# Sparse Code Filtering for Action Pattern Mining

Wei Wang[1], Yan Yan[1] Liqiang Nie[2], Luming Zhang[3], Stefan Winkler[4], and Nicu Sebe[1]

[1] University of Trento, Italy
[2] National University of Singapore
[3] Advanced Digital Sciences Center, Singapore
[4] Hefei University of Technology, China

**Abstract.** Action recognition has received increasing attention during the last decade. Various approaches have been proposed to encode the videos that contain actions, among which self-similarity matrices (SSMs) have shown very good performance by encoding the dynamics of the video. However, SSMs become sensitive when there is a very large view change. In this paper, we tackle the multi-view action recognition problem by proposing a sparse code filtering (SCF) framework which can mine the action patterns. First, a class-wise sparse coding method is proposed to make the sparse codes of the between-class data lie close by. Then we integrate the classifiers and the class-wise sparse coding process into a collaborative filtering (CF) framework to mine the discriminative sparse codes and classifiers jointly. The experimental results on several public multi-view action recognition datasets demonstrate that the presented SCF framework outperforms other state-of-the-art methods.

## 1 Introduction

Action recognition has wide applications, such as human-computer interactive games, search engines, and online video surveillance systems. Videos can be summarized by labels if the actions can be annotated automatically. Then a search engine can make better recommendations (*e.g., finding dunks in basketball games*). Usually, the same action observed from different viewpoints has considerable differences. Therefore, an efficient method to extract robust view-invariant features is essential for multi-view action recognition. The features can be roughly grouped into two types, the 2D features [1] and 3D features [2].

Many works employed 3D models to tackle the multi-view action recognition problem. First, the geometric transitions are utilized to obtain projections across different viewpoints. Then the observations are compared with the projections to find the viewpoint that best matches the observations [3]. However, how to accurately find body joints to build the 3D model remains an open problem. Besides, the built model has too many degree-of-freedom parameters, which must be carefully calibrated. Moreover, the model requires high resolution videos to locate body joints and sometimes may require mocap data [4]. An alternative solution for multi-view action recognition is to design view-invariant 2D features.

Farhadi *et .al* [5] proposed split-based representations by clustering the similar video frames into splits. The split-based representations can be transferred among different viewpoints as the change dynamics of the multi-view videos are the same. Similarly, Junejo *et. al* [6] employed SSMs to encode the frame-to-frame relative changes. However, the SSMs are robust to view changes only to a certain extent.

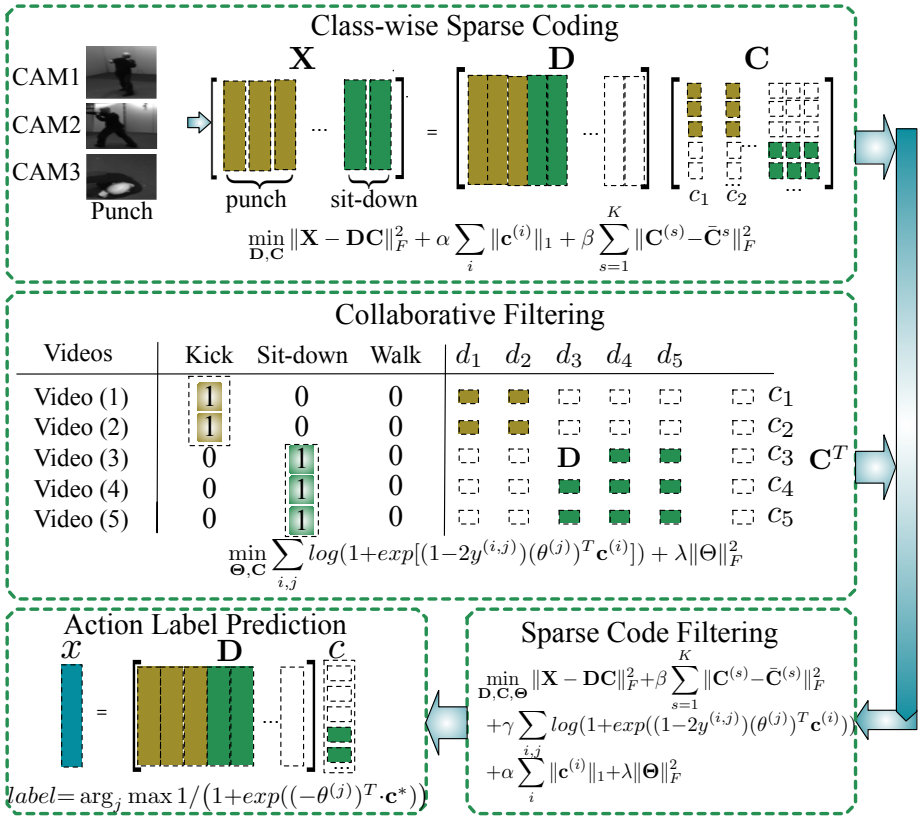


**Fig. 1.** Overview of sparse code filtering: (top) Class-wise sparse coding. (middle) Collaborative Filtering. (bottom right) Sparse code filtering framework. (bottom left) Label prediction.

In this paper, to tackle the multi-view problem, we propose a class-wise sparse coding approach to maintain label consistency. We employ SSM feature to represent each video. The sparse coding learns a dictionary from SSM representations of the video collections. The dictionary consists of typical action patterns, and each video is encoded to a code as a linear combination of action patterns. The label consistency is achieved by penalizing the within class variance of the codes. Thus, the codes of the within-class videos will lie close by, and accordingly, only

the view-invariant action patterns will be learned while the view-dependent information will be suppressed. Then we rely on the codes as video features to do action classification.

