Dynamic Image Networks for Action Recognition

Hakan Bilen[†*] Basura Fernando[‡*] Efstratios Gavves[§] Andrea Vedaldi[†] Stephen Gould[‡]
[†]University of Oxford [‡]The Australian National University [§]QUVA Lab, University of Amsterdam

Abstract

We introduce the concept of dynamic image*, a novel compact representation of videos useful for video analysis especially when convolutional neural networks (CNNs) are used. The dynamic image is based on the rank pooling concept and is obtained through the parameters of a ranking machine that encodes the temporal evolution of the frames of the video. Dynamic images are obtained by directly applying rank pooling on the raw image pixels of a video producing a single RGB image per video. This idea is simple but powerful as it enables the use of existing CNN models directly on video data with fine-tuning. We present an efficient and effective approximate rank pooling operator, speeding it up orders of magnitude compared to rank pooling. Our new approximate rank pooling CNN layer allows us to generalize dynamic images to dynamic feature maps and we demonstrate the power of our new representations on standard benchmarks in action recognition achieving state-of-the-art performance.*

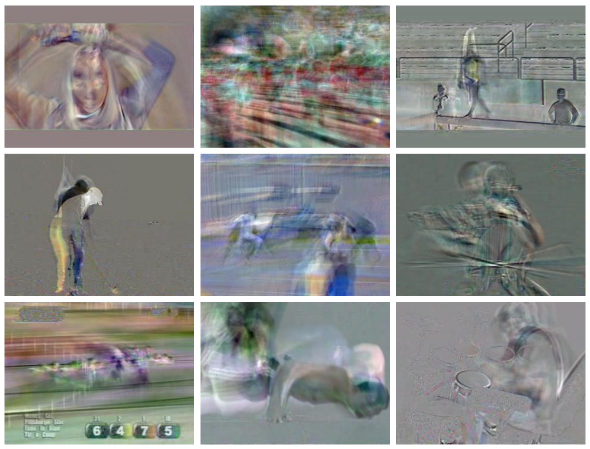

Figure 1: Dynamic images summarizing the actions and motions that happen in images in standard 2d image format. *Can you guess what actions are visualized just from their dynamic image motion signature?* [1]

1. Introduction

Videos comprise a large majority of the visual data in existence, surpassing by a wide margin still images. Therefore understanding the content of videos accurately and on a large scale is of paramount importance. The advent of modern learnable representations such as deep convolutional neural networks (CNNs) has improved dramatically the performance in many image understanding tasks. Since videos are composed of a sequence of still images, some of these improvements have been shown to transfer to videos directly. However, it remains unclear *how videos could be optimally represented*. For example, one can look at a video as a sequence of still images, perhaps enjoying some form of temporal smoothness, or as a subspace of images or image features, or as the output of a neural network encoder. Which one among these and other possibilities results in the best representation of videos is not well understood.

In this paper, we explore a new powerful and yet simple representation of videos in the context of deep learning. As a representative goal we consider the standard problem of recognizing human actions in short video sequences. Recent works such as [5, 6, 7, 9, 23] pointed out that long term dynamics and temporal patterns are a very important cues for the recognition of actions. However, representing complex long term dynamics is difficult, particularly if one seeks compact representations that can be processed efficiently. Several efficient representations of long term dynamics have been obtained by *temporal pooling* of image features in a video. Temporal pooling has been done using temporal templates [1] or using ranking functions for video frames [6] or subvideos [9] or by more traditional pooling operators [23].

In this paper we propose a new long-term pooling operator which is simple, efficient, compact, and very powerful in a neural network context. Since a CNN provides a whole hierarchy of image representations, one for each intermediate layer, the first question is where temporal pooling should take place. For example, one could use a method such as

[*]Equal contribution

[1]From left to right and top to bottom: "blowing hair dry", "band marching", "balancing on beam", "golf swing", "fencing", "playing the cello", "horse racing", "doing push-ups", "drumming".

rank pooling [6] to pool the output of the fully-connected layers of a standard CNN architecture pre-trained on still images and applied to individual frames. A downside of this solution is that the CNN itself is unaware of the lower level dynamics of the video. Alternatively, one can model the dynamics of the response of some intermediate network layer. In this case, the lower layers are still computed from individual frames, but the upper layers can reason about the overall dynamics in the video. An extreme version of this idea is to capture the video dynamics *directly at the level of the image pixels*, considering those as the first layer of the architecture.

Here we build on the latter intuition and build dynamics directly at the level of the input image. To this end, our **first contribution** (section 2) is to introduce the notion of a *dynamic image*, i.e. a RGB still image that summarizes, in a compressed format, the gist of a whole video (sub)sequence (fig. 1). The dynamic image is obtained as a ranking classifier that, similarly to [6, 7], sorts video frames temporally; the difference is that we compute this classifier directly at the level of the image pixels instead of using an intermediate feature representation.

