Vis Comput
DOI 10.1007/s00371-016-1347-4

ORIGINAL ARTICLE

Space–time image layout

Shahar Ben-Ezra[1] · **Daniel Cohen-Or**[2]

Abstract Cameras are now ubiquitous in our lives. A given activity is often captured by multiple people from different viewpoints resulting in a sizable collection of photograph footage. We present a method that effectively organizes this spatiotemporal content. Given an unorganized collection of photographs taken by a number of photographers, capturing some dynamic event at a number of time steps, we would like to organize the collection into a *space–time table*. The organization is an embedding of the photographs into clusters that preserve the viewpoint and time order. Our method relies on a self-organizing map (SOM), which is a neural network that embeds the training data (the set of images) into a discrete domain. We introduce BiSOM, which is a variation of SOM that considers two features (space and time) rather than a single one, to layout the given photograph collection into a table. We demonstrate our method on several challenging datasets, using different space and time descriptors.

Keywords Image organization · Spatial ordering · Temporal ordering · Self-organizing maps

Electronic supplementary material The online version of this article (doi:10.1007/s00371-016-1347-4) contains supplementary material, which is available to authorized users.

✉ Shahar Ben-Ezra
shaharb@mail.tau.ac.il; sharkybe@gmail.com

Daniel Cohen-Or
dcor@tau.ac.il

[1] Electrical Engineering School, Tel Aviv University, Tel Aviv, Israel

[2] Computer Science School, Tel Aviv University, Tel Aviv, Israel

1 Introduction

We consider the following space–time layout problem: Given an unorganized collection of N photographs taken by N_H photographers, capturing some dynamic event at N_V time steps roughly simultaneously, we would like to organize the collection into a *space–time* table. An example of such a space–time-organized table is illustrated in Fig. 1a. Each row represents the set of photographs taken from about the same position (likely by an individual photographer), and each column represents photographs taken at about the same time. We assume that the photographers took shots of a rather short event that occurred in a rather limited space–time domain, so the commonality among the photographs is high. Such a collection is likely to be found in social media networks, where it lacks any image metadata (GPS location, EXIF time tag, etc.). Moreover, as shown in [9], even if the image *does* contain a time tag, it *cannot* be used for our task due to insufficient accuracy.

We were inspired by two recent works. Dekel et al. [9] presented a photograph sequencing technique that, given a set of images that were captured roughly from the same viewpoint, they determine the temporal order of the images. Averbuch et al. [2] present a spatial ordering technique, where the images are captured roughly at the same moment, but from different unknown viewpoints, and the challenge is to find the spatial order of the images. Our work aims to do both spatial and temporal ordering, simultaneously.

Organizing such a collection of photographs boils down into two apparently independent clustering problems: One clusters the photographs by their spatial features, which is the view direction, and the other by the temporal features, which relies on the dynamic object in the scene. Without enforcing any constraints on the clustering, a typical result of such two independent clusterings is imperfect.

Published online: 20 January 2017

Fig. 1 **a** *Space–time* table generated by our algorithm using the WALL dataset. **b** The embedding of the input points into two 3D domains. The *coloring of the points* represents the ground-truth clustering of the images into space and time clusters

The space–time layout problem can also be regarded as an embedding problem. Here, the photograph collection has to be embedded into a given table. In this sense, our work is most similar to the recent works [12,30] that present a method for arranging images into a grid according to their similarities. However, unlike these methods, we use two separate features to organize the images into rows and columns. The premise of our work is that better results can be achieved by using both features together than by treating each separately.

The technique that we present builds on a self-organizing map (SOM) [16], which is a neural network that embeds the high-dimensional training data into a two-dimensional discrete domain. We introduce BiSOM, which is variation of SOM that considers two features (space and time) rather than one, to embed the given photograph collection in the table. As we shall show, BiSOM does not consider the two features independently. The key advantage of using a SOM is that, unlike other clustering schemes, the intrinsic order among the clusters is accounted for in the organization of clusters across the table. We develop an objective function that evaluates the quality of a given space–time table and define an optimization problem to distribute the input images into a coherent table.

2 Related work

Space–time Analysis Temporal analysis of visual data has been studied extensively in the context of video synchronization and alignment [5,10]. These methods try to align the sequences captured by uncalibrated cameras of the same dynamic scene from different viewpoints. Unlike these methods, in our problem, the temporal order of the photographs is unknown. Dekel et al. [8,9] introduced a novel method for temporal ordering a set of still images taken asynchronously by a set of uncalibrated cameras that are colocated in space and time.

Viewpoint analysis has received considerably more attention. It has primarily been studied in the context of shape-from-motion (SFM) [31]. The viewing parameters are recovered together with the reconstruction of a scene by considering the correspondence among many images of the same scene. More recent methods [4,13,20,29], known as Photo Tourism, reconstruct a static scene from an unorganized set of tourist photographs, including the spatial position of the photographers.

Averbuch et al. [2] present a technique to spatially order a set of images capturing a static scene, taken simultaneously by a group of people situated around the scene. This technique can deal with a single column in our space–time table, independently of the other, without analyzing the temporal domain.

The above space–time organization schemes are imperfect, so our technique focus is in employing global constraints to improve their separation performance.

Self-Organizing Maps Self-organizing maps (SOMs) were first introduced by Kohonen [16] as a method for visualizing low-dimensional views of high-dimensional data. Kiang [15] later showed how to extend self-organizing map networks for clustering analysis. SOM has been further used in the computer vision field for image segmentation [23], image classification [18], clustering 2D shapes [11], face recognition [3] and more.

An important result is that SOMs are topology preserving, i.e., data vectors that are close in the input space will be close in the resulting map. This aspect makes SOMs extremely

valuable in data exploration tasks. Moehrmann et al. [21] use arbitrary features for clustering the images, and the resulting SOM is displayed to the user with one representative image per unit, allowing to navigate and explore the image data collection fast and intuitively.

Over the years, many modifications have been offered to the traditional SOM [1,6,17], all presenting changes that only affect the learning rule. In our work, we rely on the SOM intuition and develop a novel technique that considers two features, rather than only one, when organizing the input elements into a low-dimensional domain.

Grid Layout Arranging images on a grid is a fundamental task in infographics, data visualization and collection exploration fields. A common approach for such task is to first perform a standard graph layout without the grid constraint and then align the images onto the grid points [12,24,30]. All of the above methods use a *single* metric function to measure the pairwise distance between the images. Reinert et al. [25] introduce a method for organizing images on a grid according to *two* features, but their method is user supervised and uses multiple image selections to define the separating features.

In our work, we solve the grid layout problem by using two separate and independent features and arrange the images into rows and columns according to space and time features, respectively. Furthermore, it should be emphasized that all the above methods consider a rather fuzzy measure of the correct grid layout, while in our work the ground-truth layout is well defined.

3 Overview

Given an unorganized set of N images $\{I_1, \ldots, I_N\}$, we would like to organize the images into a space–time table of size $N_H \times N_V$, where each row represents images taken from about the same location and each column represent images taken at about the same time. Defining such a table is equivalent to clustering the images into (horizontal) rows $\{H_i\}_1^{N_H}$ and into (vertical) columns $\{V_i\}_1^{N_V}$.

