Received: March 15, 2022. Revised: May 24, 2022.

International Journal of
Intelligent Engineering & Systems

http://www.inass.org/

Deep Video Hashing Using 3DCNN with BERT

Salah Mostafa El Abyad[1]* **Mona M. Soliman[1,2]** **Khaled Mostafa El Sayed[1]**

[1]*Faculty of Computer and Artificial Intelligence, Cairo University, Egypt*
[2]*Member of Scientific Research Group in Egypt, Egypt*
* Corresponding author's Email: s.elabyad@fci-cu.edu.eg

Abstract: Deep video hashing (DVH) is a very appealing way to decrease storage costs and query times. In this work we propose a hashing model using two separated modules. A 3DCNN is proposed with a bidirectional encoder representations from transformers (BERT) layer. And a hashing neural network (NN) module will learn to encode those features into hash codes. The proposed model that separates feature extraction from hash generation process results in better performance with respect to training time consumption and accuracy. We achieve a significant improvement in video retrieval performance on two benchmark datasets compared to state-of-the-art deep learning models for video retrieval that use convolutional neural networks (CNN)s or 3DCNNs along with other temporal feature extraction techniques and supervised hashing methods. For UCF101, HMDB51 datasets, more than 2 % mAP and 24 % improvement is achieved respectively for tested bit sizes.

Keywords: Video hashing, Deep learning, Bidirectional encoder representations from transformers (BERT), 3DCNN.

1. Introduction

In the modern world the amount of video data has increased significantly, especially with YouTube, Facebook and other cloud-based websites that provide the ability to store recording or live streams. Videos usually contain many images and a significant percentage of redundant information [1]. Images only contain a lot of different visual patterns made up of low-level visual features. Videos contains in addition to visual patterns, temporal, and spatial information, which forms high-level features or structures such as an event or an action happening across the frames [2, 3].

With the emergence of large amounts of video data, an efficient way of representing the videos is needed to be able to search for it quickly when given a query video. This leads us to video hashing, which is easy to search with and requires a small amount of storage. Hashing is the process of mapping given numerical features into binary codes, while still preserving similarity in the original features domain. The method used to map the features is called a hashing function. The binary hash codes are usually small in the size, thus saving a lot of storage cost.

Due to the usual amount of data to be searched in real applications, the nearest neighbor search is usually the go to for quick retrievals in many fields of application like computer vision, data mining and deep learning [4]. But in many cases when retrieving the results to a query, there is no need for the retrieved matches returned to be exact, and sometimes what is asked to be retrieved is only similar matches. Thus, the approximate nearest neighbor (ANN) search is used instead which achieves satisfying retrieval performance [4, 5].

ANN searches with binary codes can be achieved in a sub-linear to constant time complexity [6]. Therefore, hash codes have been used for an efficient ANN search on large datasets, to achieve low query time and low storage cost [6-9]. Images usually contain a lot of data, and videos, being made of multiple images, contain even more data. Most of the data contained in the video is usually redundant due to repetition of similar images with minor differences and same scenery (the meaning is usually within the flow of the images) and generating hash codes by hashing videos is a very appealing way of reducing cost of storing features (by storing binary hash code features instead) and by efficiently reducing search time using ANN search.

Exploiting spatial-temporal information in videos to generate hash methods is still very desirable. Existing hashing approaches for videos use deep learning models to learn hashes by first extracting features from the videos then learn to hash those features [5, 10, 11]. While others incorporate the hashing in the feature learning process to make the hash guide the feature learning process too [12-15].

Advanced deep CNN methods have been used to learn to extract representative features from images [16, 17], some have started adapting those methods to be able to learn extract features from videos [11, 14, 15]. This is through learning temporal information either by using sequential data processing deep learning models like recurrent neural network (RNN), long short-term memory (LSTM) or transformers [14, 15, 43].

Another way is extending the CNNs to incorporate temporal information by adding a temporal dimension to become 3DCNNs. 3DCNNs are CNN with an added dimension to include the temporal information, so they usually span, in addition to the regular 2D dimension of an image in a CNN, a series of frames (which adds the temporal factor) depending on the size of the filter being used.

Deep learning techniques such as CNN models have gone through a lot of advancement and are now able to achieve very high results on various datasets [18-20]. But require a lot of memory and computing power during training due to having a lot of parameters. And 3DCNNs, which one of their uses is in videos feature learning, cost even more memory and compute power. This makes training a model harder.

In this work we try to solve the problem of video retrieval through extracting video features and hashing those features. The hashes are then used for comparing videos in the retrieval process. We utilize a deep learning model to produce hash codes for short video clips. The trained model can then be applied on both the database clips and any query clips to produce hash codes. The hash code of the query video clip is then used to retrieve videos similar to it through a hash code search. Learning to produce hash codes for videos is more complex than images because the increase of diversity of information than what the images provide.

To address the problem of video retrieval we propose a deep learning model made up of two sub-modules that are trained separately. The first module is made up of a 3DCNN and a bidirectional encoder representations from transformers (BERT) layer, this module aims to extract main features from video clips. The 3DCNN is used to extract spatial and temporal features from the input video clip, and the BERT layer is a temporal attention mechanism that can learn to extract temporal features. BERT efficiently learns to fuse contextual information from both temporal directions. BERT has also shown to be better than the just the average pooling layer at the end of a 3DCNN [21]. We use only one layer of BERT in our model as to decrease some computational cost. BERT can be fine-tuned for various tasks, and in here we use it to extract contextual information [22].

The second module, which handles feature hashing, is a neural network made up of a few fully connected layers to hash the features extracted by the first module. Both modules can be fused after training and then applied directly on database videos and query videos to produce hash codes.

They are separate in the training phase to be able to fine-tune each learning processes individually. And to decrease the variance in the result between each hash code size as can be noticed in [42]. Other works that relied on the model learning to hash end to end through hashing losses show minor to noticeable variance between the results [34, 37].

The reported results are evaluated using UCF101 [18] and HMDB51 [19] datasets. The UCF101 dataset contains 13320 short clips distributed in 101 action classes and 27 hours of video data, while the HMDB51 dataset contains a total of 7000 short clips distributed in 51 action classes. Experimental results show how BERT's success in improving 3DCNNs to achieve better results compared with state-of-the-art models. Furthermore, the proposed separation of the two modules allows each module to learn its task faster without overhead. The running of the feature extraction module every iteration during the learning process incurs a large overhead on its own to run and leaves less room for experimenting the hashing part.

The main contributions of this work can be summarized as follow:

(1) A deep video hashing model made up of two sub-modules for video retrieval. The first module is made up of a 3DCNN and a BERT layer, this module handles feature extraction. The second module, which handles feature hashing, is a neural network made up of a few fully connected layers to hash the features extracted by the first module.

(2) We apply transfer learning and use a

pretrained 3D CNN model on an extremely large dataset of video clips.

(3) Generate hash codes of different sizes which yields higher accuracy with little variance between them compared to other models. This is due to the separation of the feature learning process from the hash learning process.

(4) We evaluate the model on two datasets for video action recognition UCF101 [18] and HMDB51 [19] to show that the suggested model outperforms numerous state-of-the-art approaches.

This paper is organized as follows: First we provide a basic background related to the used methods in section 2. We briefly review the related work of deep video hashing and retrieval in section 3. In section 4 we elaborate on the details of the proposed model. Section 5 discusses the used database, retrieval and evaluation methods. Finally, experimental results are provided in section 6, followed by our conclusions in section 7.

