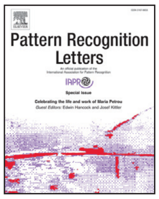
Pattern Recognition Letters 99 (2017) 48–56

Contents lists available at ScienceDirect

Pattern Recognition Letters

journal homepage: www.elsevier.com/locate/patrec



Active Incremental Recognition of Human Activities in a Streaming Context

Rocco De Rosa[a,*], Ilaria Gori[b], Fabio Cuzzolin[c], Nicolò Cesa-Bianchi[d]

[a] Rank Group, Data Science Lab, UK
[b] iCub Facility, Istituto Italiano di Tecnologia, Italy
[c] Department of Computing and Communication Technologies, Oxford Brookes University, UK
[d] Dipartimento di Informatica, Università degli Studi di Milano, Italy

ARTICLE INFO

Article history:
Available online 6 March 2017

Keywords:
Activity recognition
Streaming video
Active learning
Incremental learning
Constant Budget
Continuous action recognition
Non-parametric classification

ABSTRACT

Recognising human activities from streaming sources poses unique challenges to learning algorithms. Predictive models need to be scalable, incrementally trainable, and must remain bounded in size even when the data stream is arbitrarily long. In order to achieve high accuracy even in complex and dynamic environments methods should be also nonparametric, i.e., their structure should adapt in response to the incoming data. Furthermore, as tuning is problematic in a streaming setting, suitable approaches should be parameterless (as initially tuned parameter values may not prove optimal for future streams). Here, we present an approach to the recognition of human actions from streaming data which meets all these requirements by: (1) incrementally learning a model which adaptively covers the feature space with simple and local classifiers; (2) employing an active learning strategy to reduce annotation requests; (3) achieving good accuracy within a fixed model size. Although in this work we focus on human activity recognition, our approach is completely independent from the feature extraction and can deal with any supervised matrix (set of feature vectors). Hence, it can be adapted to a wide range of applications (e.g., speech recognition, image classification, object recognition, pose recognition, and image matching). Extensive experiments on standard benchmarks show that our approach is competitive with state-of-the-art non-incremental methods, while outperforming the existing active incremental baselines.

1. Introduction

The pervasive presence of cameras and mobile devices in our everyday lives has created a strong demand for automated methods able to analyse data streams in real time. This is especially challenging in the case of videos capturing human activities, as in TV footages and videos from surveillance cameras. Another natural application is human robot interaction, which requires the machine to learn and recognise human behavioural patterns in real time. Nevertheless, the mainstream approaches to action and activity recognition are typically based on an offline training phase (for a review of previous work in activity recognition we refer the reader to Sec. 2). Such a setting leads to several critical issues when dealing with streaming videos:

How to incrementally learn activities from the incoming data? The dynamic nature of the streaming video setting implies that, at each time instant, new data is made available to the system, which needs to incrementally learn from it. This implies both refining the current models of known human activities and adding on the fly new models of previously unseen activities.

How to minimise the required annotation effort? The issue of how many video fragments should be annotated is strictly related to the ability of learning new activities. While for newly observed activities one might assume that all video frames should be —at least initially— manually annotated, when analysing footage of known action classes only a fraction of the video input will likely bring in new information. In this context, the system should automatically select which video fragments are the most informative, and asks human annotators for help only in those cases.

How to optimise the algorithm heuristics dynamically? System components, such as the chosen feature representation and the learning algorithm parameters, have a crucial impact on the final performance of any framework. In a continuous learning setting, however, design choices and parameter tuning are —if possible at all— more difficult than in offline settings, as we just cannot anticipate what new activities the system will be asked to learn.

* Corresponding author:
E-mail addresses: rocco.derosa1982@gmail.com, Rocco.DeRosa@rankinteractive.com (R. De Rosa).

http://dx.doi.org/10.1016/j.patrec.2017.03.005
0167-8655/© 2017 Elsevier B.V. All rights reserved.

The main contribution of this paper is an approach for dealing with human activity recognition in a streaming context. To the best of our knowledge, our approach is the first one to address all the challenges listed above in a principled manner. Our starting point is a recently proposed local algorithm for classification of data streams [5] which is incrementally trainable and nonparametric (i.e., the model structure is not specified a priori, but determined by the data in such a way that the number of parameters is not fixed in advance), while exhibiting theoretical guarantees on its performance. Here we leverage on this result, and extend it to the active learning setting. This leads to a framework that meets all the above requirements: (1) it incrementally and efficiently learns the incoming data stream while being robust to the addition of new classes; (2) the active learning component evaluates the informative content of the incoming data items with respect to the current level of confidence, thus allowing to decide when the cost of manual annotation is worthwhile; (3) the nonparametric nature of the approach allows for fully data-driven learning.

Next, we illustrate the workflow of our approach in the specific case of activity recognition from streaming videos:

1. Each video is associated with a variable number of feature vectors in a given feature space.
2. The feature space gets sequentially covered with balls centered on samples selected from the stream.
3. Each ball is associated with an estimate of the conditional class probabilities obtained by collecting statistics around its centre; a new unlabeled sample is predicted using the estimate of the closest ball.
4. A sample falling outside its closest ball becomes the center of a new ball.
5. The radius of each ball is adjusted according to how well each ball predicts the class label of the new samples that fall close to it.
6. Ball centres are incrementally adjusted to fit the actual data distribution.
7. The set of balls is organized in a tree-like structure [17], so that the ball nearest to the current sample can be found in time logarithmic in the number of balls.

We call our algorithm Fast active Incremental Visual covERing (FIVER). Extensive experiments on several publicly available databases show that our approach outperforms all existing algorithms for activity recognition from streaming data. Furthermore, we show that by combining FIVER with the robust temporal segmentation algorithm presented in [7], we obtain a system able to deal, in a straightforward manner, with a realistic continuous active recognition scenario. A significant contribution of this work is the extension of [5] to an active learning setting. This is key to the practical application of incremental learning in streaming settings for at least two reasons. Firstly, active learning systems allow to substantially save on costly ground truth annotations. Secondly, the confidence score plays a crucial role in continuous activity recognition tasks in domains such as surveillance and human-robot interaction (see Sec. 4.5).

2. Related Work

Within the vast literature related to action recognition — see [25] and references therein— research focusing on the streaming setting has gained momentum only recently [8]. Desirable features in this context are: *(1) Incremental Updating:* typically, a large amount of data is sequentially presented in a stream, and so it is desirable for algorithms to incrementally update the model rather than re-training it from scratch. *(2) Incremental Learning of New Classes (activities):* algorithms should be able to accommodate on the fly any new class. *(3) Bounded Size Models:* as the data stream may be very large, models should keep a bounded memory footprint, allowing for real-time prediction while avoiding storage issues. This implies the ability of discarding useless or old data, and is critical to the tracking of *drifting concepts* (i.e., settings where the optimal decision surface changes over time, requiring repeated adjustments of the model [28]). *(4) Data-Driven behaviour:* because parameter tuning is problematic in streaming settings, systems with few or no parameters are preferable. *(5) Nonparametric behaviour:* since the true structure of the data is progressively revealed as more examples from the stream are observed, nonparametric algorithms [10], which are not committed to any specific family of decision surfaces, are preferable. *(6) Active Learning:* in a streaming setting, the system needs to learn from each incoming data point. However, training labels are provided by human annotators, who should be invoked only when the system has low confidence in its own prediction for the current label. *(7) Bounded Request rate:* since querying human annotators is expensive, any practical active learning system for streaming settings should impose a bound on the query rate.

