Learning Graph Representation with Randomized Neural Network for Dynamic Texture Classification
Lucas C Ribas, Jarba Joaci de Mesquita Sá Junior, Antoine Manzanera, Odemir M Bruno

To cite this version:
Lucas C Ribas, Jarba Joaci de Mesquita Sá Junior, Antoine Manzanera, Odemir M Bruno. Learning Graph Representation with Randomized Neural Network for Dynamic Texture Classification. Applied Soft Computing, 2021. hal-03431533

HAL Id: hal-03431533
https://hal.science/hal-03431533
Submitted on 16 Nov 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Abstract

Dynamic textures (DTs) are pseudo periodic data on a space × time support, that can represent many natural phenomena captured from video footages. Their modeling and recognition are useful in many applications of computer vision. This paper presents an approach for DT analysis combining a graph-based description from the Complex Network framework, and a learned representation from the Randomized Neural Network (RNN) model. First, a directed space × time graph modeling with only one parameter (radius) is used to represent both the motion and the appearance of the DT. Then, instead of using classical graph measures as features, the DT descriptor is learned using a RNN, that is trained to predict the gray level of pixels from local topological measures of the graph. The weight vector of the output layer of the RNN forms the descriptor.

Several structures are experimented for the RNNs, resulting in networks with final characteristics of a single hidden layer of 4, 24, or 29 neurons, and input layers 4 or 10 neurons, meaning 6 different RNNs. Experimental results on DT recognition conducted on Dyntex++ and UCLA datasets show a
high discriminatory power of our descriptor, providing an accuracy of 99.92%, 98.19%, 98.94% and 95.03% on the UCLA-50, UCLA-9, UCLA-8 and Dyntex++ databases, respectively. These results outperform various literature approaches, particularly for UCLA-50. More significantly, our method is competitive in terms of computational efficiency and descriptor size. It is therefore a good option for real-time dynamic texture segmentation, as illustrated by experiments conducted on videos acquired from a moving boat.

**Keywords:** Dynamic Texture, Complex Networks, Learned Features, Randomized Neural Networks

1. Introduction

Dynamic textures (DT) are visual patterns that vary both spatially and temporally. Therefore, they can represent a wide range of time-varying natural phenomena, such as fire, sea waves, fountains, smoke, among others. The dynamics of the textures are related to the objects or processes present in the videos, that exhibit a certain stationarity with respect to both space and time [1]. In general, they can be characterized by simple and repetitive patterns and non-rigid motion guided by non linear and stochastic dynamics such as swaying branches or bubbling water [2, 3]. Over the last years, DTs have received significant attention from the computer vision community, due to their several applications, such as face spoofing detection [4], traffic monitoring [5], crowd behavior analysis [6], fire detection [7], etc.

The basic task of DT recognition consists in classifying a piece of video (i.e. a 3D space × time volume of data), with respect to meaningful classes such as those mentioned earlier. This task can then be extended to further goals like semantic segmentation of dynamic surfaces, or discriminative detection of objects over non stationary backgrounds. The notorious difficulty of DT recognition comes from the large variability and the diffuse character of natural textures that, unlike objects, cannot - in general - be precisely located in space and time. To these challenges are added those specific to the dynamic nature
of the problem: the model should represent the different types of movement
that can affect the texture: flow, swell, growth, waving, etc. In addition, as a
basic step, DT recognition is expected to work in a fast and reactive manner. It
must then be able to run in real-time and to adapt itself on line to a changing
environment. Motivated by these challenges, a range of approaches has been
proposed, that can be divided into the following five categories: (i) filter-based,
(ii) motion-based, (iii) discrimination-based, (iv) learning-based and (v) model-
based methods.

The filter-based methods extend still texture analysis techniques to charac-
terize the dynamic textures at different scales in the spatio-temporal domain.
In [8, 9], the authors used oriented energy filters to extract the spatio-temporal
characteristics of the dynamic textures. Then, Arashloo and Kittler [10] devel-
oped a multiresolution binarized statistical image features approach that gener-
ates binary code by filtering operations on different regions of the space × time
and in three orthogonal planes. Independent component analysis learns the fil-
ters for each orthogonal plane, which generally represents a significant comput-
tational cost. Feichtenhofer et al. [11] presented a bag of space × time energies
framework that extracts primitive features from a bank of spatio-temporally
steerable filters. In [12] is proposed the spatio-temporal Directional Number
transitional Graph (DNG) descriptor. The feature extraction is based on the
direction of the temporal flow to compute the structure of the local neigh-
borhood and the transition of the principal directions between frames. Other
approaches explore high-order features from Gaussian gradients [13] or use un-
supervised 3D filter learning and local binary encoding [14]. In the same way, in
[15], the authors introduced a novel filtering kernel, named difference of deriva-
tive Gaussians, which is based on high-order derivative of a Gaussian kernel.
The methods of this category have achieved good results in several databases,
however, they are generally limited both in terms of computational complexity
and performance.

The motion-based techniques essentially use the kinematic information es-
imated from frames to describe the dynamic textures. Many techniques use
the optical flow due to its efficiency in the description of the motion, like [16]
that used the global magnitude and direction of the vector field of normal flow.
Other approaches are based on the combination of normal flow features and pe-
riodicity [17], multi-resolution histogram of the velocity and acceleration fields
[18] or rotation and scale invariant features of the image distortions computed
using optical flow [19]. More recently, Nguyen et al. [20] presented an operator
based on local vector patterns that encode the local motion features from beams
of dense trajectories with good results. In [21], the authors developed a method
that is based on the time-varying vector field and used singular patterns pooled
from the bag-of-keypoint-based coefficients dictionary. Although the methods
from this category are efficient for motion description, they often neglect most of
the appearance information, which is essential in many problems. Furthermore,
the optical flow techniques suppose brightness constancy and local smoothness
in the dynamic textures, which can be a very strong constraint [12].

Discrimination-based methods generally use local features such as the Local
Binary Patterns (LBP), widely used in image analysis. In fact, most of the
methods based on discrimination for dynamic textures are extensions of LBP
methods to the space × time domain. In this sense, Zhao and Pietikäinen
[22, 23] proposed the two most popular methods, which have the advantage
of simplicity. The Volumetric LBP (VLBP) [23] encodes the local feature by
means of 3D neighborhood and uses as feature vector a very large histogram
(i.e., 16,384 descriptors). Next, the authors proposed the LBP-TOP [22], which
applies the LBP operator on three orthogonal planes (two temporal planes and
one spatial plane) and combines the three histograms, reducing the size of the
feature vector. Recently, other works also have extended the LBP operator such
as multiresolution edge-weighted local structure pattern [24], helix local binary
pattern [25] and rotation-invariant version [26]. On the other hand, in [27] the
authors introduced the LTGH descriptor, which combines LBPs and gray-level
co-occurrence matrix (GLCM) on orthogonal planes (TOP). Nguyen et al. [28]
proposed the momental directional patterns framework that extends the Local
Derivative Pattern operator to improve the capture of directional features. In
it is proposed the local tetra pattern operator on three orthogonal planes, which computes feature codes based on the central pixel and directions of the neighbors. In general, these methods provide promising performances, however, they have some limitations such as sensitivity to noise and large feature vectors.