2. Basic background

2.1 2D convolutional neural network

2D CNNs apply 2D convolution between a 2D input image or feature map and a filter to extract features to the next layer's feature map. For a feature map f of size X width, and Y height, the value at position (x, y) where x ∈ {1, X} and y ∈ {1, Y} is calculated as show in Eq. (1).

$$f_{xy} = g(\sum_{i,j} w_{ij} \; m_{(i+x)\,(j+y)} + b) \qquad (1)$$

Where m is the input feature map, and w_{ij} are the weights of the 2D filter of size I, J, and i ∈ {1, I} and j ∈ {1, J} and b is the bias. And g is an activation function for non-linearity. Since the same filter passes over the entire image to extract the feature map, the weights needed are reduced, the filter becomes translation invariant, and its generality is increased since it is repeated over the input image. A convolution layer usually contains multiple filters to extract multiple feature maps. Pooling layers are sometimes added to reduce the size of the feature maps. The most used pooling layers are max and average pooling. A CNN is usually made up of multiple convolutional layers and pooling layers, usually by pooling every few convolutional layers. The weights of the filters are updated through training, by back propagating the errors of the results. This way the CNN has its filters adapt to the inputs to be able to extract good spatial features.

2.2 3D convolutional neural network

3D CNNs apply 3D convolution between an input video (series of frames/images) and a 3D filter to extract features to the next layer's feature map. The different here is that 3DCNNs incorporate not only spatial features but also temporal features. For a feature map f of size X width, Y height, and temporal depth T, the value at position (x, y, t) where x ∈ {1, X}, y ∈ {1, Y}, and t ∈ {1, T} is calculated as show in Eq. (2).

$$f_{xyt} = g(\sum_{i,j,k} w_{ijk} \; m_{(i+x)\,(j+y)\,(k+t)} + b) \qquad (2)$$

Where m is the input feature map, and w_{ijk} are the weights of the 3D filter of size I, J, K, and i ∈ {1, I}, j ∈ {1, J}, and k ∈ {1, K} and b is the bias. And g is an activation function for non-linearity. The filter here moves, in addition to vertically and horizontally, across multiple frames to get temporal features in addition to the spatial features that a 2D CNN extracts. Multiple 3D filters are used to extract multiple feature maps in each 3D convolutional layer. 3DCNNs are trained by back propagation of error like regular CNNs.

The downside of 3DCNNs is that they are more expensive computation, memory and storage wise, but they can capture temporal information. This makes them able to classify videos more accurately than regular CNNs. Another challenge in 3DCNNs is frame selection, which can affect how good or bad the 3DCNN is trained and its results while testing [1].

2.3 Bidirectional encoder representations from transformers (BERT)

BERT [22] is the state-of-the-art model for natural language processing (NLP), this is due not only to its architecture, but the way it is trained. BERT uses the same multi-head attention mechanism used in the transformer model [22-23], but it is organized and trained differently. The attention function has the goal of mapping a query and a set of key-value pairs to an output, where all the keys, values, query and output are all vectors. The attention is calculated as a scaled dot-product of the query Q and the keys K, and then the result is divided by the square root of dk (the dimension of the keys). A softmax is applied to the attention (relation between the keys K and the query Q) and then it is multiplied by the Value vector to get the value that the query maps to:

$$Attention\ (Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right) V \qquad (3)$$

This attention mapping is based on a model used in databases to retrieve matching results of a query [24]. The query is matched to the nearest key(s) and the values of those nearest keys are retrieved. In regard to the Neural Network model of the attention-head layer, all 3 Q, K, V are calculated through a linear model which learns to calculate them through backpropagation. Then the value the query Q maps to is calculated based on its relation to the available keys K. The multi-head attention is one layer, and there can be multiple of it with different head sizes (dependent on input size) and different linear layer sizes (most of the linear layers must be of equal sizes).

BERT is different from the regular transformer in that it only encodes, there is no decoding part. BERT is also trained in a bidirectional fashion. The input has some parts of it masked while training, and the model tries to predict those masked parts (Usually words in a sentence in the NLP domain). This random masking helps BERT to learn to use all the words in a sentence at the same time to predict the missing word. And hence it is not limited by one direction and can learn contextual information. BERT can be fine-tuned to solve a variety of problems in NLP, and in here we use it to learn to extract more temporal features from the features extracted from the 3DCNN.

3. Related work on deep hashing

Deep video hashing is an area of research that has gained a lot of attention of the past few years, and research on it is still an appealing topic as the amount of video data generated is growing every day due to many websites that allow upload of recorded videos and even streaming videos (along with chat messages). So, it is becoming more and more of a necessity to organize these videos and search for them with minimal amount of space and time, and this is exactly what hashing provides by generating hash codes.

Present methods for hashing videos are mainly used in two ways. The first is near-duplicate video search where hashing techniques are used to identify near-duplicate videos. The other method hashes are used for is content based video retrieval which addresses retrieving videos that are similar to a supplied query video.

3.1 Near-duplicate video search

Finding duplicate videos has multiple uses, some of the most patent uses would be indexing and searching, copyright protection and copy right infringement. Hashing techniques in [10, 12, 13] were used to search for and identify duplicate videos efficiently. [12] preserves the global and local structures of video features by using multi-view hashing (done by using multiple video features) while learning hashing functions. The proposed learning method is efficient and less time consuming because it is done through mapping to an eigenvalue decomposition problem and solving it. It does not require iterations for hash learning. [10] relies on extracting multi-view hashing by extracting multiple features from keyframes extracted from videos. The hash codes are then computed through a mapping function, which is updated through the errors calculated by a regularized probabilistic model based on pairwise similarities between the videos and their extracted frames. [6] uses kernels which are trained in a supervised manner to extract hash codes from already extracted features. [41] applies a disaggregation hashing in which they incorporate PCA in learn different hashing functions for different parts of the hash code, that are applied to different groups of dimensions in the extracted features.

A summary of methods mentioned in [13] includes:

Feature combination methods such as Gaussian estimation over pairwise distances for multiple feature combination, global view hashing which uses relations between multiple views of a video.

Matching methods such as: A- frame-level matching by computing the probability of the videos matching through the number of matched frames between videos, a similarity measurement used to contribute to the probability score is bin-to-bin comparison of a bag of words feature of a frame. B- video-level matching where the video is usually represented by a global feature like an indexing pattern or an aggregate feature vector, and a similarity measure like euclidean distance or hamming distance is used to compare similarity. C- hybrid-level matching which combines parts of the previous two, usually to preserve global and local features of the video.

Hashing methods which can be deep hashing methods using neural networks (through 2D convolutional neural networks (CNN) or 3D CNN). Or a learned mapping function through a probabilistic model.

3.2 Content based video retrieval

The amount of video data that is being generated has increased significantly in terms of both the quantity and categories. And just like image search which has boomed in the past few years, video search based on content of a query video will also gradually be in demand too. [5, 11, 14, 15] use different techniques to retrieve videos similar to a query video. [14] uniformly sample frames and applies a CNN to extract features from them then the features are fed through an RNN for hashing, then the frames are reconstructed again from the code (for training purposes). The model is trained via 3 losses, neighborhood similarity, visual content reconstruction and neighborhood information reconstruction losses. [5] uses quantization-based hashing. The hashing is done through multiplying by an affinity matrix. The matrix values are updated through minimizing similarity error (hamming distance) and quantization error which the l2 norm between the hash code cluster centers and the data point. Both errors contain a hyper parameter for tuning. The models are trained on labeled and non-labeled datasets.

