Combined Static and Motion Features for Deep-Networks Based Activity Recognition in Videos

Sameera Ramasinghe, *Member, IEEE,* Jathushan Rajasegaran, Vinoj Jayasundara, Kanchana Ranasinghe, Ranga Rodrigo, *Member, IEEE,* and Ajith A. Pasqual, *Member, IEEE*

S. Ramasinghe, J. Rajasegaran, K. Ranasinghe, R. Rodrigo, and A. Pasqual were with the Department of Electronic and Telecommunication Engineering, the University of Moratuwa, 10400, Sri Lanka, e-mail: (ranga@uom.lk).

Manuscript received January XX, 201X; revised XXXX XX, 201X.

Combined Static and Motion Features for Deep-Networks Based Activity Recognition in Videos

***Abstract*—Activity recognition in videos in a deep-learning setting—or otherwise—uses both static and pre-computed motion components. The method of combining the two components, whilst keeping the burden on the deep network less, still remains uninvestigated. Moreover, it is not clear what the level of contribution of individual components is, and how to control the contribution. In this work, we use a combination of CNN-generated static features and motion features in the form of motion tubes. We propose three schemas for combining static and motion components: based on a variance ratio, principal components, and Cholesky decomposition. The Cholesky decomposition based method allows the control of contributions. The ratio given by variance analysis of static and motion features match well with the experimental optimal ratio used in the Cholesky decomposition based method. The resulting activity recognition system is better or on par with existing state-of-the-art when tested with two popular data sets. The findings also enable us to characterize a data set with respect to its richness in motion information.**

***Index Terms*—Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN), Long Short-Term Memory (LSTM).**

I. Introduction

AUTOMATIC activity recognition in video an intensely researched area in computer vision due to its wide range of real-world applications in sports, health care, surveillance, robot vision, and human-computer interaction. Furthermore, the rapid growth of digital video data demands automatic classification and indexing of videos. Despite the increased interest, state-of-the-art systems are still far from human-level performance, in contrast to the success in image classification [**?**], [**?**]. This is partially due to the complex intra-class variations in videos, some obvious causes being the view point, background clutter, high dimensionality of data, lack of large data sets, and low resolution.

Despite these reasons more-or-less affecting automatic image classification, it has been quite successful in recent years, largely owing to the rise of deep learning techniques. This is not the case in video classification, although deep learning is starting to be widely applied. Therefore, it is worthwhile investigating what is holding back video classification. In this study, we address three key areas: exploiting the underlying dynamics of sub-events for high-level action recognition, crafting self-explanatory, independent static and motion features—in which, motion features should capture micro-level actions of each actor or object independently, such as arm or leg movements—, and optimum fusing of static and motion features for better accuracy.

A complex activity typically comprises several sub activities. The existing approaches try to classify a video treating it as a single, high-level activity [**?**], [**?**], [**?**], [**?**]. As the action becomes complex, the behavior and the temporal evolution of its underlying sub-events become complicated. For example, cooking may involve sub-events: cutting, turning on the cooker, and stirring. It may not always preserve this same lower order for the same action class. Instead, they may contain a higher-order temporal relationship among them. For example, turning on the cooker and cutting may appear in reverse order in another video in the same class. Therefore, this temporal pattern of sub-events is not easy to capture through a simple time series analysis. These patterns can only be identified by observing a large number of examples, using a system which has an infinite dynamic response. Therefore, it is important to model this higher-order temporal progression, and capture temporal dynamics of these sub-events for better recognition of complex activities.

In contrast to image classification, the information contained in videos are not in a single domain. Both the motion patterns of the actors and objects, as well as the static information—such as, background and still objects that the actors interact with—are important for determining an action. For example, the body movements of a group of people fighting may closely relate to the body movements of a sports event, e.g., wrestling. In such a case, it is tough to distinguish between the two activities solely by looking at the motion patterns. Inspecting the background setting and objects is crucial in such a scenario. Therefore, it is necessary to engineer powerful features from both motion and static domains. In addition, these features must be complementary, and one feature domain should not influence the other domain. In other words, they should be self-explanatory, and mutually exclusive, as much as possible. Also, these motion features should be able to capture activities of each actor independently. If so, the distributional properties of these actions, over short and long durations, can be used to identify high level actions. In the proposed method, we craft static and motion features to have this property.

Furthermore, how to optimally combine or fuse these motion and static descriptors remains a challenge for three key reasons: both static and motion information provide cues regarding an action in a video, the contribution ratio from each domain affects the final recognition accuracy, and optimum contribution ratio of static and motion information may depend on the data set. The combined descriptor should contain the essence of both domains, and should not have a negative influence on each other. Recently, there has been attempts to answer this question [**?**], [**?**]. However, these existing methodologies lack the insight in to how much this ratio affects the final accuracy, and has no control over this ratio. One

major requirement of the fusion method is to have a high-level intuitive interpretation on how much contribution each domain provides to the final descriptor, and have the ability to control level of contribution. This ability makes it possible to deeply investigate the optimum contribution of each domain for a better accuracy. In this study, we investigate this factor extensively.

In this work, we focus on video activity recognition using both static and motion information. We address three problem areas: crafting mutually exclusive static and motion features, optimal fusion of the crafted features, and modeling temporal dynamics of sub-activities for high-level activity recognition. In order to examine the temporal evolution of sub-activities subsequently, we create video segments with constant overlapping durations. Afterwards, we combine static and motion descriptors to represent each segment. We propose *motion tubes*, a dense trajectory [**?**] based tracking mechanism, to identify and track candidate moving areas separately. This enables independent modeling of the activities in each moving area. In order to capture motion patterns, we use histogram oriented optic flows (HOOF) [**?**] inside motion tubes. Then we apply a bag-of-words method on the generated features to capture the distributional properties of micro actions, such as body movements, and create high level, discriminative motion features.

Inspired by the power of object recognition of convolutional neural networks (CNNs), we create a seven-layer deep CNN, and pre-train it on the popular Imagenet data set [**?**]. Afterwards, we use this trained CNN to create deep features, to synthesize static descriptors. Then, using a computationally efficient, yet powerful, mathematical model, we fuse static and motion feature vectors. We propose three such novel methods in this paper: based on Cholesky decomposition, variance ratio of motion and static vectors, and principal components analysis (PCA). The Choleskey decomposition based model provides the ability to precisely control the contribution of static and motion domains to the final fused descriptor. Using this intuition, we investigate the optimum contribution of each domain, experimentally. The variance ratio based method also provides us this ability, and additionally, lets us obtain the optimum contribution ratio mathematically. We show that the optimum contribution ratio obtained experimentally using the Choleskey based method, matches with the ratio obtained mathematically from the variance based method. Furthermore, we show that this optimum contribution may vary depending on the data set, and affects the final accuracy significantly.

In order to model the temporal progression of sub events, we feed the fused vectors to a long short-term memory (LSTM) network. The LSTM network discovers the underlying temporal patterns of the sub events, and classifies high level actions. We feed the same vectors to a classifier which does not capture temporal dynamics to show that modeling temporal progression of sub events indeed contribute for a better accuracy. We carry out our experiments on the two popular action data sets, UCF-11 [**?**] and Hollywood2 [**?**]. Our results surpass existing state-of-the-art results on these data sets, to the best of our knowledge.

The key contributions of this paper are as follows:

- We propose an end-to-end system, which extracts both static and motion information, fuses, and models the temporal evolution of sub-events, and does action classification.
- We propose a novel, moving actor and object tracking mechanism, called *motion tubes*, which enables the system to track each actor or object independently, and model the motion patterns individually over a long time period. This allows the system to model actions occurring in a video in micro level, and use these learned dynamics at a high level afterwards.
- We propose three novel, simple, and an efficient mathematical models for fusing two vectors, in two different domains, based on Choleskey transformation, variance ratio of motion and static vectors, and PCA. The first two methods provide the ability to govern the contribution of each domain for the fused vector precisely and find the optimum contribution of the two domains, mathematically or empirically. Using this advantage, we prove that both static and motion information are complementary and vital for activity recognition through experiments.
- We prove that the final recognition accuracy depends on the ratio of contribution of static and motion domains. Also we show that this optimum ratio depends on the data set. Hence, it is beneficial to exploit this optimum ratio for a better accuracy.
- We model the underlying temporal evolution of sub-events for complex activity recognition using a LSTM network. We experimentally prove that capturing these dynamics indeed benefits the final accuracy.

