neural networks [15]. It normalizes $\mathbf{c}_t$ to zero-mean unit-variance Gaussian distribution and then a linear transformation: $c_t \leftarrow \frac{c_t - \mu_t}{\sigma_t}$, $c_t \leftarrow \gamma_1 c_t + \gamma_2$, where $c_t$ is any entry of $\mathbf{c}_t$, $\mu_t$ and $\sigma_t$ are the mean and unbiased variance of the training minibatch at time $t$, $\gamma_1$ and $\gamma_2$ are trainable transformation parameters. Note that in testing, $\mu_t$ and $\sigma_t$ are estimated from the whole training data. By denoting matrix $\mathbf{C}_t$ as the memory variables of a minibatch, BN approximates the constraints $\mathbf{C}_t^T\mathbf{C}_t = \mathbf{I}$ and $\mathbf{C}_t^T\mathbf{1} = \mathbf{0}$, which respectively impose decorrelation (*i.e.*, bits should be as independent as possible) and balance (*i.e.*, each bit should split the data as balanced as possible) on the resultant codes $\mathbf{B}_t = \text{sgn}(\mathbf{O}_t \circ \mathbf{C}_t)$. In fact, these constraints are shown to maximize the information entropy in binary codes [34], *i.e.*, BN helps BLSTM yielding informative binary codes (cf. Figure 2(b)).

## 3.3 Learning Objective

We use the temporal order of the video sequence as a self-supervision for learning to hash. Essentially, our learning objective falls in the RNN encoder-decoder framework for unsupervised sequence learning [28, 27]. In particular, we propose to implement the framework as follows. First, an encoder RNN with BLSTM first runs through a video sequence $(\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_m)$ to come up with a hash code $\mathbf{b}_m$. Then, a decoder composed of two independent RNNs with LSTM: forward and reverse reconstruction RNNs, decodes the hash code to features that reconstruct the input frames in the forward order $(\tilde{\mathbf{v}}_1, \tilde{\mathbf{v}}_2, ..., \tilde{\mathbf{v}}_m)$ and reverse order $(\hat{\mathbf{v}}_m, \hat{\mathbf{v}}_{m-1}, ..., \hat{\mathbf{v}}_1)$, respectively. Specifically, we adopt linear reconstructions for the output of the decoder LSTMs:

$$\begin{cases} \tilde{\mathbf{v}}_t = \tilde{\mathbf{W}}\tilde{\mathbf{h}}_t + \tilde{\mathbf{a}}, \ \tilde{\mathbf{h}}_t = \tilde{f}(\tilde{\mathbf{h}}_{t-1}, \mathbf{0}), \\ \hat{\mathbf{v}}_t = \hat{\mathbf{W}}\hat{\mathbf{h}}_t + \hat{\mathbf{a}}_r, \ \hat{\mathbf{h}}_t = \hat{f}(\hat{\mathbf{h}}_{t+1}, \mathbf{0}), \end{cases} \tag{4}$$

where $\{\tilde{\mathbf{W}}, \tilde{\mathbf{a}}\}$ and $\{\hat{\mathbf{W}}, \hat{\mathbf{a}}\}$ are the parameters of the forward and reverse reconstruction; $\tilde{f}$ and $\hat{f}$ are the functions of the forward and reverse LSTMs. Particularly, $\tilde{\mathbf{h}}_0 = \mathbf{b}_m$ or $\hat{\mathbf{h}}_m = \mathbf{b}_m$ denotes that the decoder starts from the hash code and the $\mathbf{0}$ input highlights that the decoder RNNs do not take any input features. The missing input feature $\mathbf{0}$ prevents the decoder from learning trivial reconstruction that is directly from the input. As a result, the encoder is encouraged to generate hash codes that retain sufficient information for valid reconstructions.

The proposed forward and reverse RNNs focus on learning different aspects of video dynamics. On one hand, the forward RNN attempts to memorize long-term event evolution since the reconstruction starting from the first frame will encourage the memory not to forget too much about the early frames; on the other hand, reverse RNN is more likely to capture short-term temporal relations among video semantics, *e.g.*, objects and motion changes, since the reconstruction from the last frame, which has been just viewed, is a relatively easy task if the memory holds sufficient information about recent frames. Based on the above analysis, the overall loss function for our SSTH can be formulated as:

$$L = \underbrace{\sum_{t=1}^{m} \|\tilde{\mathbf{v}}_t - \mathbf{v}_t\|_2^2}_{\text{forward order}} + \underbrace{\sum_{t=m}^{1} \|\hat{\mathbf{v}}_t - \mathbf{v}_t\|_2^2}_{\text{reverse order}}. \tag{5}$$

However, minimizing the above loss function is challenging since the deeply nested binary codes $(\mathbf{b}_1, ..., \mathbf{b}_m)$ are non-differentiable. Later, we will introduce an efficient binary optimization method based on backpropagation of the binary codes.

## 3.4 Architecture Details

The detailed architecture of the proposed SSTH is illustrated in Figure 3. Both the encoder and two decoders adopt a two-layer structure in order to enhance the expressive power of the model. On the encoder side, the first-layer is a conventional vanilla LSTM [12], where the output hidden variable $\mathbf{h}$ is used as the input to the second-layer BLSTM and the next LSTM. Specifically, the dimensions of $\mathbf{h}$ is $2k$, *i.e.*, two times as the bit size of the binary code. By doing this, the first-layer LSTMs of the encoder can be viewed as higher-level feature extractors for the frames. After the hash code $\mathbf{b}_m$ being encoded, *i.e.*, the BLSTM output of the last frame, we directly feed it to the first layer of the two decoder RNNs. On the decoder side, the dimension of the first and second layer are $k$ and $2k$, respectively. In particular,