[11] uses CNN to detect 5 concepts in the image (Who is in the image in terms of the quantity of persons and what is in the image, where is the image taken, when was it taken (its daytime) and how was it taken in terms of camera shot type). The CNN is trained to both extract those features through classification error, and to extract a long and short binary hash code from those features while still having the hashing process contributing to the learning at the same time as the classification. An addition of face recognition and identification is also used alongside the hash codes to extract more information. The system after training retrieves similar clips to the queried one based on the concepts extracted (through short binary code matching then long binary code matching which are extracted through the CNN) and the face recognition is added to further improve the relevancy of the retrieved results by having the same person(s).

[15] uses CNN followed by gated recurrent units (GRU)s for learning temporal information to achieve a deep hashing model. The weights of the model are trained and updated through quantization and triplet loss.

There have been several methods that adapted the use of 2D CNNs along with a sequential data processing NN layer(s) in addition to additional losses to obtain a video hashing deep model that hashes in an end-to-end manner [11, 14, 15, 26-31]. The additional sequential data processing NNs are used to obtain temporal features that are not extracted from the CNNs. [14] used VGG16 CNN to extract features from M uniformly sampled frames, and then uses LSTM auto encoders to generate hash codes. They also add neighborhood similarity loss on the hash codes and visual content reconstruction loss for the decoders to help train the model. [11] uses VGG16 CNN to extract features of the place (or environment) in the video and learns to generates long and short hash codes for them, in addition to other features that are extracted using other CNNs to be able to identify similarities in videos and get similar videos for any query video. [15] uses Resnet34 CNN for features and deep gated recurrent units GRUs to get hash codes.

In [26] after extracting features using VGG19 an attention-based LSTM is used to further process the features then a fully connected (FC) layer to get the hashes. [27] uses differential LSTM (DLSTM) along with a variation of AlexNet to encode the features into hashes. [28] uses Bi-directional LSTM (BLSTM) along with LSTM in some of the layers with VGG19 CNN to extract features and then get the hash codes through the LSTMs, depending on the LSTM to obtain temporal information. [29] uses VGG16 and resnet50 each for a different dataset to extract features then pass them into 1-D CNNs networks to encode and decode the hash codes and for the decoding addition LSTM layers are used for decoding. The 1D-CNNs and LSTM are used to auto encode, the LSTM is only present in decoding. A stack of CNNs which they call HetConv-MK (stacked multi kernel convolutional models) is used in [30] to extract features through the multiple CNN and then the features are fused and passed through a BLSTM and then an FC to get the hash codes. [31] uses Alexnet CNN with LSTM to get the hash codes.

Some works have tried making slight variations in the CNN architecture. [32] applies a CNN on multiple frames to extract features then a weighted sum function is applied to the features, the weights are learned during training, the result is passed through a layer fully connected layer to produce hash codes, the hash codes are evaluated to generate masks for each category which are used to further increase accuracy. [33] uses slow fusion architecture which is a mixture of early and late fusions on a CNN architecture to generate hash codes. The CNN is applied on multiple frames then features are fused and are passed through fully connected layers to generate hash codes. The location of the fusing of the features determines whether it is an early or late fusion architecture. [34] use a fully connected layer on CNN features to generate hash codes, the codes are used to filter the query input through hash code

Received: March 15, 2022. Revised: May 24, 2022.

similarity. Then the query is ranked according to the similarity of the feature vectors extracted from the CNN layer before hashing them. [35] slice the features extracted from the CNN into segments, each segment is fully connected to one neuron, each neuron generates 1 bit in the hash code.

Other works have suggested that just working with 2DCNN is insufficient [36] and that sometimes even when LSTMs are used the model is unable to model complex motion features [37]. So those works used 3DCNN instead and modified it to get an end-to-end hashing model. [38] Uses pretrained 3DCNN MFNet, it is fine-tuned after adding a hashing FC to get hash codes. Additional losses are added on the generated hashes, losses include learning to map each hash code to a pre-generated hash center to represent the class. [36] uses C3DCNN to obtain features then passes it through a feature selection method. A global average pool is used after feature selection along with a principal component analysis (PCA) model to generate the final hash code.

Some models tend to train an end-to-end fashion where the feature extraction and hashing is trained together to guide the feature extraction through the hashes. But this usually achieves a high variation of the accuracy of the results when increasing or decreasing the hash code size, that is why we chose to separate learning 3D feature extraction from the learning of hash generation. Another advantage of separating the two learning modules is the ability to train the hash generation module only when the objective is adjusting the hashing model's size or hash code output size instead of retaining the whole model.

4. Proposed method

We propose a deep video hashing model to obtain hash codes for short video clips. Our model is trained and tested on action recognition videos. The model is made up of two sub-modules. The first sub-module is the feature extraction module that uses

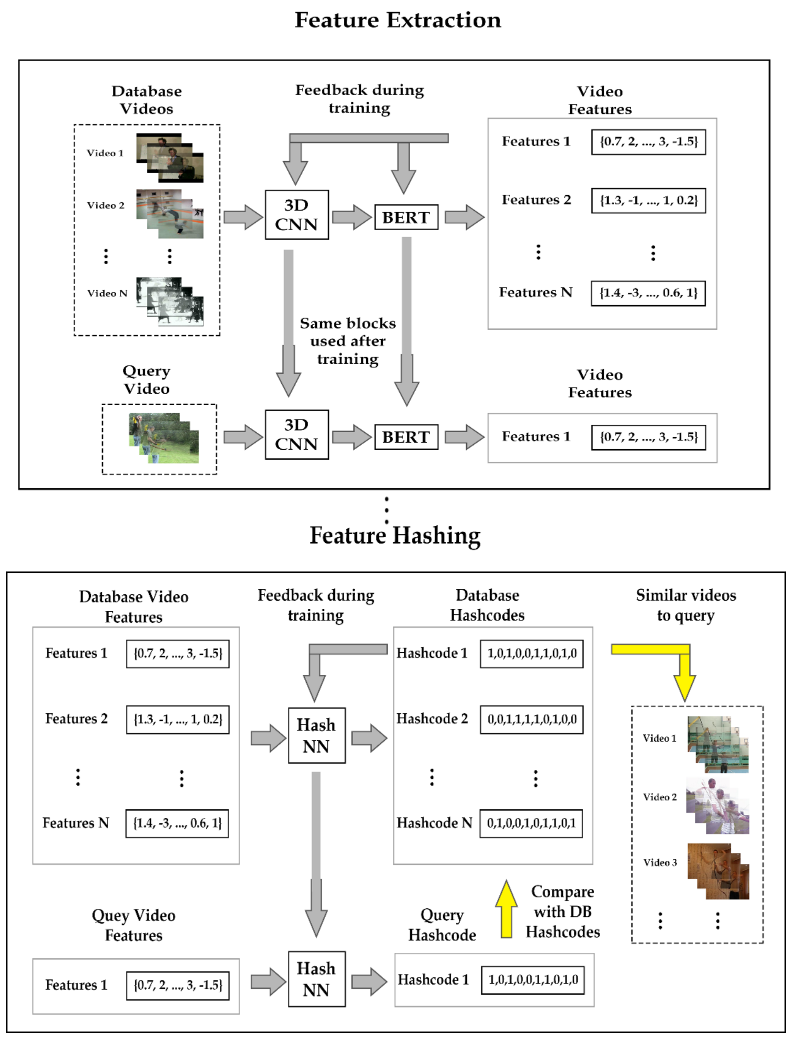


Figure. 1 Framework of the proposed deep hashing model

3DCNN to extract spatial and temporal features, followed by BERT to further process the features extracted and learn to extract contextual information. The second sub-module is a series of linear (Fully connected) neural network layers, that takes in the features extracted from the first model and learns to generate hash codes from them.