With the proposed technique we outperform the existing best results for the popular data sets UCF-11 [**?**] and Hollywood2 [**?**].

II. Related work

There has not been many approaches in activity recognition, which highlight the importance of exclusively engineered static and motion features. Most of the work rely on generating spatio-temporal interest regions, such as, action tubes [**?**], tubelets [**?**], dense trajectory based methods [**?**], [**?**], spatio-temporal extensions of spatial pyramid pooling [**?**], or spatio-temporal low level features [**?**], [**?**], [**?**], [**?**], [**?**], [**?**]. Action tubes [**?**] is quite similar to our motion tubes, but our motion candidate regions are chosen based on more powerful dense trajectories [**?**] instead of raw optic flows. Also, we employ a tracking mechanism of each moving area through motion tubes isolating actions of each actor throughout the video. Our static interest regions are independent from motion, unlike in Gkioxari *et al.* [**?**], where we are able to extract background scenery information using CNNs, for action recognition. A common attribute of these methods is that motion density is the dominant factor for identifying candidate regions. In contrast, we treat motion and static features as two independent domains, and eliminate the dominance factor.

A few attempts has recently been made on exclusive crafting and late fusion of motion and static features. Simonyan *et al.* [**?**] first decomposes a video in to spatial and temporal

components based on RGB and optical flow frames. Then they apply two deep CNNs on these two components separately to extract spatial and temporal information. Each network operates mutually exclusively and performs action classification independently. Afterwards, softmax scores are coalesced by late fusion.

Work done in Feichtenhofer *et al.* [**?**] is also similar. Instead of late fusion, they fuse the two domains in a convolutional layer. Both these approaches rely explicitly on automatic feature generation in increasingly abstract layers. While this has provided promising results on static feature generation, we argue that motion patterns can be better extracted by hand-crafted features. This is because temporal dynamics extend to a longer motion duration unlike spatial variations. It is not possible to capture and discriminate motion patterns in to classes by a system which has a smaller temporal support. There are models which employ 3D convolution [**?**], [**?**], which extends the traditional CNNs into temporal domain. Ramasinghe *et al.* [**?**] apply CNNs on optic flows, and Kim *et al.* [**?**] on low level hand-crafted inputs (spatio-temporal outer boundaries volumes), to extract motion features. However, even by generating hierarchical features on top of pixel level features, it is not easy to discriminate motion classes as the duration extent is short. Also, tracking and modeling actions of each actor separately in longer time durations is not possible with these approaches. Our motion features, on the other hand, are capable of capturing motion patterns in longer temporal durations. Furthermore, with the aid of *motion tubes* our system tracks and models the activities of each moving area separately.

Regarding video evolution, Fernando *et al.* [**?**] postulate a function capable of ordering the frames of a video temporally. They learn a ranking function per video using a ranking machine and use the learned parameters as a video descriptor. Several other methodologies, e.g., HMM [**?**], [**?**], CRF-based methods [**?**], also have been employed in this aspect. These methods model the video evolution in frame level. In contrast, attempts for temporal ordering of atomic events also has been made [**?**], [**?**]. Rohrbach *et al.* [**?**], encode transition probabilities of a series of events statistically with a HMM model. Bhattacharya *et al.* [**?**] identify low level actions using dense trajectories and assign concept identity probabilities for each action. They apply a LDS on these generated concept vectors to exploit temporal dynamics of the low level actions. Li *et al.* [**?**] uses simple dynamical systems [**?**], [**?**] to create a dictionary of low-level spatio-temporal attributes. They use these attributes later as a histogram to represent high level actions. Our method too follows a similar approach, as we also generate descriptors for sub-events and then extract temporal progression of these sub-events. However, instead of a simple statistical model, which has a finite dynamic response, we use a LSTM network [**?**] to capture these dynamics. In action recognition literature, such models are starting appear. In Yue-Hei *et al.* [**?**] the LSTM network models the dynamics of the CNN activations, and in Donahue *et al.* [**?**], the LSTM network learns the temporal dynamics of low level features generated by a CNN.

III. Methodology

A. Overview

This section outlines our approach. The overall methodology is illustrated in Fig. 1.

First, we segment a video in to smaller segments of 15 frames with a constant frame overlap and carry out feature construction pipelining for each of these sub segments, as shown in Fig. 1. We create features for describing both motion and static domains.

For extracting motion features we create *motion tubes* across frames, where we track each moving area along the frames using "action boxes". Action boxes are square regions, which contain significant motion in each frame. We choose candidate areas by first creating dense trajectories for each frame, and then clustering trajectory points preceded by a significant amount of pre-processing. This process is explained in sub-section III-B. These action boxes create motion tubes by linking across the frames. Then we calculate HOOF [**?**] features within these motion tubes and apply a bag-of-features method on these to create a descriptor for each video segment.

For extracting static features we train a deep CNN on Imagenet. Then we apply this CNN on the frames of each video segment to retrieve deep features—output vector from the final softmax layer of the CNN—from it. Then we use these features to create a static descriptor for the video segment. Afterwards, we combine these features using one of the fusion models described in sub-section III-D.

The system can then represent a video as a vector time series, $C = [c_{t_0}, c_{t_1}, \ldots, c_{t_{n-1}}]$, where n is the number of segments. Then we apply an LSTM network on these features and exploit the dynamics of time evolution of the combined vector. Finally we classify the dynamics of this time series and predict actions.

B. Motion Features

This section discusses the detailed methodology of creating motion features, particularly, "Motion tubes", "HOOF", and "BoW" blocks as illustrated in Fig. 1.

1) Low level motion descriptor: A pixel level descriptor is required to identify moving points in the video. Dense trajectories [**?**] is a powerful pixel-level algorithm, which captures and tracks motion across several frames. In this work we choose dense trajectories as the low level descriptor for capturing raw motion.

2) Clustering: Initially we create dense trajectories for every frame in the video. Then in order to isolate each sub area in a frame which contains significant motion we apply DBScan clustering on the calculated trajectory points. Algorithm 1 illustrates our clustering approach. Empirically, we use 8 and 10 as ϵ and MinPoints parameters respectively.

Despite the presence of many regions which contain motion in a video, some are neither significant nor descriptive. Those moving regions can be neglected without loss of performance. Therefore after clustering each trajectory point in to cluster groups non-significant cluster groups are neglected. We do this in order to prevent the algorithm focusing on small random moving areas in a video as well as to limit creation of motion

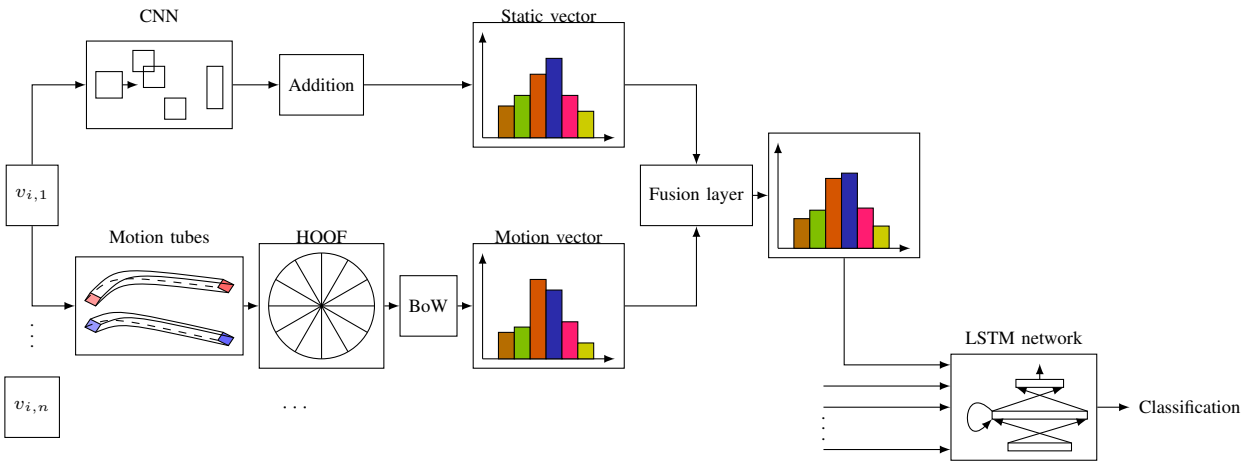

Fig. 1. **Overall methodology**. The whole process consists of five major steps: (i) segmenting a video (ii) crafting static features, (iii) crafting motion features, (iv) fusing static and motion features, and (v) capturing temporal evolution of sub events. Static and motion features are independent and complementary. We generate static and motion features based on a pre-trained CNN and motion tubes respectively, and capture the temporal evolution of sub events using a LSTM network.

tubes to areas which are significant and descriptive. Therefore all the clusters which are not at least 50% of the largest cluster of the frame are discarded.