First, the images are analyzed, and for each image two feature descriptors x_i^H and x_i^V are extracted. These descriptors aim to represent the horizontal and vertical properties of the images. Images taken from close viewpoints have similar x_i^H descriptors, while images taken from distant viewpoints have distant descriptors. Similarly, the descriptor x_i^V aims to be similar for images taken at about the same time. We assume that the features that separate time and space are known, but they are inaccurate. If they were accurate, the problem of clustering the images into rows and columns would have been trivial. Further details on feature extraction are given in Sect. 7.

The x_i^H and x_i^V descriptors may be vectors of high dimensions. To visualize them, we compute all pairs of distances to define two similarity matrices and apply MDS to embed the input images into two 3D domains (Fig. 1b). Notice how the correct clustering in each domain is non-trivial.

We then imply our BiSOM algorithm which we describe in Sect. 5. Our technique uses a grid of neurons, where each neuron is associated with two weight vectors m_i^H and m_i^V, learns the data and clusters the data by optimizing the grid network. During the training process, the neurons iteratively adjust their weights to the input points, while maintaining the inner relations, namely their grid structure. Besides the clustering of the images into space clusters and time clusters, the BiSOM algorithm outputs the order between the clusters, thus defining the temporal order of the images and the spatial order of the viewpoints. The output of the BiSOM algorithm is an assignment of the input elements to a grid of neurons, thus defining an initial guess for our space–time table.

Finally, in Sect. 6 we develop an objective function based on the Normalized Cut [28] criterion that evaluates the quality of a given table. We use a heuristic relaxation process, which locally optimizes this objective function, while maintaining the order among the clusters, and yields the final space–time table (Fig. 1a).

4 Unary-SOM

The self-organizing map (SOM), introduced by Kohonen [16], is a method for embedding high-dimensional inputs into a two-dimensional grid. Let us denote the set of input elements as $\{x_i\}_{i=1}^N$ where each element is associated with a *single* weight vector $x_i \in \mathbb{R}^D$. Similarly, each neuron is associated with a *single* weight vector with the same dimensions $m_i \in \mathbb{R}^D$. The neurons are organized in a four-connected grid structure of size $N_H \times N_V$, reflecting the target dimension of the table.

The training process is iterative, where at each iteration t all input elements are fed one by one to the network, in a random order. The closest neuron to the input element, in terms of euclidean distance, is chosen to be the winning neuron $m_c(t)$. The weights of $m_c(t)$ and the neurons around it are updated to be closer and more similar to the value of the input element. In the end of the training process, the network of neurons has weights which reflect the input data, so in fact, the input data are embedded into the structure of the neurons. We call this SOM, Unary-SOM, since it considers a single feature only, while later we develop a BiSOM which considers two features. The training algorithm of a Unary-SOM can be described as follows:

Algorithm 1 Unary-SOM

1: initialize the neurons $\{m_i\}_{i=1}^{N}$ with random weights
2: **repeat**
3: **for all** input element x **do**
4: set $m_c(t)$ to be the closest neuron to x
5: update the weights of the neurons $m_i(t)$ around $m_c(t)$ w.r.t $h_i(t)$
6: **end for**
7: **until** convergence

The update of the neighboring neurons' weights is critical and performed by the following process

$$m_i\,(t+1) = m_i\,(t) + \alpha\,(t) \cdot h_i\,(t) \cdot [x\,(t) - m_i\,(t)] \qquad (1)$$

The neighborhood function h_i can be considered as a smoothing kernel over the grid of neurons. The Unary-SOM uses an *isotropic* Gaussian centered around $m_c(t)$, where r_i are the coordinates of the i-th neuron on the neuron grid and r_c are the coordinates of $m_c(t)$:

$$h_i\,(t) = \exp\left\{-\frac{\|r_i - r_c\|^2}{2\sigma^2\,(t)}\right\} \qquad (2)$$

Figure 2a shows the architecture of a 10×10 grid of neurons and the definition of the neighborhood function h_i. In general, the farther a neuron is from $m_c(t)$, the lower the amplitude of the Gaussian is, and hence, the lower is the updating rate of the neuron's weight vector.

The Gaussian width also decreases over time, by using $\sigma\,(t) = \sigma_0 \cdot \exp\left\{-\frac{t}{\lambda}\right\}$ as an exponential time-decaying function of the Gaussian width. Generally speaking, at the first iterations of the training process, the neighborhood update region includes the entire grid of neurons, and as it progresses, the neighborhood size decreases, until eventually, it updates only a single neuron.

Furthermore, the update step amplitude is exponentially decaying over time defined by the learning rate function $\alpha\,(t) = \alpha_0 \cdot \exp\left\{-\frac{t}{\lambda}\right\}$.

5 BiSOM

The BiSOM algorithm is based upon the Unary-SOM. As the name implies, it is designed to deal with input elements that are defined by a bi-vector with *two* descriptors, whereas the Unary-SOM deals with a *single* descriptor. We denote each input point descriptors as $x_i^H, x_i^V \in \mathbb{R}^D$. Similarly, the neurons are also bi-vectors of the same dimensions $m_i^H, m_i^V \in \mathbb{R}^D$. The BiSOM algorithm is described in Algorithm 2.

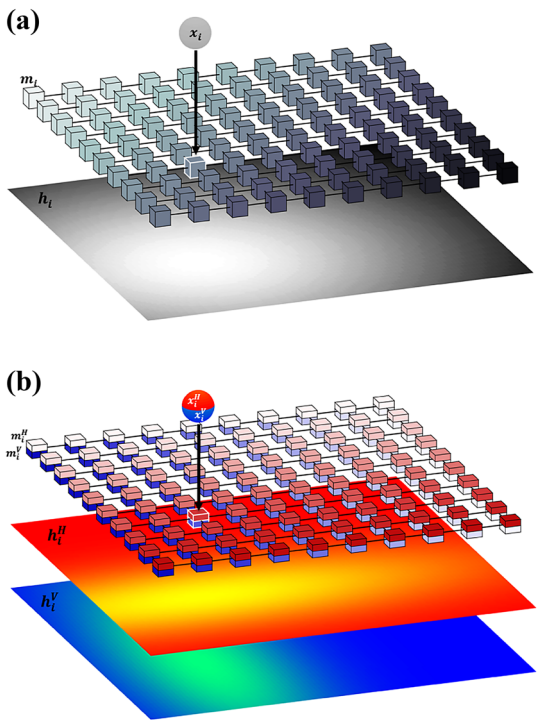


Fig. 2 Architecture of Unary-SOM (**a**) and BiSOM (**b**). The *gray tone* of the neurons represents the single feature that the Unary-SOM is trained to. The *red* and *blue tones* are the horizontal and vertical descriptors of the BiSOM algorithm. The current input element x_i is used to train the network, and the current winning neuron m_c is marked in white. Notice the isotropic (**a**) and anisotropic (**b**) neighborhood functions centered around the winning neuron

Algorithm 2 BiSOM

1: initialize the neurons $\{m_i\}_{i=1}^{N}$ with random weights
2: **repeat**
3: **for all** input element x **do**
4: set $m_c(t)$ to be the closest neuron to x
5: update the H-weights of the neurons $\{m_i{}^H(t)\}$ around $m_c(t)$ w.r.t $h_i^H(t)$
6: update the V-weights of the neurons $\{m_i{}^V(t)\}$ around $m_c(t)$ w.r.t $h_i^V(t)$
7: **end for**
8: **until** convergence