4.1 The proposed deep video hashing framework

The framework of our deep hashing model is shown in Fig. 1. The first part made up of 3DCNN and BERT which is applied to the videos to extract features and then the features are passed to the second part a neural network to generate hash codes. After training the two parts of the model, hash codes are extracted and saved from the database videos (which are the training videos for our case). Whenever there is a query video, it is passed through both parts of the model to generate its hash code, then the hash code is compared with the saved hash codes of the database videos and then the most similar videos are retrieved based on the most similar hash codes.

4.2 3DCNN and BERT for feature extraction

Our digital video hashing model is started by feature extraction from video using 3DCNN. The 3DCNN used is r(2+1)d which has shown to perform better than regular 3DCNNs in action recognition [39]. It is different in terms of splitting the 3D convolution into 2D convolution to extract spatial features followed by 1D convolution for extracting temporal features. The 3DCNN used is based of Resnet 34 in the number of layers but replaces 3D kernels with 2D followed by 1D kernels. *Fig. 2* illustrates the inner layers of the 3DCNN. Every 2D+1D convolution is considered a layer, as the purpose of one layer here is to extract both spatial information through 2D convolution on each frame and temporal information through the 1D convolution applied on multiple frames.

After each 3D convolution layer, a 3D batch normalization is applied followed by an activation function Rectified Linear Unit (ReLU). More layers can be added to the model, but that comes at a cost of memory and complexity with usually minor improvements. K frames from a video are selected and fed through the 3DCNN passing over the multiple convolutional layers to extract temporal and spatial features. At the end of the model usually an average pooling layer is used for temporal pooling layer is used for feature pooling. We keep a spatial average pooling but use BERT for temporal

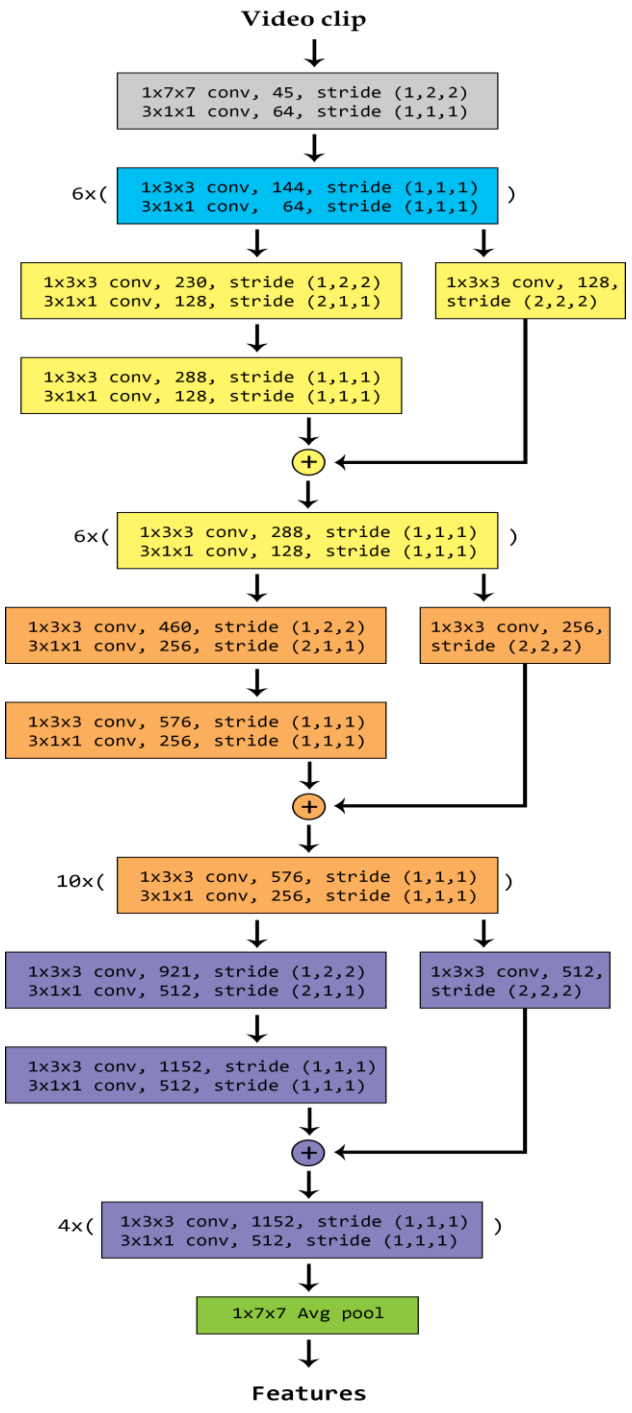

Video clip
1x7x7 conv, 45, stride (1,2,2)
3x1x1 conv, 64, stride (1,1,1)
6x(1x3x3 conv, 144, stride (1,1,1)
3x1x1 conv, 64, stride (1,1,1))
1x3x3 conv, 230, stride (1,2,2)
3x1x1 conv, 128, stride (2,1,1)
1x3x3 conv, 128, stride (2,2,2)
1x3x3 conv, 288, stride (1,1,1)
3x1x1 conv, 128, stride (1,1,1)
+
6x(1x3x3 conv, 288, stride (1,1,1)
3x1x1 conv, 128, stride (1,1,1))
1x3x3 conv, 460, stride (1,2,2)
3x1x1 conv, 256, stride (2,1,1)
1x3x3 conv, 256, stride (2,2,2)
1x3x3 conv, 576, stride (1,1,1)
3x1x1 conv, 256, stride (1,1,1)
+
10x(1x3x3 conv, 576, stride (1,1,1)
3x1x1 conv, 256, stride (1,1,1))
1x3x3 conv, 921, stride (1,2,2)
3x1x1 conv, 512, stride (2,1,1)
1x3x3 conv, 512, stride (2,2,2)
1x3x3 conv, 1152, stride (1,1,1)
3x1x1 conv, 512, stride (1,1,1)
+
4x(1x3x3 conv, 1152, stride (1,1,1)
3x1x1 conv, 512, stride (1,1,1))
1x7x7 Avg pool
Features

Figure. 2 The architecture of the 3DCNN part of the model which is made up of r(2+1)d convolutions

pooling as it can learn contextual information and to extract the better temporal features. BERT encodes data from both directions using self-attention mechanisms. It has helped in further advancement in the Natural Language Processing (NLP) domain as it can use contextual information from both sides instead of relying on just one (left-to-right or right-to-left).

A BERT layer is added after the 3DCNN to perform temporal pooling. After the features are

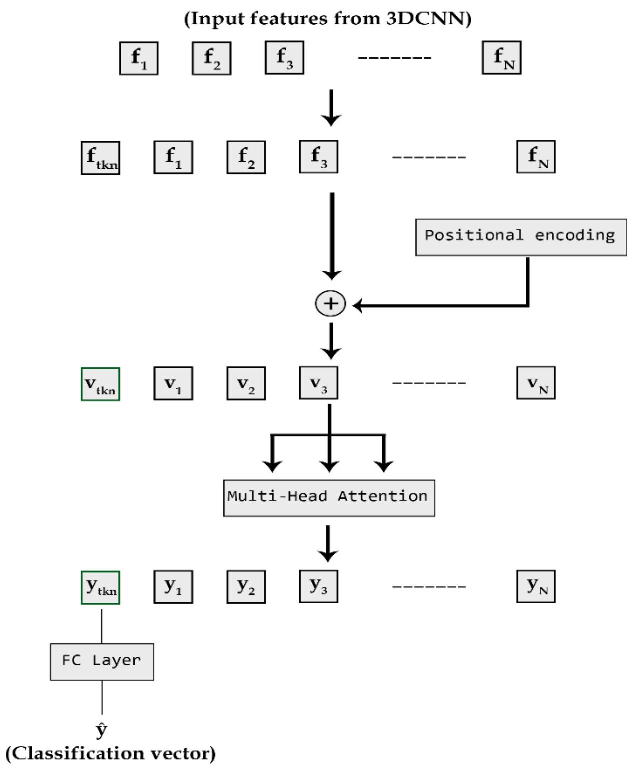

(Input features from 3DCNN)
f_1 f_2 f_3 ------- f_N
f_{tkn} f_1 f_2 f_3 ------- f_N
Positional encoding
v_{tkn} v_1 v_2 v_3 ------- v_N
Multi-Head Attention
y_{tkn} y_1 y_2 y_3 ------- y_N
FC Layer
$\hat{y}$
(Classification vector)

Figure. 3 The architecture of BERT layer

extracted from K frames in a video by passing it through the 3DCNN without temporal pooling at the end. First a classification embedding token is add (f_{tkn}) is added to the extracted feature maps from the 3DCNN. The token is stacked on the temporal dimension. A positional encoding which is learned is added at to these features. Then those features are passed into a multi-head attention layer as can be seen in Fig. 3.

