arXiv:1811.08815v5 [cs.CV] 6 Nov 2019

Learning Motion in Feature Space: Locally-Consistent Deformable Convolution Networks for Fine-Grained Action Detection

Khoi-Nguyen C. Mac[1], Dhiraj Joshi[2], Raymond A. Yeh[1], Jinjun Xiong[2], Rogerio S. Feris[2], Minh N. Do[1]
[1]University of Illinois at Urbana-Champaign, [2]IBM Research AI
[1]{knmac, yeh17, minhdo}@illinois.edu, [2]{djoshi, jinjun, rsferis}@us.ibm.com

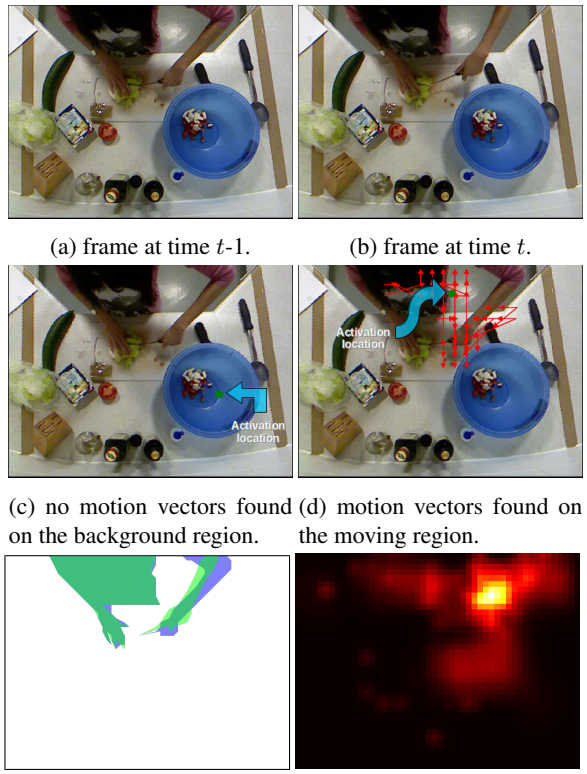
Abstract

Fine-grained action detection is an important task with numerous applications in robotics and human-computer interaction. Existing methods typically utilize a two-stage approach including extraction of local spatio-temporal features followed by temporal modeling to capture long-term dependencies. While most recent papers have focused on the latter (long-temporal modeling), here, we focus on producing features capable of modeling fine-grained motion more efficiently. We propose a novel locally-consistent deformable convolution, which utilizes the change in receptive fields and enforces a local coherency constraint to capture motion information effectively. Our model jointly learns spatio-temporal features (instead of using independent spatial and temporal streams). The temporal component is learned from the feature space instead of pixel space, e.g. optical flow. The produced features can be flexibly used in conjunction with other long-temporal modeling networks, e.g. ST-CNN, DilatedTCN, and ED-TCN. Overall, our proposed approach robustly outperforms the original long-temporal models on two fine-grained action datasets: 50 Salads and GTEA, achieving F1 scores of 80.22% and 75.39% respectively. Source code is available at: https://github.com/knmac/LCDC_release.

1. Introduction

Action detection, *a.k.a* action segmentation, addresses the task of classifying every frame of a given video, containing multiple action segments, as one out of a fixed number of defined categories, including a category for unknown actions. This is contrary to the simpler task of *action recognition*, wherein a given video is pre-segmented and guaranteed to be one of the provided action classes [13].

Fine-grained actions are a special class of actions which can only be differentiated by subtle differences in motion patterns. Such actions are characterized by high inter-class similarity [20, 23], *i.e.* it is difficult, even for humans, to distinguish two different actions just from observing individual frames. Unlike generic action detection, which can largely rely on "what" is in a video frame to perform detection, fine-grained action detection requires additional reason about "how" the objects move across several video frames. In this



(a) frame at time t-1. (b) frame at time t.

(c) no motion vectors found on the background region. (d) motion vectors found on the moving region.

(e) the person at time t-1 (blue) and t (green). (f) visualization of motion in feature space.

Figure 1: Visualization of difference of adaptive receptive fields for action *cutting lettuce* in 50 Salads dataset: (a) and (b) are two consecutive frames; (c) and (d) are motion vectors at background and moving regions (green dots indicate activation locations and red arrows indicate motion vectors); (e) is the manually defined mask of the person at time $t-1$ and t; and (f) is the energy of motion field in feature space, computed by aggregating motion vectors in all deformable convolution layers.

work, we consider the fine-grained action detection setting.