The major difference between the Unary-SOM and BiSOM is the update step, which consists of two consecutive steps:

$$\begin{aligned} m_i^H\,(t+1) &= m_i^H\,(t) + \alpha\,(t) \cdot h_i^H\,(t) \cdot \left[x^H\,(t) - m_i^H\,(t)\right] \\ m_i^V\,(t+1) &= m_i^V\,(t) + \alpha\,(t) \cdot h_i^V\,(t) \cdot \left[x^V\,(t) - m_i^V\,(t)\right] \end{aligned} \qquad (3)$$

First, the neuron's horizontal weight vector $m_i^H\,(t)$ is updated using only $x_i^H\,(t)$ and the neighborhood function

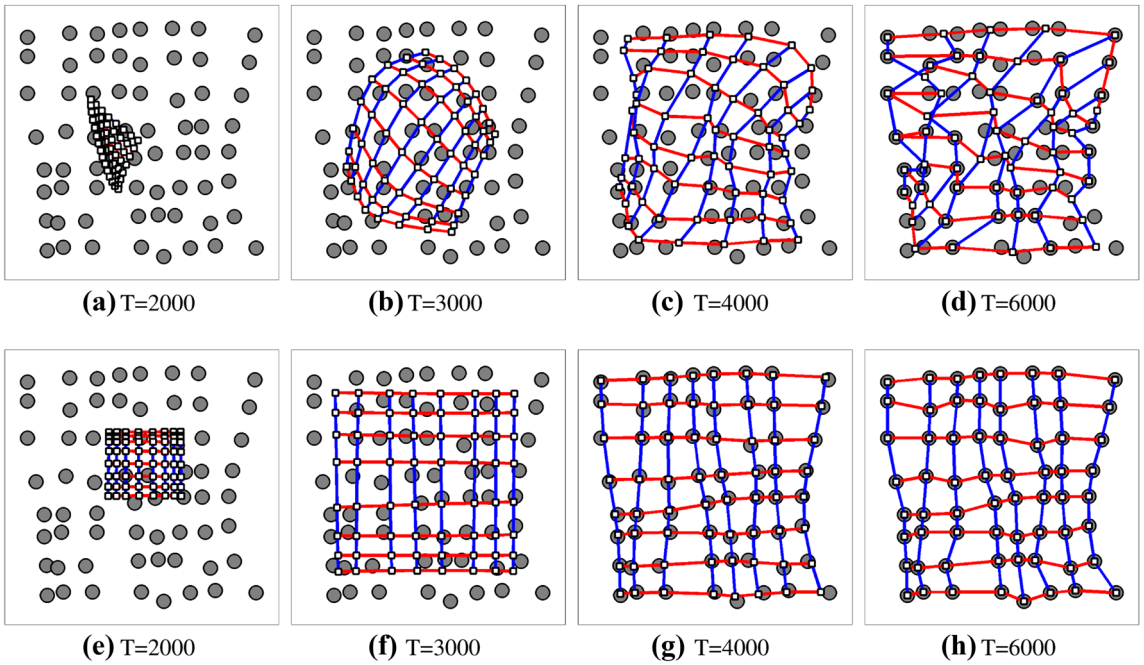


Fig. 3 Training process of the Unary-SOM (*top row*) and BiSOM (*bottom row*) algorithm, demonstrating the adjustment of the neural network grid to the input points (*gray circles*)

$h_i^H(t)$. The vertical neuron's weight is then updated in a similar manner.

Both of the neighborhood functions h_i^H, h_i^V are defined as an *anisotropic* Gaussian which has a different scale in each dimension. We will further discuss the scale factor $\mathscr{F}$ in Sect. 5.1. The location $r_i = (u_i, v_i)$ of the i-th neuron defines the neuron residing row and column on the grid.

$$\begin{aligned} h_i^H(t) &= \exp\left\{-\frac{(u_i-u_c)^2}{\mathscr{F}\cdot 2\sigma^2(t)} + \frac{(v_i-v_c)^2}{2\sigma^2(t)}\right\} \\ h_i^V(t) &= \exp\left\{-\frac{(u_i-u_c)^2}{2\sigma^2(t)} + \frac{(v_i-v_c)^2}{\mathscr{F}\cdot 2\sigma^2(t)}\right\} \end{aligned} \tag{4}$$

Figure 3 demonstrates the training process of both algorithms on a toy example. We use a set of 64 points as our input elements, where the (x, y) coordinates of each point are the x_i^V, x_i^H descriptors, respectively. To adopt the Unary-SOM to our setup, we use a concatenated vector of our two descriptors into one large single descriptor $x_i = [x_i^H, x_i^V]$. The Unary-SOM algorithm (top row) is trained according to the input elements' locations, ignoring the valuable knowledge about their horizontal and vertical distributions. Conversely, the BiSOM algorithm (bottom row) maintains the neuron's grid structure during the whole process. The time complexity of Algorithm 2 is similar to that of Algorithm 1 since the only difference between the algorithms is the update step. For a full demonstration of the training process, see the accompanying video.

5.1 $\mathscr{F}$ factor

The factor $\mathscr{F} \in (0, 1]$ defines the ratio between the major and the minor axis of the anisotropic Gaussian in the update steps. In the first update step, we are updating the horizontal weights of the neurons, using the neighborhood function h_i^H with the horizontal axis as the major axis of the Gaussian (see Fig. 2b). The update magnitude for the adjacent rows decays faster than adjacent columns, thus encouraging neurons in the m_c's row to have similar H-weights. Similarly, in the second step, the neighborhood function h_i^V results in a bigger update magnitude for neurons residing in the winning column than for adjacent columns.

Basically, BiSOM is a generalized version of Unary-SOM that allows us to give different weights to the horizontal and vertical update steps. The ratio $\mathscr{F}$ plays an important role in the training process and has a significant impact on the result.

Setting $\mathscr{F} \to 1$ causes the neighborhood function to reduce back into an isotropic Gaussian with $\sigma_H = \sigma_V = \sigma(t)$. The BiSOM algorithm then degenerates into an Unary-SOM where the descriptor is the concatenated vector $m_i = [m_i^H, m_i^V]$. The horizontal and vertical features are treated equally during the update process. Thus, each neuron is encouraged to be similar to each of its four neighbors equally. Eventually, this leads the neurons to break down the grid structure which means that neurons residing in the same row/column are not necessarily similar.

On the other hand, setting $\mathscr{F} \to 0$ degenerates the anisotropic Gaussian into a one-dimensional Gaussian. This

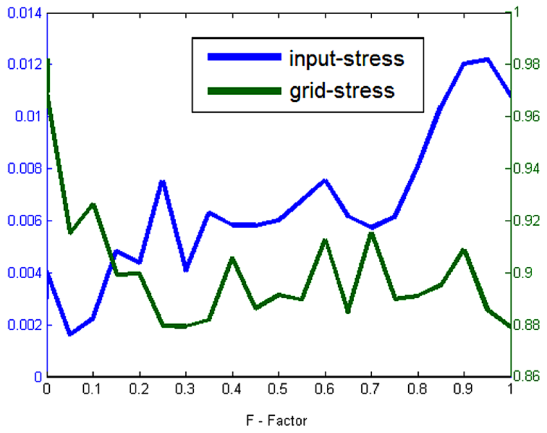


Fig. 4 Grid stress and input stress as a function of $\mathscr{F}$

implies that the first update step updates only the H-weights of the H-neighbors, and the second step similarly. Eventually, the order between the rows and columns is violated.