After identifying the initial candidate clusters for creating action boxes further processing is done to each cluster to ensure that they contain only the important moving areas according to Algorithm 2. For each point the Chebychev distance from the centroid of the cluster is calculated as in Eq. 1. We discard the furthest 20% of the points from the cluster. The reason behind the choice of Chebychev distance over Euclidean distance is due to the possibility of obtaining symmetric square shaped cluster groups as opposed to circular ones. This makes it easier to track moving areas and create motion tubes.

$$D_{Chebychev} = \max(|X_2 - X_1|, |Y_2 - Y_1|) \tag{1}$$

After identifying square-shaped interest regions (action boxes), we represent each of them with a vector, $b = (x, y, r, f)$, where x and y are the coordinates of the top left corner of the box, r is the height or width of the box, and f is the frame number.

3) Motion Tubes: Since our work models the time evolution of sub activities within a video, we divide each input video V_i into temporal segments, $f(V_i) = [v_{i,1}, v_{i,2}, \ldots, v_{i,n}]$, and create features for each individual segment separately. Therefore, after creating the action boxes for each video segment, the action boxes within a segment $v_{i,t}$ can be represented as,

$$g(v_{i,t}) = \big\{[b_{t,1,1}, b_{t,1,2}, \ldots, b_{t,1,q}], \\ [b_{t,2,1}, b_{t,2,2}, \ldots, b_{t,2,p}], \ldots, [b_{t,n,1}, b_{t,n,2}, \ldots, b_{t,n,k}]\big\} \tag{2}$$

where $b_{t,j,k}$ is the k^{th} action box in j^{th} frame of the t^{th} video segment. Note that the number of action boxes differ from frame to frame.

Therefore before linking the action boxes to create motion tubes further pre processing is needed to ensure the same number of action boxes exist in every frame within a video segment. Otherwise, different motion tubes could become joined halfway through the video and the effort to capture dynamics of each moving object separately is disturbed.

First we calculate the mean number of action boxes per frame in each segment. Then we obtain the rounded down value, N, of that number. Afterwards, we iterate through each frame starting from frame number 1 until we come to a frame W which has N number of action boxes.

Then from frame W we propagate forward and backward along the frames, either to eliminate or add action boxes. The procedure is explained below.

If the action box count in a particular frame is larger than the previous frame the smallest excess number of action boxes are removed. In the case where the action box count is lower than the previous frame, linear regression (Eq. 3) is used for each x, y and r value of vector $b = (x, y, r, f)$ up to that frame, in order to create artificial action boxes until the number of action boxes matches N.

$$\begin{bmatrix} Y_1 \\ Y_2 \\ .. \\ Y_n \end{bmatrix} = \begin{bmatrix} 1 & X_1 \\ 1 & X_2 \\ ..... \\ 1 & X_n \end{bmatrix} \times \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ .. \\ \epsilon_n \end{bmatrix} \tag{3}$$

where Y and X would be either x, y or r. Then least squares method is used to minimize the error, $\sum \epsilon^2$, and find the values for β_1 and β_0. Afterwards, using these values missing action boxes are predicted. Note that n increases as the observed action boxes increase; hence the prediction becomes more accurate. Note how this processing results in Eq. 2 being transformed in to Eq. 4, thus verifying that the number of action boxes per frame is equal for all frames within a video segment.

$$h(g(v_{i,t})) = \big\{[b_{t,1,1}, b_{t,1,2}, \ldots, b_{t,1,k}], \\ [b_{t,2,1}, b_{t,2,2}, \ldots, b_{t,2,k}], \ldots, [b_{t,n,1}, b_{t,n,2}, \ldots, b_{t,n,k}]\big\} \tag{4}$$

Algorithm 1 DBScan Clustering algorithm.

Require: $\mathcal{D}$ ▷ Data set of sub-areas in frame
Require: M ▷ Min. no. of points
Require: ϵ ▷ Max. cluster radius
$c \leftarrow 0$ ▷ initialize cluster no.
for each $P \in \mathcal{D}$ **do**
  if P is $visited$ **then**
    continue
  end if
  mark P as $visited$
  NeighborPts = GETALLPOINTS(P, ϵ)
  if size(NeighborPts) $< M$ **then**
    mark P as noise
  else
    c = next cluster
    ADDTOCLUSTER(P, NeighborPts, c, ϵ, M)
  end if
end for
function ADDTOCLUSTER(P, NeighborPts, c, ϵ, MinPoints)
  add P to cluster c
  for each point $np \in$ NeighborPts **do**
    if P is not $visited$ **then**
      mark np as $visited$
      NeighborPts' = GETALLPOINTS(np, ϵ)
      if size(NeighborPts) $\geq$ MinPoints **then**
        NeighborPts' $\leftarrow$ NeighborPts joined with NeighborPts'
      end if
    end if
    if np is not yet member of any cluster **then**
      add np to cluster c
    end if
  end for
end function
function GETALLPOINTS(P, ϵ) **return** all points within P's ϵ-neighborhood (including P)
end function

Algorithm 2 Boundary noise removal algorithm of clusters.

Require: M_d ▷ Max. Chebichev dist.
Require: $\mathcal{C}$ ▷ Input cluster
totalPoints $\leftarrow$ points within M_d of center of $\mathcal{C}$
currentPoints $\leftarrow$ totalPoints
while $true$ **do**
  if COUNT(currentPoints) $<$ COUNT(totalPoints) $\times 0.8$ **then return** currentPoints
  end if
  $M_d \leftarrow M_d - 1$
  currentPoints $\leftarrow$ points within M_d of center of $\mathcal{C}$
end while

The following procedure is used to link the action boxes in consecutive frames.

Assume $b_{t,k,1}$,$b_{t,k,2}$,...,$b_{t,k,n}$ and $b_{t,k+1,1}$,$b_{t,k+1,2}$,...,$b_{t,k+1,n}$ are action boxes in two consecutive frames at time k and $k+1$. Then the following distance matrix is calculated.

$$D = \begin{bmatrix} D_{11} & D_{12} & D_{13} & \dots & D_{1k} \\ D_{21} & D_{22} & D_{23} & \dots & D_{2k} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ D_{k1} & D_{k2} & D_{k3} & \dots & D_{kk} \end{bmatrix} \tag{5}$$

where $D_{i,j}$ is the Euclidean distance between the centroids of i^{th} action box in k^{th} frame and j^{th} action box in $(k+1)^{th}$ frame. Then u^{th} action box at $k+1$, and 1^{st} action box at k is linked, where u is calculated using,

$$u = \underset{j \in J}{\operatorname{argmin}_j}\{D_{1,j}\}, J = \{1, 2, \dots, l\} \tag{6}$$

Then the 1^{st} row and the u^{th} column are removed from the distance matrix, and we apply the same process repeatedly using Eq. 6 to link each of the action boxes at k with $k+1$.

By this removal process, we avoid combining of motion tubes half-way through the video segment and keep them isolated from each other, which is vital for capturing the dynamics separately for each moving object.