Table 1 lists the previous efforts in human activity recognition involving incremental and/or active learning components which, due to their features, are the closest alternatives to our approach.

A feature tree-based incremental recognition approach was proposed in [26], where the tree is free to grow without bounds as more examples are fed to the learner. As this requires to store all the presented instances, the method is infeasible for continuous recognition from streaming videos, where the number of activities can get very large over time. A human tracking-based incremental activity learning framework was proposed in [23] which, however, requires annotation on the location of the human body in the initial frame, heavily restricting its applicability. For these reasons [26] and [23] are not listed in Table 1. Our work shares similarities with the incremental algorithm in [3] upon which, to some extent, we build our proposal. Both methods adopt a nonparametric, incremental ball covering of the feature space strategy. FIVER, however, brings to the table crucial new features that makes it uniquely suitable for dealing with streaming data. Firstly, it does not rely on any input parameters, which are inconvenient to tune in streaming settings. Secondly, it limits the model size, thus allowing the tracking of drifting concepts. More precisely, when the number of allocated balls exceeds a given budget, FIVER discards each ball with a probability proportional to its error rate. Thirdly, it dynamically adjusts the ball centres, thus yielding very compact models while improving performance. The resulting covering resembles a visual dictionary, learned incrementally and directly usable for predictions, where the balls play the role of visual codewords. Finally, the active learning module defines the interaction between the learning system and the labeler agent, limiting the number of annotations requested.

The use of incremental active learning for activity recognition tasks was recently investigated in [11,12], where an ensemble of linear SVM classifiers is incrementally created in a sequence of mini-batch learning phases. These methods, however, are not designed to operate on individual data elements in streams, as it is required in our setting. A confidence measure over the SVM outputs is defined, where each individual classifier output is weighted by the training error. Two user-defined thresholds control the query rate of labeled videos. Non-confident instances, which are close to a class boundary, are forwarded to the annotator, while the others are discarded. Note that the set of ensemble classifiers can become very large, as an arbitrary number of SVMs can be added in each batch phase. Furthermore, the method requires model initialisation, and several parameters need to be tuned at validation time, thus making the approach unsuitable to a truly streaming context. The method in [11] initially learns features in an unsupervised manner using a deep neural network. Then, a multinomial

Table 1
Features required or desirable in human action recognition from streaming data. To the best of our knowledge, FIVER is the only algorithm that addresses all these challenges. *LEGEND*: √: exhibits the feature; ≈: partially exhibits the feature; ⊗: does not possess the feature.

STREAMING CONTEXT REQS.	Incr. Upd.	New Cls.	Bound. Size	Data Driv.	Nonpar.	Active	Bound. Rate
De Rosa et al. [3]	√	√	⊗	≈ (one param.)	√	⊗	⊗
Hasan and Roy-Chowdhury [11]	≈ (minibatch)	⊗	√	⊗	⊗	√	⊗
Hasan and Roy-Chowdhury [12]	≈ (minibatch)	⊗	⊗	⊗	⊗	√	⊗
Hasan and Roy-Chowdhury [13]	≈ (minibatch)	⊗	⊗	⊗	⊗	√	⊗
FIVER	√	√	√	√	√	√	√

logistic regression classifier is learned incrementally. The posterior class probability output is used —as in [12]— to select what videos need supervised information. This method also depends on several parameters, cannot deal with new classes, and requires initialisation. The same authors have recently presented in [13] a further extension which attempts to mine information from scene context. However, the core learning system suffers from the same drawbacks discussed above. Moreover, the active module used there cannot explicitly limit the query rate, a crucial feature for real-world applications where the cost of human annotation has to be controlled.

3. Fast Active Incremental Visual Covering

This section describes the FIVER algorithm at the heart of our framework. Although in this work we mainly focus on the problem of activity recognition from streaming videos, our framework is general-purpose, and can accept any type of input data (i.e, videos, sounds, depth maps, and so on).

The rest of the section is structured as follows: Sec. 3.1 describes our incremental visual covering approach based on [3], Sec. 3.2 shows how to keep the memory footprint bounded via a technique introduced in [5]. Sec. 3.3 introduces a mechanism that performs active learning on the input stream, while Sec. 3.4 summarizes the FIVER algorithm.

3.1. (Passive) Incremental Learning

We assume that the learner is trained on a stream of (pre-segmented) labeled videos $(\boldsymbol{V}_1, y_1), (\boldsymbol{V}_2, y_2), \ldots$. Each video $\boldsymbol{V}_i$ is associated with a set of T_i local descriptors $\{\boldsymbol{x}_t^{(i)}\}_{t=1}^{T_i}$, where each descriptor $\boldsymbol{x}_t^{(i)} \in \mathbb{R}^d$ belongs to a d-dimensional feature space. Each video label y_i denotes an activity class that belongs to a set $\mathcal{Y} = \{1, \ldots, C\}$ of possible classes. Notably, this set may change over time. The classifier is trained incrementally: every time a new labeled video is acquired, the current model is adjusted. In the following, we drop the superscript i and re-index the local features, assuming that the learner is fed a sequence $(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots$ of labeled local feature examples, where $(\boldsymbol{x}_t, y_t) = (\boldsymbol{x}_t^{(i)}, y_i) \in \mathbb{R}^d \times \mathcal{Y}$ for some t and $1 \le t < T_i$. The feature space is then adaptively covered by a set $\mathcal{S}$ of balls. The number of balls is dependent on the complexity of the classification problem. Unlike [3], where the balls were always centered on input samples, we extend the AUTO-ADJ version of ABACOC described in [5]. In particular, a K-means-like update step makes the centre of each ball shift towards the average of the training samples that were correctly predicted by the local classifier; in other words, the balls track the feature clusters. Similarly to [5], we initialize the radius of any new ball as the distance from the closest ball. These settings result in a parameter-less algorithm. Unlike [3], we set the feature space intrinsic dimension parameter to 2, as we empirically found that it does not affect performance significantly. A low value of intrinsic dimension corresponds to a "flat" data manifold when viewed in feature space. As this flatness is an intrinsic property of the data, we expect it