In our work, we compromise between these two extreme settings and allow the neurons to move independently to preserve the data structure on the one hand and maintain the grid structure on the other hand. To measure the trade-off between these two requirements, we calculate the *grid stress* which is the average distance between each neuron to its four neighbors and the *input stress* which is the average distance between each input element and its matching neuron. Obviously, we would like to minimize both of the stress parameters.

In Fig. 4, we elaborate on this and calculate the grid stress and the input stress for different values of $\mathscr{F}$ using different sets of input elements. The input stress increases linearly as $\mathscr{F}$ increases, while the grid stress is mostly constant for any value of $\mathscr{F}$, but rises fast for $\mathscr{F} < 0.1$. We set empirically $\mathscr{F} = 0.125$ and use it through all our experiments.

5.2 Assignment

At the end of the training process, we obtain a map of neurons. Each neuron represents an entry in the space–time table. Each input element is then assigned by its nearest neuron to a table entry. The assignment is not one to one, and some cells might remain empty or crowded.

6 Refinement

In this section, we define an objective function $f(T, S_H, S_V)$ that evaluates the quality of a space–time table T according to similarity matrices S_H or S_V. We then use a greedy local search to relax our initial guess and to yield the final space–time table. We briefly summarize the N-Cut technique below, and we then elaborate on how it is used to define our objective function.

6.1 Normalized Cuts

Given a fully connected undirected weighted graph $G(V, E)$, where the vertices V are points in the feature space and the edge weight $w(i, j)$ is associated with the similarity between vertex i and j, the Normalized Cut algorithm [28] partitions the vertex set V into two disjoint sets A, B by minimizing the Normalized Cut criterion:

$$N_{\text{cut}}(A, B) = \frac{\text{cut}(A, B)}{\text{ass}(A, V)} + \frac{\text{cut}(A, B)}{\text{ass}(B, V)} \tag{5}$$

A *cut* is defined as the degree of dissimilarity between the two disjoint sets A and B. Similarly, the *association* is the partial volume of A from the entire set V:

$$\text{cut}\,(A, B) = \sum_{a \in A, b \in B} w(a, b) \tag{6}$$

$$\text{ass}\,(A, V) = \sum_{a \in A, v \in V} w(a, v) \tag{7}$$

Shi and Malik [28] show that minimizing the Normalized Cut criterion is naturally related to maximizing the Normalized Association criterion, given by:

$$N_{\text{ass}}(A, B) = \frac{\text{ass}(A, B)}{\text{ass}(A, V)} + \frac{\text{ass}(A, B)}{\text{ass}(B, V)} \tag{8}$$

They further expand this criterion for K way partitioning of the data into K disjoint sets:

$$N^K_{\text{ass}}(A_1, \ldots, A_K) = \sum_{i=1}^{K} \frac{\text{ass}(A_i, A_i)}{\text{ass}(A_i, V)} \tag{9}$$

This implies that the best partition of the set V to K disjoint sets is the partition that maximizes the N^K_{ass} criterion calculated according to the similarity matrix.

6.2 The objective function

The objective function primarily aims to evaluate the clusters' quality. We use the well-known N-Cut scheme [28] that evaluates the clustering structure directly from the original data. The assignment of the input elements to a space–time table T is equivalent to two separate clusters partitions; first, partition of the elements into horizontal clusters $\{H_1, H_2, \ldots, H_{N_H}\}$ and second, vertical clusters $\{V_1, V_2, \ldots, V_{N_V}\}$. The correctness of the H/V-clusters is

measured using the N_{ass}^{K} criterion, according to the corresponding similarity matrix S_H or S_V, respectively.

$$f\left(T, S_H, S_V\right) = N_{\text{ass}}^{N_H}\left(H_1, \ldots, H_{N_H}\right) + N_{\text{ass}}^{N_V}\left(V_1, \ldots, V_{N_V}\right) - \mathscr{P} \tag{10}$$

To generate a smooth, uniform and coherent space–time table, we define a penalty term $\mathscr{P}$, which decreases the objective function when the table T does not satisfy the global constraints. Let us denote the number of elements in the i-th row of the table by n_i^r, similarly n_j^c is the number of elements in the j-th column, and $n_{i,j}^e$ is the number of elements in the cell $[i, j]$. Basically, for a uniform distribution of the input elements in the table, the optimal number of elements in each row, column and cell is: $\frac{N}{N_H}$, $\frac{N}{N_V}$, $\frac{N}{N_H \cdot N_V}$, respectively. The penalty term is then defined by:

$$\mathscr{P} = W_r \sum_{i=1}^{N_H} g\left(n_i^r - \frac{N}{N_H}\right) + W_c \sum_{j=1}^{N_V} g\left(n_j^c - \frac{N}{N_V}\right) + W_e \sum_{i,j} g\left(n_{i,j}^e - \frac{N}{N_H \cdot N_V}\right) \tag{11}$$

The penalty term penalizes the objective function for any deviation from the optimal number of elements in each row, column or cell. The *single-penalty function* $g\,()$ is an arbitrary monotonic symmetric function around 0 that satisfies $g\,(0) = 0$. In our work, we use $g\,(x) = 1 - \mathrm{e}^{-x^2}$. We set the penalty weights $W_{\text{cell}} = 1.5$, $W_{\text{row}} = W_{\text{col}} = 0.5$ empirically to generate a smooth and uniform table.

6.3 The refinement process

The refinement process assumes that most of the elements are assigned to their correct entry in the table, with respect to the ground-truth assignment. Furthermore, we assume that the mismatches, if they exist, are small and local. Namely, a mismatched element is likely to be assigned to one of the adjacent neighbors of the correct cell. These assumptions are strong because the typical performance of BiSOM guarantees that adjacent neurons have similar weights.

At each iteration of the refinement process, the *silhouette index*[1] [27] of each of the elements is computed for all clusters. We then consider only the movements which have a large $s_i\,(j)$ and try to make *small* local movements and swaps in the table aiming to maximize the objective function.

[1] The silhouette index $-1 \leq s_i\,(j) \leq 1$ provides an indication for how well the element i lies within the cluster j. A value of $s_i\,(j)$ close to positive one means that the datum i is appropriately clustered in the cluster j, conversely, a value close to negative one means that the datum i is unlikely to belong to cluster j.

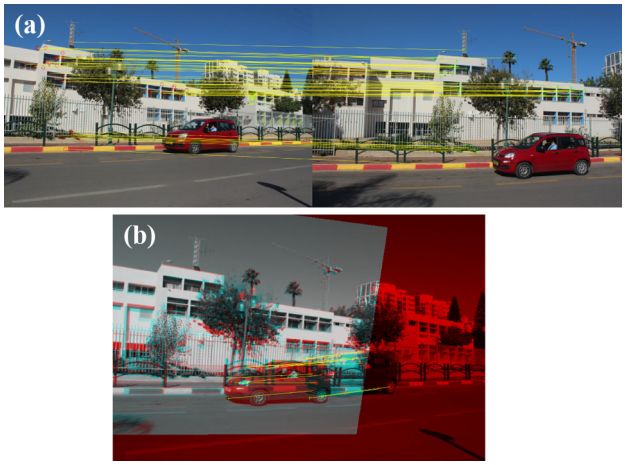


Fig. 5 Example from the RED CAR dataset. **a** Static SIFT matches used to compute to the transformation between the images. **b** Blended registered images and the dynamic SIFT matches used to compute the disparity between the images

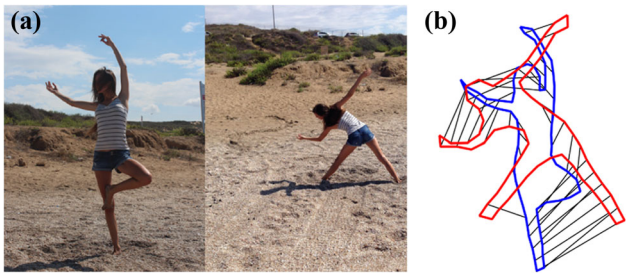


Fig. 6 Example from the DANCER dataset (**a**), and the IDSC matches found using the two contours (**b**)

7 Implementation details

Feature extraction is a initial key step to our method. We experimented with a number of spatial and temporal features according to the nature of each scene. It should be emphasized that automatic feature selection is an independent challenging problem, but is not in the scope of our work. The assumption is that the features used are leading to reasonable but imperfect clustering. We now elaborate on each feature category separately:

7.1 Temporal features

The temporal feature aims to separate the images according to their time of acquisition. We define a metric that measures the time difference between each two images in the set. Notice that we cannot predict the inner order within any two images; thus, the final order of the images in the space–time table can be reversed.