The output token (y_{tkn}) from the output of the attention layer is passed through a linear layer to get the classification vector $\hat{y}$. We use 8 attention heads in our only one attention layer. More layers can be added to the model to further increase the learning capacity of the model, though that would likely need an increase in the size of the features extracted from the 3DCNN by adding more kernels. A Binary cross entropy loss is used to backpropagate the error in classification back to the model and update its weights. A summary of this sub-module of the model can be seen on the left part of Fig. 4.

4.3 2D CNN classification layer

The second part of the model is made up of linear neural network layers. A dropout layer is added at the start of the model. All the linear layers have a batch normalization layer and a ReLU activation after them except for the hashing layer which is the 5th layer. It has Tanh as an activation function. And the last (6th) linear layer that turns the hashes into a classification vector has a softmax after it, to enable computing of binary cross entropy losses. The hashing layer is followed by a Tanh activation function because it yields 1, -1 which can be mapped to 1, 0 hash codes later when needed or used as is. It converges faster and suffers less of a vanishing gradient problem than the sigmoid. The purpose of this classification layer is training the model through classification loss. A summary of the layers for the hashing sub-module can be seen on the right part of Fig. 4.

4.4 Near-duplicate video search

The input of our model is 32 frames of a video clip, each of a size 112x112, the frames as selected from the middle of the video. We use the R(2+1)d pretrained on IG65M [40] dataset. For BERT we use the default configuration used in [21]. A FC layer is added at the end with its size dependent on the number of classes. Due to memory limit the batch size used is 21 and the weights are updated every 6 iterations. For tuning the parameters of the model, the loss used is the cross-entropy loss between the classification output of the last layer and the target class, and the optimizer is AdamW [41] which is Adam optimizer with weight decay. The model is trained for 40 epochs with a dropout of 0.8, with an initial learning rate of 1e-5 which decays by a factor

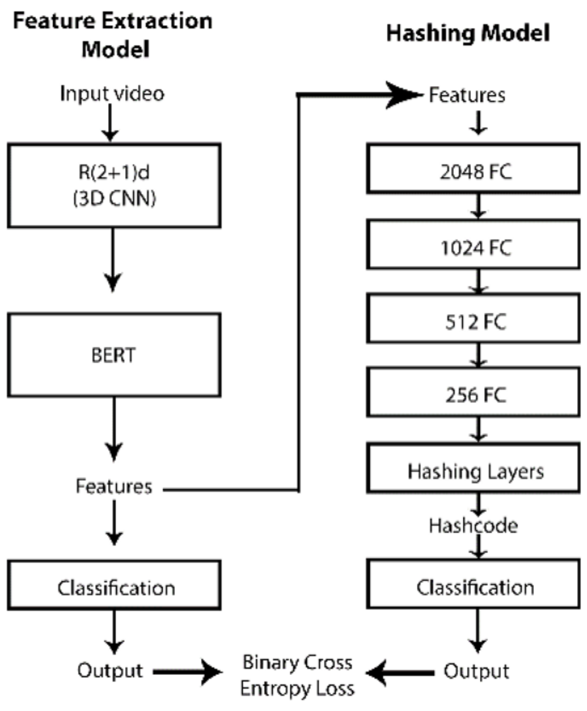


Figure. 4 The architecture of the models for deep video hashing through learning to extract features then learning to hash those features

of 10 each time the loss does not decrease for 5 epochs.

After the 3DCNN model is fine tuned to the selected dataset, we use it as a feature extractor. The output of the BERT layer which is taken as input to the classification layer becomes our feature vector. We use the average of 10 crops' feature vector for a video as the input to the hashing model, the 10 crops are from the same 32 frames selected.

The hashing model is made up of 6 FC layers the first layer starts with the size of 2048 and for each consecutive layer until the 4th layer, the size is halved, the fifth layer is the hashing layer with the size set to the preferred bit size, then a classification layer with the size depending on the dataset's number of classes. The training of the hashing model is summarized in Algorithm 1.

After the model is trained there is no use for the classification layer, and the sign function is used instead of tanh to get the hash codes in the fifth layer. This is because hash codes need to be either 1 or -1, -1 being the same as 0 in binary hash codes. The batch size is set to the size of the input dataset for training since the model is relatively small, batch sizes of 64, 100, 200, 400 have also been experimented with and yielded similar or mostly lesser results so we chose all the data as batch size.

Algorithm 1 Training Hashes

```
Input: $x_i$, the extracted video features using 3DCNN and BERT.
Output: $h_i$ the hash code representing the videos.
Output: $u_i$ the classification of the.
Step 1 Initialization:
Initialize the Hash NN weights with uniform random values.
Set dropout probability to 0.8
Set learning rate to 0.05 and momentum 0.9 for SGD optimizer
Set learning decay to 0.8 and apply it every 60 iterations
Step 2 Hash NN network learning:
for $iter = 1, 2, \ldots 360$ do
  for $t = 1, 2, \ldots$ Total batches:
    Forward Propagation:
    Set $X$ to the batch subset $t$ of features from $x_i$
    for $j = 1, 2, \ldots, layers-2$:
      Apply dropout to the input $X$.
      Multiply input by weights of the layer $j$ to generate output $O$.
      Set $O$ = ReLU($O$)
      Apply batch normalization to $O$ to get normalized output $B$.
      Set $X$ to $O$, to be the input for the next layer
    end for
    Multiply $X$ by the weights of the hash layer to get output $O$
    Apply Tanh on output $O$ to get $H$ the hash codes of the features. Every row in the matrix H is $h_i$ which is the hash code of the ith feature.
    Pass through $h_i$ through the last layer to get the classification vectors $u_i$.
    Backward Propagation:
    Compute gradients of losses using the SGD settings, the losses are computed through applying softmax loss on the classification vector $u_i$.
    Apply SGD on the losses to update the weights of the network.
  end for
end for
Return: Trained Hash NN
```

5. Datasets and experiment setup

5.1 The proposed deep video hashing framework

We use two benchmark datasets for action recognition to evaluate our method and compare them against other methods: UCF101 [18] and HMDB51 [19].

5.1.1. UCF101 dataset

The UCF101 dataset contains 13320 short clips distributed in 101 action classes and 27 hours of video data. The database is made up of realistic user uploaded videos which contains camera motion and cluttered background. The UCF101 dataset is an extension of the UCF50 dataset which contained different actions like biking, diving, drumming and adds to it more 51 more. The clips of one action are divided into 25 groups, each group contains 4 to 7 clips each. The clips in one group have some common features that they share such as the persons or backgrounds in it.