Algorithm 1 ABACOC ([5], adapted).

```
Input: feature space metric $\rho$
 1: Ball centres $\mathcal{S} = \emptyset$ and set of labels $\mathcal{Y} = \emptyset$ initialised
 2: for $i = 1, 2, \ldots$ do
 3:   Input labeled video $(\boldsymbol{V}_i, y_i)$
 4:   if $y_i \notin \mathcal{Y}$ then
 5:     Set $\mathcal{Y} = \mathcal{Y} \cup \{y_i\}$ // add class on the fly
 6:   end if
 7:   for $t = 1, \ldots, T_i$ do
 8:     if $\mathcal{S} \equiv \emptyset$ then
 9:       if $t = 1$ then
10:         save $\boldsymbol{x}_1$;
11:       else
12:         $\mathcal{S} = \{\boldsymbol{x}_2\}$, set $\varepsilon_2 = R_2 = \rho(\boldsymbol{x}_1, \boldsymbol{x}_2)$
13:         Use $y_i$ to initialize estimates $p_2$
14:       end if
15:     else
16:       Let $\boldsymbol{c}_s \in \mathcal{S}$ be the nearest neighbour of $\boldsymbol{x}_t$ in $\mathcal{S}$
17:       if $\rho(\boldsymbol{c}_s, \boldsymbol{x}_t) \le \varepsilon_s$ ($\boldsymbol{x}_t$ belongs to ball at $\boldsymbol{c}_s$) then
18:         if $y_i \neq \operatorname*{argmax}_{y \in \mathcal{Y}} p_s(y)$ then
19:           Set $m_s = m_s + 1$, $\varepsilon_s = R_s m_s^{-1/4}$ // shrink on errs
20:         else
21:           Set $\Delta = \boldsymbol{x}_t - \boldsymbol{c}_s$, $n_s = n_s + 1$
22:           Set $\boldsymbol{c}_s = \boldsymbol{c}_s + \Delta / n_s$
23:         end if
24:         Use $y_i$ to update $p_s$
25:       else
26:         $\mathcal{S} = \mathcal{S} \cup \{\boldsymbol{x}_t\}$, $\varepsilon_t = R_t = \rho(\boldsymbol{c}_s, \boldsymbol{x}_t)$, $n_t = 1$, $m_t = 0$
27:         use $y_i$ to initialize estimates $p_t$
28:       end if
29:     end if
30:   end for
31: end for
```

to remain valid, at least to some extent, even when more complex features are added.

Incremental updates. The sequence of observed training examples $(\boldsymbol{x}_t, y_t)$ is used to build a set $\mathcal{S}$ of balls that cover the region of the feature space they span. For each ball, we store an empirical distribution of the predicted classes. Furthermore, for each ball centre $\boldsymbol{c}_s \in \mathcal{S}$ we keep updated the number $n_s(y)$ of data points $\boldsymbol{x}_t$ of each class $y \in \mathcal{Y}$ that at time t belong to the ball. These counts are used to compute the class probability estimates (activity scores) for each ball centre $\boldsymbol{c}_s \in \mathcal{S}$ as $p_s(y) = \frac{n_s(y)}{n_s}$ for all $y \in \mathcal{Y}$, where $n_s = n_s(1) + \cdots + n_s(C)$. This estimate could be unstable, but using stabilizers such as the Laplace correction did not affect the results in practice.

The training algorithm operates as follows (see Alg. 1): initially, the set of balls $\mathcal{S}$ is empty. For each training example $\boldsymbol{x}_t$, we efficiently[1] compute its nearest ball centre $\boldsymbol{c}_{\pi(t)} \in \mathcal{S}$, where $\pi(t)$ de-

[1] For example, [17] embed $\mathcal{S}$ in a tree structure in which nearest neighbour queries and updates can be performed in time $\mathcal{O}(\ln |\mathcal{S}|)$ —see also [16].

notes the index s of ball $\boldsymbol{c}_s \in \mathcal{S}$ nearest to the input sample $\boldsymbol{x}_t$. If $\boldsymbol{x}_t$ does not belong to its closest ball, i.e., the distance $\rho(\boldsymbol{c}_{\pi(t)}, \boldsymbol{x}_t)$ between $\boldsymbol{x}_t$ and $\boldsymbol{c}_{\pi(t)}$ is greater than the ball's radius $\varepsilon_{\pi(t)}$, a new ball with centre $\boldsymbol{x}_t$ and initial radius R_t equal to $\rho(\boldsymbol{c}_{\pi(t)}, \boldsymbol{x}_t)$ is created and added to $\mathcal{S}$. The label y_t is used to initialise the empirical class distribution for the new ball. If $\boldsymbol{x}_t$ belongs to the nearest ball, its label y_t is used to update the error count $m_{\pi(t)}$ for that ball. The local classifier centred at $\boldsymbol{c}_{\pi(t)}$ makes a mistake on $(\boldsymbol{x}_t, y_t)$ if and only if $y_t \neq \mathrm{argmax}_{y \in \mathcal{Y}}\, p_{\pi(t)}(y)$. Whenever this happens, the radius is set back to its initial value $R_{\pi(t)}$, scaled by a polynomial function that depends on the current error count: $\varepsilon_{\pi(t)} = R_{\pi(t)}\, m_{\pi(t)}^{-1/4}$. This formula comes from the decay function defined in [3] with intrinsic dimension equal to 2. By doing that, the algorithm makes room for new balls in the feature space, allowing for a finer description of the classification function in the zones that are difficult to model.

If the prediction is correct, the ball centre is set to the average of the correctly classified instances within the ball, so that the centre moves towards the majority class centroid. Finally, the class probability estimates $p_{\pi(t)}(y)$ for the local classifier centred in $\boldsymbol{c}_{\pi(t)}$ are updated. Notably, a-priori knowledge of the full set of classes $\mathcal{Y}$ is not needed, as our incremental learning approach can add new labels to $\mathcal{Y}$ on the fly, as soon as they first appear in the stream.

Prediction. In the prediction phase, we proceed in a similar fashion: for each $\boldsymbol{x}_t$ associated with an unlabelled video $\boldsymbol{V}_i$, its nearest neighbour $\boldsymbol{c}_{\pi(t)} \in \mathcal{S}$ is efficiently computed. Then, assuming that the local features are i.i.d., the label of the test video $\boldsymbol{V}_i$ is predicted using the following maximum likelihood estimate

$$\widehat{y}_i = \underset{y \in \mathcal{Y}}{\mathrm{argmax}} \prod_{t=1}^{T_i} p_{\pi(t)}(y) \tag{1}$$

which integrates over all the local features of the video. In our experiments we used the log-likelihood in order to avoid numerical problems (note that, as the logarithm is a monotonically increasing function, replacing probabilities with log-probabilities in (1) does not change the argmax).

3.2. Constant Model Size

In order to curb the system's memory footprint, we adopt the simple approach proposed in [5], which is based on deleting existing balls whenever a given budget parameter on the label query rate is attained. This is crucial for real-time applications, as NN search, used in both training and prediction, takes time logarithmic in the number of balls. The probability of deleting any given ball is proportional to the number of mistakes made so far by the associated classifier. In fact, if a ball is making many mistakes, its class probability estimate should not be used for computing class scores during the prediction phase. So, if the budget is reached and a new ball has to be added, an existing ball s is deleted with probability $\mathbb{P}_{\mathrm{disc}}(s) = \frac{m_s+1}{\sum_{r \in \mathcal{S}} m_r + |\mathcal{S}|}$, where m_s is the number of mistakes made by ball $s \in \mathcal{S}$.