Following the success of the deep convolutional neural networks (CNN) on image classification, there has recently been a growing interest in learning-based methods for dynamic texture analysis. Qi et al. applied pre-trained CNN to extract mid-level features from the frames. The first and second order statistics from the mid-level features are used to create the feature vector. Later, Arashloo et al. presented a deep multi-scale convolutional network (PCANet-TOP) architecture that learns filters employing the principal component analysis (PCA) on each orthogonal plane. Andrearczyk and Whelan proposed a framework based on applying CNNs on three orthogonal planes. This framework used the AlexNet and GoogleNet models that were trained on spatial frames and temporal slices from the dynamic texture videos. The output of the three CNNs are combined to obtain a feature vector. The approaches of this category are powerful and usually obtain outstanding results in DT classification. However, they have known limitations that can make them unfeasible for many real-world problems, such as their difficulty to be implemented on embedded platforms, and the need for a considerable number of training samples, that can be impossible to get, particularly when online adaptation/learning is needed. Zhao et al. explore two different approaches to learn 3D random features: learning-based Fisher vector and the learning-free binary encoding. In is proposed a DT descriptor, which employs Randomized Neural Networks (RNNs) to learn the local features from three orthogonal planes. The determining interest is that the RNN has a single feed-forward hidden layer and a fast learning algorithm, making the feature extraction extremely efficient. In the model-based category, the methods analyze the DTs through mathematical or physical models. Popular methods of this category are based on linear dynamical systems (LDS). In the estimated parameters
of the LDS model are used for characterizing the DT. However, the method has
limitations such as a poor invariance to rotation, scale and illumination [32].
To overcome the view-invariance limitation, Ravichandran et al. [36] proposed
the Bag-of-dynamical Systems (BoS), which uses the LDSs features with non-
Euclidean parameters computed from non-linear dimensionality reduction and
clustering. Later, in [37] the authors extended the method to extract interest
points with a dense sampling and used two alternative approaches for forming
the code books. Wang and Hu [38] also proposed a bag-of-words approach to
encode chaotic features. More recently, methods that use deterministic walkers
[39] and Complex Network (CN) theory [40, 3] have been used with success for
DT analysis. In particular, the CN-based methods obtained great results due
to their flexibility and ability to represent the motion and appearance in the
DTs. These methods model the DT as graphs, and extract statistical measures
from them. Despite the promising results, we believe that a more robust char-
acterization of the graph can improve the performance compared to the classic
statistical measures. Furthermore, for graph modeling, these methods have as
a drawback the need to adjust four parameters.

In summary, the main drawbacks present in the existing methods from the
literature are: (i) strong emphasis on either appearance or motion, and poor
combination of the two aspects; (ii) large feature vectors; (iii) complex and
computationally costly algorithms; (iv) many parameters to adjust; and (v)
very simple graph measures.

To address these problems, we present in this paper a DT method that
extends the approach proposed in [41] for static textures, that is, it combines
a graph based description from the Complex Network (CN) framework, and
a learned representation from the Randomized Neural Network (RNN) model.
Using a small piece of DT data, the RNN can be trained to predict the gray
level value of a pixel, from its local topology features, provided by measures on
the space × time graph of the video. The learned weight vector of the output
layer of the RNN is then used as descriptor of the video. Thus, we refer to
our method as Complex Patterns learned using Randomized Neural Networks
Our contributions are:

- A more simple directed graph modeling than [3, 40, 42] for dynamic textures based on only one parameter (radius).
- A robust learned representation for graphs using the RNN, instead of using classic statistical measures. The RNN is an extremely compact neural network that has a fast and few-shots learning algorithm, which is a determining advantage with respect to deep neural network approaches.
- A DT descriptor that provides competitive accuracy and processing time compared to many literature methods.

We evaluate this descriptor in recognition task using different classifiers on two popular DT datasets. Then, we illustrate the potential reactivity of the model on a DT segmentation example. This paper is organized as follows: Section 2 presents the related background on CN and RNN. Section 3 explains our approach in detail. Section 4 explains our experiments and shows some results. Section 5 presents and discusses quantitative results on DT recognition, and qualitative results on reactive DT segmentation. Finally, Section 6 presents the conclusions and future works.

2. Methodologies

2.1. Complex Networks

The complex network research field emerges from the intersection of the areas of graph theory, physics, statistics and computer science, in order to understand and analyze complex systems. Indeed, many systems and data are formed by a set of elements that interact with each other and that can be represented as networks by defining the entities (vertices) and the relationships among them (edges). Some examples of these systems are the internet, the WWW and social networks. In particular, researches in scale-free networks [43], identification of community structure in many networks [44] and small-world [45] networks
have drawn attention from the scientific community on the study of complex networks, which is a multidisciplinary research field. In computer vision, the flexibility and expressiveness of this framework have been used over the last years for color-texture classification with multi-layer CN [46] and dynamic texture analysis with diffusion in networks [3].

Formally, a complex network can be defined as a graph $G = (V,E)$, where $V$ is the set of vertices and $E$ the set of edges connecting vertex pairs. In this work, we adopted the term graph to refer to a complex network in order to avoid confusion with the neural network term. The graphs can be directed and weighted, which is when the edge $e_{ij}$ is directed from $i$ to $j$ and has a weight $w_{ij} \in \mathbb{R}$. For description, two important information about the graph can be used: the out-degree and the weighted out-degree or strength. The out-degree of a vertex $i$ counts the number of connections from $i$:

$$k_i = |\{v_j; e_{ij} \in E\}|,$$

where $|S|$ denotes the cardinality of set $S$. The weighted out-degree $s_i$ computes the sum of the weights of all connections from $i$:

$$s_i = \sum_{e_{ij} \in E} w_{ij}.$$

These measures quantify topological characteristics and different properties of the graph that are useful for the identification of hubs, scale-free property, etc.

2.2. Randomized Neural Networks

A well known problem in neural networks is that gradient descent based learning is slow and needs a huge quantity of data, specially when the number of neural weights is large. To tackle this issue, randomized neural networks [47, 48, 49, 50] were proposed. In their simplest version, these networks have a single hidden layer whose weights are random, and an output layer whose weights can be computed using a closed-form solution.
To mathematically describe the neural network used in this work, let $Z = [z_1, z_2, \ldots, z_N]$ be a matrix composed of the outputs of the hidden layer, according to the following equation

$$Z = \phi ([\vec{w}_1, \vec{w}_2, \ldots, \vec{w}_Q]^T [\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_N])$$

(3)

where $T$ denotes transpose operation, $\vec{x}_i = [-1, x_{i1}, x_{i2}, \ldots, x_{ip}]^T$ is an input vector $i$ with $p + 1$ attributes, $\vec{w}_q = [w_{q0}, w_{q1}, \ldots, w_{qp}]^T$ is the set of random weights of a determined neuron $q$, $N$ is the number of feature vectors $\vec{x}_i$, $Q$ is the number of neuron units of the hidden layer, and $\phi(\cdot)$ is a transfer function.