To further improve the discriminative power of the codes, we integrate the class-wise sparse coding and classifiers training process into a unified CF framework as shown in Fig.1. This is because CF can link the dictionary and classifiers together which can optimize them jointly. The dictionary can be adjusted for the classifiers while the classifiers can be adjusted for the dictionary collaboratively. In this way, the learned action patterns in the dictionary can be more discriminative with respect to different actions. Thus, we derive a novel sparse code filtering framework. In the sparse code filtering scheme, each action class is regarded as an user. For the classical collaborative filtering, the entry in the rating matrix (e.g., ranges from 0 to 5) describes how much a user likes the product. In our scheme, however, the entry in the rating matrix, ranging from 0 to 1, represents the probability that a video belongs to an action class. The sparse code filtering framework provides a trade-off between the dictionary reconstruction error and the classification error which derive from the class-wise sparse coding and the logistic classifiers respectively.

To summarize, our work makes the following contributions: (i) We propose a class-wise sparse coding approach to maintain the label consistency by encouraging the sparse codes of the multi-view videos within the same action class to lie close by. (ii) We propose a novel sparse code filtering framework in which the classifiers and dictionary can be optimized collaboratively. Thus, the view-invariant and class-discriminant sparse codes can be learned. (iii) The proposed sparse code filtering framework has a good generalization property and can be applied to other pattern recognition tasks.

## 2 Related Work

### 2.1 Action Recognition

Many 3D and 2D based approaches are proposed for action recognition. Through reconstructing 3D human bodies, features can be adapted across different viewpoints through geometric transformation. Weinland *et .al* [7] projected 3D poses into 2D to obtain arbitrary views and employed an exemplar-based HMM to model view transformations. A similar idea is proposed in [8] which employed CRF instead of HMM. Except for designing the 3D models, some works focus on designing view-invariant classifiers, such as linear discriminant analysis [9] and latent multi-task learning [10]. Matikainen *et. al* [11] suggested training models for all the views and then utilizing recommender system to find the suitable model. But the approach in [11] requires huge amount of training samples from different viewpoints. Recently, the recurrent neural network is also applied for the action recognition task [12] as it is good at dealing with signal sequences with various lengths [13, 14]. However, these methods can only tackle the single-view action recognition task.

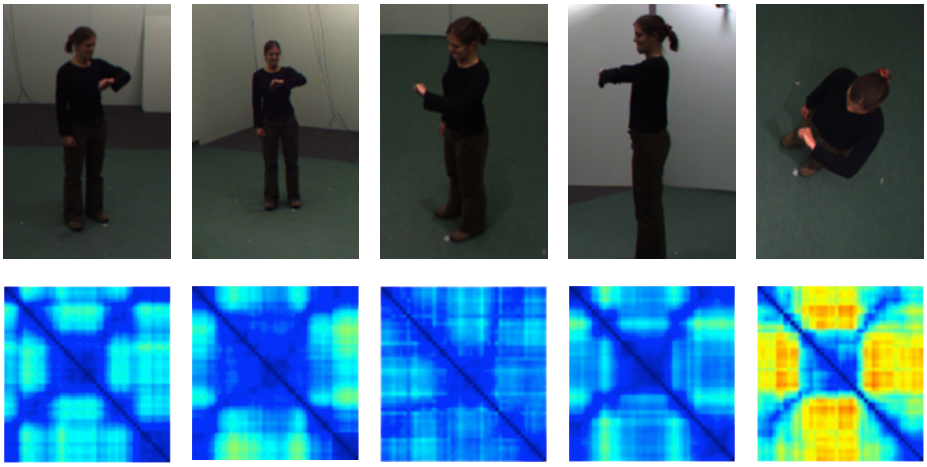

**Fig. 2.** SSM features extracted from different views.

To achieve view invariance in 2D models, many works try to extract view-invariant features. Farhadi *et. al* [5] proposed a split-based representation by clustering video frames into splits. Then videos can be represented by the statistics of the splits, and the split transfer mapping across views can be learned. Based on 2D features, the transfer learning model requires no 3D human reconstructions. Recently, a more robust view-invariant descriptor, self-similarity matrix (SSM) [6] has been proposed. It is relatively stable over the viewpoint changes compared with other features [15]. Similarly to [5], this descriptor encodes the relative changes between pairs of frames, and completely discards the absolute features of each single frame. SSMs can be calculated using different low-level features which have similar properties.

Fig.2 shows the examples from the action videos and their corresponding SSM features. From Fig.2, we can observe that the SSM features from the 4 side cameras are visually similar, while the feature from camera 5 (on the ceiling) is quite different. Yan *et. al* [9] revealed that SSMs became less reliable when there was a very large view change. Based on SSMs, Joint Self-Similarity Volume (SSV) was introduced by [16] which utilized Joint Recurrence Plot (JRP) theory to extend SSM. But different from [15], the SSM defined in [16] is the recurrence-plot matrix of the vector representation of each single frame.

### 2.2 Sparse Coding

Sparse coding, also known as dictionary learning, aims to construct efficient representations of data as a combination of a few typical patterns (dictionary bases). Wang *et. al* [17, 18] used the sparse coding for attribute detection. Raina *et. al* [19] showed that sparse coding significantly improved classification performance. [20] employed sparse coding for action recognition from depth maps. However, their approach is restricted to the videos which can provide depth information.

Qiu *et. al* [21] selected a set of more compact and discriminative bases from the dictionary using Gaussian Process. Guha *et. al* [22] investigated different sparse coding strategies, namely, an overall dictionary for all the classes, different dictionaries for each class, and their concatenation. But view changes were not considered. Zheng *et. al* [23] proposed view-specific sparse coding. But sufficient training data from each viewpoint are required. Besides, the label information is discarded. Thus, it can not preserve label consistency. In our work, we add within-class variance into the loss function to preserve the label-consistency. The learned class-wise dictionary can be considered as a more label-smooth feature space compared with the original video feature space.