This also helps addressing concept drifts: ball classifiers that accumulate a large number of mistakes are removed to make room for a more up-to-date description of the data.

3.3. Streaming Active Learning

We now introduce an active learning system for streaming settings, which bounds the rate of queries to human annotators. The technique we propose is inspired from [32]. Whenever a new segmented video is presented to the model, the system makes a prediction and then invokes the active learning module in order to determine whether the label should be requested to the human annotator. In particular, if the confidence of the prediction is below a certain threshold —i.e., the prediction is ambiguous— then a query is issued to the annotator unless the query rate budget is violated. When the label is not requested, the model is not updated. Instead of selecting a fixed confidence threshold on the query instances, we use the so-called Variable Uncertainty Strategy [32] (`VarUnStr`), which queries the least certain instances within a time interval.

Measuring prediction confidence. Eq. (1) shows that class estimates $p_{\pi(t)}(y)$, associated with ball centres near the current input instance, should be considered more reliable than those associated with faraway centres, as the corresponding region of the feature space has already been explored —see Fig. 3-right. We thus adapt the RBF kernel [2] to scale ball estimates based on their distance from the input examples: $w_t = \exp(-\frac{\rho(\boldsymbol{x}_t, \boldsymbol{c}_{\pi(t)})^2}{2\epsilon_{\pi(t)}^2})$, where the variance is set to the current ball radius $\epsilon_{\pi(t)}$. This confidence function is quite simple, and we are aware that more complex measures do exist. However, we view it as a plus that we manage to have good performance using a simple method, as we show in the experimental section. The local bandwidths of the RBF kernel are directly related to the complexity of the problem in each neighborhood of the feature space. Indeed, in more difficult areas (small radii) the kernel penalizes distant samples from the current ball covering, when compared to less troublesome ones (big radii).

Given a test video $\boldsymbol{V}_i$, we thus define a confidence measure $C_i(y)$ on the estimate of the expected class conditional probability for any given class y as:

$$C_i(y) = \frac{1}{T_i} \sum_{t=1}^{T_i} w_t \log p_{\pi(t)}(y) \qquad \forall y \in \mathcal{Y}\ .$$

Updating the confidence threshold. The `VarUnStr` strategy of [32] continuously updates the confidence threshold Θ, which triggers requests for new labels (see Alg. 2). If the prediction confidence is below the current threshold Θ over the duration of the last observed video, Θ is decreased by a fraction τ in order to query the most uncertain instances first. Otherwise, the threshold is increased to avoid interruptions of the learning process when the algorithm is not asking for labels. As explained in [32], the parameter τ can be set to a default value of 0.01. In the experimental section we follow this suggestion thus keeping our algorithm fully parameterless.

Further research should be devoted to investigating whether the confidence measure, coupled with our nonparametric model, allows to discover new classes automatically at prediction time. For instance, the confidence for input video frames that fall far from those that make up the learned model vanishes. These outlier frames may thus be associated with new activities —e.g., see the low confidence input examples in Fig. 3 (right). The resulting system can be viewed as performing a form of semi-supervised clustering. A simple approach in this direction is proposed in [4].

Algorithm 2 Variable Uncertainty Strategy.

```
Input: video $\boldsymbol{V}_i$, model, threshold $\tau \in (0, 1]$ (default is 0.01)
Output: labeling $\in \{true, false\}$
1: Init: $\Theta = 1$ and store the latest value during operation
2: Calculate the confidence of the majority class $C_i(\hat{y}_i)$
3: if $C_i(\hat{y}_i) < \Theta$ then
4:    decrease the confidence threshold $\Theta = (1 - \tau)\Theta$
5:    return true
6: else
7:    increase the confidence threshold $\Theta = (1 + \tau)\Theta$
8:    return false
9: end if
```

Algorithm 3 FIVER.

```
Input: annotation budget $B$, maximum number of balls $M$, video
    stream $(\boldsymbol{V}_1, y_1), (\boldsymbol{V}_2, y_2), \ldots$
1: for $i = 1, 2, \ldots$ do
2:   Receive video $\boldsymbol{V}_i$
3:   Predict $\hat{y}_i$ (Eq. 1)
4:   if query rate $\leq$ budget $B$ then
5:     if Query Strategy (Alg. 2) returns true then
6:       Request true label $y_i$ and update query rate
7:       Use $(\boldsymbol{V}_i, y_i)$ to update model (Alg. 1)
8:       if $|S| > M$ (memory exceeded) then
9:         Discard one ball (see Sec. 3.2)
10:      end if
11:    end if
12:  end if
13: end for
```

3.4. FIVER Algorithm

The FIVER algorithm (see Alg. 3) combines all the elements described above. Namely, FIVER trains the model over the video stream via Alg. 1, while controlling the memory footprint as described in Sec. 3.2. Alg. 2 is the active learning module, which asks only for the most informative instances while not exceeding the budget rate.

Ignoring the cost of extracting features (see Sec. 4.1 for a discussion on this issue), the prediction and update time of FIVER is dominated by the NN search, whose cost is small (i.e., logarithmic in the number of balls). The space requirement, instead, is clearly linear in the same quantity. However, the experiments we report reveal that a good classification accuracy can be achieved using a number of balls which is quite a small fraction of the training set size. This implies that, in practice, FIVER really runs in real time.

4. Experiments

To emphasize the versatility of our approach, we tested FIVER in both batch and streaming learning settings.

In the batch setting, we followed the standard evaluation protocol for each dataset: specific train-test splits or K-fold cross-validation with specific values of K —see below for details. We then compared FIVER's results to those of competing incremental and offline methods.

In the streaming setting, instead, we assessed different variants of FIVER using the online accuracy —or sequential risk [9]— as evaluation measure. This measure captures the average error made by the sequence of incrementally learned models in a procedure where we first predict the test item on the current model, and then use the result to adjust the model itself.

Note that the streaming setting used in our experiments is very strict: we do not use seed training sets, mini-batch training, cross-validation sets, or assume any preliminary knowledge on the number of classes. As the other incremental methods rely on much richer sources of information than those allowed in our streaming setting, we could only evaluate them in the batch and mini-batch setting.

Finally, since FIVER is oblivious to the actual number of classes in the training data, in all our experiments the algorithm is learning the classes truly on the fly.

4.1. Datasets and Feature Extraction

We assessed our method on the following datasets: KTH [27] (all scenarios), UCF11 [25] and VIRAT [24] for action recognition, SKIG [21] and MSRGesture3D [31] for gesture recognition, JAPVOW [22] and AUSLAN [15] for sign language recognition (UCI Repository [19]). Table 2 shows the number of videos and classes of our benchmark.