There are three keys advantages to this idea. First, the new RGB image can be processed by a CNN architecture which is structurally nearly identical to architectures used for still images, while still capturing the long-term dynamics in a video that relate to the long term dynamics therein. It is then possible to use a standard CNN architecture to learn suitable "dynamic" features from the videos. The second advantage of this method is its remarkable efficiency: the extraction of the dynamic image is extremely simple and efficient and it allows to reduce video classification to classification of a single image by a standard CNN architecture. The third advantage is the compression factor, as a whole video is summarized by an amount of data equivalent to a single frame.

Our **second contribution** is to provide a fast approximation of the ranker in the construction of the dynamic image. We replace learning the ranker by simple linear operations over the images, which is extremely efficient. We also show that, in this manner, it is possible to apply the concept of dynamic image to the intermediate layers of a CNN representation by constructing an efficient *rank pooling layer*. This layer can be incorporated into end-to-end training of a CNN for video data.

Our **third contribution** is to use these ideas to propose a novel static/dynamic neural network architecture (section 3) which can perform end-to-end training from videos combining both static appearance information from still frames, as well as short and long term dynamics from the whole video. We show that these technique result in efficient and accurate classification of actions in videos, outperforming the state-of-the-art in standard benchmarks in the area (section 4). Our findings are summarized in section 5.

1.1. Related Work

Existing video representations can be roughly broken into two categories. The first category, which comprises the majority of the literature on video processing, action and event classification, be it with shallow [31, 6, 20] or deep architectures [21, 24], has viewed videos either as a stream of still images [21] or as a short and smooth transition between similar frames [24]. Although obviously suboptimal, considering the video as bag of static frames performs reasonably well [21], as the surroundings of an action strongly correlate with the action itself (*e.g.*, "playing basketball" takes place usually in a basketball court). The second category extends CNNs to a third, temporal dimension [12] replacing 2D filters with 3D ones. So far, this approach has produced little benefits, probably due to the lack of annotated video data. Increasing the amount of annotated videos would probably help as shown by recent 3D convolution methods [29], although what seems especially important is spatial consistency between frames. More specifically, a pixel to pixel registration [24] based on the video's optical flow brings considerable improvements. Similarly, [8] uses action tubes to to fit a double stream appearance and motion based neural network that captures the movement of the actor.

We can also distinguish two architectural choices in the construction of video CNNs. The first choice is to provide as input to the CNN a sub-video of fixed length, packing a short sequence of video frames into an array of images. The advantage of this technique is that they allow using simple modifications of standard CNN architectures (*e.g.* [15]) by swapping 3D filters for the 2D ones. Examples of such techniques include [12] and [24].

Although the aforementioned methods successfully capture the local changes within a small time window, they cannot capture longer-term motion patterns associated with certain actions. An alternative solution is to consider a second family of architectures based on recurrent neural networks (RNNs) [4, 28]. RNNs typically consider memory cells [11], which are sensitive to both short as well as longer term patterns. RNNs parse the video frames sequentially and encode the frame-level information in their memory. In [4] LSTMs are used together with convolutional neural network activations to either output an action label or a video description. In [28] an autoencoder-like LSTM architecture is proposed such that either the current frame or the next frame is accurately reconstructed. Finally, the authors of [33] propose an LSTM with a temporal attention model for densely labelling video frames.

Many of the ideas in video CNNs originated in earlier architectures that used hand-crafted features. For example, the authors of [18, 30, 31] have shown that local motion patterns in short frame sequences can capture very well the short temporal structures in actions. The rank pooling idea, on which our dynamic images are based, was proposed in [6, 7] using hand-crafted representation of the frames, while in [5] authors increase the capacity of rank pooling using a hierarchical approach.

Our static/dynamic CNN uses a multi-stream architecture. Multiple streams have been used in a variety of different contexts. Examples include Siamese architectures for learning metrics for face identification [2] of for unsupervised training of CNNs [3]. Simonyan *et al.* [24] use two streams to encode respectively static frames and optical flow frames in action recognition. The authors of [19] propose a dual loss neural network was proposed, where coarse and fine outputs are jointly optimized. A difference of our model compared to these is that we branch off two streams at arbitrary location in the network, either at the input, at the level of the convolutional layers, or at the level of the fully-connected layers.

2. Dynamic Images

In this section we introduce the concept of *dynamic image*, which is a standard RGB image that summarizes the appearance and dynamics of a whole video sequence (section 2.1). Then, we show how dynamic images can be used to train dynamic-aware CNNs for action recognition in videos (section 2.2). Finally, we propose a fast approximation to accelerate the computation of dynamic images (section 2.3).

2.1. Constructing dynamic images

While CNNs can learn automatically powerful data representations, they can only operate within the confines of a specific hand-crafted architecture. In designing a CNN for video data, in particular, it is necessary to think of how the video information should be presented to the CNN. As discussed in section 1.1, standard solutions include encoding sub-videos of a fixed duration as multi-dimensional arrays or using recurrent architectures. Here we propose an alternative and more efficient approach in which the video content is summarized by a single still image which can then be processed by a standard CNN architecture such as AlexNet [15].