### 2.3 Collaborative Filtering

Collaborative filtering is widely used in recommendation systems of commercial websites, such as Amazon and eBay, to recommend products to their consumers. The most attractive characteristic of CF is that it can learn a good set of features automatically [24], which does not require hand-designed features. Taking the movie recommendation system for example, each movie has its own features and each user has its own specific feature preference weights. Given the movie-user rating matrix, CF learn a good set of features for each movie and feature preference weights for each user jointly. During the CF learning process, the features will be adjusted for the feature preference weights for each user, and feature preference weights will also be adjusted for the features iteratively. Inspired by the movie recommendation system, we employ a CF framework to learn class-wise dictionary and classifiers jointly. Thus, the codes and classifiers can be adjusted to better fit each other. The experimental results demonstrate the effectiveness of our framework.

The rest of the paper is organized as follows. We propose the class-wise dictionary learning approach in the Section 3. Section 4 presents our sparse code filtering scheme. The experiments are described in Section 5. We conclude our paper in Section 6.

## 3 Class-wise Sparse Coding

The input of the sparse model is the descriptors for $n_v$ videos, where each video is represented by a $d$-dimension vector $\mathbf{x}_d$. Let $\mathbf{X}_{d\times n_v}$ be the matrix by stacking the all the training video descriptors. In our model, $\mathbf{x}_d$ is the SSM feature. The outputs are the dictionary $\mathbf{D}$ and sparse codes $\mathbf{C}$. The loss of the *classical* sparse coding model, which considers reconstruction error and sparsity, is defined as:

$$\mathbf{L}(\mathbf{X};\mathbf{D},\mathbf{C})=\|\mathbf{X}-\mathbf{D}\mathbf{C}\|_F^2+\alpha\sum_{i=1}^{n_v}\|\mathbf{c}^{(i)}\|_1 \tag{1}$$

In Eq. 1, $\mathbf{D}_{d\times n}$ represents the learned dictionary and each column vector in the dictionary represents a typical action pattern, $n$ is the number of typical

patterns, $\mathbf{C}_{n\times n_v}$ is the sparse code matrix, whose $i$-th column, $\mathbf{c}^{(i)}$, is the sparse code of sample $i$. $l_1$-norm is a lasso constraint which encourages sparsity, and $\alpha$ balances the reconstruction error and the sparsity penalty.

In order to mine the view-invariant patterns of the SSM feature, we propose a class-wise sparse coding method to encourage the sparse codes of the multi-view within-class videos to lie close by. The closeness is measured by the within-class variance. Given the class labels of the training data, we try to reduce the within-class variance during the learning process. The within-class variance is measured by the Euclidean distance between the videos and their class center. The loss of the class-wise sparse coding model is defined as follows,

$$\mathbf{L}(\mathbf{X};\mathbf{D},\mathbf{C}) + \beta \sum_{s=1}^{K} \|\mathbf{C}^{(s)} - \bar{\mathbf{C}}^s\|_F^2 \tag{2}$$

The second term in Eq. 2 measures the within-class variance. This term enforces the multi-view within-class videos to have similar sparse codes. $K$ is the number of action classes. $\mathbf{C}^{(s)}$ represents a video collection. $s$ is the class index. Each column vector in $\mathbf{C}^{(s)}$ is the sparse code of the video which belong to action class $s$. Each column vector in $\bar{\mathbf{C}}^s$ is the mean of all the column vectors in $\mathbf{C}^{(s)}$. $\bar{\mathbf{C}}^s$ has the same size as $\mathbf{C}^{(s)}$. $\beta$ is the weight of within-class variance penalty.

## 4 Sparse Code Filtering

### 4.1 Joint Action Learning

The input to our learning scheme is (1) the learned sparse codes for $n_v$ videos, each represented as a $n$-dimension vector $\mathbf{c}^{(i)} \in \mathbb{R}^n, i = 1, 2, ..., n_v$. (2) the binary action label matrix for all the videos, which is represented as $\mathbf{Y}_{n_v \times n_a}$, $n_a$ is the number of actions. The item $y^{(i,j)}$, is either 1 or 0, which denotes whether or not video $i$ belongs to action class $j$.

We learn all action classifiers simultaneously in a multi-task learning setting, where each *task* represents one action. The output is the parameter matrix $\mathbf{\Theta}_{n\times n_a}$ whose column vector $\theta^{(j)}$ denotes the parameters of the classifier of action $j$. In our model, we employ logistic regression classifiers. Given the sparse code matrix and binary action label matrix $(\mathbf{C}_{n\times n_v}, \mathbf{Y}_{n_v \times n_a})$, the loss function is defined as:

$$\mathbf{L}(\mathbf{C}, \mathbf{Y}; \mathbf{\Theta}) = \sum_{i,j} log(1 + exp((1 - 2y^{(i,j)})(\theta^{(j)})^T \mathbf{c}^{(i)})) \tag{3}$$

Each action classifier has an tuple $\theta^{(j)}$ whose element $\theta_k^{(j)}$ corresponds to the *weight* of the sparse code which is tied to the $k$-th typical pattern in the dictionary.