Table 2
Benchmark.

dataset	Instances	Number of Classes
KTH	600	6
UCF11	1160	11
VIRAT	1545	11
SKIG	1080	10
MSRG3D	336	12
JAPVOW	370(test)/270(train)	9
AUSLAN	1865(test)/600(train)	95

The first five datasets contain mostly footage material: we decided to extract efficient local features at frame level in order to focus on truly real-time prediction. In particular, from KTH, UCF11, and VIRAT sequences we computed improved dense trajectories [30], due to their outstanding performance in action recognition tasks. For each video, three types of features were extracted, namely Histogram of Oriented Gradient (HOG), Histogram of Optical Flow (HOF) and Motion Boundary Histogram (MBH). We ran the code published on the INRIA Website[2], keeping all the default parameters except for the trajectory length, set to 8 frames, and the number of descriptor bins (16 BINs for HOG, MBHx and MBHy, and 18 BINs for HOF). Every 8 frames we obtained a variable number of active trajectories. We then accumulated all the trajectories for each descriptor, and concatenated all the descriptors, obtaining a collection of vectors of 66 dimensions for each video. In this setting, each vector is a summary of the three local descriptors extracted from each video frame. For VIRAT, we initialised the improved trajectory algorithm using the bounding boxes released along with the dataset. For the other datasets, we did not rely on any initialisation.

From SKIG we extracted the same information as [7], which consists of 3DHOF on the RBG frames and GHOG (Global Histogram of Oriented Gradient) on the depth frames. In MSRGesture3D only depth information is available, and we extracted two-level pyramidal HOG (PHOG) features using 32 bins. For all the experiments, we used the Euclidean distance as the metric ρ (see Alg. 1), since tests using an L_p norm with varying p did not show any significant improvement on these datasets.

4.2. Comparison with competitors in a batch setting

We ran a first set of experiments in a batch setting, i.e., running FIVER on a random permutation of the given training set and then applying the resulting classifier on the test set. Here, FIVER is evaluated without the active module.[3] We compared FIVER against incremental and batch approaches:

Incremental algorithms: [3,11,12], which follow an incremental learning approach similar to ours.

Batch algorithms: [1,14,20,21,29,31], which have unrestricted access to training data for learning, as opposed to incremental methods that can access the data only sequentially. Note that the performance of incremental algorithms is typically poorer than that obtained using the corresponding batch versions [6].

We also tested these features with linear SVM and 1-NN with Dynamic Time Warping, two standard approaches generally used for these kinds of problems. As these two methods are highly inef-

[2] lear.inrialpes.fr/people/wang/improved_trajectories.
[3] Base code from mloss.org/software/view/560/.

Table 3
Comparison among baselines using the same features. The methods associated to bold numbers clearly outperform the others.

dataset	FIVER	Best Batch Method	Best Incremental Method	SVM	NN-DTW
KTH	**98.50%**	95.00% [29]	97.00% [12]	96.70%	78.30%
UCF11	**79.36%**	76.10% [20]	66.00% [12]	n.d.	n.d.
VIRAT	**57.20%**	55.40% [14]	54.20% [11]	n.d.	n.d.
SKIG	**98.30%**	88.70% [21]	97.50% [3]	94.50%	95.74%
MSRG3D	**91.25%**	86.50% [31]	90.33% [3]	**91.85%**	50.65%
JAPVOW	96.75%	95.67% [1]	**98.01%** [3]	84.59%	69.72%
AUSLAN	72.60%	**83.81%** [1]	72.32% [3]	44.78%	82.58%

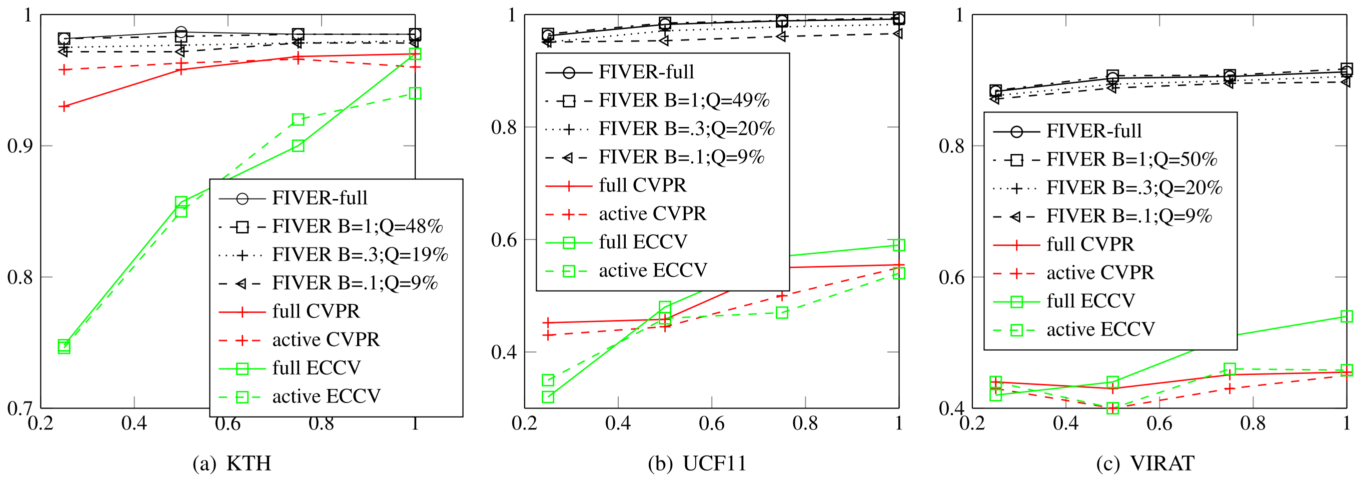


Fig. 1. Evaluation performance for the complete incremental system. The x-axis represents the percentage of the train fold used during the test in each of the four experiments (25%, 50%, 75% and 100% of the fold samples used for training). The y-axis indicates the final accuracy of the 5-fold cross validation. We show the final percentage of requested samples by the active modules only for the last experiment (100%), where the whole fold was fed to the learning module.

ficient, we did not calculate the performance on the larger UCF11 and VIRAT datasets.

We used 5-fold cross-validation averaged on ten runs for KTH, UCF11 and VIRAT, as the incremental competitors described in Sec. 2. For SKIG and MSRGesture3D we carried out a 3-fold cross-validation as in [31]. The available training and test sets were used for JAPVOW and AUSLAN. In Table 3 we reported the results that have been published in the referenced papers and the performance of our approach. Table 3 shows that FIVER is among the best methods on all the datasets, no matter whether the batch or incremental setting is considered. This demonstrates that our algorithm, combined with state-of-the-art features, provides an accurate and efficient classification system across the board.