The pipeline of fine-grained action detection generally consists of two steps: (1) spatio-temporal feature extraction and (2) long-temporal modeling. The **first step** models spatial and short-term temporal information by looking at a few consecutive frames. Traditional approaches tackle this problem by decoupling spatial and temporal information in different feature extractors and then combining the two streams with a fusion module. Optical flow is commonly used for such short-term temporal modeling [8, 9, 21, 23, 24]. However, optical flow is usually computationally expensive and may suffer from noise introduced by data compression [15, 16]. Other approaches use Improved Dense Trajectory (IDT) or Motion History Image (MHI) as an alternative to optical flow [5, 16, 28]. Recently, there have been efforts to model motion in video using variants of 3D convolutions [1, 12, 27]. In such cases, motion modeling is somewhat limited by receptive fields of standard convolutional filters [11, 29, 30].

The **second step** models long-term dependency of extracted spatio-temporal features over the whole video, *e.g*. bi-directional LSTM [23], spatial-temporal CNN (ST-CNN) with segmentation models [16], temporal convolutional networks (TCN) [15], and temporal deformable residual networks (TDRN) [17]. Recent works that focused on modeling long-term dependency have usually relied on existing features [15, 16, 17]. In this work, we create efficient short-term spatio-temporal features which are very effective in modeling fine-grained motion.

Instead of modeling temporal information with optical flow, we learn temporal information in the *feature space*. This is accomplished by utilizing our proposed *locally-consistent deformable convolution (LCDC)*, which is an extension of the standard deformable convolution [2]. At a high-level, we model motion by evaluating the local movements in adaptive receptive fields over time (as illustrated in Fig. 1). Adaptive receptive fields can focus on important parts [2] in a frame, thus using them helps focus on movements of interesting regions. On the other hand, traditional optical flow tracks all possible motion, some of which may not be necessary. Furthermore, we enforce a local coherency constraint over the adaptive receptive fields to achieve temporal consistency.

To demonstrate the effectiveness of our approach, we evaluate on two standard fine-grained action detection datasets: 50 Salads [25] and Georgia Tech Egocentric Activities (GTEA) [7]. We also show that our features, without any optical flow guidance, are robust and outperform features from original networks. Additionally, we perform quantitative evaluation of the learned motion using ablation studies to demonstrate the power of our model in capturing temporal information.

Our main contributions are: *(1) Modeling motion in feature space* using changes in adaptive receptive fields over time, instead of relying on pixel space as in traditional optical flow based methods. To the best of our knowledge, we are the first to extract temporal information from receptive fields. *(2) Introducing local coherency constraint* to enforce consistency in motion. The constraint reduces redundant model parameters, making motion modeling more robust. *(3) Constructing a backbone single-stream network to jointly learn spatio-temporal features*. This backbone network is flexible and can be used in consonance with other long-temporal models. Furthermore, we prove that the network is capable of representing temporal information with a behavior equivalent to optical flow. *(4) Significant reduction of model complexity* is achieved without sacrificing performance by using local coherency constraint. This reduction is proportional to the number of deformable convolution layers. Our single-stream approach is computationally more efficient than traditional two-stream networks, as they require expensive optical flow and multi-stream inference.

2. Related work

An extensive body of literature exists for features, temporal modeling, and network architectures within the context of action detection. In this section, we will review the most recent and relevant papers related to our approach.

Spatio-temporal features. Spatio-temporal features are crucial in the field of video analysis. Usually, the features consist of spatial cues (extracted from RGB frames) and temporal cues over a *short* period of time. Optical flow [18] is often used to model temporal information. However, it was found to suffer from noise due to video compression and insufficient to capture small motion [15, 16]. It is also generally computationally expensive. Other solutions to model temporal information include Motion History Image (MHI) [5], leveraging the difference of multiple consecutive frames, and Improved Dense Trajectory (IDT) [28], combining HOG [3], HOF [28], and Motion Boundary Histograms (MBH) descriptors [4].

To combine spatial and (short) temporal components, Lea *et al*. [16] stacked an RGB frame with MHI as input to a VGG-like network to produce features (which they refereed to as SpatialCNN features). Simonyan and Zisserman [21] proposed a two-stream network, combining scores from separate appearance (RGB) and motion streams (stacked optical flows). The original approach was improved by more advanced fusion in [8, 9]. A different school of thought models motion using variants of 3D convolutions including C3D proposed in [27]. Inflated 3D (I3D) network, leveraging 3D convolutions within a two-stream setup was proposed in [1]. To cope with egocentric motion captured by head-mounted cameras, Singh *et al*. introduced a third stream (EgoStream) in [24], capturing the relation of hands, head, and eyes motion. [23] further used four streams (two appearance and two motion streams) in Multi-Stream Network (MSN). Each domain (spatial and

temporal) has a global view (whole frame) and a local view (cropped by motion tracker).