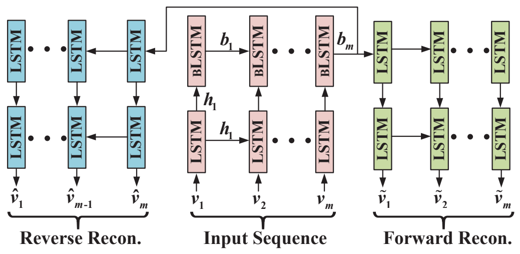


**Figure 3: The encoder-decoder architecture used in SSTH learning. Both encoder and decoder are two-layer RNNs. Red: encoder RNN. Green: forward decoder RNN. Blue: reverse decoder RNN.**

the linear transformation reconstructs the $2k$-D hidden variables of the second layer to $d$-D frame-level features. Note that this is different from [27] which aligns the two-layer encoder outputs to the two-layer decoder inputs. The reason is that we want to limit the decoder to access information from other source, *e.g.*, the first hidden layer of the encoder, in order to encourage the hash code retaining more higher-level video dynamics. One should also note that the trainable parameters of the encoder LSTM-BLSTM, forward and reverse decoder LSTM-LSTM are independent with each other since the three RNNs are responsible for different purposes.

## 4. BINARY OPTIMIZATION

Training SSTH equipped with BLSTM is essentially NP-hard as it involves binary optimization of the hash codes that requires combinatorial search space [13]. However, approximated solution by discarding the binary constraints will lead to large quantization loss [11]. In this section, we propose to directly tackle the challenging binary optimization. We first introduce how to take the derivative of the sgn binarization and then detail the learning algorithm.

### 4.1 Derivative of Binarization

As the derivative of the sgn function in BLSTM is zero almost everywhere, it is impossible to apply exact backpropagation for BLSTM. Here, we propose an approximation by estimating the derivative of the sgn function.

In fact, the binary code generated by sgn can be viewed as binary classification results, where the classifier responses are $\mathbf{h}_t$. So, $\text{sgn}(\mathbf{h}_t)$ is a set of hidden classifiers that collaboratively make predictions for a final objective [2], *e.g.*, the reconstruction loss in our case. Based on this view, our key idea is to design a delegate classifier $d$ that approximates the original classifier sgn by allowing a certain loss. Without loss of generality, we only consider a single entry $h$ of $\mathbf{h}_t$ in Eq. (3h):

- When $h \geq 0$, we use $d(h)$ to approximate $\text{sgn}(h) = 1$ by allowing some classification loss as:

$$\text{sgn}(h) \approx d(h) = \text{sgn}(h) - \ell(h), \ h \geq 0, \qquad (6)$$

where $\ell(\cdot) \geq 0$ is the loss function. Since $h \geq 0$, we can view it as a score for the positive label +1. In fact, $d(h)$ is equivalent to $\text{sgn}(h)$ when $\ell(h) = \mathbb{1}(1 \cdot h \leq 0)$, *i.e.*, the loss is 0 if the response $h$ is consistent with the label +1; and 1 otherwise. However, such 0–1 loss is useless since we constrain $h \geq 0$. A popular way to soften the "hard" 0–1 loss is to use the hinge loss with a margin: $\ell(h) = \max(0, 1 - 1 \cdot h)$, which is widely used in penalizing large-margin classifiers such as SVM. Specifically, it allows a certain violation $1 - (+1) \cdot h$ to the ideal response +1 when $0 \leq h \leq 1$. Thus, Eq. (6) can be rewritten as:

$$d(h) = 1 - \max(0, 1 - 1 \cdot h), h \geq 0, \qquad (7)$$

- When $h < 0$, similar to the above analysis, we can view it as a score for the negative label −1. By allowing a hinge loss to the ideal binary response −1, $\text{sgn}(h) = -1$ can be approximated as:

$$d(h) = -1 + \max(0, 1 - (-1) \cdot h), h < 0, \qquad (8)$$

where a violation as $1 + h$ is allowed when $-1 \leq h < 0$.

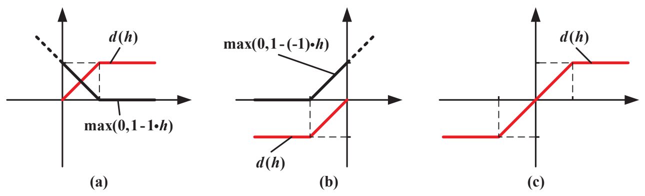


**Figure 4: Illustrative process of how $d(\cdot)$ (red line) approximates $\text{sgn}(\cdot)$ using a hinge loss function (black line). (a) when $h \geq 0$, $d(h) = \textbf{sgn}(h) - \textbf{max}(0, 1 - 1 \cdot h)$; (b) when $h < 0$, $d(h) = \textbf{sgn}(h) + \textbf{max}(0, 1 - (-1) \cdot h)$; (c) $d(h)$.**

Combining Eq. (7) and Eq. (8), the resultant approximation of $\text{sgn}(h)$ can be written as:

$$\text{sgn}(h) \approx d(h) = \begin{cases} -1, & h < -1, \\ h, & -1 \leq h \leq 1, \\ 1, & h > 1. \end{cases} \qquad (9)$$

So far, we are ready to define the derivative of $\text{sgn}(h)$ as:

$$\text{sgn}'(h) := d'(h) = \mathbb{1}(|h| \leq 1). \qquad (10)$$

Figure 4 illustrates how we use the hinge loss to approximate the sgn function.

The derivative $d'(h)$ states a simple backpropagation rule for BLSTM: when the gradients back propagate to the sgn function, we only allow gradients, whose neural responses are between $[-1, +1]$, to pass through. Intuitively, this rule encourages BLSTM to update if the hidden value $h$ is not well separated with respect to a margin, resulting in hash functions that are more certain about binarization. In fact, $d'$ can be considered as a "straight-through estimator" which has been used in SGD for discrete neurons [4]. Though $d'$ is not an exact gradient for sgn, the convergence of SGD can be achieved [1].

### 4.2 Algorithmic Details

Thanks to the derivative of the sgn function introduced in Eq. (10), we are now ready to develop forward propagation (FP) and back propagation (BP) for training the deep RNNs. Due to space limit, we only summarize FP and BP for the key component: the proposed BLSTM.