Summarizing the video content in a single still image may seem difficult. In particular, it is not clear how image pixels, which already contain appearance information in the video frames, could be overloaded to reflect dynamic information as well, and in particular the long-term dynamics that are important in action recognition.

We show here that the construction of Fernando *et al.* [6] can be used to obtain exactly such an image. The idea of their paper is to represent a video as a *ranking function* for its frames $I_1, \ldots, I_T$. In more detail, let $\psi(I_t) \in \mathbb{R}^d$ be a representation or feature vector extracted from each individual frame I_t in the video. Let $V_t = \frac{1}{t}\sum_{\tau=1}^{t} \psi(I_\tau)$ be time average of these features up to time t. The ranking function associates to each time t a score $S(t|\mathbf{d}) = \langle \mathbf{d}, V_t \rangle$, where $\mathbf{d} \in \mathbb{R}^d$ is a vector of parameters. The function parameters $\mathbf{d}$ are learned so that the scores reflect the rank of the frames in the video. Therefore, later times are associated with larger scores, *i.e.* $q > t \implies S(q|\mathbf{d}) > S(t|\mathbf{d})$. Learning $\mathbf{d}$ is posed as a convex optimization problem using the RankSVM [26] formulation:

$$\begin{aligned} \mathbf{d}^* &= \rho(I_1, \ldots, I_T; \psi) = \operatorname*{argmin}_{\mathbf{d}} E(\mathbf{d}), \\ E(\mathbf{d}) &= \frac{\lambda}{2}\|\mathbf{d}\|^2 + \\ &\frac{2}{T(T-1)} \times \sum_{q>t} \max\{0, 1 - S(q|\mathbf{d}) + S(t|\mathbf{d})\}. \end{aligned} \tag{1}$$

The first term in this objective function is the usual quadratic regularizer used in SVMs. The second term is a hinge-loss soft-counting how many pairs $q > t$ are *incorrectly* ranked by the scoring function. Note in particular that a pair is considered correctly ranked only if scores are separated by at least a unit margin, i.e. $S(q|\mathbf{d}) > S(t|\mathbf{d}) + 1$.

Optimizing eq. (1) defines a function $\rho(I_1, \ldots, I_T; \psi)$ that maps a sequence of T video frames to a single vector $\mathbf{d}^*$. Since this vector contains enough information to rank all the frames in the video, it aggregates information from all of them and can be used as a video descriptor. In the rest of the paper we refer to the process of constructing $\mathbf{d}^*$ from a sequence of video frames as *rank pooling*.

In [6] the map $\psi(\cdot)$ used in this construction is set to be the Fisher Vector coding of a number of local features (HOG, HOF, MBH, TRJ) extracted from individual video frames. Here, we propose to apply rank pooling *directly to the RGB image pixels* instead. While this idea is simple, in the next several sections we will show that it has remarkable advantages.

The $\psi(I_t)$ is now an operator that stacks the RGB components of each pixel in image I_t on a large vector. Alternatively, $\psi(I_t)$ may incorporate a simple component-wise non-linearity, such as the square root function $\sqrt{\cdot}$ (which corresponds to using the Hellinger's kernel in the SVM). In all cases, the descriptor $\mathbf{d}^*$ is a real vector that *has the same number of elements as a single video frame*. Therefore, $\mathbf{d}^*$ can be interpreted as standard RGB image. Furthermore, since this image is obtained by rank pooling the video frames, it summarizes information from the whole video sequence.

A few examples of dynamic images are shown in fig. 1. Several observations can be made. First, it is interesting to note that the dynamic images tend to focus mainly on the acting objects, such as humans or other animals such as horses in the "horse racing" action, or objects such as

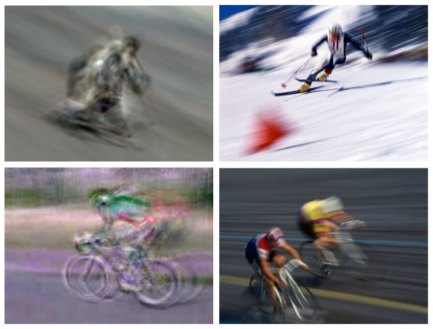

Figure 2: Left column: dynamic images. Right column: motion blur. Although fundamentally different both methodologically, as well as in terms of applications, they both seem to capture time in a similar manner.

drums in the "drumming" action. On the contrary, background pixels and background motion patterns tend to be averaged away. Hence, the pixels in the dynamic image appear to focus on the identity and motion of the salient actors in videos, indicating that they may contain the information necessary to perform action recognition.

Second, we observe that dynamic images behave differently for actions of different speeds. For slow actions, like "blowing hair dry" in the first row of fig. 1, the motion seems to be dragged over many frames. For faster actions, such as "golf swing" in the second row of fig. 1, the dynamic image reflects key steps in the action such as preparing to swing and stopping after swinging. For longer term actions such as "horse riding" in the third row of fig. 1, the dynamic image reflects different parts of the video; for instance, the rails that appear as a secondary motion contributor are superimposed on top of the horses and the jockeys who are the main actors. Such observations were also made in [7].