Long-temporal modeling. While spatio-temporal features are usually extracted over short periods of time, some form of long-temporal modeling is performed to capture long-term dependencies within the entirety of a video containing an action sequence. In [15] Spatio-temporal CNN (ST-CNN) was introduced to combine SpatialCNN features using a 1D convolution that spans over a long period of time. Singh *et al*. learned the long-term dependency from MSN features (four-stream) using bi-directional LSTMs [23]. More recently, [15] proposed two Temporal Convolution Networks (TCN): DilatedTCN and Encoder-Decoder TCN (ED-TCN). These networks fused SpatialCNN features and captured long-temporal patterns by convolving them in the time-domain. A Temporal Deformable Residual Networks (TDRN) was proposed in [17] to model long-temporal information by applying a deformable convolution in the time domain. The TCN model was also further improved with multi stage mechanism in Multi-Stage TCN (MS-TCN) [6].

Network architectures. Pre-trained architectures for image classification, such as VGG, Inception, ResNet [10, 22, 26] are the most important determinants of the performance of the main down-stream vision tasks. Many papers have focused on improving the recognition accuracy by innovating on the network architecture. In standard convolutions, the convolutional response always comes from a local region. Dilated convolutions have been introduced to overcome this problem by changing the shape of receptive fields with some dilation patterns [11, 29, 30]. In 2017, Dai *et al*. [2] introduced deformable convolutional networks with adaptive receptive fields. The method is more flexible since the receptive fields depend on input and can approximate an arbitrary object's shape. We leverage on the advances of [2], specifically the adaptive receptive fields from the model to capture motion in the *feature space*. We further add a local coherency constraint on receptive fields in order to ensure that the motion fields are consistent. This constraint also plays a major role in reducing model complexity.

3. Locally-Consistent Deformable Convolution Networks

Our architecture builds upon deformable convolutional networks with an underlying ResNet CNN. While a deformable convolutional network has been shown to succeed in the task of object detection and semantic segmentation, it is not directly designed for fine-grained action detection. However, we observe that deformable convolution layers have a byproduct, the *adaptive receptive field*, which can capture motion very naturally.

At a high level, an adaptive receptive field in a deformable convolution layer can be viewed as an aggregation of important pixels, as the network has the flexibility to

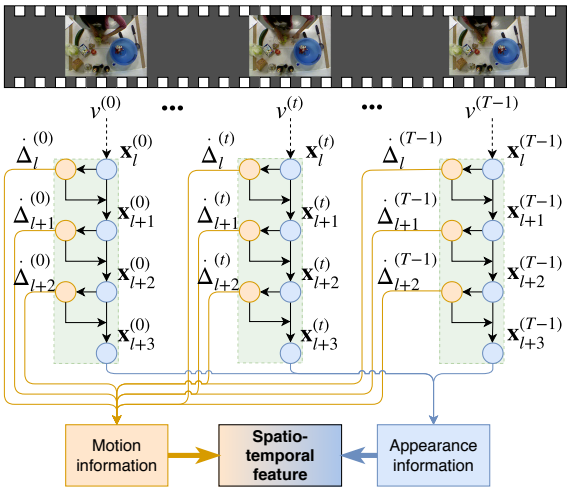


Figure 2: Network architecture of the proposed LCDC across multiple frames $v^{(t)}$. Appearance information comes from the last layer while motion information is extracted directly from deformation $\dot{\Delta}$ in the feature space instead of from a separate optical flow stream. Weights are shared across frames over time.

change where each convolution samples from. In a way, the adaptive receptive fields are performing some form of key-points detection. Therefore, our hypothesis is that, if the key-points are consistent across frames, we can model motion by taking the difference in the adaptive receptive fields across time. As a deformable convolution can be trained end-to-end, our network can learn to model motion at hidden layers of the network. Combining this with spatial features leads to a powerful spatio-temporal feature.

We illustrate the intuition of our method in Fig. 1. The motion here is computed using difference in adaptive receptive fields on multiple *feature spaces* instead of pixel space as in optical flow. Two consecutive frames of action *cutting lettuce* from 50 Salads dataset are shown in Fig. 1a and Fig. 1b. Fig. 1e shows masks of the person to illustrate how the action takes place. We also show the motion vectors corresponding to different regions in Fig. 1c and Fig. 1d. Red arrows are used to describe the motion and green dots are used to show the corresponding activation units. We suppress motion vectors with low values for the sake of visualization. In Fig. 1c, the activation unit lies on a background region (cut ingredients inside the bowl) and so there is no motion recorded as the difference between two adaptive receptive fields of background region over time is minimal. However, we can find motion in Fig. 1d (the field of red arrows) because the activation unit lies on a moving region, *i.e*. the arm region. The motion field at all activation units is seen in Fig. 1f, where the field's energy corresponds to the length of motion vectors at each location. The motion field is excited around the moving region (the arm) while suppressed in the background. Therefore, this highly suggests that the motion information we extract can be used as