### 4.2 Formulation of Sparse Code Filtering

Usually, the dictionary and classifiers are trained separately. Thus, there is no guarantee that the learned patterns in the dictionary can serve the classification task well. In order to mine the class-discriminative action patterns, we propose a sparse code filtering (SCF) scheme. In our scheme, the prediction function is logistic function whose output denotes the probability that a video belongs to an action. Besides, the parameters are learned by minimizing both the dictionary reconstruction error and classification error. Thus, the dictionary and classifiers are optimized jointly. The learned sparse codes are expected to be view-invariant and class-discriminative. By integrating all the tasks, we can obtain the following loss function:

$$\mathbf{L}(\mathbf{X},\mathbf{Y};\mathbf{D},\mathbf{C},\mathbf{\Theta})=\mathbf{L}(\mathbf{X};\mathbf{D},\mathbf{C})+\gamma\mathbf{L}(\mathbf{C},\mathbf{Y};\mathbf{\Theta})+\beta\sum_{s=1}^{K}\|\mathbf{C}^{(s)}-\bar{\mathbf{C}}^{(s)}\|_F^2+\lambda\|\mathbf{\Theta}\|_F^2 \quad (4)$$

In Eq. 4, $\gamma$ balances the dictionary reconstruction error and the classification error, the Frobenius norm of $\mathbf{\Theta}$ is employed to prevent overfitting. By minimizing the loss function, Eq. 4, a view-invariant and class-discriminative dictionary $\mathbf{D}$, and an action classification parameter matrix $\mathbf{\Theta}$ are learned jointly.

**Optimization** The input of the SCF framework is video descriptor matrix and binary action label matrix: $[\mathbf{X},\mathbf{Y}]$. The outputs are the dictionary, sparse codes, and parameter matrix for the classifiers: $[\mathbf{D},\mathbf{C},\mathbf{\Theta}]$. We propose the following algorithm (Alg. 1) to solve the framework. When only one variable is left to optimize and the rest are fixed, the problem becomes convex. Thus, we optimize the variables alternatively by fixing the rest.

*Initialization* in Algorithm 1: we employ k-means clustering to find k centroids as the bases in dictionary $\mathbf{D}_0$. $\mathbf{\Theta}_0$ and $\mathbf{C}_0$ are set to $\mathbf{0}$.

*The loop* in Algorithm 1 consists of three parts:

1. Fix $\mathbf{C}$, $\mathbf{\Theta}$, Optimize $\mathbf{D}$. In Eq. (4), only the first term is related to $\mathbf{D}$, and it is a least square problem when the other parameters are fixed. By setting the derivative of Eq. (4) equal to $\mathbf{0}$ with respect to $\mathbf{D}$, we can obtain:

$$(\mathbf{DC}-\mathbf{X})\mathbf{C}^T=0 \Rightarrow \mathbf{D}=\mathbf{XC}^T(\mathbf{CC}^T)^{-1} \quad (5)$$

Then we employ the following equation to update $\mathbf{D}$:

$$\mathbf{D}=\mathbf{XC}^T(\mathbf{CC}^T+\lambda\mathbf{I})^{-1} \quad (6)$$

$\lambda$ is a small constant and it guarantees that the matrix $\mathbf{CC}^T+\lambda\mathbf{I}$ is invertible in case $\mathbf{CC}^T$ is singular.

**Algorithm 1:** Solution Structure

*Initialization*: $\mathbf{D} \leftarrow \mathbf{D}_0$ , $\mathbf{C} \leftarrow \mathbf{C}_0$ , $\mathbf{\Theta} \leftarrow \mathbf{\Theta}_0$
**repeat**
  *fix* $\mathbf{D}, \mathbf{\Theta}$, *update* $\mathbf{C}$:
  **for** $\mathbf{C}^{(s)} \in \mathbf{C}$ **do**
    $ratio \leftarrow 1$
    **while** $ratio > threshold$ **do**
      *run* **FISTA(modified)**
      **update** $ratio$
    **end while**
  **end for**
  *fix* $\mathbf{D}, \mathbf{C}$, *update* $\mathbf{\Theta}$:
  **parallelgradientdescent**
  *fix* $\mathbf{C}, \mathbf{\Theta}$, *update* $\mathbf{D}$:
  $\mathbf{least-squaressolution}$
**until** converges

2. Fix $\mathbf{D}$, $\mathbf{C}$, Optimize $\mathbf{\Theta}$. When $\mathbf{D}$ & $\mathbf{C}$ are fixed, we employ the parallel gradient descent method to tackle the problem. Since $\theta^{(j)}$ are independent from each other, we optimize them in parallel. The updating formula is as follows:

$$\theta^{(j)} = \theta^{(j)} - \delta \frac{\partial}{\partial \theta^{(j)}} \mathbf{L}(\mathbf{X}, \mathbf{Y}; \mathbf{D}, \mathbf{C}, \mathbf{\Theta}) \tag{7}$$

3. Fix $\mathbf{D}$, $\mathbf{\Theta}$, Optimize $\mathbf{C}$. Beck *et. al* [25] proposed the Fast Iterative Soft-Thresholding Algorithm (FISTA) to solve the classical dictionary learning problem. A soft-threshold step is incorporated into FISTA to guarantee the sparseness of the solution. The complexity for the classical ISTA method is $O(1/k)$, in which $k$ denotes the iteration times. FISTA converges in function values as $O(1/k^2)$, which is much faster. FISTA optimizes $\mathbf{c}^{(i)} \in \mathbf{C}$ independently. However, in our model, $\mathbf{c}^{(i)}$ and $\mathbf{c}^{(j)}$ within the same action class depend on each other. Thus, $\mathbf{c}^{(i)}$, $\mathbf{c}^{(j)}$ must be updated jointly until all of them converge. Thus, we decompose our objective function and modify the original FISTA algorithm to tackle the decomposed sub-objectives.
In Eq. (4), the sparse code matrices with respect to different action classes are independent. Thus, when updating $\mathbf{C} = [\mathbf{C}^{(1)}, ..., \mathbf{C}^{(K)}]$, we decompose the objective function into $K$ sub-objectives, shown as follows:

$$\min_{\mathbf{C}} \sum_{s=1}^{K} \mathbf{L}(\mathbf{C}^{(s)}) = \sum_{s=1}^{K} \min_{\mathbf{C}^{(s)}} \mathbf{L}(\mathbf{C}^{(s)}) \tag{8}$$