First, we extract the dynamic object in each image using the Grabcut algorithm [26]. Next, according to the nature of

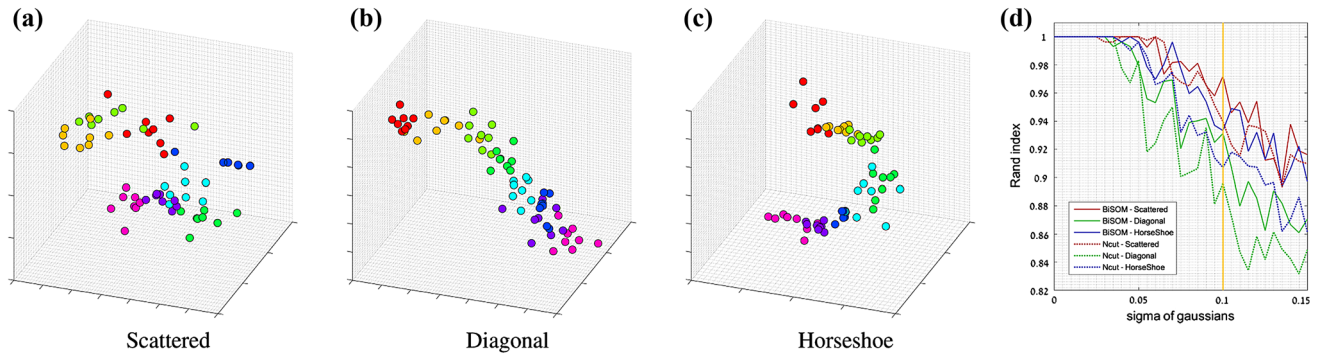


Fig. 7 **a**, **b** and **c** Three typical layouts of the synthetic input points using $\sigma = 0.1$. **d** Rand index computed on the clustering result on each layout as a function of σ

the motion in each scene, we manually choose a different kind of descriptor.

For scenes with a fixed object moving along a linear path (i.e., a car traveling on a road), we use SIFT point tracking, similar to [32]. First, we extract the SIFT points from each image and divide them into static and dynamic points according to their locations on the foreground or background (Fig. 5a). Next, we compute a projective transformation between each pair of images according to the matches between the static points only. We use this transformation to align each pair of images. Finally, after registering both images, we compute the disparity between the dynamic matches that reflects the temporal difference between the two images (Fig. 5b).

For scenes demonstrating a human in a fixed position changing his posture during the motion (i.e., ballet dancer), we use contour-based descriptors (Fig. 6). We use the foreground/background separation and extract the contour of the dynamic object. We compare the contours of each two images by using the inner-distance shape context (IDSC) [19].

7.2 Spatial features

The spatial feature aims to separate the images according to the viewpoint of the photographer.

We used the number of static SIFT matches as a metric for measuring the similarity between two images. The farther apart the images were captured, and the less SIFT matches between them will be found. We also used the GIST descriptor [22] to measure the spatial distance between each two images. Furthermore, we experimented with simpler metrics such as a Chi-square metric on the color histogram of each image.

We display the results using the first feature, although our method showed similar performance using any other choice of spatial feature.

8 Results

8.1 Synthetic data

First, we demonstrate the strengths of our method on synthetic data. We generate sets of 64 points divided into 8 clusters. The points in each cluster are sampled from a 3D anisotropic Gaussian where the average variance of the Gaussian is denoted with σ. The cluster centers are typically organized in one of the three configurations: scattered, diagonal and horseshoe. We observed that diagonal and horseshoe configurations are more typical when dealing with space or time features like in our work (see Fig. 1b).

For each configuration layout, we generate two different point clouds defining our horizontal and vertical descriptors x_i^H, x_i^V, with different values of σ. We compare our BiSOM algorithm result to the result of two separate N-Cut schemes on the horizontal and vertical descriptors separately. To measure the correctness of the clustering result, we compute the *Rand index*[2] [14], comparing the clustering result to the ground truth.

Figure 7d shows the average Rand index of the horizontal and vertical clustering result as a function of σ. We can see that our BiSOM algorithm which considers the two features together outperform the N-Cut scheme which considers each feature independently.

8.2 Real visual data

To demonstrate the effectiveness of our method and to evaluate it, we generated 11 datasets[3] of outdoor and indoor

[2] The Rand index $0 \leq R_I \leq 1$ is a measure of the similarity between two clustering results, with 0 indicating that the two data clusters do not agree on any pair of points and 1 indicating that the data clusters are exactly the same.

[3] Datasets BOATS and SLIDES in the courtesy of Dekel et al. [9]

Table 1 Dataset evaluation. Each dataset consists of N images organized into a table of N_H by N_V. The table displays the horizontal and vertical Rand index and the total swapping distance of the resulting table relative to the ground truth

Dataset			BiSOM			Unary-SOM		N-Cut		IsoMatch	
Name	N	$N_H \times N_V$	Ri^H	Ri^V	Swaps	Ri^H	Ri^V	Ri^H	Ri^V	Ri^H	Ri^V
BEER	20	5×4	1.00	1.00	0	0.70	0.61	0.96	0.74	0.74	0.66
DANCER	30	5×6	1.00	1.00	0	0.74	0.85	0.98	0.89	0.72	0.79
RED CAR	32	4×8	1.00	1.00	0	0.81	0.85	1.00	0.90	0.64	0.83
BOATS	20	2×10	1.00	1.00	1	0.48	0.88	1.00	0.94	0.47	0.90
BAG	18	3×6	1.00	1.00	1	0.71	0.77	1.00	0.91	0.6	0.82
BALLET	100	10×10	1.00	1.00	0	0.84	0.83	0.92	0.91	0.87	0.85
MALL	21	3×7	1.00	0.96	0	0.54	0.76	1.00	0.84	0.57	0.83
SLIDES	32	4×8	1.00	1.00	1	0.66	0.77	1.00	0.85	0.66	0.86
LADDER	66	6×11	0.98	0.97	0	0.72	0.79	1.00	0.69	0.73	0.86
BRIDGE	90	6×15	1.00	0.96	0	0.77	0.88	1.00	0.85	0.77	0.90
WALL	84	7×12	0.97	0.98	0	0.83	0.88	0.95	0.86	0.77	0.86