Last, it is interesting to note that dynamic images are reminiscent of some other imaging effects that convey motion and time, such as *motion blur* or *panning*, an analogy is illustrated in fig. 2. While motion blur captures the time and motion by integrating over subsequent pixel intensities defined by the camera shutter speed, dynamic images capture the time by integrating and reordering the pixel intensities over time within a video.

2.2. Using dynamic images

Given that the dynamic images are in the format of standard RGB images, they can be used to apply any method for still image analysis, and in particular any state-of-the-art CNN, to the analysis of video. In particular, we experiment with two usage scenarios.

Single Dynamic Image (SDI). In the first scenario, a dynamic image summarizes an entire video sequence. By training a CNN on top of such dynamic images, we implicitly capture the temporal patterns contained in the video. However, since the CNN is still applied to images, we can start from a CNN *pre-trained for still image recognition*, such as AlexNet pre-trained on the ImageNet ILSVRC data, and fine-tune it on a dataset of dynamic images. Fine-tuning allows the CNN to learn features that capture the video dynamics without the need to train the architecture from scratch. This is an important benefit of our method because training large CNNs require millions of data samples which may be difficult to obtain for videos.

Multiple Dynamic Images (MDI). While fine-tuning does not require as much annotated data as training a CNN from scratch, the domain gap between natural and dynamic images is sufficiently large that an adequately large fine-tuning dataset of dynamic images may be appropriate. However, as noted above, in most cases there are only a few videos available for training.

In order to address this potential limitation, in the second scenario we propose to generate multiple dynamic images from each video by breaking it into segments. In particular, for each video we extract multiple partially-overlapping segments of duration τ and with stride s. In this manner, we create multiple video segments per video, essentially multiplying the dataset size by a factor of approximately T/s, where T is the average number of frames per video. This can also be seen as a data augmentation step, where instead of mirroring, cropping, or shearing images we simply take a subset of the video frames. From each of the new video segments, we can then compute a dynamic image to train the CNN, using as ground truth class information of each subsequence the class of the original video.

2.3. Fast dynamic image computation

Computing a dynamic image entails solving the optimization problem of eq. (1). While this is not particularly slow with modern solvers, in this section we propose an approximation to rank pooling which is much faster and works as well in practice. Later, this technique, which we call *approximate rank pooling*, will be critical in incorporating rank pooling in intermediate layers of a deep CNN and to allow back-prop training through it.

The derivation of approximate rank pooling is based on the idea of considering the first step in a gradient-based optimization of eq. (1). Starting with $\mathbf{d} = \vec{0}$, the first approximated solution obtained by gradient descent is $\mathbf{d}^* = \vec{0} - \eta \nabla E(\mathbf{d})|_{\mathbf{d}=\vec{0}} \propto -\nabla E(\mathbf{d})|_{\mathbf{d}=\vec{0}}$ for any $\eta > 0$, where

$$\begin{aligned}\nabla E(\vec{0}) &\propto \sum_{q>t} \nabla \max\{0, 1 - S(q|\mathbf{d}) + S(t|\mathbf{d})\}|_{\mathbf{d}=\vec{0}} \\ &= \sum_{q>t} \nabla \langle \mathbf{d}, V_t - V_q \rangle = \sum_{q>t} V_t - V_q.\end{aligned}$$

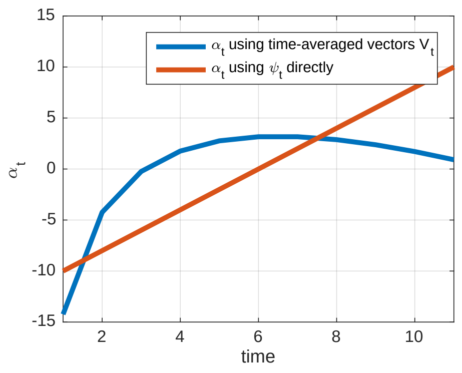

Figure 3: The graph compares the approximated rank pooling weighting functions α_t (for $T = 11$ samples) of eq. (2) using time-averaged feature frames V_t to the variant eq. (4) that ranks directly the feature frames ψ_t as is.

We can further expand $\mathbf{d}^*$ as follows

$$\mathbf{d}^* \propto \sum_{q>t} V_q - V_t = \sum_{q>t} \left[\frac{1}{q} \sum_{i=1}^{q} \psi_i - \frac{1}{t} \sum_{j=1}^{t} \psi_j \right] = \sum_{t=1}^{T} \alpha_t \psi_t$$

where the coefficients α_t are given by

$$\alpha_t = 2(T - t + 1) - (T + 1)(H_T - H_{t-1}), \qquad (2)$$

where $H_t = \sum_{i=1}^{t} 1/t$ is the t-th Harmonic number and $H_0 = 0$. Hence the rank pooling operator reduces to

$$\hat{\rho}(I_1, \ldots, I_T; \psi) = \sum_{t=1}^{T} \alpha_t \psi(I_t). \qquad (3)$$

which is a weighted combination of the data points. In particular, the dynamic image computation reduces to accumulating the video frames after pre-multiplying them by α_t. The function α_t, however, is non-trivial, as illustrated in fig. 3.