Finally, we create a $(z \times n)$-by-5 matrix M_i—z and n are number-of-frames and number-of-action-boxes-per-frame, respectively—which encodes all the information of motion tubes, in a particular video segment i. The rows of M_i for the k^{th} frame is shown in Eq. 7,

$$M_i = \begin{bmatrix} k & 1 & x_{z,1} & y_{z,1} & r_{z,1} \\ k & 2 & x_{z,2} & y_{z,2} & r_{z,2} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ k & n & x_{z,n} & y_{z,n} & r_{z,n} \end{bmatrix} \tag{7}$$

where the columns represent the frame number, action box number, x coordinate of the top left corner of the action box, y coordinate of the top left corner of the action box, and the width/height of the action box, respectively.

4) Histogram Oriented Optic Flows (HOOF): Since each action box in a particular motion tube may differ in size, we take $R = \max(r_i)$, for $\forall i$, where r_i is the i^{th} action box of the motion tube. Then we redraw the action boxes around their centroids having width or length as R. After identifying the k number of motion tubes (k is a variable) for each video segment $v_{i,n}$, we calculate the optic flows along each motion tube using Lucas *et. al* [**?**]. After that we create HOOF [**?**] for every action box within a motion tube. Each optic flow vector within a spatio-temporal action box within a motion tube is binned according to its primary angle from the horizontal axis and weighted according to its magnitude. Therefore, all optical flow vectors, $z = [x, y]^T$ with direction, $\theta = \tan^{-1}(\frac{x}{y})$ in the range,

$$-\frac{\pi}{2} + \pi\frac{b-1}{B} \leq \theta < -\frac{\pi}{2} + \pi\frac{b}{B} \tag{8}$$

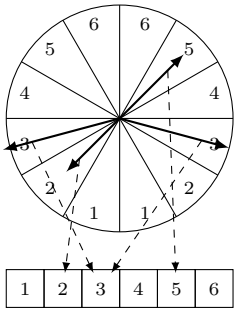

Fig. 2. Example HOOF generation with 6 bins

will contribute a weight of $\sqrt{x^2+y^2}$ to the sum in bin b, $1 \leq b \leq B$ out of a total of B bins. Finally, the histogram is normalized. We choose 100 number of bins. Example HOOF creation for 6 bins is illustrated in Fig. 2.

5) Bag of HOOFs: We use a bag of features method to create a motion descriptor for each video segment. First, we create a code book for HOOF vectors. 100,000 vectors are randomly selected from all the HOOF vectors of all the video segments in all video classes. Then these 100,000 vectors are clustered using k-means clustering and 1000 cluster heads are identified. We choose the number of cluster heads as 1000, because the dimensions of final motion descriptors are needed to be the same as the static descriptors, which is explained in section V. Then for each video segment $v_{i,n}$, a histogram is calculated as follows.

we calculate,

$$p = \underset{j \in J}{\operatorname{argmin}}_j (T_j - h_{n,k}), J = \{1, 2, \ldots, 1000\} \tag{9}$$

for each k in $\{1, 2, \ldots, l\}$, where $h_{n,k}$ is the k^{th} HOOF vector of the n^{th} video segment, and T_j is the j^{th} cluster head. Then we increment the histogram values as,

$$H_n(p) = H_n(p) + 1 \tag{10}$$

where $H_n(p)$ is the p^{th} value, $1 \leq p \leq 1000$, of histogram of the n^{th} video segment $v_{i,n}$. After calculating the histogram vector H_n for every video segment $v_{i,n}$ this $H = [H_1, H_2, \ldots, H_n]$ is the vector time series, which encodes the time evolution of motion information in the video.

C. Static Features

In this sub section we discuss crafting of static features. This section relates to the "CNN" and "Addition" blocks in Fig. 1.

In order to create static descriptors, we create a CNN of 1000 output classes, and train it on the Imagenet data set. The architecture is shown in figure 3. After training, we apply the trained model on each individual frame of each video segment. Then we average the output vectors of the CNN along indexes and obtain a static descriptor s_i for each video segment v_i. Following the same procedure for every v_i, finally, we develop a vector time series, $S = [s_1, s_2, \ldots, s_n]$, representing the static time evolution of the whole video.

D. Fusing of Static and Motion Features

This work depends on three factors; both static and motion information are vital for action recognition, but the final accuracy depends on the ratio of contribution of each domain, and the optimum contribution may depend on the data set. We derive our fusion models addressing all these aspects. The three mathematical models we used to fuse the static and motion vectors are described next. This sub section relates to the "Fusion layer" block in Fig. 1.

1) Cholesky Transformation Based Method: This derivation is based on the Cholesky transformation. An abstract version of Cholesky transformation is described below.

Let P and Q be two random variables of unknown correlation. These random variables can be transformed into two new random variables (R and S) with a known correlation of ρ, where the value of ρ can be chosen at will. The transformation can be performed as follows.

$$\begin{bmatrix} Y \\ Z \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ \rho & \sqrt{1-\rho^2} \end{bmatrix} \times \begin{bmatrix} P \\ Q \end{bmatrix} \tag{11}$$

Therefore,

$$Y = P \tag{12}$$

and

$$Z = \rho P + \sqrt{1-\rho^2} Q \tag{13}$$

The Cholesky transformation guarantees that the correlation between the two random variables Y and Z is ρ.

Based on the above properties of the Cholesky transformation, we propose the following methodology to fuse the static and motion vectors.

Let S and M be static and motion vectors, respectively. Cholesky transformation can be applied to the two vectors S and M with the correlation value ρ_1.

$$\begin{bmatrix} Y \\ Z \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ \rho_1 & \sqrt{1-\rho_1^2} \end{bmatrix} \times \begin{bmatrix} S \\ M \end{bmatrix} \tag{14}$$

$$Y = S \tag{15}$$

$$Z = \rho_1 M + \sqrt{1-\rho_1^2} M \tag{16}$$

Similarly, the transformation can be applied to M and S with the correlation value ρ_2.

$$\begin{bmatrix} A \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ \rho_2 & \sqrt{1-\rho_2^2} \end{bmatrix} \times \begin{bmatrix} M \\ S \end{bmatrix} \tag{17}$$

$$A = M \tag{18}$$

$$B = \rho_2 S + \sqrt{1-\rho_2^2} S \tag{19}$$

Again, the Cholesky transformation guarantees the following two properties.

1) The correlation between S and Z is ρ_1.
2) The correlation between M and B is ρ_2.

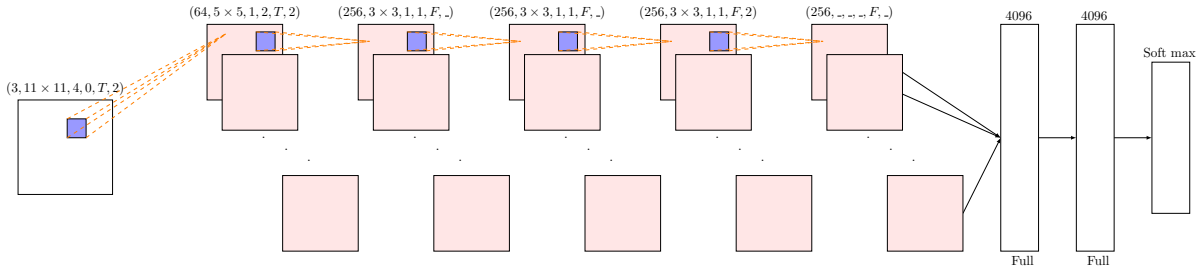

Fig. 3. CNN architecture used for generating static features. The CNN consists of five convolution layers, two fully connected layers, and one softmax layer. The details of each convolutional layer are provided on top of each layer according to the following format:(number of convolution layers × filter width × filter height, convolution stride, spatial padding, is Local Response Normalization added, max-pooling factor). Value above fully connected layers indicate the dimensionality of the layer. We use ReLu as the activation function.