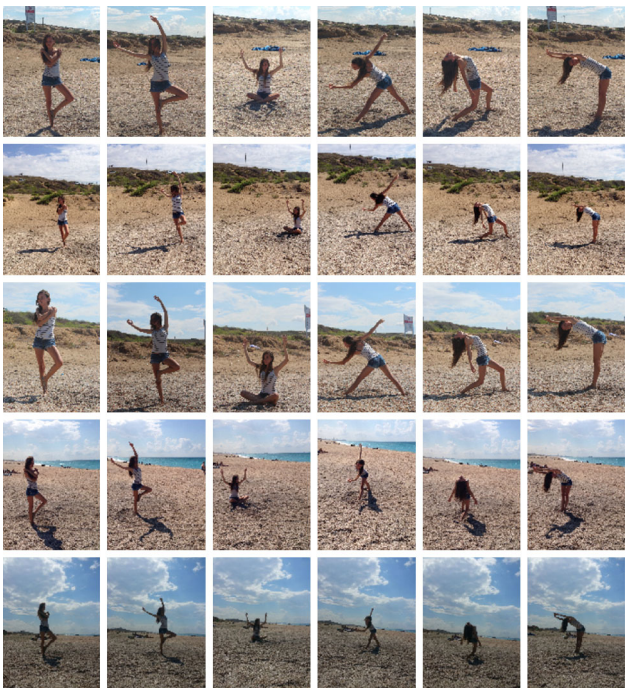


Fig. 8 DANCER—a girl poses dancing stances at the beach. Notice the large variance in viewpoints and poses. The temporal order is not recovered in this example, due to the nature of the scene

scenes. These datasets capture real-life dynamic scenes that occur in a rather limited space–time domain. The events were asynchronously captured with a multitude of capture devices including DSLR cameras and mobile phones. The photographers might have moved slightly during the dynamic event. Therefore, the images taken from the same photographer may differ slightly in the viewpoint. Furthermore, the photographers did not capture the dynamic event synchronously; hence, the input images may have temporal differences in acquisition.

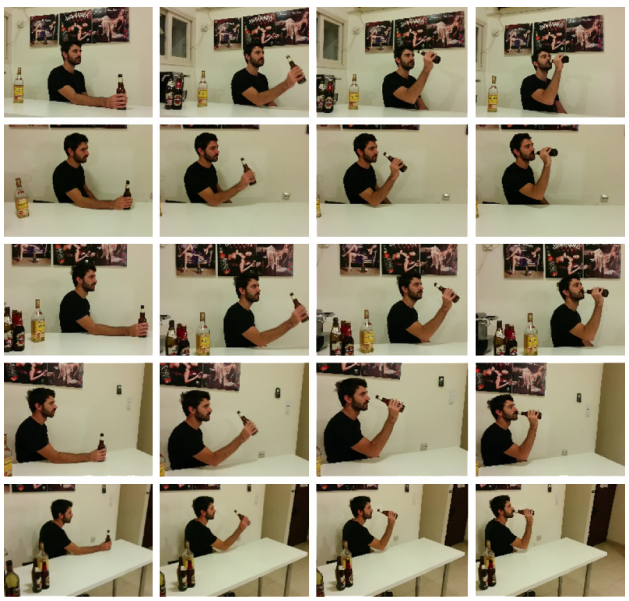


Fig. 9 BEER—a guy picking up a beer bottle to drink. Notice the temporal order (*right to left*) and the spatial order of the viewpoints (*top to bottom*)

Fig. 10 BAG—a girl bending to pick up her bag

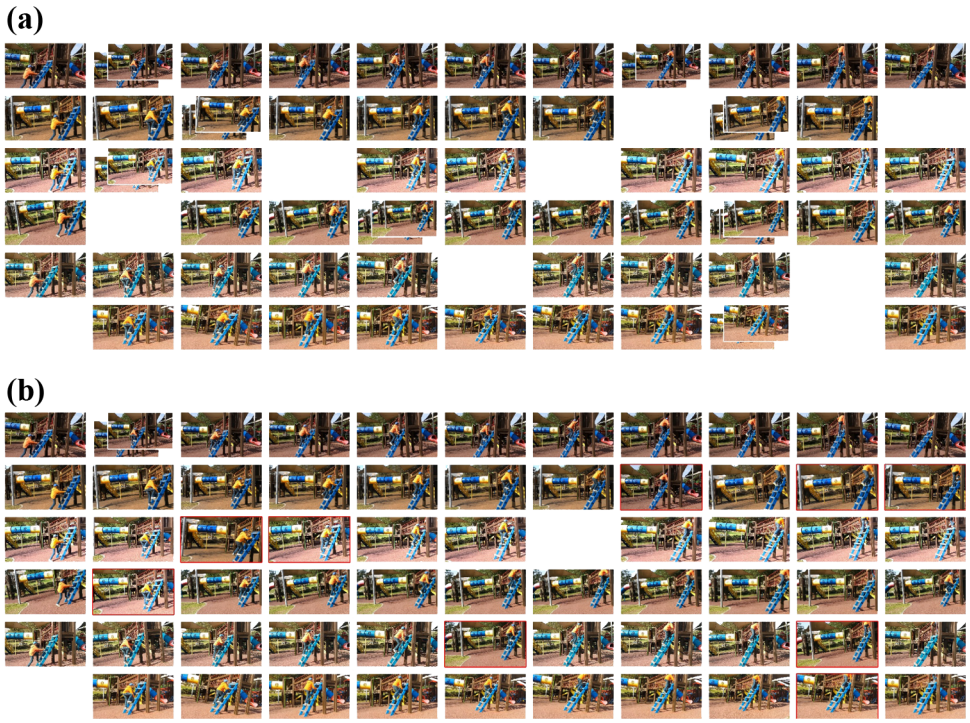


Fig. 11 Ground-truth table of LADDER dataset (**a**), and the generated space–time table (**b**)

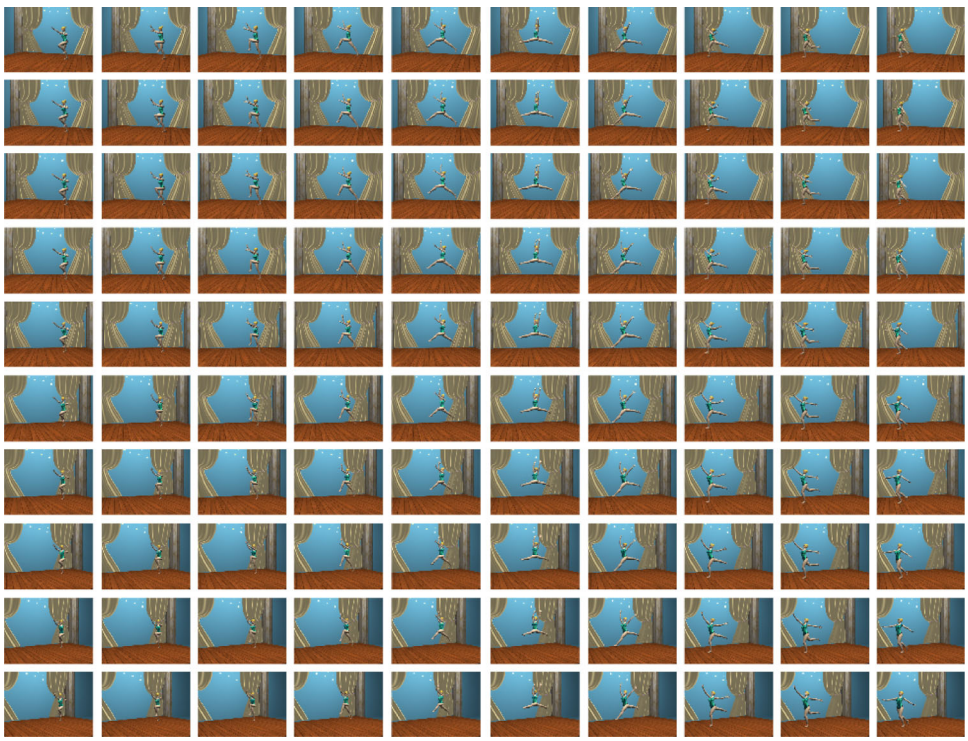


Fig. 12 BALLET—a ballet dancer performing a jump on stage, captured by an audience seated in an amphitheater around the stage. This dataset is computer rendered

The problem of assigning the photographs into the space–time table can be separated into two problems and the evaluation, respectively: (i) correctly clustering the elements into space clusters and time clusters and (ii) correctly ordering the elements within the clusters, namely the temporal and spatial order among the clusters.