The FP for BLSTM is illustrated in Algorithm 1. For notational simplicity, we divide BLSTM into four steps: 1) subroutine BLSTMHEAD, parameterized by $\alpha$, runs from Eq. (3a) to Eq. (3d). Note that the input $\mathbf{v}_t$ can be video feature or the output of the first layer LSTM; 2) subroutine BATCHNORM, parameterized by $\gamma$, runs for the batch normalization in Eq. (3e); 3) subroutine BLSTMTAIL, pa-

**Table 1: Time expense (millisecond) of LSTM and BLSTM on TITAN X. Batch size $B = 50$, frame length $l = 25$, feature dimension $d = 4096$, bit size $k = 128$.**

| | FP | BP | BatchNorm | BatchNormBP | Hashing |
|---|---|---|---|---|---|
| **LSTM** | 23.6 | 93.4 | N.A. | N.A. | 17.1 |
| **BLSTM** | 23.7 | 144 | $3.03e^{-2}$ | 50.6 | 17.3 |

rameterized by $\beta$, runs from Eq. (3f) to Eq. (3g); and 4) binarization in Eq. (3h). The BP for BLSTM is detailed in Algorithm 2. The input $\nabla \mathbf{b}_m = \frac{\partial L}{\partial \mathbf{b}_m}$ is the derivative of the loss function $L$ in Eq. (5) with respect to hash code $\mathbf{b}_m$. Line 2 is the binarization BP according to Eq. (10). Then, from Line 3 to Line 9, we assume that the three gradient subroutines are available: BLSTMheadBP, BLSTMtailBP and BatchNormBP. Note that Line 4, 6, and 9 are the gradients of the model parameters updated using chain rules.

After applying FP and BP over a minibatch of training videos, we use SGD to update the model parameters $\{\alpha, \beta, \gamma\}$ with momentum and dynamic learning rate. Denote frame length as $l$, frame-level feature dimension as $d$, bit size as $k$, and the SGD batch size as $B$, it is easy to see that it takes $\mathcal{O}(Bdkl)$ for FP and BP, and $\mathcal{O}(\frac{1}{2}Bdkl))$ for generating hash codes. Thus, the training and testing time is linear to the frame length, bit size and sample size. In particular, Table 1 lists the training/testing time expense comparison between LSTM and BLSTM on a TITAN X GPU. We only need to pay off a little overhead for BLSTM due to the introduction of batch normalization.

**Algorithm 1:** Forward Propagation for BLSTM

```
  Input : $(\mathbf{v}_1, ..., \mathbf{v}_m)$: frame-level feature sequence
          $\alpha$, $\beta$: BLSTMhead,tail parameters
          $\gamma$: BatchNorm parameters
  Output: $\mathbf{b}_m$: the binary code
  Init  : $\mathbf{c}_0 \leftarrow \mathbf{0}$, $\mathbf{b}_0 \leftarrow \mathbf{0}$
1 for $t = 1$ to $m$ do
2     $\mathbf{c}_t \leftarrow$ BLSTMhead $(\mathbf{c}_{t-1}, \mathbf{b}_{t-1}, \mathbf{v}_t; \alpha)$
3     $\mathbf{c}_t \leftarrow$ BatchNorm $(\mathbf{c}_t; \gamma)$
4     $\mathbf{h}_t \leftarrow$ BLSTMtail $(\mathbf{c}_t, \mathbf{b}_{t-1}, \mathbf{v}_t; \beta)$
5     $\mathbf{b}_t \leftarrow \mathrm{sgn}(\mathbf{h}_t)$
6 end
7 return $\mathbf{b}_t$
```

## 5. EXPERIMENTS

As the proposed SSTH is a novel unsupervised video hashing framework, the goal of our experiments is to answer the following three research questions:

**RQ1**. Why is SSTH designed to what we have proposed? How do different components of it affect the performance?

**RQ2**. How does SSTH perform as compared to other state-of-the-art video hashing methods?

**RQ3**. What is the generalization ability of SSTH, *e.g.*, less training data and cross-dataset performance?

### 5.1 Datasets

We used two challenging large-scale video datasets for unsupervised training and retrieval[4]:

[4]Details of the two datasets are in the supplementary material.

**Algorithm 2:** Back Propagation for BLSTM

```
   Input : $\nabla \mathbf{b}_m$: gradient of $\mathbf{b}_m$ propagated from the decoder
           $\nabla \mathbf{c}_m$: gradient of $\mathbf{c}_m$ propagated from the decoder
   Output: $\nabla\alpha$, $\nabla\beta$: gradients of BLSTMhead,tail
           parameters
           $\nabla\gamma$: gradient of BatchNorm parameters
   Init  : $\nabla\alpha_{m+1} \leftarrow \mathbf{0}$, $\nabla\beta_{m+1} \leftarrow \mathbf{0}$, $\nabla\gamma_{m+1} \leftarrow \mathbf{0}$
   // Backpropagation through time
 1 for $t = m$ to 1 do
 2     $\nabla \mathbf{h}_t \leftarrow \nabla \mathbf{b}_t \circ \mathbb{1}(|\mathbf{h}_t| \leq \mathbf{1})$
 3     $\left(\nabla \hat{\mathbf{c}}_t, \nabla \hat{\mathbf{b}}_{t-1}, \nabla\alpha_t\right) \leftarrow$ BLSTMtailBP$(\nabla \mathbf{h}_t, \nabla \mathbf{c}_t)$
 4     $\nabla\alpha_t \leftarrow \nabla\alpha_{t+1} + \nabla\alpha_t$
 5     $(\nabla \mathbf{c}_t, \nabla\gamma_t) \leftarrow$ BatchNormBP$(\nabla \hat{\mathbf{c}}_t)$
 6     $\nabla\gamma_t \leftarrow \nabla\gamma_{t+1} + \nabla\gamma_t$
 7     $(\nabla \mathbf{c}_{t-1}, \nabla \mathbf{b}_{t-1}, \nabla\beta_t) \leftarrow$ BLSTMheadBP$(\nabla \mathbf{c}_t)$
 8     $\nabla \mathbf{b}_{t-1} \leftarrow \nabla \hat{\mathbf{b}}_{t-1} + \nabla \mathbf{b}_{t-1}$
 9     $\nabla\beta_t \leftarrow \nabla\beta_{t+1} + \nabla\beta_t$
10 end
11 return $\nabla\alpha_1$,$\nabla\beta_1$,$\nabla\gamma_1$
```