The average clip length is around 7 seconds. The dataset has 3-fold cross validation splits supplied with the videos, each split contains around more than 9000 clips for training and more than 3000 for validation. A subset of actions of the UCF101 data set is shown in Fig. 5.

5.1.2. HMDB51

The HMDB51 dataset contains a total of 7000 short clips distributed in 51 action classes. Each of the 3 splits for the dataset contains a subset of 5100 clip. Each action gets 100 clips in every split, 70 for training and 30 for validation. HMDB51 is used to evaluate performance of human action recognition

Received: March 15, 2022. Revised: May 24, 2022.

Figure. 5 Sample actions of UCF101 dataset

Figure. 6 Sample actions of HMDB51 dataset

Table 1. Comparison of mAP results with other hashing methods on UC101 and HMDB51 datasets

	(a) UCF 101 mAP					
Method used \ Bit length	16	32	64	128	256	512
DH[41]	0.300	0.290	0.470	-	-	-
SRH[30]	0.716	0.692	0.754	0.781	-	-
DVH[32]	0.701	0.705	0.712	-	-	-
KSH[6]	-	-	0.716	0.786	0.810	0.848
DBH[33]	-	-	0.681	0.736	0.766	0.785
DNNH[34]	-	-	0.740	0.789	0.817	0.835
SVHM[42]	-	-	0.798	0.801	0.806	-
CSQ[37]	**0.838**	**0.875**	0.874	-	-	-
DHWCM[31]	-	0.857	**0.901**	**0.949**	**0.959**	**0.953**
R(2+1) BERT (Proposed)	**0.985**	**0.986**	**0.986**	**0.985**	**0.986**	**0.986**
	(b) HMDB51 mAP					
Method used \ Bit length	16	32	64	128	256	512
DH[41]	0.360	0.360	0.310	-	-	-
SRH[30]	0.491	0.503	0.509	-	-	-
DVH[32]	0.441	0.456	0.518	-	-	-
KSH[6]	-	-	0.431	0.464	0.450	0.473
DBH[33]	-	-	0.389	0.391	0.386	0.346
DNNH[34]	-	-	0.487	0.503	0.493	0.480
CSQ[37]	**0.527**	**0.565**	0.579	-	-	-
SVHM[42]	-	-	0.562	0.565	0.575	-
DHWCM[31]	-	0.487	0.**605**	**0.588**	**0.588**	**0.672**
R(2+1) BERT (Proposed)	**0.848**	**0.849**	**0.854**	**0.852**	**0.849**	**0.849**

system. The actions in this dataset are divided into 5 groups: General facial actions, general body movements, facial actions, body movements and body movement for human interaction.

This dataset although smaller is harder than UCF101 to achieve a high accuracy on, usually because of some of the facial actions included in it. The sources of the video clips used are digitized movies, public databases, YouTube, and other videos available on the internet. A subset of actions of the HMDB51 dataset can be seen in Fig. 6.

5.2 Evaluation measures

The we use the mean average precision (mAP) to evaluate our results and compare them with the work of others. The mAP can be calculated by first getting the average precision (AP) for every test clip. Then we get the mean of the average precisions we get from all the test clips.

The AP is defined as the mean of the precision scores after each relevant document is retrieved. The precision score is the precision of the top *r* retrieved document. The Precision score when retrieving *r* documents is as follows calculated as follows:

$$P@r = \frac{Relevant\ documents}{r} \quad (4)$$

The AP when the total number of relevant documents available is *R*:

$$AP = \frac{\sum_r^R P@r}{R} \quad (5)$$

Sometimes not all the relevant documents for each query are the same. And sometimes we are not interested in all of them, just the top *k* of them. In this case mAP@k is used. This is the same as mAP, but instead of the total number of relevant documents being *R*. The total number becomes the minimum of *k* and *R*. Where we test it on the top k, but if there aren't enough then we test on all that is available. The mAP for the HMDB51 dataset is mAP@70 and for UCF101 mAP@100.

We also use precision-recall (PR) curve to evaluate the performance of the proposed model. The PR curve is calculated through calculating the precision and recall when predicting classes at different thresholds on their similarity score making the ones that pass the threshold belong to the class. The clips that are predicted as belonging to the class are positives and the ones that aren't, are negatives. If that prediction is correct then that prediction is true, otherwise it would be false.

The precision is calculated as the number of true positives (TP) over the number of true and false positives (FP), which is percentage of positives being correct. And the recall is the number of true positives over the number of true positives and false negatives (FN), which is also the number of samples belonging to the class recalled over all the samples that belong to the class that exist.

$$Precision = \frac{TP}{TP+FP} \quad (6)$$

$$Recall = \frac{TP}{TP+FN} \quad (7)$$

The hash code generated from the model are rounded to 1, -1 due to the Tanh. The similarity of the hash code is thus calculated by multiplying every bit in the query hash code with the bit in the database hash codes in its same position and then the result is summed. The higher the sum, the more similar the codes are. The hash codes can be mapped to the binary 1, 0, and then XORed to help get the hamming distance. But it was faster to use them as they are for testing. After finding the similarity and ordering the results from most similar to least similar the mAP is calculated.

6. Experimental results

The comparison for the UCF101 and HMDB51 is illustrated in Table 1. Values of other compared methods are taken from [31, 37, 42] or the referenced papers directly. The metric compared being the mean average precision (mAP).

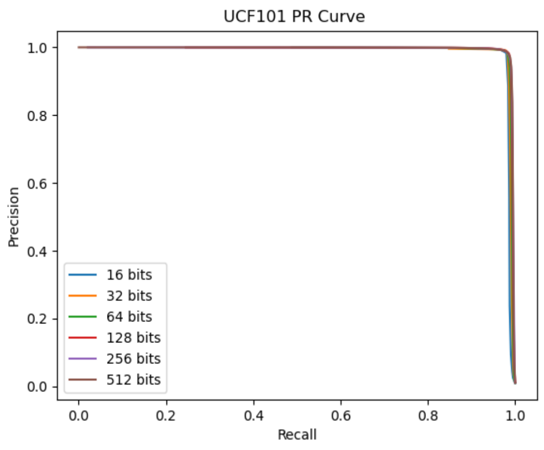
Table 1 compare the proposed hashing model against different state of the art models based on mAP using different bit length (e.g. 16, 32, 64, 128, 256, and 512). As shows in table 1 the proposed model outperforms the state-of-the-art by at least 2 % for each bit length in UCF101 and more than 20 % for the HMDB51. This is because the 3DCNN can extract better features through the help of a transformer layer BERT which learns contextual information which is not present in other models. And it was trained solely on extracting features to classify each video's actions, not being affected by hashing training.

Although 3DCNN part can train on both feature extraction and hashing at the same time and using one module but the stability in the mAP values across different bit lengths will vary. You'll find the least variance in mAP in our model because of separating the feature learning and hash learning process. And minor variance in [42]. This is due to having the model train on feature extraction first, then the model was modified to incorporate hash learning as fine-tuning of the model. But more variance in [31, 32, 34, 37] where the model learns end to end.