An alternative construction of the rank pooler does not compute the intermediate average features $V_t = (1/t) \sum_{q=1}^{T} \psi(I_q)$, but uses directly individual video features $\psi(I_t)$ in the definition of the ranking scores (1). In this case, the derivation above results in a weighting function of the type

$$\alpha_t = 2t - T - 1 \qquad (4)$$

which is linear in t. The two scoring functions eq. (2) and eq. (4) are compared in fig. 3 and in the experiments.

3. Dynamic Maps Networks

In the previous section we have introduced the concept of dynamic image as a method to pool the information contained in a number of video frames in a single RGB image.

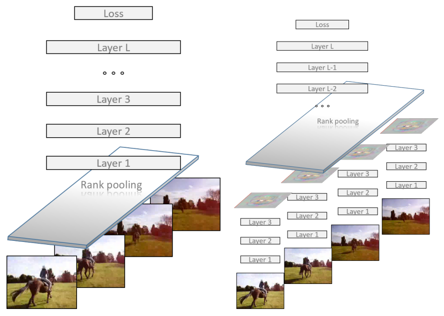

Figure 4: Dynamic image and dynamic map networks on the left and the right pictures respectively, after applying a rank pooling operation on top of the previous layer activations.

Here, we notice that every layer of a CNN produces as output a *feature map* which, having a spatial structure similar to an image, can be used in place of video frames in this construction. We call the result of applying rank pooling to such features a *dynamic feature map*, or *dynamic map* in short. In the rest of the section we explain how to incorporate this construction as a rank-pooling layer in a CNN (section 3.1) and how to accelerate it significantly and perform back-propagation by using approximate rank pooling (section 3.2).

3.1. Dynamic maps

The structure of a dynamic map network is illustrated in fig. 4. In the case seen so far (left in fig. 4), rank pooling is applied at the level of the input RGB video frames, which we could think of as layer zero in the architecture. We call the latter a **dynamic image network**. By contrast, a **dynamic map network** moves rank pooling higher in the hierarchy, by applying one or more layers of feature computations to the individual feature frames and applying the same construction to the resulting feature maps.

In particular, let $\mathbf{a}_1^{(l-1)}, \ldots, \mathbf{a}_T^{(l-1)}$ denote the feature maps computed at the $l-1$ layers of the architecture, one for each of the T video frames. Then, we use the rank pooling equation (1) to aggregate these maps into a single dynamic map,

$$\mathbf{a}^{(l)} = \rho(\mathbf{a}_1^{(l-1)}, \ldots, \mathbf{a}_T^{(l-1)}). \qquad (5)$$

Note that, compared to eq. (1), we dropped the term ψ; since networks are already learning feature maps, we set this term to the identity function. The dynamic image network is obtained by setting $l = 1$ in this construction.

Rank pooling layer (RankPool) & backpropagation. In order to train a CNN with rank pooling as an intermediate layer, it is necessary to compute the derivatives of eq. (5) for the backpropagation step. We can rewrite eq. (5) as a linear combination of the input data $V_1, \dots, V_T$, namely

$$\mathbf{a}^{(l)} = \sum_{t=1}^{T} \beta_t(V_1, \dots, V_T) V_t \tag{6}$$

In turn, V_t is the temporal average of the input features and is therefore a linear function $V_t(\mathbf{a}_1^{(l-1)}, \dots, \mathbf{a}_t^{(l-1)})$. Substituting, we can rewrite $\mathbf{a}^{(l)}$ as

$$\mathbf{a}^{(l)} = \sum_{t=1}^{T} \alpha_t(\mathbf{a}_1^{(l-1)}, \dots, \mathbf{a}_T^{(l-1)}) \mathbf{a}_t^{(l-1)}. \tag{7}$$

Unfortunately, we observe that due to the non-linear nature of the optimization problem of equation (1), the coefficients β_t, α_t depend on the data $\mathbf{a}_t^{(l-1)}$ themselves. Computing the gradient of $\mathbf{a}^{(l)}$ with respect to the per frame data points $\mathbf{a}_t^{(l-1)}$ is a challenging derivation. Hence, using dynamic maps and rank pooling directly as a layer in a CNN is not straightforward.

We note that the rank pooling layer (RankPool) constitutes a new type of portable convolutional network layer, just like a max-pooling or a ReLU layer. It can be used whenever dynamic information must be pooled across time.

3.2. Approximate dynamic maps.

Constructing the precise dynamic maps, or images, is in theory optimal, but not necessarily practical. On one hand computing the precise dynamic maps via an optimization is computationally inefficient. This is especially important in the context of CNNs, where efficient computations are extremely important for training on large datasets, and the optimization of eq. (5) would be slow compared to other components of the network. On the other hand, computing the gradients would be non trivial.