**FCVID**. It is Fudan-Columbia Video Dataset [16]. FCVID is one of the largest datasets for video categorization with accurate manual annotations in generic domain. This dataset contains 91,223 Youtube videos annotated manually according to 239 categories, covering a wide range of topics like events (*e.g.*, “Tailgate Party”), objects (*e.g.*, “Panda”) and scenes (*e.g.*, “Beach”). The average video length is about 167 seconds. Its train/test split is 45,611/45,612. We used the train split for unsupervised learning and the test split for retrieval.

**YFCC**. It is Yahoo Flickr Creative Common dataset [29], the largest public multimedia collection that has ever been released. It is officially announced with 0.8M videos from Flickr, however, we only collected 700,882 videos by filtering out invalid urls and corrupted video files. The average video length is about 37 seconds. Particularly, we also contributed to the community a large video scene dataset of 100,000 videos selected from YFCC, manually annotated according to the most popular 80 scenes from the third level of MIT SUN secene hierarchy [36], such as indoor (*e.g.*, “Coffee Shop”) and outdoor (*e.g.*, “Golf Court”). We used the unlabeled 600,882 videos for unsupervised learning and the rest 100,000 labeled videos for retrieval.

Note that unlike FCVID where a considerable amount of the Youtube videos are taken by professionals, YFCC videos are mostly taken by the mobile phones of Flickr casual users. Therefore, the visual quality of YFCC videos is much lower and hence it is more challenging for video content understanding. For each video, we uniformly sampled 25 frames as video sequences. As a result, our SSTH is a very deep RNN that has 75 layers after unfolding (*i.e.*, 25-layer encoder, 25-layer forward decoder and 25-layer reverse decoder). Though we set the frame number to 25 as a comprise for training time and GPU memory, we believe that higher frame rate will lead to stronger models. For frames, we used VGG-fc19 [23] to extract 4,096-D CNN features as the frame-level representations. Note that we did not use motion features such as dense trajectories [31] for all the methods, since we intended to investigate whether they can capture video dynamics by modeling sequences.

## 5.2 Experimental Setup

### 5.2.1 Evaluation Metrics

We adopted Average Precision at top $K$ retrieved videos (AP@K) for retrieval performance evaluation [20]. Denote $R$ as the number of relevant videos in the database. At any ranked position $j$ ($1 \leq j \leq K$), let $R_j$ be the number of relevant videos in the top $j$ results and let $I_j = 1$ if the $j$-th video is relevant and 0 otherwise, then AP@K is defined as $\frac{1}{min(R,K)} \sum_{j=1}^{K} \frac{R_j}{j} \times I_j$. For each class label, we considered every test video that belongs to the label as query and the rest of test data as the database; then, we used the mean of AP@K of the queries (mAP@K) as the performance metric for the label. We also slightly abused mAP@K to be the mean of all the label-specific mAP@K as an overall metric.

### 5.2.2 Search Protocols

We adopted two search protocols which are widely used in search with binary codes. We evaluated code length $k \in \{8, 16, 32, 64, 128, 256\}$.

**Hamming Ranking:** Videos are ranked according to their Hamming distance (or similarity) from the query user. Although the search complexity of Hamming ranking is still linear, it is very fast in practice since the Hamming distance calculation can be done by fast bit XOR operations and the sorting is constant time due to integer distance.

**Hashtable Lookup:** A lookup table is constructed using the video hash codes and all the items in the buckets that fall within a small Hamming radius (*e.g.*, 2) of the query are returned. Therefore, search is performed in constant time. However, a single table would be insufficient when the code length is larger than 32 since it would require over $\mathcal{O}(2^{32})$ space to store the table in memory. We adopted Multi-Index Hashing (MIH) table [19], which builds one table for each code subsegment. Items are aggregated by all the tables and then conducted Hamming ranking for the items. By doing this, the search time is significantly reduced to sublinear. We empirically set the substring length as $\{1, 1, 1, 2, 4, 8\}$ for bit size $\{8, 16, 32, 64, 128, 256\}$ as suggested in [19].

It is worth mentioning that the above two search protocols focus on different characteristics of hash codes. Ranking provides a better measurement of the learned Hamming space, *i.e.*, the accuracy upper bound that the codes can achieve since it linearly scans the whole data. Lookup, on the other hand, emphasizes the practical speed of large-scale search. However, a common issue in this protocol is that it may not return sufficient items for recommendation, as a query lookup may miss items due to the sparse Hamming space. In our experiments, if a query returns no videos, we treated it as a failed query with an AP of zero.

### 5.2.3 Compared Methods

To validate the architecture design of the proposed **SSTH**, we compared the following possible architectures that can also be viewed as temporal-aware hashing methods:

**LSTM**. We used the conventional vanilla LSTMs [12] for both the encoder and decoder. After training, the real-valued output of the encoder is binarized as the hash code. Note that this setting can be considered as a relaxed deep hashing model.

**LSTM-sgn**. We directly added a sgn function to the output of every LSTM in the encoder. This method includes the tanh function for the memory cell and does not use batch normalization. Its optimization adopts the same binary backpropagation as BLSTM.

**Enc-sgn**. The encoder consists of LSTM with batch normalization for the memory. Moreover, we added a sgn function at the end of the encoder to generate binary codes. This method is mentioned at the end of Section 3.1. Its optimization adopts the same binary backpropagation as the proposed BLSTM.