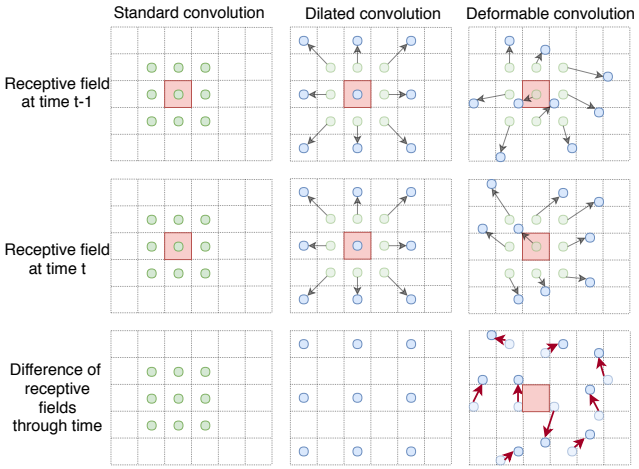


Figure 3: Illustration of temporal information modeled by the difference of receptive fields at a single location in 2D. Only deformable convolution can capture temporal information (shown with red arrows). Related to Eq. (2) and Eq. (3), n is red square, n+k are green dots, $\ddot{\Delta}_{n,k}$ are black arrows, n+k+$\ddot{\Delta}_{n,k}$ are blue dots, and $\ddot{\mathbf{r}}$ are red arrows.

an alternative solution to optical flow. A schematic of the proposed network architecture is shown in Fig. 2.

3.1. Deformable convolution

We first briefly review the deformable convolution layers, before going into a concrete description of the construction of the network architecture. Let $\mathbf{x}$ be the input signal such that $\mathbf{x} \in \mathbb{R}^N$. The standard convolution is defined as:

$$\mathbf{y}[n] = \sum_k \mathbf{w}[-k]\mathbf{x}\left[n+k\right], \tag{1}$$

where $\mathbf{w} \in \mathbb{R}^K$ is the convolutional kernel, n and k are the signal and kernel indices (n and k can be treated as multidimensional indices). The deformable convolution proposed in [2] is thus defined as:

$$\mathbf{y}[n] = \sum_k \mathbf{w}[-k]\mathbf{x}\left(n+k+\ddot{\Delta}_{n,k}\right), \tag{2}$$

where $\ddot{\Delta} \in \mathbb{R}^{N\times K}$ represents the deformation offsets of deformable convolution. These offsets are learned from another convolution with $\mathbf{x}$ *i.e.* $\ddot{\Delta}_{n,k} = (\mathbf{h}_k * \mathbf{x})[n]$, where $\mathbf{h}$ is a different kernel. Note that we use parentheses $(\cdot)$ instead of brackets $[\cdot]$ for $\mathbf{x}$ in Eq. (2) because the index n+k+$\ddot{\Delta}_{n,k}$ requires interpolation as $\ddot{\Delta}$ is fractional.

3.2. Modeling temporal information with adaptive receptive fields

We define the adaptive receptive field of a deformable convolution at time t as $\ddot{\mathbf{F}}^{(t)} \in \mathbb{R}^{N\times K}$ where $\ddot{\mathbf{F}}^{(t)}_{n,k} = n + k + \ddot{\Delta}^{(t)}_{n,k}$. To extract motion information from adaptive receptive fields, we take the difference of the receptive fields through time, which we denote as:

$$\ddot{\mathbf{r}}^{(t)} = \ddot{\mathbf{F}}^{(t)} - \ddot{\mathbf{F}}^{(t-1)} = \ddot{\Delta}^{(t)} - \ddot{\Delta}^{(t-1)}. \tag{3}$$

It can be seen that the locations $n+k$ are canceled, going from $t-1$ to t in Eq. (3), leaving only the difference of deformation offsets. Given T input feature maps with spatial dimension $H \times W$, we can construct T different $\ddot{\Delta}^{(t)}|_{t=0}^{T-1}$, resulting in $T-1$ motion fields $\ddot{\mathbf{r}}^{(t)}|_{t=0}^{T-2}$ with the same spatial dimension. Therefore, we can model different motion at different positions n and time t.

Fig. 3 further illustrates the meaning of $\ddot{\mathbf{r}}^{(t)}$ in 2D for different types of convolutions. Red square shows the current activation location, green dots show the standard receptive fields, and blue dots show the receptive fields after adding deformation offsets. In the last row, red arrows show the changes of receptive field from time $t-1$ (faded blue dots) to time t (solid blue dots). Readers should note that there are no red arrows for standard convolution and dilated convolution because the offsets are either zero or identical. Red arrows only appear in deformable convolution, which motivates modeling of temporal information.