To this end we replace once again rank pooling with approximate rank pooling. With the approximate rank pooling we significantly accelerate the computations, even by a factor of 45 as we show later in the experiments. Secondly, and more importantly, the approximate rank pooling is also a linear combination of frames, where the per frame coefficients are given by eq. (2). These coefficients are independent of the frame features V_t and $\psi(I_t)$. Hence, the derivative of the approximate rank pooling is much simpler and can be easily computed as the vectorized coefficients of eq. (2), namely

$$\frac{\partial \operatorname{vec} \mathbf{a}^{(l)}}{\partial (\operatorname{vec} \mathbf{a}_t^{(l-1)})^\top} = \alpha_t I \tag{8}$$

where I is the identity matrix. Interestingly, we would obtain the same expression for the derivative if α_t in eq. (7) would be constant and did not depend on the video frames.

We conclude that using approximate rank pooling in the context of CNNs is not only practical, but also necessary for the optimization through backpropagation.

4. Experiments

4.1. Datasets

We explore the proposed models on two state-of-the-art datasets used for evaluating neural network based models for action recognition, namely *UCF101* [27] and HMDB51 [16].

UCF101. The UCF101 dataset [27] comprises of 101 human action categories, like "Apply Eye Makeup" and "Rock Climbing" and spans over $13,320$ videos. The videos are realistic and relatively clean. They contain little background clutter and contain a single action. Also the videos are trimmed, thus almost all frames relate to the action in the video. The standard evaluation is average accuracy over three parts provided by the authors.

HMDB51. The HMDB51 dataset [16] comprises of 51 human action categories , such as "backhand flip" and "swing baseball bat" and spans over $6,766$ videos. The videos are realistic, downloaded from Youtube contain a single action. The dataset is split in three parts and accuracy is averaged over all three parts, similar to *UCF101*.

4.2. Implementation details

To maintain the same function domain and range we select for non-linear operations $\psi(\cdot)$ the square rooting kernel maps $\sqrt{\cdot}$ and time varying mean vectors [6]. We generate dynamic images for each color channel separately and then merge them so that they can be directly used directly as input to a CNN. As the initial dynamic images are not in the natural range of $[0, 255]$ for RGB data, we apply minmax normalization. We use BVLC reference CaffeNet model [13] trained on ImageNet images as a starting point to train our dynamic image networks. We fine-tune all the layers with the learning rate to be 10^-3 and gradually decrease it per epoch. We use a maximum of 20 epoch during training.
Sharing code, data, models. We share our code, models and data [2]. Furthermore, we have computed the dynamic images of the Sports1M dataset [14], and share the Alexnet and VGGnet dynamic image networks trained on the Sports1M.

4.3. Mean, max and dynamic images

First, we compare "single image per video", namely the proposed Single Dynamic Image (SDI) with the per video sequence mean and max image. For all methods we first

[2]https://github.com/hbilen/dynamic-image-nets

Method	SPLIT1	SPLIT2	SPLIT3	AVERAGE
Mean Image	52.6	53.4	51.7	52.6
Max Image	48.0	46.0	42.3	45.4
SDI	57.2	58.7	57.7	**57.9**

Table 1: Comparing several video representative image models using UCF101

Method	Speed	Accuracy
Appr. Rank Pooling	5920 fps	96.5 ± 0.9
Rank Pooling	131 fps	99.5 ± 0.1

Table 2: Approximate rank pooling vs rank pooling.

compute the single images per video offline, and for SDI specifically we use SVR [26]. Then we train and test on action recognition using CaffeNet network. Results are reported in Table 1.

From all representations we observe that SDI achieves the highest accuracy. We conclude that SDI model is a better single image model than the mean and max image models.

4.4. Approximate Rank Pooling vs Rank Pooling

Next, we compare the approximate rank pooling and rank pooling in terms of speed (frames per second) and pairwise ranking accuracy, which is the common measure for evaluating learning-to-rank methods. We train on a subset of 10 videos that contain different actions and evaluate on a new set of 10 videos with the same type of actions respectively. We report results with the mean and the standard deviations in Table 2.

We observe that approximate rank pooling is $45\times$ faster than rank pooling, while obtaining similar ranking performance. Further, in Figure 5 we plot the score distributions for rank pooling and approximate rank pooling. We observe that their score profiles are also similar. We conclude that approximate rank pooling is a good approximation to rank pooling, while being two magnitudes faster as it involves no optimization.

4.5. Evaluating the effect of end-to-end training

Next, we evaluate in Table 3 rank pooling dynamic images with and without end-to-end training. We also evaluate rank pooling with dynamic maps. The first method generates multiple dynamic images on the RGB pixels as earlier. These dynamic images can be computed offline, then we train a network from end to end. The second method passes these dynamic images through the network, computes the `fc6` activations using a pre-trained Alexnet and aggregates them with max pooling, then trains SVM classifiers per action class. The third method considers a `RankPool` layer after the `conv1` to generate multiple dynamic maps (MDM) based on approximate rank pooling. To generate the multiple dynamic images or maps we use a window size of 25 and a stride of 20 which allows for 80% overlap.