Next, after adding a constant value $-1$ to each vector $\vec{z}_i$ for the bias weights of the output layer neurons, the aim of the closed-form training is to find a matrix $M$ that satisfies $D = MZ$, where $D = [\vec{d}_1, \vec{d}_2, \ldots, \vec{d}_N]$ is a matrix of (ground truth) label vectors, each one corresponding to its respective input $\vec{x}_i$. To compute $M$, it is possible to use the Moore-Penrose pseudo-inverse [51, 52] with the Tikhonov regularization [53, 54], thus resulting in the following equation

$$M = DZ^T(ZZ^T + \lambda I)^{-1}$$

(4)

where $0 < \lambda < 1$ and $I$ is an identity matrix of size $(Q + 1) \times (Q + 1)$.

3. Proposed Approach

In this section, we describe the CPNN approach, which extends to dynamic textures the static texture characterization approach proposed in [41]. In the first step, the dynamic texture video is modeled into two directed graphs: spatial and temporal graphs. Then, information from these graphs are used to train the neural networks and build a signature.

3.1. Modeling Dynamic Texture in Directed Graphs

In order to analyze the dynamic textures, it is important to capture features that represent the appearance and motion properties of the video. To this end,
in the proposed approach, we model the dynamic texture video into two directed graphs: the spatial graph $G_S = (V_S, E_S)$ that represents the appearance properties and the temporal graph $G_T = (V_T, E_T)$ that contains the motion characteristics. Each pixel $i = (x_i, y_i, t_i)$ of the graphs is represented by a vertex $i \in V$, such that $x_i$ and $y_i$ are the spatial coordinates and $t_i$ the temporal index of the pixel $i$.

The two graphs have as main difference the definition of the set of edges. In the spatial graph, in order to characterize appearance, we connect only vertices that are from the same frame. Thus, the set $E_S$ is given by the connection of all vertices whose distance is smaller than or equal to a given radius $r$ and whose time coordinates $t_i$ and $t_j$ are equal (as illustrated in Figure 1(a)),

$$e_{ij} \in E_S \iff ((x_i - x_j)^2 + (y_i - y_j)^2 + (t_i - t_j)^2) \leq r \text{ and } t_i = t_j \ (5)$$

On the other hand, in the temporal graph, we focus on the relationships between frames to analyze the motion characteristics. In this way, in the set $E_T$, we connect all vertices whose distance is smaller than or equal to $r$ and their time coordinates are different (as exemplified in Figure 1(b)),

$$e_{ij} \in E_T \iff ((x_i - x_j)^2 + (y_i - y_j)^2 + (t_i - t_j)^2) \leq r \text{ and } t_i \neq t_j \ (6)$$

Each pixel represented by a vertex has a different gray-level, which relates to the texture patterns. To add this information to the graph topology, we transform it to a directed and weighted graph, as follows: First, for each edge a direction is defined based on the order of gray-level. Specifically, in an edge $e_{ij}$ the vertex $i$ points to the vertex $j$ if $I(i) \leq I(j)$. In addition, a weight $w_{ij}$ is defined from the difference of intensities and distance between the two pixels.
that represent the vertices:

\[
    w_{ij} = \begin{cases} 
        \frac{|I(i)-I(j)|}{L} & \text{if } r = 1 \\
        \frac{\text{dist}(i,j)-1}{r-1} + \left( \frac{|I(i)-I(j)|}{L} \right)^2 & \text{otherwise}.
    \end{cases}
\]  

(7)

where \( I(i) \in [0, 255] \) is the gray-level of the pixel \( i \), \( \text{dist}(i, j) \) is the Euclidean distance between the pixels \( i \) and \( j \) and \( L \) is the highest gray-level. When the radius is \( r = 1 \) then the weight \( w_{ij} \) is the difference of gray levels normalized by the maximum gray-level \( L \), producing a value in \([0, 1]\). On the other hand, if \( r > 1 \) then the weight \( w_{ij} \) is given by the average of the distance between the pixels normalized by the maximum distance \( r \), and the normalized gray scale difference, which also provides a value in \([0, 1]\). Thus, this weight function includes information about the pixel neighborhood and the difference of gray-level in order to balance the importance between geometric information and color in the texture representation \([10]\).

3.2. Proposed Signature

In this approach, we train the randomized neural network with topological characteristics from graphs that model the videos of dynamic textures. For training the RNNs, the out-degree \( k_i \) and the weighted out-degree \( s_i \) of each vertex from the modeled graph compose the input matrices. The learned weights from the output layer form the feature vector for dynamic texture representation. The main steps of the proposed method are summarized in the flowchart diagram in Figure 3.

In order to create the matrix of input vectors for randomized neural network, we use the evolution of the graph for different values of modeling parameter \( r \). Therefore, for each vertex \( i \) of the graph an input vector and its corresponding output label are defined as follows: the out-degree values of the vertex for different radii \( \{r_1, r_2, ..., r_R\} \) are considered as the input vector \( \vec{e}_i = [k_i^1, k_i^2, k_i^3, ..., k_i^R] \), where \( R \) is the maximum radius value. The output label is simply the gray-level \( d_i = I(i) \). A matrix of input vectors \( X(k) \) for the out-degree and a vector of output labels \( D \) is obtained considering all vertices.
Figure 1: Modeling of a three frame video in a (a) spatial graph and a (b) temporal graph (using $r = 1$).

of the graph. The number of columns of $X_{(k)}$ (and the size of vector $D$) is then
the number of samples $N$ which corresponds to all the pixels, but could also be
any sub-sample of the video. In this way, it is possible to statistically analyze
the topology evolution of the vertices that represent the pixels with different
gray-levels. Figure 2(a) illustrates the step for obtaining the matrices of input
vector and label from a spatial graph. In addition to the matrix $X_{(k)}$, we also
build the matrix of input vectors for the weighted out-degree $X(s)$.

$$X^S(k) = \begin{bmatrix} k_1^1 & 1 & \cdots & 1 \\ k_2^1 & 1 & \cdots & 1 \\ \vdots & & \ddots & \vdots \\ k_R^1 & 1 & \cdots & 1 \end{bmatrix}$$

$$D = [I(0) \mid 1 \cdots 1]$$

$$\vec{f}_k = DZ^T(ZZ^T + \lambda I)^{-1}$$

Figure 2: Construction of the output vector and label from a spatial graph by modeling dynamic textures with different values of $r$ and using the out-degree $k_i$.

In the next step, the weights of the matrix $W$ of the input layer of the randomized neural network are generated randomly. However, in the methods for dynamic texture analysis it is important that the feature vector be always the same for a given sample. In this sense, in order to always obtain the same weight values, we use the Linear Congruent Generator (LCG) [55, 56] with fixed parameters to generate the uniform pseudo random numbers for the matrix $W$,

$$V(n + 1) = (a \ast V(n) + b) \mod c, \quad (8)$$

where $V$ is the random sequence of length $E = Q \ast (p + 1)$ and started by $V(1) = E + 1$. The values of $a, b$ and $c$ are parameters defined as $a = E + 2$, $b = E + 3$ and $c = E^2$ (these values were adopted in [57]). Thus, the matrix $W$ is composed of the vector $V$ divided into $Q$ segments of values $p + 1$. The values of the matrix $W$ and $X$ (each row) are normalized using standard score (zero mean and unit variance).