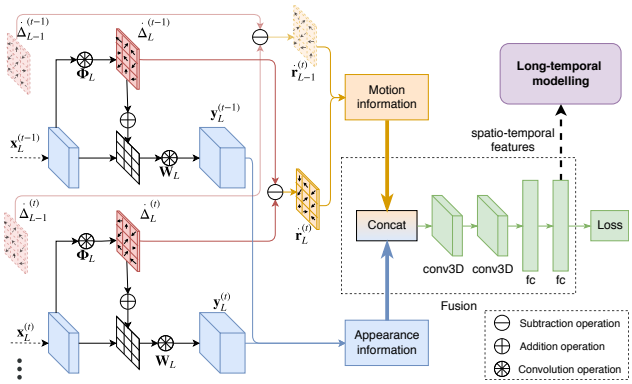


Figure 4: A more detailed view of our network architecture with the fusion module. Appearance information comes from output of the last layer while motion information comes from aggregating $\dot{\mathbf{r}}$ from multiple layers. Outputs of the final fc layer can be flexibly used as the features for any long-temporal modeling networks.

3.3. Locally-consistent deformable convolution

Directly modeling motion using $\ddot{\mathbf{r}}$ is not very effective because there is no guarantee of local consistency in receptive fields in the original deformable convolution formulation. This is because $\ddot{\Delta}_{n,k}$ is defined on both location (n) and kernel (k) indices, which essentially corresponds to $\mathbf{x}[m]$, where $m = n+k$. However, there are multiple ways to decompose m, *i.e.* $m = n+k = (n-l)+(k+l)$, for any l. Therefore, one single $\mathbf{x}[m]$ is deformed by multiple $\ddot{\Delta}_{n-l,k+l}$, with different l. This produces inconsistency

when we model $\ddot{\mathbf{r}}^{(t)}$ in Eq. (3), as there can be multiple motion vectors corresponding to the same location. While local consistency could be learned as a side-effect of the training process, it is still not explicitly formulated in the original deformable convolution formulation.

In order to enforce consistency, we propose a locally-consistent deformable convolution (LCDC):

$$\mathbf{y}[n] = \sum_k \mathbf{w}[-k]\mathbf{x}\left(n + k + \dot{\Delta}_{n+k}\right), \tag{4}$$

for $\dot{\Delta} \in \mathbb{R}^N$. LCDC is a special case of deformable convolution where

$$\ddot{\Delta}_{n,k} = \dot{\Delta}_{n+k}, \quad \forall n, k. \tag{5}$$

We name this as *local coherency constraint*. The interpretation of LCDC is that instead of deforming the receptive field as in Eq. (2), we can deform the input signal instead. Specifically, LCDC in Eq. (4) can be rewritten as:

$$\mathbf{y}[n] = \sum_k \mathbf{w}[-k]\tilde{\mathbf{x}}[n+k] = (\tilde{\mathbf{x}} * \mathbf{w})[n], \tag{6}$$

where

$$\tilde{\mathbf{x}}[n] = (D_{\dot{\Delta}}\{\mathbf{x}\})[n] = \mathbf{x}\left(n + \dot{\Delta}_n\right) \tag{7}$$

is a deformed version of $\mathbf{x}$ and $*$ is the standard convolution ($D_{\dot{\Delta}}\{\cdot\}$ is defined as the deforming operation by offset $\dot{\Delta}$).

Both $\ddot{\Delta}$ and $\dot{\Delta}$ are learned via a convolution layer. Recall that $\ddot{\Delta}_{n,k} = (\mathbf{h}_k * \mathbf{x})[n]$, where $\mathbf{x} \in \mathbb{R}^N$ and $\ddot{\Delta} \in \mathbb{R}^{N \times K}$. $\dot{\Delta}$ is constructed similarly, *i.e.*

$$\dot{\Delta}_n = (\Phi * \mathbf{x})[n], \tag{8}$$

where $\dot{\Delta} \in \mathbb{R}^N$. Since $\ddot{\Delta}$ and $\dot{\Delta}$ share the same spatial dimension N and they can be applied for different time frames, $\dot{\Delta}$ can also model motion at different positions and times.

Furthermore, $\dot{\Delta}$ only needs a kernel Φ, while $\ddot{\Delta}$ requires multiple $\mathbf{h}_k$. Therefore, LCDC is more memory-efficient as we can reduce memory cost K times. Implementation-wise, given input feature map $\mathbf{x} \in \mathbb{R}^{H \times W \times C}$, then $\ddot{\Delta} \in \mathbb{R}^{(H \times W) \times (G \times K_h \times K_w \times 2)}$, where G is the number of deformable groups, K_h and K_w are the height and width of kernels, and 2 indicates that offsets are 2D vectors. However, the dimensionality of LCDC offsets $\dot{\Delta}$ is only $\mathbb{R}^{H \times W \times 2}$. We also drop the number of deformable groups G since we want to model one single type of motion between two time frames. Therefore, the reduction in this case is $G \times K_h \times K_w$ times. The parameter reduction is proportional to the number of deformable convolution layers that are used.