Thus, the original objective function is decomposed into K sub-objective functions with respect to each action class. The following shows the details of the deduction of decomposition of Eqn. 4. The first two terms in Eqn. 4 can be reformulated as follows:

$$\begin{cases} \mathbf{L}(\mathbf{X};\mathbf{D},\mathbf{C}) = \sum_{s=1}^{K} \left( \|\mathbf{D}\mathbf{C}^{(s)} - \mathbf{X}^{(s)}\|^2 + \alpha \|\mathbf{C}^{(s)}\|_1 \right) \\ \mathbf{L}(\mathbf{C}, \mathbf{Y}; \mathbf{\Theta}) = \sum_{s=1}^{K} \sum_{j=1}^{n_a} \log(1+\exp(1-2y^{(i,j)})(\theta^{(j)})^T \mathbf{C}^{(s)}) \end{cases} \tag{9}$$

Putting the transformed terms from Eq. (9) back into the loss function Eq. (4), we can obtain a new form of the objective function. Because $\mathbf{D}$ and $\mathbf{\Theta}$ are fixed, the term $\lambda\|\mathbf{\Theta}\|_F^2$ becomes a constant. By removing the constant term, we can obtain the loss function as Eq. (8) where

$$\begin{aligned} \mathbf{L}(\mathbf{C}^{(s)}) = & \|\mathbf{D}\mathbf{C}^{(s)} - \mathbf{X}^{(s)}\|_F^2 + \alpha\|\mathbf{C}^{(s)}\|_1 + \beta\|\mathbf{C}^{(s)} - \bar{\mathbf{C}}^s\|_F^2 \\ & + \gamma \sum_{j=1}^{n_a} \log(1+\exp(1-2y^{(i,j)})(\theta^{(j)})^T \mathbf{C}^{(s)}) \end{aligned} \tag{10}$$

The modified FISTA algorithm is applied to solve the sub-objective functions. The details of the modified FISTA algorithm is as follows:
In the classical dictionary learning model, the sparse codes of training data are independent from each other. Thus, each $\mathbf{c}$ can be optimized independently. However, our new sub-objective needs to optimize a group of training data jointly because these data have dependencies among each other as shown in Eq. (10). For training data $\mathbf{x}^{(i)} \in \mathbf{X}^{(s)}$ in the equation above, its sparse code $\mathbf{c}^{(i)}$ ($\mathbf{c}^{(i)} \in \mathbf{C}^{(s)}$) dependents on other $\mathbf{c}^{(k)}$ ($\mathbf{c}^{(k)} \in \mathbf{C}^{(s)}$). We modify the classical FISTA algorithm to optimize the sub-objectives jointly.
When update $\mathbf{C}^{(s)}$, instead of updating $\mathbf{c}^{(i)}$ independently, all $\mathbf{c}^{(i)} \in \mathbf{C}^{(s)}$ are updated simultaneously using the following form,

$$\mathbf{c}^{(i)} := \mathbf{c}^{(i)} - \delta \frac{\partial L}{\partial \mathbf{c}^{(i)}} \tag{11}$$

This updating procedure of $\mathbf{C}^{(s)}$ will repeat until it converges. Then we apply a soft-threshold step to set the entries in $\mathbf{C}^{(s)}$ whose absolute value is less than the threshold to 0. We repeat the process above until the whole algorithm converges.

**Label Prediction** As shown in Fig.1, in the classical CF framework, when the features of a new movie are given, its ratings by different users can be predicted based on the movie features and the learned feature preference weights. The basic underlying assumption of CF is that users will rate movies which share the similar features with similar scores [26] as we assume that the preferences of the users remain the same. Similarly, each action class can be regarded as one user, and the action videos can be regarded as the movies. The label prediction for a new video $\mathbf{x}$ consists of two steps: sparse coding and probability calculation.

$$\mathbf{c}^* = \arg_{\mathbf{c}} \min \mathbf{L}(\mathbf{x}, \mathbf{D}; \mathbf{c}) \tag{12}$$

$$label = \arg_j \max 1/\left(1 + exp((-\theta^{(j)})^T \cdot \mathbf{c}^*)\right) \tag{13}$$

First, given the dictionary $\mathbf{D}$, and video descriptor $\mathbf{x}$ which is the SSM feature, the sparse code $\mathbf{c}^*$ of the new video is calculated by solving the classical sparse coding model as shown in Eq. (12). Then the probability that the new video belongs to action class $j$ can be calculated. The action label is the one which maximizes the probability as shown in Eq. (13).

## 5 Experiments and Results

### 5.1 Datasets

We evaluate our framework on three largest public *multi-view* action recognition datasets, as shown in Fig.3, which are the IXMAS dataset [27], the NIXMAS dataset, and the OIXMAS dataset [28] in which the actions are partially occluded. IXMAS dataset consists of 12 action classes, (*e.g., check watch, cross arms, scratch head, sit down, get up, turn around, walk, wave, punch, kick, point and pick up*). Each action is performed 3 times by 11 actors and is recorded by 5 cameras which observe the actions from 5 different viewpoints. The NIXMAS dataset is recorded with different actors, cameras, and viewpoints, and about 2/3 of the videos have objects which partially occlude the actors. Overall, it contains 1148 sequences.