**SSTH-F/R**. This is SSTH without the reverse (SSTH-F) or the forward reconstruction decoder (SSTH-R).

**SSTH-1/2**. This is SSTH with one or two layer architecture. Note that all the aforementioned methods are one-layer except SSTH-2.

As we will show that the two-layer SSTH-2 introduced in Section 3.4 performs better, we use SSTH instead of SSTH-2 when the context is clear. To compare with the state-of-the-art unsupervised hashing methods, we have:

**ITQ**. This is perhaps the most popular unsupervised hashing method called Iterative Quantization [11]. It first uses average pooling for video data representations, then applies PCA to reduce the dimensionality of the data to the target bit size (*i.e.*, $k$), and finally iteratively learns a rotation that minimizes the quantization loss.

**SubMod**. This is Submodular video hashing [3]. It first uses average pooling for video data representations, then it applies traditional hashing methods such as LSH to hash the videos into long codes (*e.g.*, we used 1024), and then it greedily selects $k$ most informative hash functions, where the informativeness is measured using the training data.

**MFH**. This is Multi-Feature Hashing for videos [26]. It learns hash functions based on the similarity graph of the frames. Since the hash function is learned on frame-level, it uses average pooling to pool the real-valued outputs of hash functions and then binarizes them for hash codes.

**DH**. This is a Deep Hashing framework [8]. Its key idea for binary code learning is to add a binarization loss function at the top layer of a neural network. For fair comparison, we used the original encoder-decoder RNN [27] with the loss function imposing at the end of the encoder RNN. By doing this, DH is able to hash temporal data like videos.

Although there are other video hashing methods, they are either based on supervised hashing [38] or specialized non-binary hash codes [21], hence do not apply in our comparable experiments. For deep models such as DH and SSTH, we used a TITAN X GPU with Theano implementation. The minibatch size of all the deep models was set to 50, training for 30 epochs.

## 5.3 Result Analysis

### 5.3.1 Architecture Investigation (RQ1)

Figure 5 shows how the possible designs of SSTH affect the overall performances. Due to space limit, we only report the results of 128-bit codes. From the results on both datasets, we can arrive at the following observations: 1) only LSTM leads to considerable performance drop when using table lookup since it is the only two-stage method with large quantization loss. On the contrary, we do not observe significant performance drop of the other models using binary backpropagation, which is a joint framework; 2) by naively adding sgn function for LSTM, LSTM-sgn achieves the worst

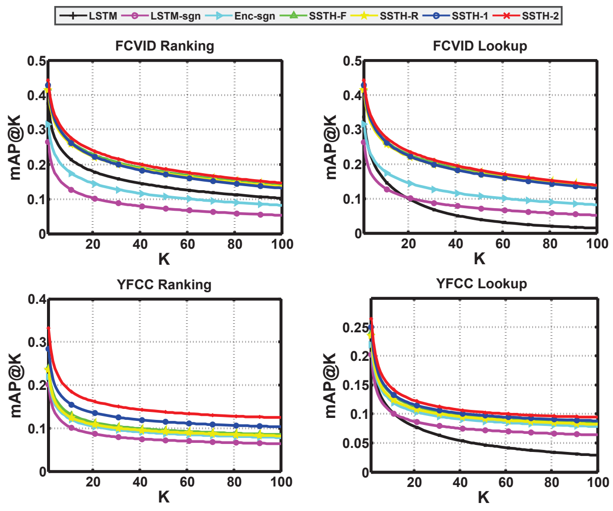


**Figure 5: Performances (mAP@K) of 128-bit hash codes learned from various RNN architectures.**

**Table 2: Cross-dataset mAP@20 gain (%) by Hamming ranking of various methods using different bit sizes.**

| Dataset | Training: FCVID, Testing: YFCC | | | | | |
|---|---|---|---|---|---|---|
| Bits | 8 | 16 | 32 | 64 | 128 | 256 |
| SubMod | **-2.13**↓ | -3.64↓ | **-7.32**↓ | -14.8↓ | -15.1↓ | -33.8↓ |
| MFH | -23.8↓ | -7.29↓ | -16.5↓ | -34.3↓ | -30.0↓ | -24.7↓ |
| ITQ | -14.7↓ | -10.1↓ | -22.7↓ | **-5.76**↓ | **-2.47**↓ | -4.76↓ |
| DH | -7.03↓ | **-3.16**↓ | -10.0↓ | -12.9↓ | -13.8↓ | **-2.04**↓ |
| SSTH | -8.71↓ | -5.03↓ | -12.5↓ | -11.4↓ | -10.0↓ | -11.6↓ |
| Dataset | Training: YFCC, Testing: FCVID | | | | | |
| SubMod | -1.78↓ | -3.54↓ | -5.07↓ | -12.5↓ | -12.1↓ | -20.3↓ |
| MFH | 12.6↑ | 13.6↑ | -7.28↓ | 5.03↑ | 3.64↑ | 2.38↑ |
| ITQ | 14.5↑ | 15.9↑ | -8.83↓ | -7.62↓ | 5.17↑ | -8.26↓ |
| DH | 11.3↑ | 11.6↑ | 13.8↑ | -8.02↓ | -9.70↓ | -3.93↓ |
| SSTH | **28.1**↑ | **27.7**↑ | **22.9**↑ | **10.1**↑ | **8.64**↑ | **7.58**↑ |

results. This is because of the absence of batch normalization as discussed in Section 3.2: without batch normalization, the hidden variables are likely to be imbalanced and correlated, *e.g.*, some of the hidden variables may be out of the range $[-1, 1]$. Thus, the zero gradients will result in premature hash codes; 3) the temporal hashing models such as SSTH-F/R/1/2 are better than Enc-sgn that only deploys binarization at the end of the encoder. This demonstrates the effectiveness of the temporal-aware design as defined in Eq. (1); 4) by combining both forward and reverse reconstruction, SSTH-1 is better than SSTH-F and SSTH-R; 5) by adding more layers, SSTH-2 can enhance the expressive power of SSTH. Therefore, based on the above observations, we have demonstrated the effectiveness of our particular design for SSTH.