We now show that LCDC can effectively model both appearance and motion information in a single network, as the difference $\dot{\mathbf{r}}^{(t)} = \dot{\Delta}^{(t)} - \dot{\Delta}^{(t-1)}$ has a behavior equivalent to motion information produced by optical flow.

Proposition 1. *Suppose that two inputs $\mathbf{x}^{(t-1)}$ and $\mathbf{x}^{(t)}$ are related through a motion field,* i.e.

$$\mathbf{x}^{(t)}(s) = \mathbf{x}^{(t-1)}\left(s - o(s)\right), \tag{9}$$

where $o(s)$ is the motion at location $s \in \mathbb{R}^2$, and $\mathbf{x}^{(t)}$ is assumed to be locally varying. Then the corresponding LCDC outputs with $\mathbf{w} \neq 0$:

$$\mathbf{y}^{(t)} = (D_{\dot{\Delta}^{(t)}}\{\mathbf{x}^{(t)}\}) * \mathbf{w},$$
$$\mathbf{y}^{(t-1)} = (D_{\dot{\Delta}^{(t-1)}}\{\mathbf{x}^{(t-1)}\}) * \mathbf{w}$$

are consistent, i.e. $\mathbf{y}^{(t-1)} = \mathbf{y}^{(t)}$, if and only if $\forall n$,

$$\dot{\mathbf{r}}_n^{(t)} = \dot{\Delta}_n^{(t)} - \dot{\Delta}_n^{(t-1)} = o\left(n + \dot{\Delta}_n^{(t)}\right). \tag{10}$$

Notice that in pixel space, $\mathbf{x}$ are input images and $o(s)$ is the optical flow at s. In latent space, $\mathbf{x}$ are intermediate feature maps and $o(s)$ is the motion of feature.

Proof. With the connection of LCDC to standard convolution, under the assumption that $\mathbf{w} \neq 0$, we have:

$$\begin{aligned} &\mathbf{y}^{(t)} = \mathbf{y}^{(t-1)} \\ \Leftrightarrow\ &D_{\dot{\Delta}^{(t)}}\{\mathbf{x}^{(t)}\} = D_{\dot{\Delta}^{(t-1)}}\{\mathbf{x}^{(t-1)}\} \\ \Leftrightarrow\ &\mathbf{x}^{(t)}\left(n + \dot{\Delta}_n^{(t)}\right) = \mathbf{x}^{(t-1)}\left(n + \dot{\Delta}_n^{(t-1)}\right), \forall n. \end{aligned}$$

Substituting the LHS in the motion relation in Eq. (9), we obtain the following equivalent conditions $\forall n$:

$$\begin{aligned} \mathbf{x}^{(t-1)}\left(n + \dot{\Delta}_n^{(t)} - o(n + \dot{\Delta}_n^{(t)})\right) &= \mathbf{x}^{(t-1)}\left(n + \dot{\Delta}_n^{(t-1)}\right) \\ \Leftrightarrow \dot{\Delta}_n^{(t)} - o(n + \dot{\Delta}_n^{(t)}) &= \dot{\Delta}_n^{(t-1)} \\ \Leftrightarrow o\left(n + \dot{\Delta}_n^{(t)}\right) &= \dot{\Delta}_n^{(t)} - \dot{\Delta}_n^{(t-1)} = \dot{\mathbf{r}}_n^{(t)}. \end{aligned}$$

(since $\mathbf{x}^{(t)}$ is locally varying). □

The above result shows that by enforcing consistent output and sharing weights $\mathbf{w}$ across frames, the learned deformed map $\dot{\Delta}_n^{(t)}$ encodes motion information, as in Eq. (10). Hence, we can effectively model both appearance and motion information in a single network with LCDC, instead of using two different streams.

3.4. Spatio-temporal features

To create the spatio-temporal feature, we further concatenate across channel dimensions the learned motion information $\dot{\mathbf{r}}^{(t)}$ from multiple layers with appearance features (output of the last layer $\mathbf{y}_L^{(t)}$). We illustrate this process in Fig. 4. To model the fusion mechanism, we used two 3D convolutions followed by two fc layers. Each 3D convolution unit was followed by batch normalization, ReLU activation, and 3D max pooling to gradually reduce temporal dimension (while the spatial dimension is retained). Outputs of the final fc layer can be flexibly used as the features for any long-temporal modeling networks, such as ST-CNN [16], Dilated-TCN [15], or ED-TCN [15].