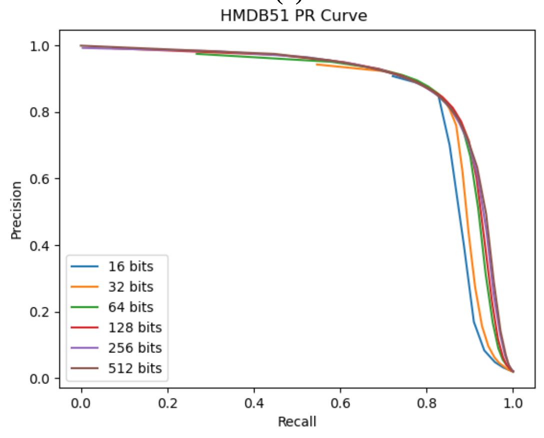
Training a 3DCNN to generate hash codes end to end can be difficult to fully optimize. Such a model usually requires other losses like classification loss in addition to hashing losses to get better results [15, 25, 27]. This, although better than training on hash losses alone, may require grid search to find optimal values to combine the losses to achieve the best results. And it also faces the challenge that the hashing part of the model is learning to generate hash codes for features that keep slightly changing. As the feature extraction part of the model keeps updating itself as well to learn to generate better features as well. So, the features are constantly changing and thus the hashing layer keeps needing to change to adapt as well.

The average time for extracting the features of 10 crops of a single video is 3.475s for the UCF101

Received: March 15, 2022. Revised: May 24, 2022.



(a)



(b)

Figure. 7 Comparison of PR curves for different hash code bit sizes for both (a) UCF101 and (b) HMBD51 datasets

dataset and 1.628s for the HMDB51 dataset. The average for a single crop of a single video is 0.544s for the UCF101 and 0.272 for the HMDB51. This variation in time is probably due to the UCF101 videos having more frames, as the time calculated includes loading the videos and running them. The time also does not include running a batch of videos (except for the 10 crops of a single video), which is faster than extracting features from videos one by one.

The average time for generating has codes is around 2 to 3 Microsecond for both UCF101 and HMDB51. Features are passed through the hashing layer's neural network in batches between 1000 and 2000, because it takes up less memory.

Fig. 7 shows the precision-recall (PR) curve for the HMDB51 dataset, and the UCF101 dataset using different hash code sizes. The PR curve is a plot of the precision against the recall. The area under the curve (AUC) in a PR curve helps determine how good a classifier is, the bigger the AUC, the better the classifier.

We have observed that incorporating the hash learning into the model instead of separating it causes variations in the hashing performance across different bit sizes. This because the hash layers don't get enough training due to the features being extracted changing with no freezing for feature extraction layers in the learning process. Once, they are frozen the hashing layer is not gaining enough training due to the time and processing cost. Separating feature extraction module from hash generation module will result in better training of hashing values from extracted features by allowing more processing power for the tuning and testing different hyper parameters. State-of-the-art models incorporate the hashing process in training the model and thus result in high variations among mAP values. The learning process of these models is affected by the hashing layer's performance.

7. Conclusion

We proposed using 3DCNN with BERT model as a deep hashing model for action recognition. We separate the task of hashing onto two models that can be combined after training. One for feature extraction and one for hashing. This separation allowed the model to learn each task to its best individually. Experimental results show that the proposed model yields state-of-the-art performance on video retrieval. It improves the previous best mAP by more than 2 % and 24 % for UCF101 and HMDB51 datasets respectively. We also achieve a decreased variance among mAP in different bit sizes compared with other models' results.

8. Future work

We plan to improve the model in future works by trying to reduce the feature extraction model size to increase the model's speed, while keeping the same or close accuracy. We will also experiment with using more frame samples of a video and adding additional transformer layers like BERT. We plan to experiment with other video datasets as well.

Conflicts of interest

The authors declare that they have no conflict of interest.

Author contributions

The conceptualization, methodology, software, validation, analysis, comparison, writing draft, and visualizations have been done by first author. The supervision, writing and work review and project administration, have been done by second and third authors.

Received: March 15, 2022. Revised: May 24, 2022.

References

[1] S. Chakraborty, O. Tickoo, and R. Iyer, "Adaptive Keyframe Selection for Video Summarization", In: *Proc. of 2015 IEEE Winter Conference on Applications of Computer Vision*, pp. 702-709, 2015, doi: 10.1109/WACV.2015.99.

[2] W. Hu, N. Xie, L. Li, X. Zeng, and S. Maybank, "A Survey on Visual Content-Based Video Indexing and Retrieval", *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, Vol. 41, No. 6, pp. 797-819, Nov. 2011, doi: 10.1109/TSMCC.2011.2109710.

[3] D. Huang, V. Ramanathan, D. Mahajan, L. Torresani, M. Paluri, L. F. Fei, and J. C. Niebles, "What Makes a Video a Video: Analyzing Temporal Information in Video Understanding Models and Datasets", In: *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 7366-7375, 2018.

[4] A. Adoni, P. Indyk, and I. Razenshteyn, "Approximate Nearest Neighbor Search in High Dimensions", In: *Proc. of the International Congress of Mathematicians (ICM 2018)*, pp. 3287-3318, 2019, doi: 10.1142/9789813272880_0182.

[5] J. Song, L. Gao, L. Liu, X. Zhu, and N. Sebe, "Quantization-based hashing: a general framework for scalable image and video retrieval", *Pattern Recognition*, Vol. 75, pp. 175-187, 2018, ISSN 0031-3203, https://doi.org/10.1016/j.patcog.2017.03.021.

[6] W. Liu, J. Wang, R. Ji, Y. Jiang, and S. Chang, "Supervised hashing with kernels", In: *Proc. of 2012 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2074-2081, 2012, doi: 10.1109/CVPR.2012.6247912.

[7] A. Gionis, P. Indyk, and R. Motwani, "Similarity search in high dimensions via hashing", In: *Proc. of International Conference on Very Large Data Bases*, pp. 518–529, 1999.

[8] B. Kulis and K. Grauman, "Kernelized locality-sensitive hashing for scalable image search", In: *Proc. of 2009 IEEE 12th International Conference on Computer Vision*, pp. 2130-2137, 2009, doi: 10.1109/ICCV.2009.5459466.

[9] Y. Guo, G. Ding, L. Liu, J. Han, and L. Shao, "Learning to hash with optimized anchor embedding for scalable retrieval", *IEEE Transactions on Image Processing*, Vol. 26, No. 3, pp. 1344–1354, 2017.

[10] Y. Hao, T. Mu, R. Hong, M. Wang, N. An, and J. Y. Goulermas, "Stochastic Multiview Hashing for Large-Scale Near-Duplicate Video Retrieval", *IEEE Transactions on Multimedia*, Vol. 19, No. 1, pp. 1-14, 2017, doi: 10.1109/TMM.2016.2610324.

[11] M. Mühling, N. Korfhage, E. Müller, C. Otto, M. Springstein, T. Langelage, U. Veith, R. Ewerth, and B. Freisleben, "Deep learning for content-based video retrieval in film and television production", *Multimed Tools Appl.*, Vol. 76, pp. 22169–22194, 2017, https://doi.org/10.1007/s11042-017-4962-9.

[12] X. Nie, W. Jing, C. Cui, C. J. Zhang, L. Zhu, and Y. Yin, "Joint Multi-View Hashing for Large-Scale Near-Duplicate Video Retrieval", *IEEE Transactions on Knowledge and Data Engineering*, Vol. 32, No. 10, pp. 1951-1965, 2020, doi: 10.1109/TKDE.2019.2913383.

[13] L. Shen, R. Hong, and Y. Hao, "Advance on large scale near-duplicate video retrieval", *Front. Comput. Sci.* Vol. 14, 145702, 2020, https://doi.org/10.1007/s11704-019-8229-7.

[14] S. Li, Z. Chen, J. Lu, X. Li, and J. Zhou, "Neighborhood Preserving Hashing for Scalable Video Retrieval", In: *Proc. of 2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 8211-8220, 2019, doi: 10.1109/ICCV.2019.00830.