4.3. Comparison with active incremental baselines

In Fig. 1 we show the experimental results on the same complete incremental setting as described in [11,12]. In this setting, a 5-fold cross validation is performed, and the algorithms learn incrementally (or in mini-batches, as the competitors do) on the training folds only on the samples requested by the active learning modules. The aim of these experiments is to show how to save annotation costs while keeping good prediction performance. Differently from [11,12], we can control the maximum percentage of requested sample (budget) on the stream. We run our method with three different budget rates $B = \{1, .3, .1\}$, which we show in the plots. The competitors reported the final percentage of requested labels only for KTH, which is around 15%, and UCF11, which is around 25%. In Fig. 1 the values Qs in the legend show the real percentage of samples requested by our methods. We remark that the budget rate parameter is just an upper bound on the possibly requested labels. We show in the figure the values reported in [11,12].

We did not run any experiment using our competitors' algorithms, we simply report the results that have been made public in the referenced works. Both the competitors that we denote as *ECCV* [11] and *CVPR* [12] (full and active version) are clearly outperformed on the accuracy and on the percentage of requested samples by FIVER.

4.4. FIVER performance on different scenarios

In a pure streaming setting the data come in sequentially, and the number of activities depicted in each video is not known a priori. In order to show the importance of all the modules proposed in Sec. 3 for dealing with the hard constraints of such a streaming context – see Sec. 2 – we conducted extensive testing exploring the following scenarios, which correspond to different variants of FIVER:

Full. This is the least realistic case, where we assume that all the incoming instances are manually annotated, there are no memory requirements, and we use each incoming training sample to incrementally update the model.

VarUn. In this case, we use the active learning component described in Alg. 3, including the `VarUnStr` strategy described in Alg. 2, to decide what instances require manual annotation. The query rate, calculated as the fraction of videos for which a label was requested among those observed so far [32], is upper bounded by an input budget parameter $B \in (0, 1]$.

Rnd. The Random strategy queries the labels of incoming instances with probability equal to the query rate budget B. In this case the algorithm does not use any information to decide which samples are important to improve the performance.

VarUnFix. In this very realistic scenario we make the additional assumption that limited memory is available to store the labeled training instances. We apply both the active module and

Algorithm 4 Active Stream Validation Protocol.

```
Input: labeling budget $B$, Active Module, video stream $(\boldsymbol{V}_1, y_1), (\boldsymbol{V}_2, y_2), \ldots$
 1: Initialize online accuracy $A_0 = 0$
 2: for $i = 1, 2, \ldots$ do
 3:   Receive video $\boldsymbol{V}_i$
 4:   Predict $\hat{y}_i$
 5:   Update $A_i = \left(1 - \frac{1}{i}\right)A_{i-1} + \frac{1}{i}\mathbb{I}\{\hat{y}_i = y_i\}$
 6:   if FULL setting then
 7:     Receive true label $y_i$
 8:     Update model using new example $(\boldsymbol{V}_i, y_i)$
 9:   else
10:     // ACTIVE setting
11:     if budget $B$ not exceeded then
12:       if ActiveModule($\boldsymbol{x}_i$,model) then
13:         Request true label $y_i$
14:         Update query rate (fraction of input samples)
15:         Update model using new example $(\boldsymbol{V}_i, y_i)$
16:       end if
17:     end if
18:   end if
19: end for
```

the method of Sec. 3.2 to limit the number of balls stored in the model. In all our tests we set the model size to 5000 instances. Note that in BoW methods, in opposition, a large amount of codewords are generally necessary to successfully predict video labels – see for instance [30], where the authors use four different visual dictionaries of 100,000 words (one for each local descriptor).

Since the method in [5] is natively incremental, it does not have a batch counterpart that we can compare against in the experiments. We performed ten random permutations of the videos in each dataset. The algorithm had to predict the label of each new incoming video —see Alg. 4. After each prediction, if the active learning system requested the true label, the video along with its label were fed to the model as a new training example. We ran all the competing algorithms with the same range $B \in \{.05, .1, .15, .2, .25, .3, .35, .4, .45, .5, .75, 1\}$ of budget values, and plotted the resulting online accuracy, averaged over ten different streams, against the average query rate. Importantly, the budget is only an upper limit to the actual query rate: algorithms generally ask for a smaller number of annotations.

Note that FIVER does not need any validation set as it has no parameters to tune. This is very important in the streaming context, where non-adaptive methods which tune their parameters in an initial validation stage may perform suboptimally on unseen data. Plots *a* to *g* in Fig. 2 illustrate the recorded performance on the various benchmarks for all the presented scenarios. The figure shows that *VarUn* performs as well as *Full* on most datasets, even though it queries only around 50% of all the labels. On KTH, for example, *VarUn* achieves 90% online accuracy while accessing only less than 20% of the labels. The *Rnd* method performs typically worse and needs all the labels to reach the performance of *Full*. *VarUnFix* works almost as well as *VarUn* on the simplest datasets and slightly worse on the complex ones; this is due to the fixed budget control that has to discard information in order to keep the model size fixed. For example, both *VarUn* and *Rnd* use 4% of the input data around the 50% query rate for UCF11, whereas *VarUnFix* use only 0.2% of the data – this is shown in the red and green boxes in Fig. 2 as final percentage of input examples used as model centres. Therefore, *VarUnFix* is extremely good at compressing the data, and allows for efficient computation at the cost of limited performance degradation. It is worth stressing that the *Rnd* setting can be compared only against the *VarUn* setting as they can freely grow the ball covering, as opposed to *VarUnFix* which can use only a fixed amount of support centroids.

4.5. Active Continuous Activity Recognition

Although in Sec. 3 we assumed that the incoming videos V_i are pre-segmented, whenever feature vectors $\{\boldsymbol{x}_t^{(i)}\}_{t=1}^{T_i}$ are extracted on a frame-by-frame basis, we can exploit the activity scores (1) computed over a short temporal window to perform automated temporal segmentation. This segmentation procedure is based on the evolution of class probabilities over time [3] (Fig. 3, left), where transitions between action instances can be associated with local minima of the standard deviation of class scores (pink curve) over the temporal window (Fig. 3, middle-top). In addition, unlike what was done in [3], we use here the confidence measure $C_i(y)$ to discard or send to supervision any detected activity with confidence below a certain threshold (Fig. 3, middle-bottom), as discussed in Sec. 3.3. This is crucial in applications such as human-robot interaction, where it is preferable for the robot not to perform any action when prediction confidence is low, as this may lead to safety issues or communication errors. We tested this active approach to temporal segmentation on the same dataset of ten manipulative actions used in [3]. Each action was recorded 60 times in two different illumination settings and backgrounds, and 3DHOF and HOG descriptors were extracted for each frame. We excluded four out of ten gestures from the learning phase, and evaluated our algorithm on sequences representing pick and place activities formed by grasping, moving and releasing actions. The system was evaluated on its ability to predict the correct class when a known gesture was performed, and to request supervision when an unknown gesture was observed. To compare the estimated class sequence with the ground truth we employed the Levenshtein distance [18]: $\frac{S+D+I}{N}$. In this case, each action is treated as a symbol in a sequence: S represents the number of substitutions (misclassifications), D the number of deletions (false negatives) and I the number of insertions (false positives). Over 20 test sequences, we achieved a Levenshtein distance error of 0.14, compared to the 0.36 reported in [3].