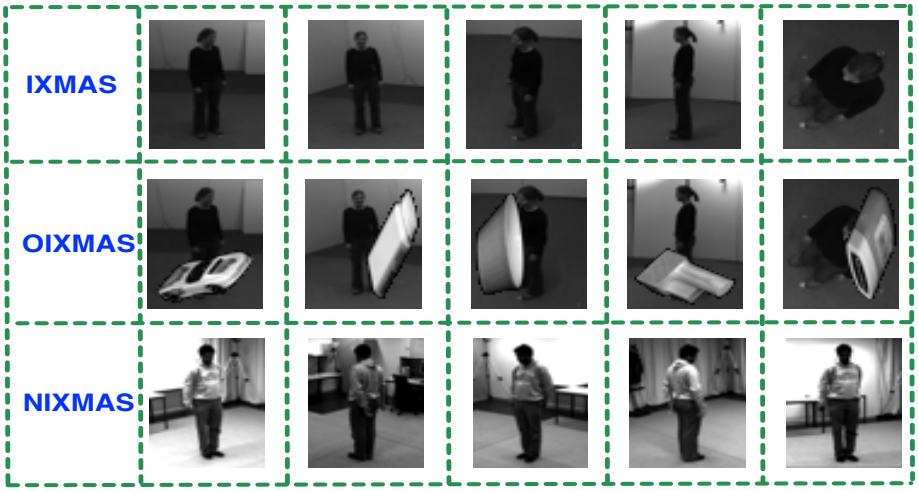


**Fig. 3.** Multi-view action recognition datasets.

### 5.2 Implementation Details

The sparse code filtering is based on SSM descriptors using HOG/HOF features to describe each individual frame. Each video is represented by a 500-

dimension vector. Fig.2 shows an example from IXMAS dataset and the corresponding extracted SSM feature. In our experiments, the dictionary size is set to $[600, 700, ..., 1000]$, and all regularization parameters $\alpha$, $\beta$, $\gamma$, $\lambda$ are tuned from $[10^{-3}, 10^{-2}, ..., 10^{3}]$.

We employ two settings for the experiment, which are *multi-view* setting and *cross-view* setting. For the *multi-view* setting, we have access to the videos from all the viewpoints for training, and use the standard experimental protocol described in [29]: two-thirds and one-third split for training and testing. This experimental protocol is widely used for action recognition. For the *cross-view* setting, one camera view is missing in the training data and we train the model using the data from other four camera views. Then we perform prediction on the missing view.

### 5.3 Baselines

To evaluate the contribution of the class-wise sparse coding (CWSC), we put the raw features and the codes into two classification scheme: (1) standard radial basis kernel SVM [6] which learns each action classifiers separately, and (2) the multi-task learning approach [9] which learns the action classifiers jointly. The codes and the classifiers are learned separately. We name the two baselines which take the codes as input as (3) CWSC+SVM, and (4) CWSC+MTL, and they are employed as baselines. Then through the comparison between (CWSC+MTL) and our SCF framework, we can observe the extra gain we obtained by training the class-wise dictionary and classifiers jointly. (5) We also choose some other action recognition baselines, such as [30], [9], and [29].

### 5.4 Results

**Multi-view Action Recognition.** For the multi-view setting, we use the standard two-thirds and one-third split for training and testing. Table 1 shows the mean action recognition accuracy of all the cameras using different approaches.

We observe that the baselines CWSC+SVM and CWSC+MTL outperform SVM and MTL with raw features respectively. This indicates that the class-wise sparse coding can help encode the view-invariant action patterns which preserve the label consistency. From Table 1, we can also observe that our method has the best performance. This is because our sparse code filtering scheme optimizes the classifiers and dictionary jointly, and it helps learn a class-wise label-discriminative dictionary. Fig.4 shows some qualitative results on IXMAS dataset for our proposed SCF framework and multi-task learning approach for multi-view action recognition.

**Cross-view Action Recognition** Table 2, 3 and 4 show the performances of different approaches on IXMAS, OIXMAS, and NIXMAS dataset.

From Table 2, 3 and 4, we can observe that our framework achieves better performance compared with other baselines which shows the effectiveness of our

**Table 1.** Multi-view action recognition accuracy of different approaches for 3 datasets.

| **Methods** | IXMAS | OIXASM | NIXMAS |
|---|---|---|---|
| SVM [6] | 0.6425 | 0.4809 | 0.5680 |
| CWSC+SVM | 0.6537 | 0.5235 | 0.6026 |
| MTL [9] | 0.6883 | 0.5608 | 0.6163 |
| CWSC+MTL | 0.6889 | 0.6082 | 0.6228 |
| Farhadi *et. al* [5] | 0.5810 | - | - |
| Huang *et. al* [29] | 0.5730 | - | - |
| Liu *et. al* [31] | 0.7380 | - | - |
| Reddy *et. al* [32] | 0.7260 | - | - |
| Li *et. al* [30] | 0.8120 | - | - |
| Baumann *et. al* [33] | 0.8055 | - | - |
| Ashraf *et. al* [34] | 0.8140 | - | - |
| **SCF** | **0.8594** | **0.7803** | **0.8083** |

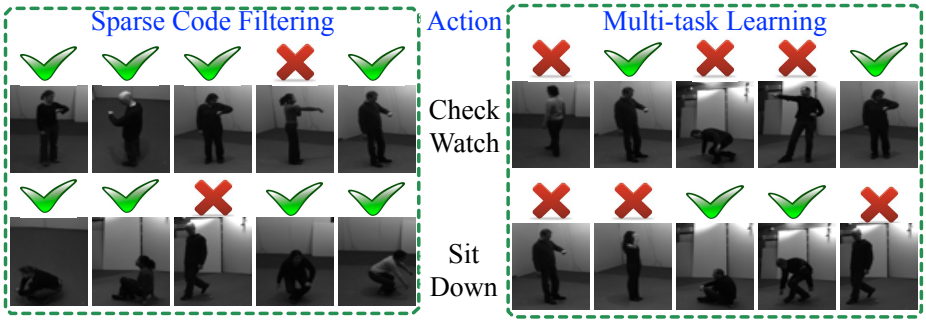


**Fig. 4.** Qualitative results on IXMAS dataset.

**Table 2.** Cross-View action recognition performance on the IXMAS dataset

| | **Missing Viewpoints** | | | | | |
|---|---|---|---|---|---|---|
| **Methods** | Cam 1 | Cam 2 | Cam 3 | Cam 4 | Cam 5 | Avg |
| Junejo *et. al* [6] | 0.6663 | 0.6554 | 0.6500 | 0.6243 | 0.4963 | 0.6185 |
| CWSC+SVM | 0.6880 | 0.6577 | 0.6701 | 0.6187 | 0.5110 | 0.6291 |
| Yan *et. al* [9] | 0.7554 | 0.7462 | 0.7710 | 0.6973 | 0.6332 | 0.7206 |
| CWSC+MTL | 0.7559 | 0.8257 | 0.8003 | 0.7759 | 0.6417 | 0.7599 |
| **SCF** | **0.8285** | **0.8322** | **0.8053** | **0.7941** | **0.7384** | **0.7997** |