Therefore, if the values of ρ_1 and ρ_2 are chosen in such a way that they obey the following rule,

$$\rho_2 = \sqrt{1 - \rho_1^2} \tag{20}$$

it can be guaranteed that $Z = B, \forall S, M, \rho_1, \rho_2$. Hence, the resultant vector C can be obtained by,

$$C = Z = B \tag{21}$$

where the correlation between C and S is ρ_1 and the correlation between C and M is ρ_1. Here S and M represent the static and the motion vectors whereas C represents the resultant vector. This derivation leads us to an important intuition: by choosing the value of ρ_1, we can choose the degree to which the static features and the motion features contribute, in deriving the resultant vector. In section 4, it is shown, how this property is used to explore, the optimal contribution of static and motion domain information for recognizing actions.

2) Variance Ratio Based Method: The second method employed to combine the motion and static vectors is based on a Gaussian probability model. We model each vector as a histogram. Further, we assume the two vectors are independent to each other. This assumption is fair as the value of the correlation coefficient between the two vectors is negligible. Using the histogram model, the mean and variance of each vector are calculated in order to fit the two vectors into Gaussian distributions. The joint Gaussian distribution is then computed based on this data.

$$G_{sm}(N) = \frac{1}{2\pi\sigma_m\sigma_s} e^{-\left[\frac{[N-\mu_s]^2}{2\sigma_s^2} + \frac{[N-\mu_m]^2}{2\sigma_m^2}\right]} \tag{22}$$

This computation corresponds to the evaluation line obtained when equating the two random variables. This evaluation line is the diagonal through the origin of the static vector vs motion vector plot. The combined distribution is then obtained through this process.

It must be noted that both histograms (corresponding to static and motion components) do not contain equal information. Hence varying the contribution of each histogram to the resultant distribution is necessary. This requires varying of the evaluation line which can be achieved through scaling of the motion and static axes. This scaling process is carried out by the following matrix.

$$\begin{bmatrix} \frac{\sigma_s}{\sigma_s+\sigma_m} & 0 \\ 0 & \frac{\sigma_m}{\sigma_s+\sigma_m} \end{bmatrix}$$

We may conclude that higher variance of a component along one axis reflects lower detail in the model with regards to the other axis. Considering the motion axis, the contribution of this vector towards the resultant vector may be defined by $(1 - \frac{\sigma_m}{\sigma_m+\sigma_s})$. A high variance always corresponds to a flatter histogram containing less detail.

This parameter we derive is significant as it defines the contribution of each individual motion and static vector pair independent of explicit terms. Hence the optimum ratio for combination of motion and static components of a given data set can be mathematically evaluated. With regards to the data sets used for experimenting, 30% of motion vector and 70% of static vector constitute this parameter on average. This mathematical inference is further verified through the experiment results in section IV.

Defining new parameters N_s and N_s as follows, we build a new distribution which is a scaled version of the joint Gaussian distribution obtained previously.

$$N_s = N\frac{\sigma_m}{\sigma_m + \sigma_s}$$
$$N_s = N\frac{\sigma_s}{\sigma_m + \sigma_s}$$

Dropping our initial assumption of independence between the motion and static vectors, we define the following distribution representative of the combined vector.

$$G_{sm}(N) = \frac{e^{-\frac{1}{2(1-\rho^2)}\left[\frac{[N_s-\mu'_s]^2}{2\sigma'^2_s} + \frac{[N_m-\mu'_m]^2}{2\sigma'^2_m} - \frac{2\rho[N_s-\mu'_s][N_m-\mu'_m]}{\sigma'_s\sigma'_m}\right]}}{2\pi\sigma'_m\sigma'_s\sqrt{1-\rho^2}} \tag{23}$$

The corresponding mean and variance of this newly computed distribution are denoted by μ'_s, σ'_s and μ'_m, σ'_m for the static and motion vector respectively.

3) PCA Based Method: The third fusion method is based on Principle Component Analysis (PCA). PCA is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set

of values of linearly uncorrelated variables called principal components. This transformation is defined in such a way that the first principal component has the largest possible variance (that it accounts for as much of the variability in the data as possible), and each succeeding component in turn has the highest variance possible under the constraint that it is orthogonal to the preceding components. The resulting vectors are an uncorrelated orthogonal basis set.

If there are n number of features in the input vector of the PCA, the output of the PCA will provide a new set of n features which are orthogonal and uncorrelated. Also, if $\text{output} = [a_1, a_2, ..., a_n]$, then $\text{var}(a_1) > \text{var}(a_2) > \cdots > \text{var}(a_n)$. Due to the properties of the PCA, the original set of n features can be represented precisely using the first $k(n > k)$ principal components of the output vector, given that the total squared reconstruction error is minimized.

In other words, the essence of the original n dimensional data set is now almost completely represented by the new k dimensional data set with a minor data loss. Thus, the dimension of the data set is reduced from n to k.

In our work, the dimensionality of the data set T is 2 (number of feature domains: static and motion). Using PCA on T, we receive a new set T', with 2 new features domains. We need only one new feature domain in the feature space to represent motion and static domains. Therefore, it is our aim to extract only the first principal component. In order to do this, we need to justify that the first principal component contains a significant majority of the essence of the original data set. In other words, only a negligible amount of data is lost by eliminating the second principal component.

Therefore, we perform PCA on over 15,000 samples of motion and static vectors and plot the variance percentage of the total variance explained by the first and the second principal components respectively, as in figure and figure

It is evident from the figure that the first principal component almost always accounts for over 97% of the essence of the original data set, except for only a negligible amount of samples. The lowest percentage registered is approximately 85%, which is still a significantly high value. Figure 4 shows the percent standard deviation values for the first and second components of the PCA for the two data sets.

Therefore, by eliminating the second principal component, only less than 5% of data is lost in average, and hence only the first principal component can be used to accurately represent the original data set.

E. Capturing Temporal Evolution

This sub section relates to the "LSTM network" block in Fig. 1.

In our work, we represent a video as n fixed-length segments (n differs for videos of different lengths) with overlapping frames. Each segment is represented with a fused vector c_t. Therefore, each video can be represented as a vector time series.

Now each vector time series could be analyzed using traditional time series modeling techniques, such as Auto Regressive Moving Average, to obtain features or model parameters

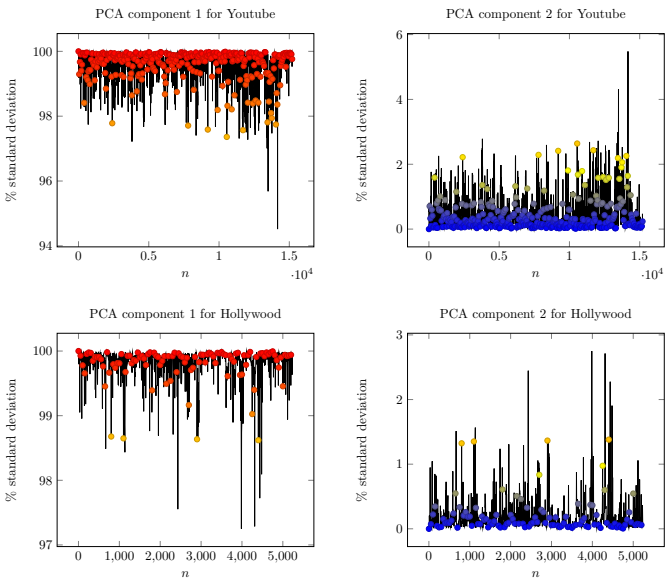

Fig. 4. Percent standard deviation values for the first and second components of the PCA. n is the feature vector index.

that can describe the vector time series. But the main drawback of these methods is that they model current values of a series as a function of past values and have finite dynamic response to time series input. Also they lack the ability to grasp the internal state representations of a complex time series. RNNs maintain hidden layers with directed feedback connections, and hence, have an infinite dynamic response. While training, it learns internal states of a sequence and usually performs better in modeling complex dynamic temporal patterns of long sequences.

However, it is not ideal to train standard RNNs to solve problems, which require learning of long-term temporal dependencies. This is because of the vanishing-gradient problem, which occurs due to the exponential decay of gradient loss of the function with respect to time.

In practice, LSTM networks typically perform better in such cases. LSTM networks are a special type of RNN, which include a "memory cell", and as the name suggests, it can maintain a certain state in memory for longer periods of time. It also has a set of control gates for controlling the removal or addition of information to the cell state. This special architecture gives them the ability to capture longer-term dependencies. First, we revise the operation of an LSTM network.