5. Conclusion and future work

We defined a truly streaming context for human activity recognition. We presented an incremental active recognition framework, well suited for streaming recognition problems, especially when the amount of data to process is large. Our approach is simple and exhibits a number of desirable features: it deals with sets of local descriptors extracted from videos, it learns in an incremental fashion, it embeds an active learning module, it is capable of learning new classes on the fly, it limits memory usage, and it predicts new data in real-time. In addition, the method is nonparametric and does not require expensive validation sessions for training, as it has no parameters to be tuned. Results demonstrate its competitiveness in terms of accuracy with respect to traditional batch approaches, as well as promising performance in a truly streaming scenario. The main two time consuming procedures are: 1) the feature extraction and 2) the NN search for the ball centroids. As previously mentioned, our method is independent from the feature extractor, that could be chosen based on the final application. In our experiments, we used dense trajectories, which is a real time feature extractor. On the other hand, the NN search could be easily parallelized using modern Big Data technologies (such as SPARK, Mahout, etc.). There are several possible directions for improvement: defining more complex local learners, learning a specific metric [4] (e.g for high-dimension features as for CNN-feature extractors), deriving a more sophisticated confidence measure, using an ensemble of our base learner (like Random Forest for Deci-

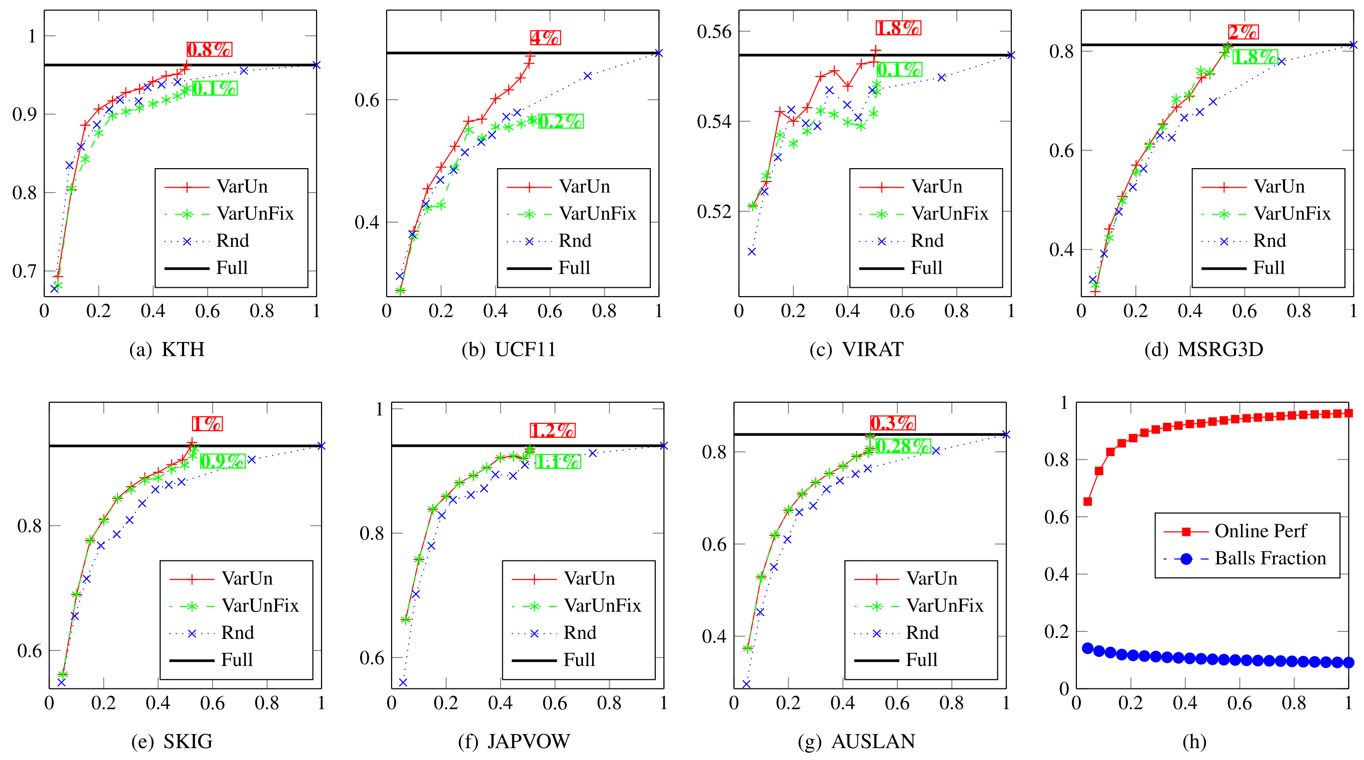


Fig. 2. Plots (a) to (g) show the active online performance of the Full, Rnd, VarUn and VarUnFix variants of FIVER on different benchmarks. The x-axis is the percentage of label requested by the active learning module, while the y-axis is the average online accuracy over ten random permutations of the videos. In the coloured boxes, we show the percentage of the input data selected as centres by *VarUn*(red) and *VarUnFix*(green) with budget $B = 1$. Plot *h* represents the evolution of accuracy and model size over the sequentially fed videos on the KTH dataset. The blue (circle) curve shows the fraction of the input data selected as centres, and the red (square) curve the online accuracy. Notably, the fraction of centers added diminishes over time as the accuracy improves. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

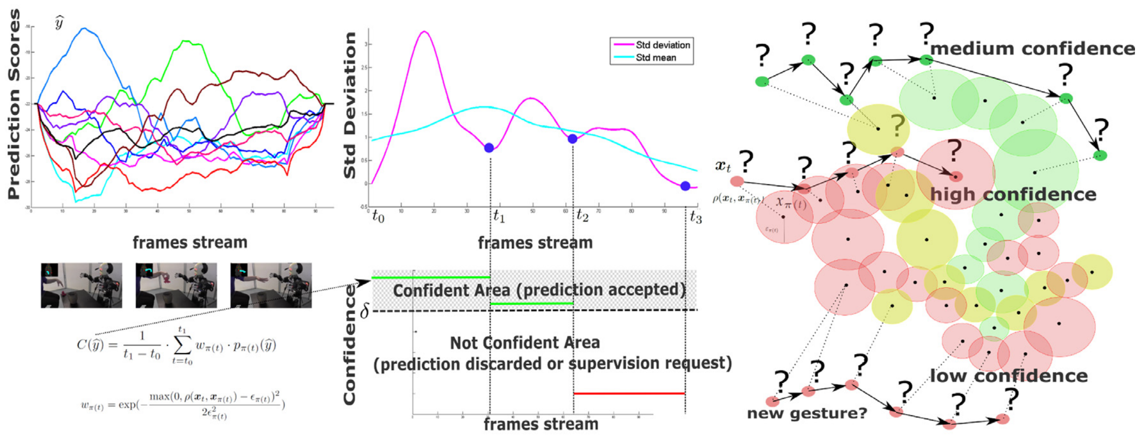

Prediction Scores
$\hat{y}$
frames stream
Std Deviation
Std deviation
Std mean
t_0
t_1
t_2
t_3
frames stream
Confidence
Confident Area (prediction accepted)
δ
Not Confident Area
(prediction discarded or supervision request)
frames stream
$C(\hat{y}) = \frac{1}{t_1 - t_0} \cdot \sum_{t=t_0}^{t_1} w_{\pi(t)} \cdot p_{\pi(t)}(\hat{y})$
$w_{\pi(t)} = \exp(-\frac{\max(0, \rho(x_t, x_{\pi(t)}) - \epsilon_{\pi(t)})^2}{2\epsilon_{\pi(t)}^2})$
medium confidence
high confidence
low confidence
new gesture?
x_t
$\rho(x_t, x_{\pi(t)})$
$x_{\pi(t)}$