The feature vector that represents the dynamic textures is built based on matrix $M$, which becomes here a vector $\vec{f}$ (since the output labels are scalar),
which is computed as: \( \vec{f} = DZ^T(ZZ^T + \lambda I)^{-1} \), such that \( \lambda = 10^{-3} \) (Figure 2(b)) and the length of \( \vec{f} \) is \( Q + 1 \) because of the bias value. To characterize appearance and movement of dynamic textures, we propose to use the spatial graph and temporal graph as inputs to train the randomized neural network. Therefore, firstly, two randomized neural networks are trained, each one with a different matrix of input vectors \( X^S_{(k)} \) and \( X^S_{(s)} \) extracted from the spatial graphs. From these trained randomized neural networks, we obtain two vectors \( \vec{f}^S_k \) and \( \vec{f}^S_s \). For the temporal graph, the same procedure is performed and two vectors are obtained \( \vec{f}^T_k \) and \( \vec{f}^T_s \) using the matrices \( X^T_{(k)} \) and \( X^T_{(s)} \), respectively. In this way, to represent the appearance and motion of the dynamic texture, the following concatenation is proposed:

\[
\vec{\Upsilon}(Q,R) = \begin{bmatrix} \vec{f}^S_k, \vec{f}^S_s, \vec{f}^T_k, \vec{f}^T_s \end{bmatrix}, \tag{9}
\]

where \( Q \) is the number of hidden layer neurons and \( R \) is the maximum radius of graph modeling, i.e., the number of radii used to construct the matrix \( X \).

Figure 3 illustrates the steps to obtain the vector \( \vec{\Upsilon}(Q,R) \), which is built using a single value of \( Q \) and \( R \). These two parameters influence the training of the neural network and, therefore, different characteristics are learned for different parameter values. Thus, the vector \( \vec{\Psi}(Q)_{R_1,R_2} \) that concatenates the vectors \( \vec{\Upsilon}(Q,R) \) for different values of \( R \) is used:

\[
\vec{\Psi}(Q)_{R_1,R_2} = \begin{bmatrix} \vec{\Upsilon}(Q)_{R_1}, \vec{\Upsilon}(Q)_{R_2} \end{bmatrix}. \tag{10}
\]

Finally, we propose a feature vector \( \vec{\Theta}(R)_{Q_1,Q_2,Q_m} \) that combines the vector \( \vec{\Psi}(Q)_{R_1,R_2} \) for different numbers of neurons:

\[
\vec{\Theta}_{Q_1,Q_2,...,Q_m} = \begin{bmatrix} \vec{\Psi}(Q_1)_{R_1,R_2}, \vec{\Psi}(Q_2)_{R_1,R_2}, ..., \vec{\Psi}(Q_m)_{R_1,R_2} \end{bmatrix}. \tag{11}
\]

The overall algorithm of the proposed method is described in Algorithm 1.
Figure 3: Flowchart diagram of the proposed method.
Algorithm 1: CPNN method

Data: Video $V$

Result: Feature Vector $\tilde{\Upsilon}(Q)_R$

Parameter: Number of hidden neurons $Q$, maximum radius $R$

/* Feature Vector for a video $V$ and parameters $Q$ and $R$ */
$\tilde{\Upsilon}(Q)_R \leftarrow \text{CPNN}(V, Q, R)$

Function $\text{CPNN}(V, Q, R)$:

/* computes the graph measures of the vertices for each radius and builds the input matrices */
for $r \leftarrow 1$ to $R$ do

$X_{(k)}^S(r,:)$ $\leftarrow$ SpatialGraphDegree $(V,r)$
$X_{(s)}^S(r,:)$ $\leftarrow$ SpatialGraphStrength $(V,r)$
$X_{(k)}^T(r,:)$ $\leftarrow$ TemporalGraphDegree $(V,r)$
$X_{(s)}^T(r,:)$ $\leftarrow$ TemporalGraphStrength $(V,r)$
end

$D \leftarrow V$ // the label vector is the gray-scale values of the video pixels

/* trains the RNN for each input matrix and obtains the output weights */
$\vec{f}_k^S \leftarrow \text{trainRNN}(X_{(k)}^S,D,Q)$
$\vec{f}_s^S \leftarrow \text{trainRNN}(X_{(s)}^S,D,Q)$
$\vec{f}_k^T \leftarrow \text{trainRNN}(X_{(k)}^T,D,Q)$
$\vec{f}_s^T \leftarrow \text{trainRNN}(X_{(s)}^T,D,Q)$

$\tilde{\Upsilon}(Q)_R \leftarrow [\vec{f}_k^S, \vec{f}_s^S, \vec{f}_k^T, \vec{f}_s^T]$ // combines the output weights in a unique feature vector

return $\tilde{\Upsilon}(Q)_R$
Function trainRNN(X, D, Q):

X = Zscore(X)  // normalize the input matrix

W = LCG(Q,P+1,Q*(P+1))  // generate the random weights

X = addBias(X,-1)  // add the bias in the input matrix

Z = Activation(W*X)  // activation function of the hidden layer

Z = addBias(Z,-1)  // add bias in the Z

lambda = 0.001

M = (D*Z')/(Z*Z' + lambda * eye(Q+1))  // calculates the output weights with the Moore-Penrose pseudo-inverse

return M

4. Validation Setup

To evaluate our method, two benchmark databases were used. They are:

- Dyntex++ [58]: this database, which is a compilation of the Dyntex databases [59], has 3,600 videos divided into 36 classes, 100 videos per class, each one of size 50 × 50 × 50. Figure 4(a) shows examples of the first frame of samples from Dyntex++.

- UCLA-50 [60]: this database is composed of 50 classes, 4 videos per class, each one of size 75 × 48 × 48. Also, two variations from the UCLA-50, both proposed in [36], were used in our experiments. The first (UCLA-9) combines videos taken from different viewpoints and groups them into 9 classes: smoke (4 samples), flowers (12), boiling water (8), sea (12), fire (8), water (12), fountains (20), waterfall (16) and plants (108). The second (UCLA-8) discards the class “plants” because it has a large set of samples when compared to the other classes. The first frame of some samples from the UCLA database is shown in Figure 4(b).
In the validation procedure, we used the 1-NN classifier (we used a implementation from Weka [61]) in order to compare our results to other descriptors available in the literature. As evaluation protocol, we employed the procedure adopted in [36, 58, 25] for the UCLA-50, UCLA-9, and UCLA-8; and adopted in [40] for the Dyntex++. Thus, we used 4-fold and 10-fold cross-validation for the UCLA-50 and Dyntex++, respectively. In this case, we use 10 trials for a statistically more reliable result. On the UCLA-9 and UCLA-8 databases, we used hold-out (50% of samples for training and the remainder for test). Average accuracy and standard deviation of 20 trials were used to quantify the performance of our method. The graph modeling step of the proposed method was programmed in the C language, while for training and feature extraction with the RNN was used the Matlab 9.2 software. The trained weights of the output layer (i.e., the feature vectors) computed by the proposed method in all databases are available in GitHub [1].