### 5.3.2 *Comparison with State-of-The-Arts (RQ2)*

Figure 8 shows how SSTH performs as compared to state-of-the-art video hashing methods. We can see that SSTH consistently performs the best on both datasets. Specifically, we have the following observations:
1) Compared to pooling based hashing methods such SubMod, MFH and ITQ, SSTH explicitly models the temporal information of videos. For example, on FCVID, by using only 128 bits, SSTH achieves 24.5% mAP@20, which is

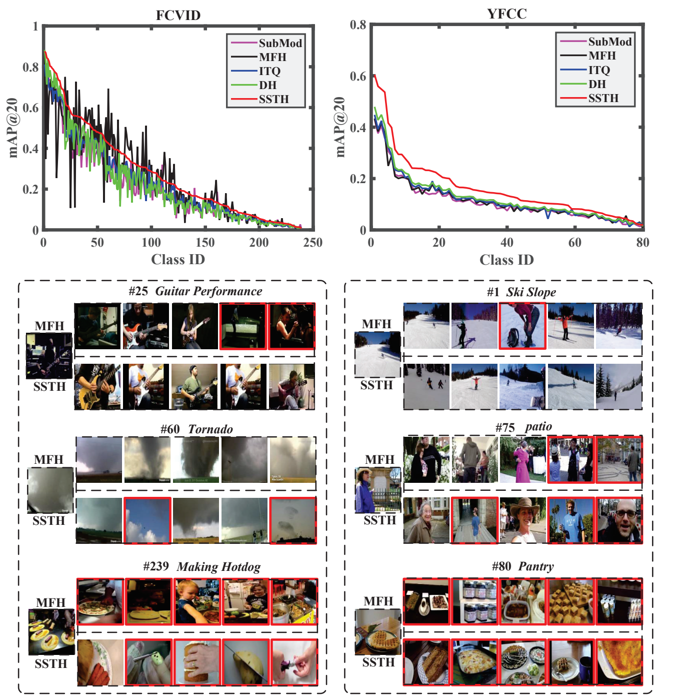


**Figure 6: Detailed performance (mAP@20 of 256 bits) of all the methods on both datasets. The Class ID is sorted according to the descending order of performances of SSTH. The illustrative examples are the top 5 retrieval results of SSTH and the most competitive MFH queried by three categories of different performances. Red borders denote wrong results. Left: FCVID; Right: YFCC.**

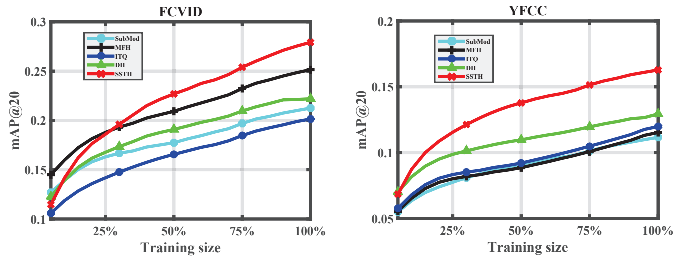


**Figure 7: Performance (mAP@20 of 256 bits) on two datasets using various percentages of training data.**

comparable to 25.1% of the most competitive MFH using 256 bits. Figure 6 detailed the mAP@20 of various methods using 256 bits on both datasets. We can see that modeling temporal information is beneficial to discriminate concepts that involves human actions like "Guitar Performance" in FCVID. More interestingly, we can observe that SSTH consistently outperforms the pooling-based methods on the scene retrieval task on YFCC. This demonstrates that the temporal information of videos are not only crucial for events but also indispensable for recognizing scenes. For the example of YFCC "Ski Slope" in Figure 6, pooling-based methods like MFH will inevitably mis-classify frames of "Snow" as "Ski Slope" while SSTH successfully captures the informative "Ski" motion.
2) However, we should note that pooling-based methods such as MFH are also powerful, especially for video categories that are not likely to be distinguished by temporal information. As illustrated in Figure 6, MFH that only