4. Experiments

4.1. Implementation details

We implemented our approach using ResNet50 with deformable convolutions as backbone (at layers `conv5a`, `conv5b`, and `conv5c` as in [2]). Local coherency constraints were added on all existing deformable convolutions layers. For the fusion module, we used a spatial kernel with size 3 and stride 1; and temporal kernel with size 4 and stride 2. We also used pooling with size 2 and stride 2 in 3D max pooling. Temporal dimension was collapsed by averaging. The network ended with two fully connected layers. Standard cross-entropy loss with weight regularization was used to optimize the model. After training, LCDC features (last fc layer) were extracted and incorporated into long-temporal models. All data cross-validation splits followed the settings of [15]. Frames were resized to 224x224 and augmented using random cropping and mean removal. Each video snippet contained 16 frames after sampling. For training, we downsampled to 6fps on 50 salads and 15 fps on GTEA, because of different motion speeds, to make sure one video snippet contained enough information to describe motion. For testing, features were downsampled with the same frame rates as other papers for comparison. We used the common Momentum optimizer [19] (with momentum of 0.9) and followed the standard procedure of hyper-parameter search. Each training routine consisted of 30 epochs; learning rate was initialized as 10^{-4} and decayed every 10 epochs with a decaying rate of 0.96.

4.2. Datasets

We evaluate our approach on two standard datasets, namely, 50 Salads dataset and GTEA dataset.

50 Salads Dataset [25]: This dataset contains 50 salad making videos from multiple sensors. We only used RGB videos in our work. Each video lasts from 5-10 minutes, containing multiple action instances. We report results for *mid* (17 action classes) and *eval* granularity level (9 action classes) to be consistent with results reported in [15, 16, 17].

Georgia Tech Egocentric Activities (GTEA) [7]: This dataset contains 28 videos of 7 action classes, performed by 4 subjects. The camera in this dataset is head-mounted, thus introducing more motion instability. Each video is about 1 minute long and has around 19 different actions on average.

4.3. Baselines

We compare LCDC with several baselines including (1) methods which do not involve long-temporal modeling where comparison is at spatio-temporal feature level (SpatialCNN) and (2) methods with long-temporal modeling (ST-CNN, DilatedTCN, and ED-TCN).

SpatialCNN [16]: a VGG-like model that learns both spatial and *short-term* temporal information by stacking an RGB frame with the corresponding MHI (the difference between frames over a *short* period of time). MHI is used for both 50 Salads and GTEA datasets instead of optical flow as optical flow was observed to suffer from small motion and data compression noise [15, 16]. SpatialCNN features are also used as *inputs* for ST-CNN, DilatedTCN, ED-TCN, and TDRN.

ST-CNN [16], DilatedTCN [15], and ED-TCN [15]: are long-temporal modeling frameworks. Long-term dependency was modeled using a 1D convolution layer in ST-CNN, stacked dilated convolutions in DilatedTCN, and an encoder-decoder with pooling and up-sampling in ED-TCN. All three frameworks were originally proposed with SpatialCNN features as their input. We incorporated LCDC features into these long-temporal models and compared with the original results.

We obtained the publicly available implementations of ST-CNN, DilatedTCN, and ED-TCN from [14]. On incorporating LCDC features into these models, we observed that training from scratch can become sensitive to random initialization. This is likely because these long-temporal models have a low complexity (*i.e.* only a few layers) and the input features are not augmented. We ran each long-temporal model (with LCDC features) five times and report means and standard deviations over multiple metrics. For completeness, we have also included original results from TDRN (where the input was SpatialCNN features as well) [17]. However, TDRN's implementation was not publicly available so we were unable to incorporate LCDC with TDRN.

4.4. Results

We benchmark our approach using three standard metrics reported in [15, 17]: frame-wise accuracy, segmental edit score, and F1 score with overlapping of 10% (F1@10). Since edit and F1 scores penalize over-segmentation, accuracy metric is more suitable to evaluate the quality of short-term spatio-temporal features (SpatialCNN and LCDC). All mentioned metrics are sufficient to assess the performance of long-temporal models (ST-CNN, DilatedTCN, ED-TCN, and TDRN). We have also specified inputs for spatial and short-term temporal components, as well as the long-temporal model in each setup (Tab. 1 and Tab. 2).