5. Results and Discussions

5.1. Dynamic Texture Classification

Firstly, we perform a parametric analysis of our approach. For this, a feature vector is extracted of each sample from UCLA-50 and Dyntex++ databases using different values of $Q$ and $R$. Figure 5 summarizes the average accuracy for the UCLA-50 and Dyntex++ databases using the feature vector $\Psi(Q)_{R_1,R_2}$ with $Q = 4$ and different values and combinations of $R$. In the plot, the rows represent the values of $R_1$ and the columns the values of $R_2$. The main diagonal corresponds to a single value of $R$ used to build the feature vector. In this last case, the highest average accuracy, 94.51%, is obtained using $R = 4$. On the other hand, when using two values of $R$ combined, the highest average accuracy is achieved using the combination $(R_1, R_2) = (4, 10)$. This indicates that the combination of local (small radius) and regional (large radius) information from

[1]https://github.com/lucascorreiaribas/CPNN
Figure 4: Initial frames of dynamic texture samples from the databases. Each row represents a class.
Figure 5: Accuracy for different values and combination of one or two radii, averaged on the datasets UCLA-50 and Dyntex++.

Figure 6 presents the accuracies using the feature vector $\vec{\Theta}_{Q_1, Q_2, \ldots, Q_m}$ with one and two values of $Q$ on the UCLA-50 and Dyntex++ databases. It can be seen that, on the UCLA-50 database, the higher accuracies are obtained using large values of $Q$ (all combinations with $Q = 29$), while small values of $Q$ provide better accuracies on Dyntex++ database. In a second experiment, we test the feature vector $\vec{\Theta}_{Q_1, Q_2, \ldots, Q_m}$ combining three different values of $Q$, as it can be seen in Table 1. Note that, as we increase the values of $Q$ in the combinations, the accuracy tends to stabilize and then decrease, meanwhile the size of the feature vector is increased. In this sense, the combination $\{4, 24, 29\}$ can be considered as a good trade-off between the size of feature vectors and accuracy.
Table 1: Accuracy for the feature vector $\hat{\Theta}_{Q_1, Q_2, \ldots, Q_m}$ using three value combinations for $Q$.

<table>
<thead>
<tr>
<th>$Q$</th>
<th># of features</th>
<th>UCLA</th>
<th>Dyntex++</th>
</tr>
</thead>
<tbody>
<tr>
<td>{04, 09, 14}</td>
<td>240</td>
<td>99.10 (± 1.21)</td>
<td>95.06 (± 1.04)</td>
</tr>
<tr>
<td>{04, 09, 19}</td>
<td>280</td>
<td>99.10 (± 1.02)</td>
<td>95.08 (± 1.14)</td>
</tr>
<tr>
<td>{04, 09, 24}</td>
<td>320</td>
<td>99.50 (± 0.89)</td>
<td>95.19 (± 1.10)</td>
</tr>
<tr>
<td>{04, 09, 29}</td>
<td>360</td>
<td>99.50 (± 0.89)</td>
<td>94.71 (± 1.21)</td>
</tr>
<tr>
<td>{04, 14, 19}</td>
<td>320</td>
<td>99.30 (± 0.98)</td>
<td>95.38 (± 0.97)</td>
</tr>
<tr>
<td>{04, 14, 24}</td>
<td>360</td>
<td>99.40 (± 0.94)</td>
<td>95.32 (± 1.09)</td>
</tr>
<tr>
<td>{04, 14, 29}</td>
<td>400</td>
<td>99.60 (± 0.82)</td>
<td>95.05 (± 1.16)</td>
</tr>
<tr>
<td>{04, 19, 24}</td>
<td>400</td>
<td>99.10 (± 1.21)</td>
<td>95.10 (± 1.10)</td>
</tr>
<tr>
<td>{04, 19, 29}</td>
<td>440</td>
<td>99.50 (± 0.89)</td>
<td>94.86 (± 1.18)</td>
</tr>
<tr>
<td>{04, 24, 29}</td>
<td>480</td>
<td>99.92 (± 0.37)</td>
<td>95.03 (± 1.27)</td>
</tr>
<tr>
<td>{09, 14, 19}</td>
<td>360</td>
<td>99.10 (± 1.02)</td>
<td>95.18 (± 0.91)</td>
</tr>
<tr>
<td>{09, 14, 24}</td>
<td>400</td>
<td>99.50 (± 0.89)</td>
<td>95.18 (± 1.04)</td>
</tr>
<tr>
<td>{09, 14, 29}</td>
<td>440</td>
<td>99.50 (± 0.89)</td>
<td>94.90 (± 1.10)</td>
</tr>
<tr>
<td>{09, 19, 24}</td>
<td>440</td>
<td>98.80 (± 1.36)</td>
<td>94.81 (± 1.15)</td>
</tr>
<tr>
<td>{09, 19, 29}</td>
<td>480</td>
<td>99.10 (± 1.02)</td>
<td>94.69 (± 1.21)</td>
</tr>
<tr>
<td>{09, 24, 29}</td>
<td>520</td>
<td>99.30 (± 0.98)</td>
<td>94.87 (± 1.12)</td>
</tr>
<tr>
<td>{14, 19, 24}</td>
<td>480</td>
<td>98.80 (± 1.36)</td>
<td>94.94 (± 1.02)</td>
</tr>
<tr>
<td>{14, 19, 29}</td>
<td>520</td>
<td>99.10 (± 1.02)</td>
<td>94.72 (± 1.19)</td>
</tr>
<tr>
<td>{14, 24, 29}</td>
<td>560</td>
<td>99.40 (± 0.94)</td>
<td>94.72 (± 1.01)</td>
</tr>
<tr>
<td>{19, 24, 29}</td>
<td>600</td>
<td>99.00 (± 1.21)</td>
<td>94.24 (± 1.11)</td>
</tr>
</tbody>
</table>

on both databases. However, it is important to emphasize that the method obtained good results for a large number of parameter values, demonstrating robustness to parameter changes. For example, the combination \{4, 14\} produces 160 features, and the accuracies are 99.60% and 95.31% on the UCLA-50 and Dyntex++ databases, respectively.

We also compare our experimental results in the classification task to the results obtained by other existing methods. Table 2 presents the accuracy obtained for different methods on UCLA-50 database. In these experiments, the results indicate that the proposed method achieved the highest accuracy, which is followed by the CNN-GoogLeNet [33] approach with an accuracy of 99.50%. The next accuracy of 97.15% was obtained by the Randomized Neural Network based signature (RNN-DT) [35] method.

A statistical hypothesis test was performed to evaluate the significance of the difference in performance between the proposed method and the compared
Figure 6: Classification results for the feature vector $\vec{\Theta}_{Q_1, Q_2, \ldots, Q_m}$ with one single ($m = 1$) or two different ($m = 2$) values of $Q$ on the Dyntex++ and UCLA-50 databases. The diagonal cells correspond to single values of $Q$, the others to combinations of two distinct values of $Q$. 
methods on the UCLA-50 database. For this purpose, we employ the paired-sample t-test and Wilcoxon signed-rank test [62] from the Matlab implementation with a significance level equal to $\alpha = 0.05$. To perform this test, the accuracy of the proposed method, Diffusion, DPSWNet, and RNN-DT methods were computed as the average of 20 trials. Thus, the null hypothesis is that the performance of the proposed method is equal to the compared methods, while in the alternative hypothesis the proposed method is statistically superior to the others. When our hypothesis is tested in the comparison with the Diffusion method, the $p$-value is $p = 5.687e-19$ and $p_w = 1.255e-5$ to the t-test and Wilcoxon test, respectively. Since $p$ and $p_w < \alpha$ the null-hypothesis can be rejected in favor of alternative hypothesis, confirming that the performance (mean and median accuracy for t-test and Wilcoxon test, respectively) of the proposed method is statistically superior to the Diffusion method in UCLA-50 database. The $p$-values for the comparison with the RNN-DT ($p = 1.844e-17$ and $p_w = 3.094e-5$) and DPSWNet ($p = 6.683e-20$ and $p_w = 1.614e-5$) methods are also much lower than $\alpha$ in the two tests, so we can reject the null-hypothesis. Regarding the GoogLenet method, the result of the statistical test must be taken with caution because we use the result of the original paper. Therefore, our hypothesis is that the proposed method improved the result in relation to the GoogleLenet in 0.25%, so the null-hypothesis is the negation of this hypothesis. Based on the t-test and Wilcoxon test, the $p$-values are $p = 0.0002$ and $p_w = 0.00093$, respectively. These values reject the null-hypothesis and indicate the superiority of the proposed method.