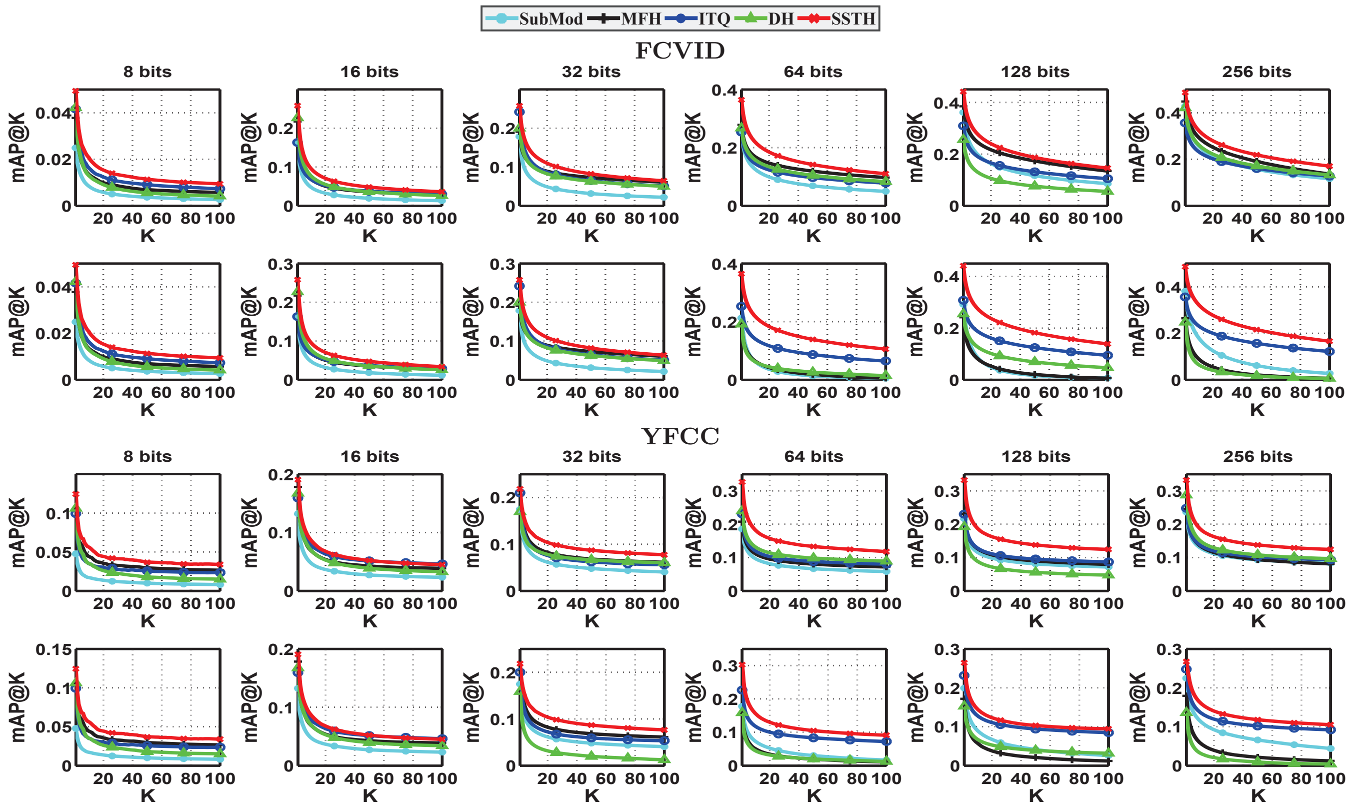


**Figure 8: Performance (mAP@K) of various video hashing methods at different bit sizes. For each dataset, the top row is the performance by Hamming ranking and the bottom row is by table lookup.**

captures visual appearances seems sufficient for the "Tornado" category in FCVID; while SSTH over-emphasizes on the cloud motion. As another example, SSTH overfits to the motion of individual selfies in YFCC "Patio", and thus performs worse than MFH. We also find some hard cases for all the methods. One example is the food related events like FCVID "Making Hotdog", where MFH is biased to the general concepts of "Food", and SSTH is more likely favored the motion of "Making" like "Nail Painting" and "Cutting Potatoes". Another example is the YFCC "Pantry", where MFH and SSTH are both biased to the appearance of "Waffle".
3) DH, which also attempts to capture temporal information, performs much worse than SSTH and even some pooling-based methods such as MFH and ITQ. The reason is similar to why Enc-sgn in the previous experiment performed worse than SSTH: although DH exploits RNN to model the temporal order of frames, it is essentially a pooling-based hashing method since the hash codes are not temporal-aware as defined in Eq. (1).
4) As shown in Figure 8, when retrieval is performed by using hash table lookup, relaxed methods—SubMod, MFH and DH—show significant performance drop using longer hash codes, *e.g.*, 128 and 256 bits. This demonstrates the effectiveness of binary optimization adopted in ITQ and SSTH, for the minimization of the binarization loss.

### 5.3.3 *Generalization Ability (RQ3)*

Generalization ability is especially crucial for learning-to-hash methods since the queries provided by users are usually out-of-domain as compared to the training data. Therefore, it is important to investigate how SSTH performs given limited training data and how SSTH generalizes to cross-dataset retrieval tasks, *e.g.*, training on FCVID but testing on YFCC and vice versa.

Figure 7 shows the mAP@20 of all the methods using 256 bits on two datasets trained by various percentages of training data. Obviously, all the methods gain better performance when using more training data. In particular, the generalization ability of the proposed SSTH is significantly better than others. For example, by using only the half training data, SSTH achieves similar performance to the most competitive MFH on FCVID and outperforms all the methods on YFCC. We believe that this is due to the strong expressive ability of the deep architecture of SSTH. Table 2 lists the cross-dataset performance of all the hashing methods. We can see that all of them suffer a considerable performance drop when training on FCVID but testing on YFCC. This demonstrates that when training data are relatively small, *e.g.*, around 50 thousands of FCVID, the practical use of data-driven hash models is limited due to the train/test dataset discrepancy. However, if the training data are sufficient, *e.g.*, around 600 thousands of YFCC, we can observe that all the methods, except SubMod, achieves much better performance gain as compared to that of training on FCVID; in particular, SSTH achieves consistently increasing gain when using all the bit sizes. This demonstrates that the expressive power of the SSTH architecture is strong. The reason why SubMod achieves the worst generalization is perhaps that the binary code selection procedure is heavily biased to particular datasets. In summary, based on the above observations, we believe that SSTH has a great potential in real-world CBVR due to its generalization ability in unsupervised learning from the inexhaustible videos on the Web.

## 6. CONCLUSIONS

In this paper, we proposed a novel video hashing framework called Self-Supervised Temporal Hashing (SSTH). In sharp contrast to existing video hashing methods that are generally based on frame pooling, SSTH explicitly models video temporal information and directly optimizes the binary code learning problem without relaxation. Through extensive experiments on two large consumer video datasets, we have demonstrated that the effectiveness of SSTH is due to its three distinguished designs: Binary Long-Short Term Memory, Self-Supervision Learning Strategy and Binary Backpropagation. To the best of our knowledge, SSTH is the first unsupervised deep video hashing model.

SSTH is essentially a generic end-to-end framework for binary representation learning of any sequential multimedia data. In this view, a possible future direction is to apply SSTH to other temporal multimedia applications where images and texts can be traced by time, such as multimedia user profiling and news event prediction. Further, we can also extend SSTH to supervised and semi-supervised hashing framework if labels are available.

## Acknowledgements

NExT research is supported by the National Research Foundation, Prime Minister's Office, Singapore under its IRC@SG Funding Initiative.

## 7. REFERENCES

[1] Y. Bengio, N. Léonard, and A. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

[2] A. Bergamo, L. Torresani, and A. W. Fitzgibbon. Picodes: Learning a compact code for novel-category recognition. In *NIPS*, 2011.