To evaluate the clustering result, we use the Rand index described above. To evaluate the ordering result, we use the swapping metric[4] [7], which measures the distance from a given permutation of the clusters to the correct one. Table 1 displays the Rand index and the swapping distance, for each of the sets.

[4] The swapping distance is defined to be the minimum number of swaps, or transpositions, of two adjacent clusters that transforms one permutation into another.

Fig. 13 RED CAR—a red car driving down the street from right to left

Fig. 14 BOATS—a couple of small boats traveling down the river

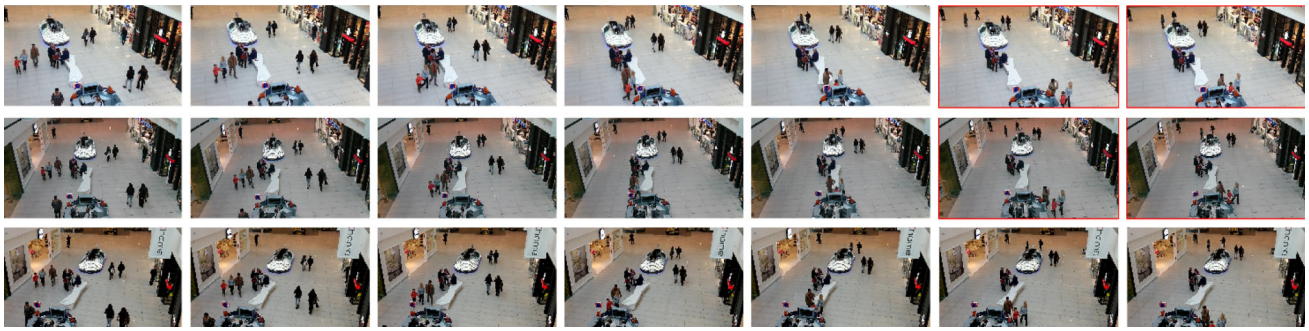


Fig. 15 MALL—people walking at the mall. Notice the three different groups of people walking in different directions

Fig. 16 SLIDES—kids slide down the slide

Fig. 17 BRIDGE—a boy walking on a bridge at the playground

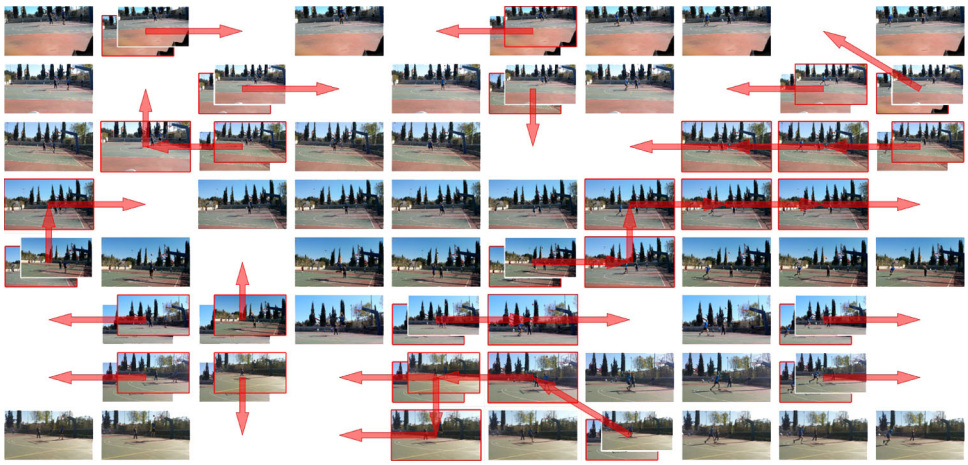


Fig. 18 BASKETBALL—failure case. In this dataset the audience capturing the scene was spread sparsely on the side of the field. The spatial descriptor failed to reflect the distances between the viewpoints, and as a result our method failed to organize the images in a coherent table

Figures 8, 9, 10, 11, 12, 13, 14, 15, 16 and 17 show the resulting space–time tables generated by our algorithm. For a full resolution of the images, see our supplementary material. Notice how the inner order between the rows defines the spatial order of the photographers, while the inner order between the columns defines the temporal order of the event.

Our method can also handle more realistic cases where the ground-truth table contains a different number of images in each row and column. To generate such dataset, we modified the LADDER dataset by removing some of the images and duplicating others. Figure 11 shows the ground-truth table and the resulting space–time table generated by our algorithm. Although the objective function leads to a result with some local mismatches, the final space–time table is more uniform and coherent than the ground-truth table.

8.3 Comparison to other methods

First, we compare our results to the Unary-SOM method introduced in Sect. 4. We use the space descriptor and the time descriptor as a single concatenated descriptor, and the rest of the parameters of the neural network and the training process are identical to the BiSOM method. Next, we compare our results with the N-Cut [28] method by first applying the N-Cut algorithm on the space descriptors obtaining the space clusters, and similarly the time clusters. We further experiment with IsoMatch [12], a grid layout technique which organizes the set of images into a full grid. Although the resulting table is always full and uniform, it doesn't reflect the temporal and spatial content of the event. Table 1 displays the comparison results on all datasets. A major drawback of all the alternative methods is their inability to provide the temporal or spatial order among the clusters, meaning that the rows and columns of the space–time table are not ordered correctly.

9 Discussion and conclusions

We have presented a technique to organize a set of photographs into a space–time table. The embedding of the photographs into the table reveals information regarding the clustering of the images into space and time clusters and the order within the clusters.

We introduced BiSOM, that is built upon SOM, which considers two independent features rather than one, and showed that it outperforms SOM. We also showed that using the space and time features independently to solve two independent clustering problems is inferior to using both the features coupled with BiSOM. Furthermore, unlike other

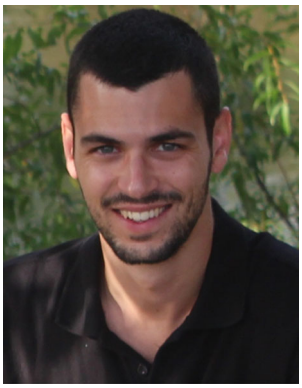
methods, our method reveals the temporal and spatial order within the clusters.

Regarding the limitations of our work, we assume that the number of clusters is given and use it to initialize the size of our grid. Furthermore, note that our method relies on the extracted features and descriptors used to describe the distances among the images. This implies that we are bounded to common limitations of feature extraction methods. Figure 18 shows a failure case, where the images are captured from a wide baseline, which means that dynamic object appearance in adjacent viewpoints is significantly different.