Table 3 provides a comparison of literature methods on the UCLA-8 and UCLA-9 databases, which consider videos taken from different viewpoints in the same class. On the UCLA-8 database, the proposed method obtained the second-highest accuracy (98.94%), which is very similar to the highest accuracy achieved by the CNN-GoogLeNet (99.02%). On the other hand, on the UCLA-9 database, the DPSWNet [42] method achieved the highest accuracy (99.10%) while the proposed method obtained 98.19%. Table 4 presents the comparison results for the Dyntex++ database. On this database, our pro-
Table 2: Comparison in the 50-class UCLA database (1-NN classifier and 4-fold cross validation). For the methods with '*', the results are taken from [25], [24] and [40], the results with '+' are obtained from the original paper, while the others were generated.

<table>
<thead>
<tr>
<th>Descriptor</th>
<th>ACC (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>KDT-MD* [63]</td>
<td>89.50</td>
</tr>
<tr>
<td>DFS* [2]</td>
<td>89.50</td>
</tr>
<tr>
<td>3D-OTF* [64]</td>
<td>87.10</td>
</tr>
<tr>
<td>CVLBP* [30]</td>
<td>93.00</td>
</tr>
<tr>
<td>HLBP* [25]</td>
<td>95.00</td>
</tr>
<tr>
<td>MEWLSP* [24]</td>
<td>96.50</td>
</tr>
<tr>
<td>LBP-TOP* [22]</td>
<td>94.50</td>
</tr>
<tr>
<td>VLBP* [23]</td>
<td>89.5</td>
</tr>
<tr>
<td>DPSW [39]</td>
<td>94.60 (± 1.98)</td>
</tr>
<tr>
<td>CNN-GoogLeNet+ [33]</td>
<td>99.50</td>
</tr>
<tr>
<td>CNDT [40]</td>
<td>94.62 (± 1.04)</td>
</tr>
<tr>
<td>Diffusion+ [3]</td>
<td>98.50 (± 3.37)</td>
</tr>
<tr>
<td>DPSWNet+ [42]</td>
<td>98.00 (± 3.50)</td>
</tr>
<tr>
<td>RNN-DT+ [35]</td>
<td>97.05 (± 1.87)</td>
</tr>
<tr>
<td>CPNN</td>
<td>99.92 (± 0.37)</td>
</tr>
</tbody>
</table>

posed method was surpassed by the Local Binary Patterns (LBP-TOP) [22], RNN-DT [35] and RI-VLBP [23]. In this way, we can affirm that our result is not statistically superior to the others in these two databases. However, it is important to emphasize some aspects of the methods. CNNs are currently among the most powerful approaches in image analysis and have high computational cost. Also, they require a large number of samples for training, which are drawbacks compared to our method. The RNN-DT is a state-of-the-art method that shares with the CPNN the same fundamental core of using neural weights as descriptors. Furthermore, our proposed approach produces a smaller feature vector (480 descriptors but the combination \{4,14\} produces 160 descriptors with competitive accuracies) when compared to the LBP-TOP (768 descriptors) and RI-VLBP (16,384 descriptors), MBSIF (6,144 descriptors) and MEWLSP (1,536 descriptors). The feature vector size can be crucial in some applications which require low computational cost to classify the DT and memory consumption.

We also compared the results of the proposed method with other approaches in the literature that are more similar to ours. In particular, the RNN-DT
method uses the same neural network architecture considered in our work, but in a three orthogonal planes scheme to train the model, without the graph modeling information. The RNN-DT method obtained a slightly higher accuracy in the 9-class UCLA and Dyntex++ databases, while the proposed approach improves the accuracy in 2.95% and 1.2% when compared with the RNN-DT method on the 50-class and 8-class UCLA databases, respectively. On the other hand, the CNDT and DPSWNet methods employed different ways to model the dynamic texture in graphs with topological statistical measures as descriptors. In relation to these methods, with the exception of the 9-class UCLA database, the proposed method obtained higher accuracies in all databases. These results indicate that the combination of the graph-based description and learned representation from RNNs can improve the characterization of the dynamic texture.

Table 3: Comparison of the proposed method with other dynamic texture methods in the 9-class and 8-class UCLA databases (1-NN classifier and half of the samples for training and the remainder for testing). The results with ‘*’ are taken from [25] and [24], the results with ‘+’ are obtained from the original paper, while the others were generated.

<table>
<thead>
<tr>
<th>Descriptor</th>
<th>ACC (%)</th>
<th>9-class UCLA</th>
<th>8-class UCLA</th>
</tr>
</thead>
<tbody>
<tr>
<td>3D-OTF* [64]</td>
<td>96.32</td>
<td>95.80</td>
<td></td>
</tr>
<tr>
<td>CVLBP* [30]</td>
<td>96.90</td>
<td>95.65</td>
<td></td>
</tr>
<tr>
<td>HLB0* [25]</td>
<td>98.35</td>
<td>97.50</td>
<td></td>
</tr>
<tr>
<td>MEWLS* [24]</td>
<td>98.55</td>
<td>98.04</td>
<td></td>
</tr>
<tr>
<td>MBSIF* [10]</td>
<td>98.75</td>
<td>97.80</td>
<td></td>
</tr>
<tr>
<td>High level feature* [65]</td>
<td>92.67</td>
<td>85.65</td>
<td></td>
</tr>
<tr>
<td>DNGP* [72]</td>
<td>98.10</td>
<td>97.00</td>
<td></td>
</tr>
<tr>
<td>WMFS* [60]</td>
<td>96.95</td>
<td>97.18</td>
<td></td>
</tr>
<tr>
<td>Chaotic vector* [38]</td>
<td>85.10</td>
<td>85.00</td>
<td></td>
</tr>
<tr>
<td>VLBP* [23]</td>
<td>96.30</td>
<td>91.96</td>
<td></td>
</tr>
<tr>
<td>LBP-TOP* [22]</td>
<td>96.00</td>
<td>93.67</td>
<td></td>
</tr>
<tr>
<td>CNN-GoogLeNet+</td>
<td>98.35</td>
<td>99.02</td>
<td></td>
</tr>
<tr>
<td>DPSW [39]</td>
<td>96.33 (± 2.46)</td>
<td>93.41 (± 6.01)</td>
<td></td>
</tr>
<tr>
<td>CNDT [40]</td>
<td>95.61 (± 2.72)</td>
<td>94.32 (± 4.18)</td>
<td></td>
</tr>
<tr>
<td>DPSWNet+ [32]</td>
<td>99.10 (± 0.86)</td>
<td>96.55 (± 7.13)</td>
<td></td>
</tr>
<tr>
<td>Diffusion+ [3]</td>
<td>97.80 (± 1.53)</td>
<td>96.22 (± 4.80)</td>
<td></td>
</tr>
<tr>
<td>RNN-DT+ [35]</td>
<td>98.54 (± 1.56)</td>
<td>97.74 (± 2.99)</td>
<td></td>
</tr>
<tr>
<td>CPNN</td>
<td>98.19 (± 2.27)</td>
<td>98.94 (± 1.42)</td>
<td></td>
</tr>
</tbody>
</table>