The most important structure of an LSTM unit is its memory cell c_t, which preserves the state. Basic structure of an LSTM unit is shown in figure 5. The memory cell is self connected, and it has three gates(multiplicative units), i.e., input gate, forget gate and output gate, which are used to control how much to store, remove or output long range contextual information of a temporal sequence.

The detailed activation process of the memory cell and three gates, as shown in Fig. 5 is illustrated as follows:

$$i^t = \sigma(W_{xi}x^t + W_{hi}h^{t-1} + W_{ci}c^{t-1} + b_i) \qquad (24)$$

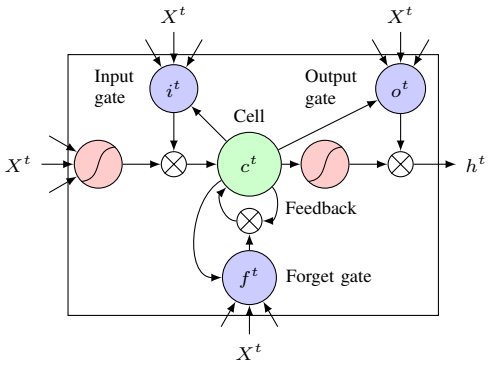

Fig. 5. Long short-term memory (LSTM) block cell. Source [**?**].

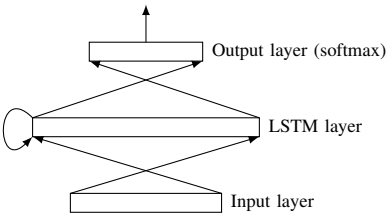

Fig. 6. A simple illustration of the LSTM network. The network consists of an input layer, a 128 unit LSTM layer with 0.8 dropout, and a fully connected softmax output layer.

$$f^t = \sigma(W_{xf}x^t + W_{hf}h^{t-1} + W_{cf}c^{t-1} + b_f) \tag{25}$$

$$c^t = f^t c^{t-1} + i^t tanh(W_{xc}x^t + W_{hc}h^{t-1} + b_c) \tag{26}$$

$$o^t = \sigma(W_{xo}x^t + W_{ho}h^{t-1} + W_{co}c^{t-1} + b_o) \tag{27}$$

$$h^t = o^t \tanh(c^t) \tag{28}$$

where W is the connection weight between two units and $\sigma(\cdot)$ is the sigmoid function.

Since the LSTM network is used only for capturing the temporal dynamic patterns between sub actions, one LSTM layer is enough. Our LSTM network is shown in Fig. 6. The network consists of an input layer, a 128 unit LSTM layer with 0.8 dropout, and a fully connected softmax output layer. As we have a sequence of activities per classification, we use a many to one approach for feeding the fused vectors to the network, as shown in Fig. 7.

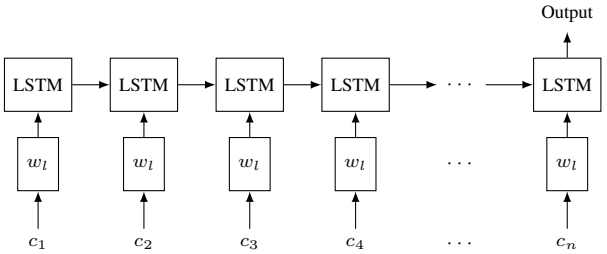

Fig. 7. The process of feeding fused vectors to the LSTM network. c_i indicates the fused vector representing the i_{th} video segment.

IV. Experiments and Results

This section details our experimental methodology and the video data sets used. We evaluate our approach on the two popular data sets UCF-11 and Hollywood2. On both data sets, we show that our work exceeds the current state-of-the-art results. We also vary the contribution of static and motion features for the calculation of combined vector series and explore what is optimum contribution from each domain. We show that optimum contribution may vary depending on the data set. We also show that static and motion features are complementary, and provide vital information about the actions occurring in a video. We compare our three fusion models and show that all the methods are better or on par with existing state-of-the-art. Furthermore, we highlight the importance of considering the time evolution of sub activities in order to identify complex events by comparing the results of LSTM and Random Forest Classification algorithm(which does not capture the temporal dynamics), when applied on our features.

A. Data Sets

Holywood 2 [**?**]: This consists of 12 classes of human actions distributed over 1500 video clips: *answer phone, drive car, eat, fight person, get out car, hand shake, hug person, kiss, run, sit down, sit up,* and *stand up*. The data set is composed of video clips from 69 movies and provides a challenging task, in automatic action detection.

UCF-11 [**?**]: This consists over 1000 sports and home videos from YouTube. This data set contains 11 action classes: *basketball shooting, cycle, dive, golf swing, horse back ride, soccer juggle, swing, tennis swing, trampoline jump, volleyball spike*, and *walk with a dog*. Each of the action sets is subdivided into 25 groups sharing similar environment conditions. This is a challenging data set with camera jitter, cluttered backgrounds and variable illumination.

B. Contribution of Static and Motion Domains

The derivation done in Cholesky based method for fusing the static and motion vectors, provides us an insightful intuition: we can control the contribution of motion and static domains to the fusion vector by varying the ρ value. The derivation of ρ values for different contribution ratios is illustrated in table I

Results for these different contribution values for UCF-11 and Hollywood2 data sets, are shown in table II and table III. We use accuracy and mean average precision as performance metrics, for UCF-11 and Hollywood2, respectively. For UCF-11, we obtain the optimum contribution ratio as 80:20 between static and motion vectors. For Hollywood2, it is 60:40.

An overview distribution of the overall performance over different contribution levels, from static and motion domains, for both data sets is shown in Fig. 9. We can see that the performance change for different contribution percentages of motion and static domain. Also, the optimum contribution may change depending on the nature of the data set. For example, if the motion patterns are indistinguishable across actions, static

TABLE I
DERIVATION OF ρ VALUES FOR DIFFERENT CONTRIBUTION LEVELS OF STATIC AND MOTION DOMAINS TO THE FUSED VECTOR

Contribution to Z	**ρ value**	**Fusion vector**
80% Motion, 20% Static $\rho_1 = 4\rho_2$	$\frac{1}{4}\rho_1 = \sqrt{1-\rho_1^2}$ $\rho_1 = \frac{4}{\sqrt{17}}$	$Z = \frac{4}{\sqrt{17}}M + \frac{1}{\sqrt{17}}S$
60% Motion, 40% Static $2\rho_1 = 3\rho_2$	$\frac{2}{3}\rho_1 = \sqrt{1-\rho_1^2}$ $\rho_1 = \frac{3}{\sqrt{13}}$	$Z = \frac{3}{\sqrt{13}}M + \frac{2}{\sqrt{13}}S$
50% Motion, 50% Static $\rho_1 = \rho_2$	$\rho_1 = \sqrt{1-\rho_1^2}$ $\rho_1 = \frac{1}{\sqrt{2}}$	$Z = \frac{1}{\sqrt{2}}M + \frac{1}{\sqrt{2}}S$
40% Motion, 60% Static $3\rho_1 = 2\rho_2$	$\frac{3}{2}\rho_1 = \sqrt{1-\rho_1^2}$ $\rho_1 = \frac{2}{\sqrt{13}}$	$Z = \frac{2}{\sqrt{13}}M + \frac{3}{\sqrt{13}}S$
80% Motion, 20% Static $4\rho_1 = \rho_2$	$4\rho_1 = \sqrt{1-\rho_1^2}$ $\rho_1 = \frac{1}{\sqrt{17}}$	$Z = \frac{1}{\sqrt{17}}M + \frac{4}{\sqrt{17}}S$

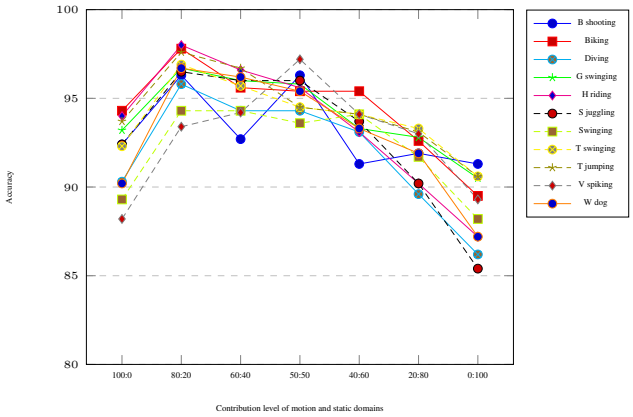

Fig. 8. Accuracy distribution for different contribution levels of motion and static domains. This figure illustrates that motion:static ratio affects the accuracy and the optimum contribution depends on the UCF-11 dataset for each classes.

information plays a critical role, for determining the action, and vise versa. This highlights our hypothesis, that being able to control this contribution explicitly, is vital for an action recognition system.