[3] L. Cao, Z. Li, Y. Mu, and S.-F. Chang. Submodular video hashing: a unified framework towards video pooling and indexing. In *MM*, 2012.

[4] M. Courbariaux and Y. Bengio. Binarynet: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

[5] R. Datta, D. Joshi, J. Li, and J. Z. Wang. Image retrieval: Ideas, influences, and trends of the new age. *ACM Computing Surveys*, 2008.

[6] T.-T. Do, A.-D. Doan, and N.-M. Cheung. Learning to hash with binary deep neural network. In *ECCV*, 2016.

[7] J. Donahue, L. Anne Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell. Long-term recurrent convolutional networks for visual recognition and description. In *CVPR*, 2015.

[8] V. Erin Liong, J. Lu, G. Wang, P. Moulin, and J. Zhou. Deep hashing for compact binary codes learning. In *CVPR*, 2015.

[9] A. Gionis, P. Indyk, R. Motwani, et al. Similarity search in high dimensions via hashing. In *VLDB*, 1999.

[10] X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In *ICAIS*, 2011.

[11] Y. Gong, S. Lazebnik, A. Gordo, and F. Perronnin. Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval. *TPAMI*, 2013.

[12] A. Graves and J. Schmidhuber. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks*, 2005.

[13] J. Håstad. Some optimal inapproximability results. *JACM*, 2001.

[14] W. Hu, N. Xie, L. Li, X. Zeng, and S. Maybank. A survey on visual content-based video indexing and retrieval. *TSMC Part C*, 2011.

[15] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[16] Y.-G. Jiang, Z. Wu, J. Wang, X. Xue, and S.-F. Chang. Exploiting feature and class relationships in video categorization with regularized deep neural networks. *arXiv preprint arXiv:1502.07209*, 2015.

[17] M. S. Lew, N. Sebe, C. Djeraba, and R. Jain. Content-based multimedia information retrieval: State of the art and challenges. *TOMM*, 2006.

[18] H. Li. Multimodal visual pattern mining with convolutional neural networks. In *ICMR*, 2016.

[19] M. Norouzi, A. Punjani, and D. J. Fleet. Fast search in hamming space with multi-index hashing. In *CVPR*, 2012.

[20] P. Over, J. Fiscus, G. Sanders, D. Joy, M. Michel, G. Awad, A. Smeaton, W. Kraaij, and G. Quénot. Trecvid 2014–an overview of the goals, tasks, data, evaluation mechanisms and metrics. In *Proceedings of TRECVID*, page 52, 2014.

[21] J. Revaud, M. Douze, C. Schmid, and H. Jégou. Event retrieval in large video collections with circulant temporal encoding. In *CVPR*, 2013.

[22] F. Shen, C. Shen, W. Liu, and H. Tao Shen. Supervised discrete hashing. In *CVPR*, 2015.

[23] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[24] A. W. Smeulders, M. Worring, S. Santini, A. Gupta, and R. Jain. Content-based image retrieval at the end of the early years. *TPAMI*, 2000.

[25] C. G. Snoek and M. Worring. Concept-based video retrieval. *FTIR*, 2008.

[26] J. Song, Y. Yang, Z. Huang, H. T. Shen, and R. Hong. Multiple feature hashing for real-time large scale near-duplicate video retrieval. In *MM*, 2011.

[27] N. Srivastava, E. Mansimov, and R. Salakhudinov. Unsupervised learning of video representations using lstms. In *ICML*, 2015.

[28] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *NIPS*, 2014.

[29] B. Thomee, D. A. Shamma, G. Friedland, B. Elizalde, K. Ni, D. Poland, D. Borth, and L.-J. Li. The new data and new challenges in multimedia research. *arXiv preprint arXiv:1503.01817*, 2015.

[30] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan. Show and tell: A neural image caption generator. In *CVPR*, 2015.

[31] H. Wang and C. Schmid. Action recognition with improved trajectories. In *ICCV*, 2013.

[32] J. Wang, S. Kumar, and S.-F. Chang. Semi-supervised hashing for large-scale search. *TPAMI*, 2012.

[33] J. Wang, W. Liu, S. Kumar, and S.-F. Chang. Learning to hash for indexing big data: A survey. *Proceedings of the IEEE*, 2016.

[34] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. In *NIPS*, 2009.

[35] Z. Wu, Y. Fu, Y.-G. Jiang, and L. Sigal. Harnessing object and scene semantics for large-scale video understanding. In *CVPR*, 2016.

[36] J. Xiao, J. Hays, K. A. Ehinger, A. Oliva, and A. Torralba. Sun database: Large-scale scene recognition from abbey to zoo. In *CVPR*, 2010.

[37] Y. Yang, Z.-J. Zha, Y. Gao, X. Zhu, and T.-S. Chua. Exploiting web images for semantic video indexing via robust sample-specific loss. *TMM*, 2014.

[38] G. Ye, D. Liu, J. Wang, and S.-F. Chang. Large-scale video hashing via structure learning. In *CVPR*, 2013.

[39] Z.-J. Zha, M. Wang, Y.-T. Zheng, Y. Yang, R. Hong, and T.-S. Chua. Interactive video indexing with statistical active learning. *TMM*, 2012.

[40] Z.-J. Zha, H. Zhang, M. Wang, H. Luan, and T.-S. Chua. Detecting group activities with multi-camera context. *TCSVT*, 2013.

[41] H. Zhang, F. Shen, W. Liu, X. He, H. Luan, and T.-S. Chua. Discrete collaborative filtering. In *SIGIR*, 2016.

[42] H. Zhang, Z.-J. Zha, Y. Yang, S. Yan, Y. Gao, and T.-S. Chua. Attribute-augmented semantic hierarchy: towards bridging semantic gap and intention gap in image retrieval. In *MM*, 2013.

[43] F. Zhao, Y. Huang, L. Wang, and T. Tan. Deep semantic ranking based hashing for multi-label image retrieval. In *CVPR*, 2015.