In addition to 1-NN, we also consider other classifiers to evaluate the poten-
Table 4: Comparison of the proposed method and others in the DynTex++ database (1-NN classifier and 10-fold cross validation). The results with "*" are obtained from [35], the results with "+" are taken from the original paper, while the others were generated.

<table>
<thead>
<tr>
<th>Descriptor</th>
<th>ACC (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>VLBP [23]*</td>
<td>96.14 (± 0.77)</td>
</tr>
<tr>
<td>LBP-TOP [22]*</td>
<td>97.72 (± 0.43)</td>
</tr>
<tr>
<td>DPSW [39]*</td>
<td>91.39 (± 1.29)</td>
</tr>
<tr>
<td>CNDT [40]*</td>
<td>83.86 (± 1.40)</td>
</tr>
<tr>
<td>DPSWNet [42]+</td>
<td>93.50 (± 1.27)</td>
</tr>
<tr>
<td>Diffusion [3]+</td>
<td>93.80 (± 1.08)</td>
</tr>
<tr>
<td>RNN-DT [35]+</td>
<td>96.51 (± 0.94)</td>
</tr>
<tr>
<td>CPNN</td>
<td>95.03 (± 1.27)</td>
</tr>
</tbody>
</table>

potential of the methods. The classifiers are: Random Forest [67], Deep Random Vector Functional Link - D-RVFL [68] and Linear Discriminant Analysis - LDA [69].

We consider the dynamic texture descriptors with the features and source codes available. Table 5 shows the accuracies obtained on the UCLA-50, UCLA-9 and UCLA-8 databases. In the table, the rows represent the different DT descriptors, while the columns represent the different classifiers evaluated on each database.

On the UCLA-50 database, the proposed method achieved the highest accuracy in all classifiers, which are very similar to the RNN-DT and DPSWNet methods. On the other hand, on the UCLA-9 and UCLA-8 databases, the RNN-DT method had a better performance, except for the UCLA-9 using the Random Forest and D-RVFL classifiers, where our method had a slightly higher accuracy.

In particular, we can observe that on UCLA-50 database and using the D-RVFL classifier, our descriptor CPNN achieved a good performance compared to other descriptors, such as Diffusion, RI-VLBP, and DPSW methods. The core of the CPNN method is learning features from randomized neural networks, which can be viewed as a variant of the RVFL network. This can explain the good performance of these methods since their features are the weights of the output layer.

This argument motivates the investigation for extending our method to other neural network architectures with more layers, although this may penalize the computational competitiveness of the method. On the DynTex++ database, the proposed method also obtained the highest accuracy for the D-RVFL classifier.
Table 5: Accuracy obtained on the UCLA databases by different dynamic texture descriptors (represented in the rows) using the Random Forest, D-RVFL and LDA classifiers.

<table>
<thead>
<tr>
<th>Descriptor</th>
<th>Random Forest</th>
<th>D-RVFL</th>
<th>LDA</th>
</tr>
</thead>
<tbody>
<tr>
<td>UCLA-50</td>
<td>80.37 (1.49)</td>
<td>86.79 (4.03)</td>
<td>82.84 (6.97)</td>
</tr>
<tr>
<td>UCLA-9</td>
<td>81.10 (1.10)</td>
<td>73.38 (4.37)</td>
<td>56.48 (4.79)</td>
</tr>
<tr>
<td>UCLA-8</td>
<td>72.87 (1.11)</td>
<td>86.32 (3.18)</td>
<td>85.00 (6.32)</td>
</tr>
<tr>
<td>VLBP</td>
<td>97.30 (0.54)</td>
<td>91.53 (4.03)</td>
<td>93.07 (4.27)</td>
</tr>
<tr>
<td>DPSWNet</td>
<td>94.15 (0.97)</td>
<td>84.18 (3.33)</td>
<td>60.22 (7.35)</td>
</tr>
<tr>
<td>CPSW</td>
<td>93.75 (0.94)</td>
<td>87.65 (3.03)</td>
<td>63.29 (4.96)</td>
</tr>
<tr>
<td>CNDT</td>
<td>97.00 (0.71)</td>
<td>85.25 (2.23)</td>
<td>57.45 (2.82)</td>
</tr>
<tr>
<td>Diffusion</td>
<td>91.72 (0.16)</td>
<td>63.13 (0.22)</td>
<td>83.22 (0.17)</td>
</tr>
<tr>
<td>RNN-DT</td>
<td>95.37 (0.10)</td>
<td>79.22 (0.22)</td>
<td>83.17 (0.16)</td>
</tr>
<tr>
<td>CPNN</td>
<td>93.44 (0.27)</td>
<td>81.17 (0.34)</td>
<td>87.74 (0.15)</td>
</tr>
</tbody>
</table>

Table 6: Accuracy generated for the Dyntex++ database by different dynamic texture descriptors (represented in the rows) using the Random Forest, D-RVFL and LDA classifiers.

<table>
<thead>
<tr>
<th>Descriptor</th>
<th>Random Forest</th>
<th>D-RVFL</th>
<th>LDA</th>
</tr>
</thead>
<tbody>
<tr>
<td>VLBP</td>
<td>84.47 (0.26)</td>
<td>79.48 (0.48)</td>
<td>89.54 (0.35)</td>
</tr>
<tr>
<td>DPSWNet</td>
<td>90.60 (0.25)</td>
<td>85.98 (0.31)</td>
<td>85.61 (0.25)</td>
</tr>
<tr>
<td>CPSW</td>
<td>90.17 (0.08)</td>
<td>63.13 (0.42)</td>
<td>83.22 (0.17)</td>
</tr>
<tr>
<td>CNDT</td>
<td>83.61 (0.51)</td>
<td>84.34 (0.26)</td>
<td>90.27 (0.17)</td>
</tr>
<tr>
<td>Diffusion</td>
<td>91.72 (0.16)</td>
<td>79.22 (0.22)</td>
<td>83.17 (0.16)</td>
</tr>
<tr>
<td>RNN-DT</td>
<td>95.37 (0.10)</td>
<td>48.22 (0.34)</td>
<td>88.89 (0.28)</td>
</tr>
<tr>
<td>CPNN</td>
<td>93.44 (0.27)</td>
<td>81.17 (0.34)</td>
<td>87.74 (0.15)</td>
</tr>
</tbody>
</table>

as can be seen in Table 6. However, for the Random Forest and LDA classifiers, our results indicate that some classifiers may not be adapted to the proposed descriptor. Indeed, the performance of the classifiers depends on several issues such as the nature of the features, tuning of the parameters, etc. Thus, in some cases, simple classifiers can obtain best performances than more complex classifiers [70].