C. Mathematical Validation of Optimum Contribution

As it is evident from the results of table II and table III, we experimentally obtain the optimum contribution ratio for each data set as 80:20 and 60:40, for UCF-11 and Hollywood2, respectively. In section III, in the derivation of the variance ratio based fusion model, we mathematically obtained values for the optimum contribution as 70:30 and 60:40 for the same data sets. It should be noted that these values closely represent the experimental values, and hence, the results are further verified.

D. Comparison of Fusion Models

The per-class accuracies obtained for each fusion model is illustrated in table IV and table V. Although all three methods give impressive results, Cholesky based fusion model

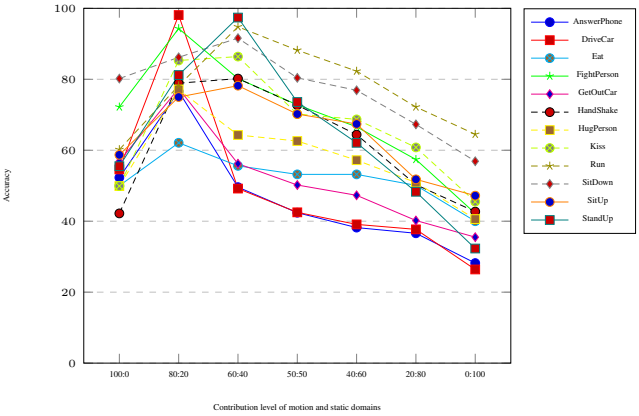

Fig. 9. Accuracy distribution for different contribution levels of motion and static domains. This figure illustrates that motion:static ratio affects the accuracy and the optimum contribution depends on the Hollywood2 data set.

is superior, and has an overall accuracy of 96.3% for UCF-11. For Hollywood2 data set, it achieves a mean average precision of .

E. Comparison with the state-of-the-art

Table VI compares our results to state of the art. We use a motion:static ratio of 20:80 for UCF-11 and 60:40 for Hollywood2 to combine the static and motion vectors, since these values gave the best results for the corresponding data sets. On UCF-11, we significantly outperform the state of the art Ramasinghe *et al.* [**?**] by 3.2%. A mean average precision of % is achieved by our system for Hollywood2, which outperforms the state-of-the-art by %.

Per-action class results, are also compared in table **??** and table VIII. In UCF-11, our method excells in all the classes, when compared with Lucas et al. [**?**], Wang et al. [**?**], Ikizler et al. [**?**] and Ramasinghe *et al.* [**?**]. In Hollywood2, we calculate the average precision of each class, and compare with Lucas et al. [**?**], Wang et al. [**?**], and []. We achieve best results in 8 out of 12 classes in this case.

F. Effectiveness of Capturing Time Evolution

As discussed in earlier sections, complex actions are composed of sub activities preserving a temporal pattern. In this work, we try to capture those underlying patterns by a LSTM network. It is interesting to verify whether this strategy has an impact on the accuracy of the classification. Here we directly feed the fused vectors to a random forest classifier, which does not capture sequential dynamic patterns, and compare it with the results obtained by the LSTM network. The results are shown in Fig. 10 and Fig. 11.

As it is evident from the results in in Fig. 10 and Fig. 11, LSTM network significantly outperforms the random forest classifier for both data sets. In Hollywood2, the LSTM network wins by a 12% margin. In UCF-11, the LSTM network wins by a 12% margin. Therefore, it can be concluded that, exploiting temporal patterns of sub activities, benefits complex action classification.

TABLE II
PER-CLASS ACCURACY FOR DIFFERENT CONTRIBUTION OF STATIC AND MOTION VECTORS FOR UCF-11. THE VECTORS ARE FUSED USING CHOLESKY METHOD. RATIOS ARE INDICATED IN THE FORMAT STATIC:MOTION. HIGHEST ACCURACY FOR UCF-11 IS ACHIEVED USING A 80:20 RATIO BETWEEN STATIC AND MOTION VECTORS.

Class	100:0	80:20	60:40	50:50	40:60	20:80	0:100
B_shooting	92.4%	**96.3%**	92.7%	96.3%	91.3%	91.9%	91.3%
Biking	94.3%	**97.8%**	95.6%	95.4%	95.4%	92.6%	89.5%
Diving	90.3%	**95.8%**	94.3%	94.3%	93.1%	89.6%	86.2%
G_swinging	93.2%	**96.7%**	96.0%	95.8%	93.3%	92.8%	90.5%
H_riding	94.0%	**98.0%**	96.6%	95.6%	93.1%	90.2%	87.2%
S_juggling	92.4%	**96.5%**	96.0%	96.0%	93.7%	90.2%	85.4%
Swinging	89.3%	**94.3%**	94.3%	93.6%	94.1%	91.7%	88.2%
T_swinging	92.3%	**96.9%**	95.7%	94.5%	94.1%	93.3%	90.6%
T_jumping	93.7%	**97.6%**	96.7%	94.5%	94.1%	93.1%	90.6%
V_spiking	88.2%	93.4%	94.2%	**97.2%**	94.1%	93.0%	89.3%
W_dog	90.2%	**96.7%**	96.2%	95.4%	93.3%	91.9%	87.2%
Accuracy	91.8%	**96.3%**	95.3%	95.3%	93.6%	91.8%	88.72%

TABLE III
MAP FOR EACH CLASS FOR DIFFERENT CONTRIBUTION OF STATIC AND MOTION VECTORS TO THE FUSED VECTOR FOR HOLLYWOOD2. RATIOS ARE INDICATED IN THE FORMAT STATIC:MOTION. HIGHEST MAP FOR HOLLYWOOD2 IS ACHIEVED USING A 80:20 RATIO BETWEEN STATIC AND MOTION VECTORS.

Class	100:0	80:20	60:40	50:50	40:60	20:80	0:100
AnswerPhone	52.3%	**76.6%**	49.6%	42.4%	38.2%	36.6%	28.2%
DriveCar	54.6%	**98.1%**	49.2%	42.5%	39.1%	37.7%	26.4%
Eat	50.0%	**62.1%**	55.6%	53.2%	53.2%	50.1%	40.0%
FightPerson	72.2%	**94.3%**	80.2%	72.8%	66.6%	57.4%	42.2%
GetOutCar	56.9%	**77.4%**	56.2%	50.2%	47.3%	40.2%	35.5%
HandShake	42.2%	78.9%	**80.2%**	72.7%	64.4%	50.3%	42.7%
HugPerson	49.9%	**77.1%**	64.3%	62.6%	57.2%	50.9%	40.6%
Kiss	49.9%	85.3%	**86.4%**	70.2%	68.7%	60.8%	45.5%
Run	60.2%	78.2%	**94.8%**	88.2%	82.3%	72.2%	64.5%
SitDown	80.2%	86.2%	**91.6%**	80.4%	76.9%	67.3%	56.9%
SitUp	58.7%	75.0%	**78.2%**	70.2%	67.4%	51.8%	47.2%
StandUp	55.5%	81.2%	**97.4%**	73.6%	62.1%	48.3%	32.3%
mAP	56.9%	80.9%	73.6%	64.9%	60.3%	51.9%	41.8%

TABLE IV
COMPARISON OF FUSION MODELS ON UCF-11 DATA SET.