learned dictionary. It is also interesting to notice that the fifth camera always has low action recognition accuracy regardless of the classification methods. One reasonable explanation is that the fifth camera is placed on the ceiling, and the motion dynamics of different actions observed from this camera are visually similar with each other.

**Table 3.** Cross-View action recognition performance on the OIXMAS dataset

| | Missing Viewpoints | | | | | |
|---|---|---|---|---|---|---|
| **Methods** | Cam 1 | Cam 2 | Cam 3 | Cam 4 | Cam 5 | Avg |
| Junejo *et. al* [6] | 0.5639 | 0.6250 | 0.5472 | 0.4677 | 0.4423 | 0.5292 |
| CWSC+SVM | 0.5688 | 0.6477 | 0.6001 | 0.5087 | 0.4511 | 0.5553 |
| Yan *et. al* [9] | 0.5422 | 0.6540 | 0.5070 | 0.5171 | 0.4730 | 0.5387 |
| CWSC+MTL | 0.5535 | 0.6826 | 0.5366 | 0.5401 | 0.4867 | 0.5599 |
| **SCF** | **0.6080** | **0.6980** | **0.6573** | **0.6957** | **0.5850** | **0.6512** |

**Table 4.** Cross-View action recognition performance on the NIXMAS dataset

| | Missing Viewpoints | | | | | |
|---|---|---|---|---|---|---|
| **Methods** | Cam 1 | Cam 2 | Cam 3 | Cam 4 | Cam 5 | Avg |
| Junejo *et. al* [6] | 0.6410 | 0.6532 | 0.5912 | 0.5924 | 0.5322 | 0.6020 |
| CWSC+SVM | 0.6759 | 0.6951 | 0.6226 | 0.6387 | 0.5560 | 0.6377 |
| Yan *et. al* [9] | 0.7170 | 0.6993 | 0.7542 | 0.6911 | 0.6792 | 0.7082 |
| CWSC+MTL | 0.7198 | 0.7391 | 0.7559 | 0.7176 | 0.6879 | 0.7240 |
| **SCF** | **0.8080** | **0.7980** | **0.7573** | **0.7357** | **0.7050** | **0.7608** |

### 5.5 Parameter Tuning

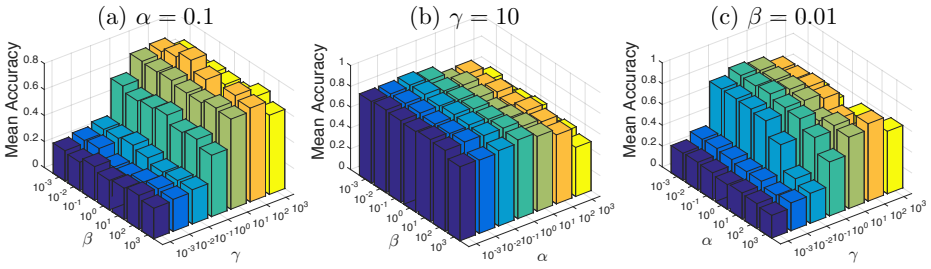


**Fig. 5.** Sensitivity study of different regularization parameters on IXMAS dataset.

Fig.5 shows the sensitivity study of regularization parameters $\gamma$, $\alpha$, $\beta$ and $\lambda$. In our model, $\gamma$ balances the dictionary learning loss and the classification loss, $\alpha$ balances the reconstruction error and the sparsity penalty, $\beta$ provides the trade-off between the dictionary reconstruction loss and intra-class variance penalty, and $\lambda$ is employed to prevent overfitting of the classifiers. The optimal classification performance can be obtained when dictionary size is set to 800. We observe that the performance changes little (within 0.0015) when we set $\lambda$ to the different values. So we focus on the other 3 parameter. As shown in Fig.5(a), when $\gamma$ is fixed, the mean accuracy varies subtly along the axis of $\beta$.

However, when $\beta$ is fixed, the mean accuracy changes dramatically along the axis of $\beta$. Thus, $\gamma$ is more sensitive than $\beta$. Similarly, Fig.5(b) shows that $\alpha$ is more sensitive than $\beta$, and Fig.5(c) shows that $\gamma$ is more sensitive than $\alpha$. Thus, we obtain the importance of these parameters $\gamma > \alpha > \beta > \lambda$.

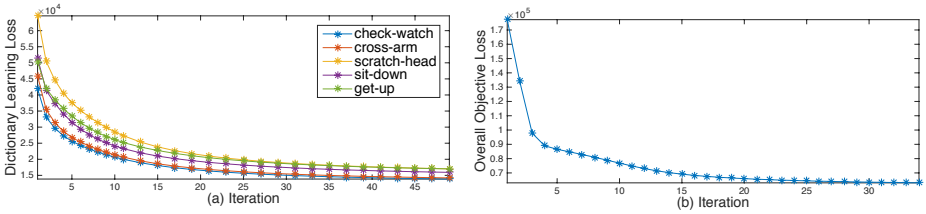


**Fig. 6.** Convergence of the sparse code filtering algorithm on IXMAS dataset.

We also analyzes the convergence of our algorithm. Fig.6 plots the convergence curves of the objectives. Fig.6(b) shows that Alg. 1 converges in 30 iterations. Fig.6(a) plots the convergence curves when updating $\mathbf{C}^{(s)}$ for action classes. It shows that the class-wise dictionary learning converges very fast.