Although our work focuses on space–time separation, the technique is applicable for any set of features. Given a set of images, we may use style and content descriptors to embed the images into a style content table. Another possible application can be organizing human shapes according to their body shape and postures. Furthermore, another possible venue for research is extending the BiSOM to higher dimensions, e.g., using three descriptors and a three-dimensional grid of neurons to organize a set of tiles according to shape, color and texture features.

References

1. Aoki, T., Aoyagi, T.: Self-organizing maps with asymmetric neighborhood function. Neural Comput. **19**(9), 2515–2535 (2007)
2. Averbuch-Elor, H., Cohen-Or, D.: Ringit: Ring-ordering casual photos of a temporal event. ACM Trans. Graph. **35**(1), 33 (2015)
3. Bashyal, S., Venayagamoorthy, G.K.: Recognition of facial expressions using gabor wavelets and learning vector quantization. Eng. Appl. Artif. Intell. **21**(7), 1056–1064 (2008)
4. Brahmachari, A.S., Sarkar, S.: View clustering of wide-baseline n-views for photo tourism. In: Graphics, Patterns and Images (Sibgrapi), 2011 24th SIBGRAPI Conference on, pp. 157–164. IEEE (2011)
5. Caspi, Y., Irani, M.: Spatio-temporal alignment of sequences. IEEE Trans. Pattern Anal. Mach. Intell. **24**(11), 1409–1424 (2002)
6. Chen, L.P., Liu, Y.G., Huang, Z.X., Shi, Y.T.: An improved som algorithm and its application to color feature extraction. Neural Comput. Appl. **24**(7–8), 1759–1770 (2014)
7. Cormode, G., Muthukrishnan, S.: The string edit distance matching problem with moves. ACM Trans. Algorithms. (TALG) **3**(1), 2 (2007)
8. Moses, Y., Avidan, S., et al.: Space-time tradeoffs in photo sequencing. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 977–984 (2013)
9. Dekel, T., Moses, Y., Avidan, S.: Photo sequencing. Int. J. Comput. Vis. **110**(3), 275–289 (2014)
10. Dexter, E., Pérez, P., Laptev, I.: Multi-view synchronization of human actions and dynamic scenes. In: BMVC, pp. 1–11. Citeseer (2009)
11. Endo, M., Ueno, M., Tanabe, T.: A clustering method using hierarchical self-organizing maps. J. VLSI Signal Process. Syst. Signal Image Video Technol. **32**(1–2), 105–118 (2002)
12. Fried, O., DiVerdi, S., Halber, M., Sizikova, E., Finkelstein, A.: IsoMatch: Creating informative grid layouts. In: Computer Graphics Forum, vol. 34, no 2, pp. 155–166. Wiley (2015)
13. Furukawa, Y., Curless, B., Seitz, S.M., Szeliski, R.: Towards internet-scale multi-view stereo. In: Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on, pp. 1434–1441. IEEE (2010)
14. Hubert, L., Arabie, P.: Comparing partitions. J. Classif. **2**(1), 193–218 (1985)
15. Kiang, M.Y.: Extending the kohonen self-organizing map networks for clustering analysis. Comput. Stat. Data Anal. **38**(2), 161–180 (2001)
16. Kohonen, T.: The self-organizing map. Proc. IEEE **78**(9), 1464–1480 (1990)
17. Lee, J.A., Verleysen, M.: Self-organizing maps with recursive neighborhood adaptation. Neural Netw. **15**(8), 993–1003 (2002)
18. Lefebvre, G., Laurent, C., Ros, J., Garcia, C.: Supervised image classification by som activity map comparison. In: Pattern Recognition, 2006. ICPR 2006. 18th International Conference on, vol. 2, pp. 728–731. IEEE (2006)
19. Ling, H., Jacobs, D.W.: Shape classification using the inner-distance. IEEE Trans. Pattern Anal. Mach. Intell. **29**(2), 286–299 (2007)
20. Mauro, M., Riemenschneider, H., Van Gool, L., Leonardi, R., Brescia, I.: Overlapping camera clustering through dominant sets for scalable 3D reconstruction. In: BMVC, vol. 1, no 2, p. 3 (2013)
21. Moehrmann, J., Bernstein, S., Schlegel, T., Werner, G., Heidemann, G.: Improving the usability of hierarchical representations for interactively labeling large image data sets. In: Human-Computer Interaction. Design and Development Approaches, pp. 618–627. Springer, Berlin (2011)
22. Oliva, A., Torralba, A.: Modeling the shape of the scene: a holistic representation of the spatial envelope. Int. J. Comput. Vis. **42**(3), 145–175 (2001)
23. Ong, S.H., Yeo, N., Lee, K., Venkatesh, Y., Cao, D.: Segmentation of color images using a two-stage self-organizing network. Image Vis. Comput. **20**(4), 279–289 (2002)
24. Quadrianto, N., Song, L., Smola, A.J.: Kernelized sorting. In: Advances in neural information processing systems, pp. 1289–1296 (2009)
25. Reinert, B., Ritschel, T., Seidel, H.P.: Interactive by-example design of artistic packing layouts. ACM Trans. Graph. (TOG) **32**(6), 218 (2013)
26. Rother, C., Kolmogorov, V., Blake, A.: Grabcut: interactive foreground extraction using iterated graph cuts. ACM Trans. Graph. (TOG) **23**(3), 309–314 (2004)
27. Rousseeuw, P.J.: Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. J. Comput. Appl. Math. **20**, 53–65 (1987)
28. Shi, J., Malik, J.: Normalized cuts and image segmentation. IEEE Trans. Pattern Anal. Mach. Intell. **22**(8), 888–905 (2000)
29. Snavely, N., Seitz, S.M., Szeliski, R.: Photo tourism: exploring photo collections in 3D. In: ACM transactions on graphics (TOG), vol. 25, no 3, pp. 835–846. ACM (2006)
30. Strong, G., Gong, M.: Self-sorting map: An efficient algorithm for presenting multimedia data in structured layouts. IEEE Trans. Multimedia. **16**(4), 1045–1058 (2014)
31. Tomasi, C., Kanade, T.: Shape and motion from image streams under orthography: a factorization method. Int. J. Comput. Vis. **9**(2), 137–154 (1992)
32. Zhou, H., Yuan, Y., Shi, C.: Object tracking using sift features and mean shift. Comput. Vis. Image Underst. **113**(3), 345–352 (2009)



Shahar Ben-Ezra is a computer vision algorithm developer in a innovative startup company in Israel. He received his B.Sc. in Electrical Engineering from Technion—Israel Institute of Technology (2009) and his M.Sc. in Electrical Engineering from Tel Aviv University (2016). His main research interests are computer vision and machine learning, in particular segmentation, tracking and clustering of images.

Daniel Cohen-Or is a professor in the School of Computer Science. He received his B.Sc. cum laude in both mathematics and computer science (1985), M.Sc. cum laude in computer science (1986) from Ben-Gurion University and Ph.D. from the Department of Computer Science (1991) at State University of New York at Stony Brook. He received the 2005 Eurographics Outstanding Technical Contributions Award. His research interests are in computer graphics, in particular synthesis, processing and modeling techniques. His main interest right now is in few areas: image synthesis, motion and transformations, shapes and surfaces, analysis and reconstruction.