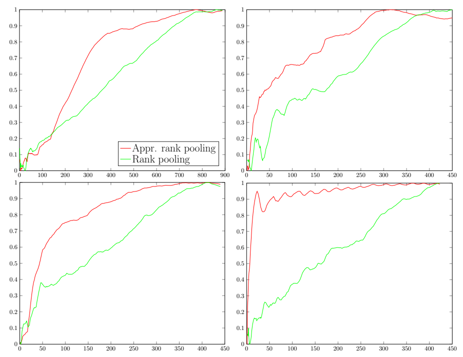

Figure 5: Comparison between score profile of ranking functions for approximate rank pooling and rank pooling. Generally the approximate rank pooling follows the trend of rank pooling.

Method	RankPool Layer	Training	HMDB51	UCF101
MDI	After images	End-to-end	35.8	70.9
MDI	`fc6+max pool`	SVM	32.3	68.6
MDM	After `conv1`	End-to-end	–	67.1

Table 3: Evaluating the effect of end-to-end training for multiple dynamic images and multiple dynamic maps after the convolutional layer 1.

We observe that compared to a unified, end-to-end training is beneficial, bringing 2-3% accuracy improvements depending on the dataset. Furthermore, approximate dynamic maps computed after the convolutional layer 1 perform slighly below the dynamic images (dynamic maps computed after layer 0). We conclude that multiple dynamic images are better to be pooled on top of the static RGB frames. Furthermore, multiple dynamic images perform better when employed in end-to-end training.

4.6. Combining dynamic images with static images

Next, we evaluate how complementary dynamic image networks and static RGB frame networks are. For both networks we apply max pooling on the per video activations at pool5 layer. Results are reported in Table 4.

As expected, we observe that static appearance information appears to be equally important to the dynamic appearance in the context of convolutional neural networks. A combination of the two, however, brings a noticeable 6% increase in accuracy. We conclude that the two representations are complementary to each other.

Method	HMDB51	UCF101
Static RGB	36.7	70.1
MDI-end-to-end	35.8	70.9
MDI-end-to-end + static-rgb	42.8	76.9

Table 4: Evaluating complementarity of dynamic images with static images.

Classes	Diff.	Classes	Diff.
SoccerJuggling	+38.5	CricketShot	-47.9
CleanAndJerk	+36.4	Drumming	-25.6
PullUps	+32.1	PlayingPiano	-22.0
PushUps	+26.7	PlayingFlute	-21.4
PizzaTossing	+25.0	Fencing	-18.2

Table 5: Class by class comparison between RGB and MDI networks, where the difference in scores using MDI and RGB are reported. A positive difference is better for MDI, a negative difference better for RGB

Method	HMDB51	UCF101
Trajectories [31]	60.0	86.0
MDI-end-to-end + static-rgb+trj	65.2	89.1

Table 6: Combining with trajectory features brings a noticeable increase in accuracy.

4.7. Further analysis

We, furthermore, perform a per analysis between static rgb networks and MDI based networks. We list in Table 5 the top 5 classes based on the relative performances for each method. MDI performs better for "PullUps" and "PushUps", where motion is dominant and discriminating between motion patterns is important. RGB static models seems to work better on classes such as "CricketShot" and "Drumming", where context is already quite revealing. We conclude that dynamic images are useful for actions where there exist characteristic motion patterns and dynamics.

Furthermore, we investigate whether dynamic images are complementary to state-of-the-art features, like improved trajectories [31], relying on late fusion. Results are reported in Table 6. We obtain a significant improvement of 5.2% over trajectory features alone on HMDB51 dataset and 3.1% on UCF101 dataset.

Due to the lack of space we refer to the supplementary material for a more in depth analysis of dynamic images and dynamic maps and their learning behavior.

4.8. State-of-the-art comparisons

Last, we compare with the state-of-the-art techniques in UCF101 and HMDB51 in Table 7, where we make a distinction between deep and shallow architectures. Note that similar to us, almost all methods, be it shallow or deep, obtain their accuracies after combining their methods with improved trajectories [31] for optimal results.

	Method	HMDB51	UCF101
deep	This paper	65.2	89.1
	Zha et al. *[34]*	–	89.6
	Simonyan et al. *[24]*	59.4	88.0
	Yue-Hei-Ng et al. *[21]*	–	88.6
shallow	*Wu* et al. *[32]*	56.4	84.2
	Fernando et al. *[6]*	63.7	–
	Hoai et al. *[10]*	60.8	–
	Lan et al. *[17]*	65.4	89.1
	Peng et al. *[22]*	66.8	–

Table 7: Comparison with the state-of-the-art. Despite being a relatively simple representation, the proposed method is able to obtain results on par with the state state-of-the-art.

Considering deep learning methods, our method performs on par and is only outperformed from [34]. [34] makes use of the very deep VGGnet [25], which is a more competitive network than that the Alexnet architecture we rely on. Hence a direct comparison is not possible. Compared to the shallow methods the proposed method is also competitive. We anticipate that combining the proposed dynamic images with sophisticated encodings [17, 22] will benefit the accuracies further.

We conclude that while being in the context of CNNs a simple and efficient video representation, dynamic images allow for state-of-the-art accuracy in action recognition.