Table 7 shows the processing time needed, on average, to compute a feature vector from a single dynamic texture for each method. In the tests, we computed the average of 20 executions of feature extraction of a single sample using a 3.60 GHz Intel(R) Core i7, 64GB RAM, and 64-bit Operating System. The proposed method took 3.07 s and 2.47 s to extract the feature vector from the UCLA and Dyntex++ database, respectively. The RNN-DT method took the lowest time, 1.35 s on the UCLA database and 1.83 s on the Dyntex++ database. Although the proposed method obtained the second-lowest time, the results demonstrate that our method is very competitive compared to the other approaches, taking a reasonable time to compute the features, and achieving high accuracies.
Table 7: Computational processing time in seconds of the proposed method and other methods to compute the feature vector.

<table>
<thead>
<tr>
<th>Descriptor</th>
<th>UCLA</th>
<th>DynTex++</th>
</tr>
</thead>
<tbody>
<tr>
<td>VLBP</td>
<td>3.67</td>
<td>2.60</td>
</tr>
<tr>
<td>DPSW</td>
<td>49.02</td>
<td>34.13</td>
</tr>
<tr>
<td>DPSWNet</td>
<td>8.97</td>
<td>48.78</td>
</tr>
<tr>
<td>CNDT</td>
<td>17.68</td>
<td>16.45</td>
</tr>
<tr>
<td>Diffusion</td>
<td>6.68</td>
<td>4.94</td>
</tr>
<tr>
<td>RNN-DT</td>
<td>1.35</td>
<td>1.83</td>
</tr>
<tr>
<td>Proposed Method</td>
<td>3.07</td>
<td>2.47</td>
</tr>
</tbody>
</table>

5.2. Dynamic Texture Segmentation

Here, we show how our CPNN descriptors (described in Section 3) can be used for real-time dynamic texture segmentation. Experiments were done with videos captured from a moving boat on the Guerlédan lake in Brittany. In order to apply our descriptor for dynamic texture segmentation, we consider an approach based on overlapping blocks. Figure 7 summarizes the approach for dynamic texture segmentation: firstly, the video is divided into overlapping blocks; for each block a feature vector is obtained using our dynamic texture descriptor; finally the feature vector is labeled using our classification approach.

- **Overlapping blocks**: the dynamic texture video of $w \times h \times T$ pixels is divided into overlapping blocks $B$ of size $p \times p \times q$ pixels (as can be seen in Figure 7(a)). The blocks are evenly sampled using steps of size $l$ between each block (horizontally, vertically and temporally). The border pixels are not considered when it is not possible to fit a block.

- **Feature extraction**: a feature vector is obtained for each block of the dynamic texture video using the proposed method for dynamic texture description in Section 3.

- **Labeling**: in this step, the blocks are labeled from their feature vector. For this, we use supervised classifiers. In this way, new blocks are predicted based on a classifier model trained using labeled blocks from other annotated videos. As we are using overlapping blocks, the pixels of the
video belong to several blocks, thus their final label is provided by majority voting.

The goal is to test our dynamic texture descriptor to segment the different semantic regions of the video: water, sky and land (forest). Figure 8(a) shows an illustration of the video used in the experiments. The main challenges of this type of video are the variability of textures due to perspective, the sun reflections, and of course, the motion of the boat. The video used in the training step has $288 \times 384 \times 1200$ pixels, while the video used in the testing step has $288 \times 384 \times 1621$ pixels. The labeled blocks used for the training of the classifier were obtained randomly from the different regions of the training video. In the experiments, we tested different sizes of step between blocks ($l = 5, 8, 11$) and a size of block equal to $p = q = 30$ pixels. We also used two classifiers (Support Vector Machine (SVM) and 1-Nearest Neighbors (1-NN)) and different numbers of labeled samples to train the classifiers.

In Figure 9 is shown the first frame of the videos segmented into three classes, with each color representing a class: blue (sky), red (land) and transparent (water). In this figure, in the first column was used 1000 samples for training the 1-NN classifier and in the second column 2000 samples. Figure 10 shows the segmented videos using the SVM classifier. As can be seen, the proposed
scheme can identify well the three different classes in all examples. However, the method has more difficulties to delimit the border regions. To improve this point, we intend to investigate new ways of applying or refine the scheme for segmentation.

The main advantages of our method are the computational simplicity and the fast processing time, which makes the approach promising for real-time segmentation task and active learning. In addition, the proposed approach uses a small number of samples for training, which is another interesting characteristic for problems with few data samples.

6. Conclusion

In this paper, we present a method for dynamic texture recognition called CPNN, which learns a representation from the graph-based features using randomized neural networks. This is achieved through the adoption of a Complex Network framework for modeling a video through directed graphs, which are able to efficiently model the appearance and motion characteristics of the dynamic texture. From this, graph-based features can be learned by the randomized
Figure 9: One frame of segmented video using our DT model with the KNN classifier, for different values of step parameter $l$, and different numbers of training samples (left: 1000, right: 2000).
Figure 10: Same as Figure 9 but using the SVM classifier.
neural network, which has a simple and fast learning algorithm, producing a representative feature vector through the trained weights of the output layer. Based on the experiments, we adopted in the proposed method six randomized neural networks of 1 hidden layer with 4, 24, and 29 number of hidden units and input layers of size 4 and 10 features.

We have tested the CPNN method on two benchmarks for the task of dynamic texture classification. The results lead to the conclusion that our method provides discriminative dynamic texture descriptors using a simple classifier. Also, experiments of computational processing time demonstrated a competitive performance of the proposed method compared to the others. Based on these findings and on some experiments in dynamic texture segmentation, the proposed method can be a valuable tool for real-time applications and could be investigated for active learning purposes. These observations motivate the future investigation of exploring the complex network frameworks and learning methods for dynamic texture analysis. As limitation and future work, new manners of creating the training set of the neural network can be explored, such as the use of clustering measures, hierarchical degree and joint-degree of the vertices. The interpretation of the physical meaning of the learned features by the proposed model should also be further investigated. Another research issue is to investigate other neural network architectures with more layers (e.g., D-RVFL) to learn the features, although computational time may increase. Furthermore, our method learns the features using a regression model from RNN. However, we believe that a classification model in which each output neuron represents one class can be employed to learn the features and improve the results.

Acknowledgments

Lucas C. Ribas gratefully acknowledges the financial support grant #2016/23763-8 and #2019/03277-0, São Paulo Research Foundation (FAPESP). Jarbas Joaci de Mesquita Sá Junior thanks CNPq (National Council for Scientific and Technological Development, Brazil) (Grant: 302183/2017-5) for the financial sup-
port of this work. O. M. Bruno acknowledges support from CNPq (Grant #307897/2018-4) and FAPESP (grant #2014/08026-1, 2018/22214-6 and 2016/18809-9). The authors wish to thank Thomas Simon and Clément Yver for allowing them to use their videos from Guerlédan’s lake.

References