## 6 Conclusion

In this paper, we propose a novel sparse code filtering framework for multi-view action recognition. First, a class-wise dictionary is learned by encoding label information into the sparse coding process. We integrate class-wise sparse coding and classifier learning into a CF framework. Thus, the classifiers and dictionary are optimized jointly, and they can be adapted for each other. The extensive experimental results illustrate that our proposed method outperforms other important baselines for multi-view action recognition. In the future work, we will take the correlation between the classifiers into consideration. For example, we can suppress the urge of feature sharing between classifiers by adding a $l_1$ norm penalty to the classifier parameters.

## References

1. Cai, Z., Wang, L., Peng, X., Qiao, Y.: Multi-view super vector for action recognition. In: CVPR. (2014)
2. Vemulapalli, R., Arrate, F., Chellappa, R.: Human action recognition by representing 3d skeletons as points in a lie group. In: CVPR. (2014)
3. Lv, F., Nevatia, R.: Single view human action recognition using key pose matching and viterbi path searching. In: CVPR. (2007)
4. Peursum, P., Venkatesh, S., West, G.: Tracking-as-recognition for articulated full-body human motion analysis. In: CVPR. (2007)

5. Farhadi, A., Tabrizi, M.K.: Learning to recognize activities from the wrong view point. In: ECCV. (2008)
6. Junejo, I.N., Dexter, E., Laptev, I., Perez, P.: View-independent action recognition from temporal self-similarities. TPAMI (2011)
7. Weinland, D., Boyer, E., Ronfard, R.: Action recognition from arbitrary views using 3d exemplars. In: ICCV. (2007)
8. Natarajan, P., Nevatia, R.: View and scale invariant action recognition using multiview shape-flow models. In: CVPR. (2008)
9. Yan, Y., Ricci, E., Subramanian, R., Liu, G., Sebe, N.: Multitask linear discriminant analysis for view invariant action recognition. TIP (2014)
10. Mahasseni, B., Todorovic, S.: Latent multitask learning for view-invariant action recognition. In: ICCV. (2013)
11. Matikainen, P., Sukthankar, R., Hebert, M.: Model recommendation for action recognition. In: CVPR. (2012)
12. Du, Y., Wang, W., Wang, L.: Hierarchical recurrent neural network for skeleton based action recognition. In: CVPR. (2015)
13. Wang, W., Cui, Z., Yan, Y., Feng, J., Yan, S., Shu, X., Sebe, N.: Recurrent face aging. In: CVPR. (2016)
14. Wang, W., Tulyakov, S., Sebe, N.: Recurrent convolutional face alignment. In: ACCV. (2016)
15. Junejo, I.N., Dexter, E., Laptev, I., Pérez, P.: Cross-view action recognition from temporal self-similarities. Springer (2008)
16. Sun, C., Junejo, I., Foroosh, H.: Action recognition using rank-1 approximation of joint self-similarity volume. In: ICCV. (2011)
17. Wang, W., Yan, Y., Winkler, S., Sebe, N.: Category specific dictionary learning for attribute specific feature selection. TIP (2016)
18. Wang, W., Yan, Y., Sebe, N.: Attribute guided dictionary learning. In: ICMR. (2015)
19. Raina, R., Battle, A., Lee, H., Packer, B., Ng, A.Y.: Self-taught learning: transfer learning from unlabeled data. In: ICML. (2007)
20. Luo, J., Wang, W., Qi, H.: Group sparsity and geometry constrained dictionary learning for action recognition from depth maps. In: ICCV. (2013)
21. Qiu, Q., Jiang, Z., Chellappa, R.: Sparse dictionary-based representation and recognition of action attributes. In: ICCV. (2011)
22. Guha, T., Ward, R.K.: Learning sparse representations for human action recognition. TPAMI (2012)
23. Zheng, J., Jiang, Z.: Learning view-invariant sparse representations for cross-view action recognition. In: ICCV. (2013)
24. Goldberg, D., Nichols, D., Oki, B.M., Terry, D.: Using collaborative filtering to weave an information tapestry. Communications of the ACM (1992)
25. Beck, A., Teboulle, M.: A fast iterative shrinkage-thresholding algorithm for linear inverse problems. SIAM Journal on Imaging Sciences (2009)
26. Goldberg, K., Roeder, T., Gupta, D., Perkins, C.: Eigentaste: A constant time collaborative filtering algorithm. Information Retrieval (2001)
27. Weinland, D., Ronfard, R., Boyer, E.: Free viewpoint action recognition using motion history volumes. CVIU (2006)
28. Weinland, D., Özuysal, M., Fua, P.: Making action recognition robust to occlusions and viewpoint changes. In: ECCV. (2010)
29. Huang, C.H., Yeh, Y.R., Wang, Y.C.F.: Recognizing actions across cameras by exploring the correlated subspace. In: ECCV. (2012)

30. Li, R., Zickler, T.: Discriminative virtual views for cross-view action recognition. In: CVPR. (2012)
31. Liu, J., Shah, M.: Learning human actions via information maximization. In: CVPR. (2008)
32. Reddy, K.K., Liu, J., Shah, M.: Incremental action recognition using feature-tree. In: ICCV. (2009)
33. Baumann, F., Ehlers, A., Rosenhahn, B., Liao, J.: Recognizing human actions using novel space-time volume binary patterns. Neurocomputing (2016)
34. Ashraf, N., Sun, C., Foroosh, H.: View invariant action recognition using projective depth. CVIU (2014)