[15] L. Shen, R. Hong, H. Zhang, X. Tian, and M. Wang, 2019, "Video Retrieval with Similarity-Preserving Deep Temporal Hashing", *ACM Trans. Multimedia Comput. Commun. Appl.*, Vol. 15, No. 4, Article 109, 2020, doi: 10.1145/3356316.

[16] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition", *arXiv* 1409.1556, 2014.

[17] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition", In: *Proc. of 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770-778, 2016, doi: 10.1109/CVPR.2016.90.

[18] K. Soomro, A. Zamir, and M. Shah, *UCF101: A Dataset of 101 Human Actions Classes From Videos in The Wild. CoRR.*, 2012

[19] H. Kuehne, H. Jhuang, E. Garrote, T. Poggio, and T. Serre, "HMDB: A large video database for human motion recognition', In: *Proc. of 2011 International Conference on Computer Vision*, pp. 2556-2563, 2011, doi: 10.1109/ICCV.2011.6126543.

[20] F. C. Heilbron, V. Escorcia, B. Ghanem, and J. C. Niebles, "ActivityNet: A large-scale video benchmark for human activity understanding", In: *Proc. of 2015 IEEE Conference on*

Computer Vision and Pattern Recognition (CVPR), pp. 961-970, 2015, doi: 10.1109/CVPR.2015.7298698.

[21] M. Kalfaoglu, S. Kalkan, and A. Alatan, “Late Temporal Modeling in 3D CNN Architectures with BERT for Action Recognition. Computer Vision – ECCV 2020 Workshops”, In: *Proc. of ECCV 2020. Lecture Notes in Computer Science*, Vol. 12539. pp. 731–747, 2020, doi: 10.1007/978-3-030-68238-5_48.

[22] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova, *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*, 2018.

[23] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, A. Kaiser, and I. Polosukhin, “Attention is all you need”, *Advances in Neural Information Processing Systems*, Vol. 2017-Decem, pp. 5999-6009, 2017.

[24] T. Guo and H. Gao, “Bidirectional Attention for SQL Generation”, *arXiv:1801.00076*, 2017.

[25] Y. Wang, X. Nie, Y. Shi, X. Zhou, and Y. Yin, “Attention-Based Video Hashing for Large-Scale Video Retrieval”, *IEEE Transactions on Cognitive and Developmental Systems*, Vol. 13, No. 3, pp. 491-502, 2021, doi: 10.1109/TCDS.2019.2963339.

[26] N. Zhuang, J. Ye, and K. A. Hua, “DLSTM approach to video modeling with hashing for large-scale video retrieval”, In: *Proc. of 2016 23rd International Conference on Pattern Recognition (ICPR)*, pp. 3222-3227, 2016, doi: 10.1109/ICPR.2016.7900131.

[27] J. Song, H. Zhang, X. Li, L. Gao, M. Wang, and R. Hong, “Self-Supervised Video Hashing With Hierarchical Binary Auto-Encoder”, *IEEE Transactions on Image Processing*, Vol. 27, No. 7, pp. 3210-3221, 2018, doi: 10.1109/TIP.2018.2814344.

[28] S. Li, Z. Chen, X. Li, J. Lu, and J. Zhou, “Unsupervised Variational Video Hashing With 1D-CNN-LSTM Networks”, *IEEE Transactions on Multimedia*, Vol. 22, No. 6, pp. 1542-1554, 2020, doi: 10.1109/TMM.2019.2946096.

[29] R. Anuranji and H. Srimathi, “A supervised deep convolutional based bidirectional long short term memory video hashing for large scale video retrieval applications”, *Digital Signal Processing*, Vol. 102, 2020, doi: 10.1016/j.dsp.2020.102729.

[30] Y. Gu, C. Ma, and J. Yang, “Supervised recurrent hashing for large scale video retrieval”, In: Proc *of the 24th ACM international conference on Multimedia (MM '16). Association for Computing Machinery*, New York, NY, USA, pp. 272–276, 2016, doi: 10.1145/2964284.2967225.

[31] X. Liu, L. Zhao, D. Ding, and Y. Dong, “Deep Hashing with Category Mask for Fast Video Retrieval”, *arXiv:1712.08315*, 2017

[32] V. E. Liong, J. Lu, Y. P. Tan, and J. Zhou, “Deep Video Hashing”, *IEEE Transactions on Multimedia*, Vol. 19, No. 6, pp. 1209-1219, 2017, doi: 10.1109/TMM.2016.2645404.

[33] K. Lin, H. F. Yang, J. H. Hsiao, and C. S. Chen, “Deep learning of binary hash codes for fast image retrieval”, In: *Proc. of 2015 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 27-35, 2015, doi: 10.1109/CVPRW.2015.7301269.

[34] H. Lai, Y. Pan, Y. Liu, and S. Yan, “Simultaneous feature learning and hash coding with deep neural networks”, In: *Proc. of 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3270-3278, 2015, doi: 10.1109/CVPR.2015.7298947.

[35] A. Ullah, K. Muhammad, T. Hussain, S. W. Baik, and V. H. C. D. Albuquerque, “Event-Oriented 3D Convolutional Features Selection and Hash Codes Generation Using PCA for Video Retrieval”, *IEEE Access*, Vol. 8, pp. 196529-196540, 2020, doi: 10.1109/ACCESS.2020.3029834.

[36] Y. Chen, Y. Kalantidis, J. Li, S. Yan, and J. Feng. (2018). “Multi-Fiber Networks for Video Recognition. Computer Vision – ECCV 2018”, In: *Proc. of ECCV 2018. Lecture Notes in Computer Science*, Vol. 11205, pp. 364-380, 2018, doi: 10.1007/978-3-030-01246-5_22.

[37] L. Yuan, T. Wang, X. Zhang, F. E. Tay, Z. Jie, W. Liu, and J. Feng, “Central Similarity Quantization for Efficient Image and Video Retrieval”, In: *Proc. of 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Seattle, WA, USA, 2020, pp. 3080-3089, doi: 10.1109/CVPR42600.2020.00315.

[38] D. Tran, H. Wang, L. Torresani, J. Ray, Y. Lecun, and M. Paluri, “A Closer Look at Spatiotemporal Convolutions for Action Recognition”, In: *Proc. of 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6450-6459, 2018, doi: 10.1109/CVPR.2018.00675.

[39] D. Ghadiyaram, M. Feiszli, D. Tran, X. Yan, H. Wang, and D. Mahajan, “Large-Scale Weakly-Supervised Pre-Training for Video Action Recognition”, In: *Proc. of 2019 IEEE/CVF*

Conference on Computer Vision and Pattern Recognition (CVPR), pp. 12038-12047, 2019, doi: 10.1109/CVPR.2019.01232.

[40] I. Loshchilov and F. Hutter, “Decoupled weight decay regularization”, In: *Proc. of 2019 International Conference on Learning Representations (ICLR 2019)*, 2019.

[41] J. Qin, L. Liu, M. Yu, Y. Wang, and L. Shao, “Fast action retrieval from videos via feature disaggregation”, *Computer Vision and Image Understanding*, Vol. 156, pp. 104-116, 2017, doi: 10.1016/j.cviu.2016.09.009.

[42] H. Chen, C. Hu, F. Lee, C. Lin, W. Yao, L. Chen, and Q. Chen, “A Supervised Video Hashing Method Based on a Deep 3D Convolutional Neural Network for Large-Scale Video Retrieval”, *Sensors*, Vol. 21, p. 3094, 2021, doi: 10.3390/s21093094.

[43] A. Hussain, T. Hussain, W. Ullah, and S. W. Baik, “Vision Transformer and Deep Sequence Learning for Human Activity Recognition in Surveillance Videos”, Vol. 2022, pp. 1687-5265, 2022, doi: 10.1155/2022/3454167.