Class	Cholesky	Variance ratio	PCA
B_shooting	**96.3%**	90.3%	90.6%
Biking	**97.8%**	90.8%	91.0%
Diving	**95.8%**	92.3%	89.3%
G_swinging	**96.7%**	90.3%	92.3%
H_riding	**98.0%**	87.4%	88.6%
S_juggling	**96.5%**	89.7%	92.8%
Swinging	**94.3%**	90.0%	88.0%
T_swinging	**96.9%**	89.4%	93.0%
T_jumping	**97.6%**	92.5%	91.0%
V_spiking	**93.4%**	91.6%	91.7%
W_dog	**96.7%**	91.6%	93.4%
Accuracy	96.3%	90.5%	91.06%

TABLE V
COMPARISON OF FUSION MODELS ON HOLLYWOOD2 DATA SET

Class	Cholesky	Variance ratio	PCA
AnswerPhone	**76.6%**	62.4%	67.6%
DriveCar	**98.1%**	72.8%	70.0%
Eat	**62.1%**	49.4%	56.5%
FightPerson	**94.3%**	78.2%	72.6%
GetOutCar	**77.4%**	46.9%	56.7%
HandShake	**78.9%**	56.9%	55.6%
HugPerson	**77.1%**	52.4.6%	60.6%
Kiss	**85.3%**	64.0%	66.6%
Run	**78.2%**	58.3%	54.3%
SitDown	**86.2%**	72.0%	68.6%
SitUp	**75.0%**	50.0%	54.7%
StandUp	**81.2%**	54.4%	50.0%
mAP	80.9%	59.8%	61.1%

V. CONCLUSION

This paper presents an end to end system for action classification which operates on both static and motion features. Our approach relies on deep features, for creating static vectors, and *motion tubes* for motion features. Motion tubes are a novel concept we introduce in this paper which can be used to track individual actors or objects across frames, and model micro level actions. We present three novel methods: Based on Cholesky transformation, variance ratio, and PCA for efficient combining of features from different domains, which is a vital requirement in action classification. Cholesky method provide

TABLE VI
COMPARISON OF OUR METHOD WITH STATE-OF-THE-ART METHODS IN THE LITERATURE. MOTION:STATIC RATIOS ARE 20:10 AND 60:40 FOR UCF-11 AND HOLLYWOOD2 RESPECTIVELY.

UCF-11		Hollywood2	
Liu *et al.* [**?**]	71.2%	Vig *et al.* [**?**]	59.4%
Ikizler-Cinbis *et al.* [**?**]	75.21%	Jiang *et al.* [**?**]	59.5%
Wang *et al.* [**?**]	84.2%	Mathe *et al.* [**?**]	61.0%
Sameera *et al.* [**?**]	93.1%	Jain *et al.* [**?**]	62.5%
		Wang *et al.* [**?**]	58.3%
		Wang *et al.* [**?**]	64.3%
Our method(Cholesky)	**96.3%**	Our method	**80.9%**

TABLE VII
PER-CLASS ACCURACY COMPARISON WITH STATE-OF-THE-ART ON UCF-11.

Class	Ours(Cholesky)	KLT [?]	Wang et al. [?]	Ikizler-Cinbis [?]	Sameera et. al. [?]
B_shooting	**96.3%**	34.0%	43.0%	48.5%	95.6%
Biking	**97.8%**	87.6%	91.7%	75.17%	93.1%
Diving	95.8%	**99.0%**	**99.0%**	95.0%	92.8%
G_swinging	96.7%	95.0%	**97.0%**	95.0%	95.0%
H_riding	**98.0%**	76.0%	85.0%	73.0%	94.3%
S_juggling	**96.5%**	65.0%	76.0%	53.0%	87.8%
S_winging	**94.3%**	86.0%	88.0%	66.0%	92.4%
T_swinging	**96.9%**	71.0%	71.0%	77.0%	94.9%
T_jumping	**97.6%**	93.0%	94.0%	93.0%	94.0%
V_spiking	93.4%	**96.0%**	95.0%	85.0%	93.2%
W_dog	**96.7%**	76.4%	87.0%	66.7%	91.4%
Accuracy	**96.3%**	79.0%	84.2%	75.2%	93.1%

TABLE VIII
PER-CLASS MAP COMPARISON WITH STATE-OF-THE-ART ON HOLLYWOOD2.

Class	Ours	KLT [?]	Wang et al. [?]	Ullah [?]
AnswerPhone	**76.6%**	18.3%	32.6%	25.9%
DriveCar	**98.1%**	88.8%	88.0%	85.9%
Eat	**62.1%**	73.4%	65.2%	56.4%
FightPerson	**94.3%**	74.2%	81.4%	74.9%
GetOutCar	**77.4%**	47.9%	52.7%	44.0%
HandShake	78.9%	18.4%	29.6%	29.7%
HugPerson	**77.1%**	42.6%	54.2%	46.1%
Kiss	85.3%	65.0%	65.8%	55.0%
Run	78.2%	76.3%	82.1%	69.4%
SitDown	86.2%	59.0%	62.5%	58.9%
SitUp	75.0%	27.7%	20.0%	18.4%
StandUp	81.2%	63.4%	65.2%	57.4%
mAP	**80.9%**	54.6%	58.3%	51.8%

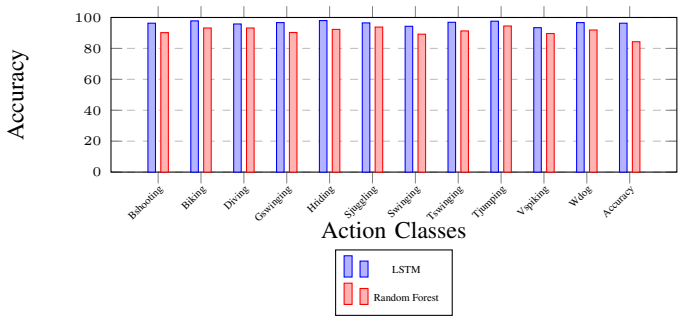

Fig. 10. Accuracy comparison between Random Forest Classifier and LSTM for UCF-11 data set. Motion:static ratio of 20:80 is used. Accuracy is significantly higher when the temporal dynamics of sub events are captured.

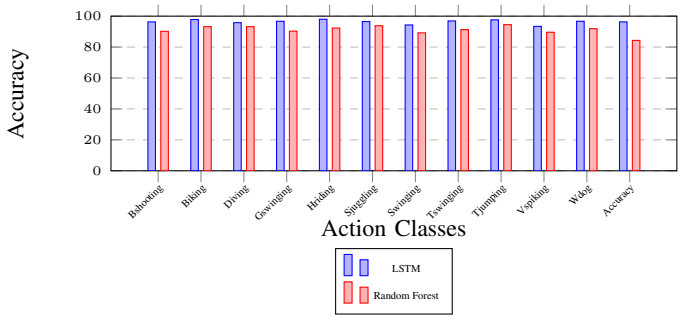

Fig. 11. mAP comparison for Random Forest Classifier and LSTM for Hollywood2 data set. Motion:static ratio of 20:80 is used. mAP is significantly higher when the temporal dynamics of sub events are captured.

the power to control the contribution of each domain in exact numbers, and variance ratio based method mathematically provides an optimum ratio for contribution. We show that these mathematical and experimental values agree with each other. We run experiments to show that the accuracy depends on the ratio of this contribution and the optimum contribution of static and motion domains may vary depending on the data set.

Through our experiments we also show that our static and motion features are complementary, and contribute to the final result. We also compare our three fusion algorithms, and show that Choleky based method is superior, although all three of them give impressive results. We also model the temporal progression of sub-events using an LSTM network. Experimental results indicate that this is indeed beneficiary, compared to using models which does not capture temporal dynamics. Comparison of our work with multiple state-of-the-art algorithms, on the two popular data sets, UCF-11 and Hollywood2, show that our system performs better.

In the future, it would be interesting to improve the motion tubes, so that, it can maintain an identity over each actor object. While it is mostly the case even in the present system, there is no guarantee. Also exploring more powerful methods to describe micro actions inside motion tubes would also be interesting, since it may increase the descriptiveness of the motion features and contribute well to the final accuracy.