Fig. 3. Left: evolution of 10 action class probabilities over time for a test sequence containing three actions. Middle top: the pink line represents the standard deviation of class probabilities for each frame of the same sequence. The cyan curve is their average standard deviation computed over a short interval of frames. For each segmented activity, we computed a confidence measure $C_i(y)$; the predicted activity label is discarded when confidence is below an adaptive threshold Θ. Right: examples of three time series associated with low, medium and high confidence, respectively.

sion Trees), to name a few. Future research will explore the use of confidence measures to automatically discover new activity classes by associating them with low confidence trajectories (see Fig. 3, right), as we did for images [4].

References

[1] A. Antonucci, R. de Rosa, A. Giusti, F. Cuzzolin, Robust classification of multivariate time series by imprecise hidden Markov models, in: Int. J. Approx. Reasoning IJAR, 56, 2015, pp. 249–263.
[2] Y.-W. Chang, C.-J. Hsieh, K.-W. Chang, M. Ringgaard, C.-J. Lin, Training and testing low-degree polynomial data mappings via linear svm, The Journal of Machine Learning Research 11 (2010).
[3] R. De Rosa, N. Cesa-Bianchi, I. Gori, F. Cuzzolin, Online action recognition via nonparametric incremental learning, in: BMVC, 2014.
[4] R. De Rosa, T. Mensink, B. Caputo, Online open world recognition, CoRR abs/1604.02275 (2016).
[5] R. De Rosa, F. Orabona, N. Cesa-Bianchi, The ABACOC algorithm: a novel approach for nonparametric classification of data streams, in: Data Mining (ICDM), 2015 IEEE International Conference on, IEEE, 2015.
[6] O. Dekel, Y. Singer, Data-driven online to batch conversions, in: Advances in Neural Information Processing Systems, 2005, pp. 267–274.
[7] S.R. Fanello, I. Gori, G. Metta, F. Odone, Keep it simple and sparse: Real-time action recognition, J. Mach. Learn. Res. 14 (1) (2013).
[8] M.M. Gaber, A. Zaslavsky, S. Krishnaswamy, A survey of classification methods in data streams, in: Data Streams, Springer, 2007, pp. 39–59.

[9] J. Gama, R. Sebastiao, P.P. Rodrigues, On evaluating stream learning algorithms, Machine Learning (2013).

[10] L. Györfi, M. Köhler, A. Krzyzak, H. Walk, A distribution-free theory of non-parametric regression, Springer series in statistics, Springer, New York, Berlin, Paris, 2002.

[11] M. Hasan, A.K. Roy-Chowdhury, Continuous learning of human activity models using deep nets, in: Computer Vision–ECCV 2014, Springer, 2014a.

[12] M. Hasan, A.K. Roy-Chowdhury, Incremental activity modeling and recognition in streaming videos, in: Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on, IEEE, 2014b.

[13] M. Hasan, A.K. Roy-Chowdhury, Context aware active learning of activity recognition models, in: Computer Vision (ICCV), 2015 IEEE International Conference on, IEEE, 2015.

[14] Y.-G. Jiang, C.-W. Ngo, J. Yang, Towards optimal bag-of-features for object categorization and semantic video retrieval, in: Proceedings of the 6th ACM international conference on Image and video retrieval, ACM, 2007.

[15] J. Kies, Empirical Methods for Evaluating Video-Mediated Collaborative Work, Virginia Tech, 1997 Ph.D. thesis.

[16] S. Kpotufe, F. Orabona, Regression-tree tuning in a streaming setting., in: NIPS, 2013.

[17] R. Krauthgamer, J.R. Lee, Navigating nets: Simple algorithms for proximity search, in: Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, in: SODA '04, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2004.

[18] V. Levenshtein, Binary codes capable of correcting deletions, insertions, and reversals, Soviet Physics - Doklady (1966).

[19] M. Lichman, Uci machine learning repository (2007).

[20] J. Liu, Y. Yang, M. Shah, Learning semantic visual vocabularies using diffusion distance, in: Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on, IEEE, 2009.

[21] L. Liu, L. Shao, Learning discriminative representations from rgb-d video data, in: Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence, in: IJCAI'13, AAAI Press, 2013.

[22] J.T. M. Kudo, M. Shimbo, Multidimensional curve classification using passing-through regions, Pattern Recognition Letters 20 (1999).

[23] R. Minhas, A.A. Mohammed, Q.J. Wu, Incremental learning in human action recognition based on snippets, Circuits and Systems for Video Technology, IEEE Transactions on 22 (11) (2012).

[24] S. Oh, A. Hoogs, A. Perera, N. Cuntoor, C.-C. Chen, J.T. Lee, S. Mukherjee, J. Aggarwal, H. Lee, L. Davis, et al., A large-scale benchmark dataset for event recognition in surveillance video, in: Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on, IEEE, 2011.

[25] R. Poppe, A survey on vision-based human action recognition, Image and vision computing 28 (6) (2010).

[26] K.K. Reddy, J. Liu, M. Shah, Incremental action recognition using feature-tree, in: Computer Vision, 2009 IEEE 12th International Conference on, IEEE, 2009.

[27] C. Schuldt, I. Laptev, B. Caputo, Recognizing human actions: a local SVM approach, in: Proc. of International Conference on Pattern Recognition, 2004.

[28] A. Tsymbal, The problem of concept drift: Definitions and related work, Technical Report TCD-CS-2004-15, Trinity College Dublin, 2004.

[29] H. Wang, A. Klaser, C. Schmid, C.-L. Liu, Action recognition by dense trajectories, in: Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on, IEEE, 2011.

[30] H. Wang, C. Schmid, Action recognition with improved trajectories, in: Computer Vision (ICCV), 2013 IEEE International Conference on, IEEE, 2013.

[31] J. Wang, Z. Liu, J. Chorowski, Z. Chen, Y. Wu, Robust 3d action recognition with random occupancy patterns, in: Proceedings of the 12th European Conference on Computer Vision - Volume Part II, in: ECCV'12, Springer-Verlag, Berlin, Heidelberg, 2012.

[32] I. Zliobaite, A. Bifet, B. Pfahringer, G. Holmes, Active learning with drifting streaming data., IEEE transactions on neural networks and learning systems 25 (1) (2014).