5. Conclusion

We present *dynamic images*, a powerful and new, yet simple video representation in the context of deep learning that summarizes videos into single images. As such, dynamic images are directly compative to existing CNN architectures allowing for end-to-end action recognition learning. Extending dynamic images to the hierarchical CNN feature maps, we introduce a novel temporal pooling layer, `Approximate-RankPool` directly. Experiments on state-of-the-art action recognition datasets demonstrate the descriptive power of dynamic images, despite their conceptual simplicity. A visual inspection outlines the richness of dynamic images in describing complex motion patterns as simple 2d images.

Acknowledgments: This work acknowledges the support of the EPSRC grant EP/L024683/1 and the Australian Research Council Centre of Excellence for Robotic Vision (project number CE140100016).

References

[1] A. F. Bobick and J. W. Davis. The recognition of human movement using temporal templates. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 23(3):257–267, 2001.

[2] S. Chopra, R. Hadsell, and Y. LeCun. Learning a similarity metric discriminatively, with application to face verification. In *CVPR*, 2005.

[3] C. Doersch, A. Gupta, and A. A. Efros. Unsupervised visual representation learning by context prediction. In *CVPR*, 2015.

[4] J. Donahue, L. A. Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell. Long-term recurrent convolutional networks for visual recognition and description. In *arXiv preprint arXiv:1411.4389*, 2014.

[5] B. Fernando, P. Anderson, M. Hutter, and S. Gould. Discriminative hierarchical rank pooling for activity recognition. In *CVPR*, 2016.

[6] B. Fernando, E. Gavves, J. Oramas, A. Ghodrati, and T. Tuytelaars. Modeling video evolution for action recognition. In *CVPR*, 2015.

[7] B. Fernando, E. Gavves, J. Oramas, A. Ghodrati, and T. Tuytelaars. Rank pooling for action recognition. *TPAMI*, 2016.

[8] G. Gkioxari and J. Malik. Finding action tubes. In *CVPR*, June 2015.

[9] M. Hoai and A. Zisserman. Improving human action recognition using score distribution and ranking. In *Proceedings of Asian Conference on Computer Vision*, 2014.

[10] M. Hoai and A. Zisserman. Improving human action recognition using score distribution and ranking. In *ACCV*, 2014.

[11] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[12] S. Ji, W. Xu, M. Yang, and K. Yu. 3d convolutional neural networks for human action recognition. *TPAMI*, 2013.

[13] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[14] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei. Large-scale video classification with convolutional neural networks. In *CVPR*, 2014.

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[16] H. Kuehne, H. Jhuang, E. Garrote, T. Poggio, and T. Serre. Hmdb: a large video database for human motion recognition. In *ICCV*, 2011.

[17] Z.-Z. Lan, M. Lin, X. Li, A. G. Hauptmann, and B. Raj. Beyond gaussian pyramid: Multi-skip feature stacking for action recognition. In *CVPR*, 2015.

[18] I. Laptev. On space-time interest points. *IJCV*, 64:107–123, 2005.

[19] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. *CVPR*, 2015.

[20] M. Mazloom, E. Gavves, and C. G. M. Snoek. Conceptlets: Selective semantics for classifying video events. *IEEE TMM*, December 2014.

[21] J. Y. Ng, M. J. Hausknecht, S. Vijayanarasimhan, O. Vinyals, R. Monga, and G. Toderici. Beyond short snippets: Deep networks for video classification. In *CVPR*, 2015.

[22] X. Peng, C. Zou, Y. Qiao, and Q. Peng. Action recognition with stacked fisher vectors. In *ECCV*, 2014.

[23] M. S. Ryoo, B. Rothrock, and L. Matthies. Pooled motion features for first-person videos. In *CVPR*, 2015.

[24] K. Simonyan and A. Zisserman. Two-stream convolutional networks for action recognition in videos. *CoRR*, abs/1406.2199:1–8, 2014.

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

[26] A. J. Smola and B. Schölkopf. A tutorial on support vector regression. *Statistics and computing*, 14:199–222, 2004.

[27] K. Soomro, A. R. Zamir, and M. Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild. *CoRR*, 2012.

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

[29] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri. Learning spatiotemporal features with 3d convolutional networks. *arXiv preprint arXiv:1412.0767*, 2014.

[30] H. Wang, A. Kläser, C. Schmid, and C.-L. Liu. Dense trajectories and motion boundary descriptors for action recognition. *IJCV*, 103:60–79, 2013.

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

[32] J. Wu, Y. Zhang, and W. Lin. Towards good practices for action video encoding. In *CVPR*, 2014.

[33] S. Yeung, O. Russakovsky, N. Jin, M. Andriluka, G. Mori, and L. Fei-Fei. Every moment counts: Dense detailed labeling of actions in complex videos. *ArXiv e-prints*, 2015.

[34] S. Zha, F. Luisier, W. Andrews, N. Srivastava, and R. Salakhutdinov. Exploiting Image-trained CNN Architectures for Unconstrained Video Classification. In *BMVC*, 2015.