Tab. 1 shows the results on 50 Salads dataset on both granularity levels. Overall performance of LCDC setups, with long-temporal models, outperform their counterparts. We highlight our LCDC + ED-TCN setups as they provided the most significant improvement over other baselines. Compared to the original ED-TCN, which used SpatialCNN features, our approach increases by 5.75%, 7.14%, 7.42% on mid-level and 3.72%, 2.36%, 5.5% on eval-level, in terms of F1@10, edit score, and accuracy. Tab. 2 shows the results on GTEA dataset and is organized in a fashion similar to Tab. 1. We achieve the best performance when incorporating LCDC features with ED-TCN framework out

	Model	Spatial comp	Temporal comp (short)	Long-temporal	F1@10	Edit	Acc
Mid	SpatialCNN [16]	RGB	MHI	-	32.3	24.8	54.9
	(SpatialCNN) + ST-CNN [16]	RGB	MHI	1D-Conv	55.9	45.9	59.4
	(SpatialCNN) + DilatedTCN [15]	RGB	MHI	DilatedTCN	52.2	43.1	59.3
	(SpatialCNN) + ED-TCN [15]	RGB	MHI	ED-TCN	68.0	59.8	64.7
	(SpatialCNN) + TDRN [17]	RGB	MHI	TDRN	(72.9)	(66.0)	(68.1)
	LCDC	RGB	Learned deformation	-	43.99	33.38	67.27
	LCDC + ST-CNN	RGB	Learned deformation	1D-Conv	60.01±0.42	51.35±0.12	68.45±0.15
	LCDC + DilatedTCN	RGB	Learned deformation	DilatedTCN	58.21±0.59	48.54±0.52	69.28±0.25
	LCDC + ED-TCN	RGB	Learned deformation	ED-TCN	**73.75±0.54**	**66.94±1.33**	**72.12±0.41**
Eval	Spatial CNN [16]	RGB	MHI	-	35.0	25.5	68.0
	(SpatialCNN) + ST-CNN [16]	RGB	MHI	1D-Conv	61.7	52.8	71.3
	(SpatialCNN) + DilatedTCN [15]	RGB	MHI	DilatedTCN	55.8	46.9	71.1
	(SpatialCNN) + ED-TCN [15]	RGB	MHI	ED-TCN	76.5	72.2	73.4
	LCDC	RGB	Learned deformation	-	56.56	45.77	77.59
	LCDC + ST-CNN	RGB	Learned deformation	1D-Conv	70.46±0.41	62.71±0.46	77.84±0.26
	LCDC + DilatedTCN	RGB	Learned deformation	DilatedTCN	67.59±0.42	58.97±0.55	78.29±0.29
	LCDC + ED-TCN	RGB	Learned deformation	ED-TCN	**80.22±0.21**	**74.56±0.70**	**78.90±0.25**

Table 1: Results on 50 salads dataset (*mid* and *eval*-level). Learned deformation is $\dot{\Delta}$ in Eq. (8). Means and standard deviations over five runs are reported for LCDC with long-temporal models. Results of baselines are directly reported from their original publications. Please note that since TDRN implementation was not publicly available, LCDC features were not incorporated into TDRN and hence the TDRN results (in parentheses) are not directly comparable with LCDC results.

Model	Spatial comp	Temporal comp (short)	Long-temporal	F1@10	Edit	Acc
SpatialCNN [16]	RGB	MHI	-	41.8	-	54.1
(SpatialCNN) + ST-CNN [16]	RGB	MHI	1D-Conv	58.7	-	60.6
(SpatialCNN) + DilatedTCN [15]	RGB	MHI	DilatedTCN	58.8	-	58.3
(SpatialCNN) + ED-TCN [15]	RGB	MHI	ED-TCN	72.2	-	64.0
(SpatialCNN) + TDRN [17]	RGB	MHI	TDRN	(79.2)	(74.1)	(70.1)
LCDC	RGB	Learned deformation	-	52.42	45.38	55.32
LCDC + ST-CNN	RGB	Learned deformation	1D-Conv	62.23±0.69	55.75±0.94	58.36±0.45
LCDC + DilatedTCN	RGB	Learned deformation	DilatedTCN	62.08±0.85	55.13±0.79	58.07±0.30
LCDC + ED-TCN	RGB	Learned deformation	ED-TCN	**75.39±1.33**	**72.84±0.84**	**65.34±0.54**

Table 2: Results on GTEA dataset. Table format follows the same convention as in Tab. 1.

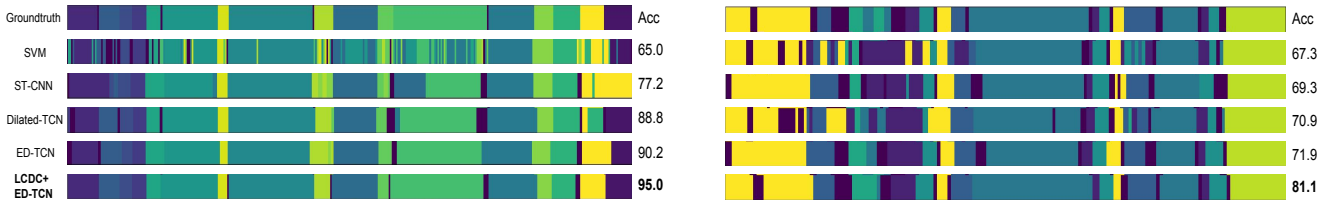
of the three baselines. LCDC + ED-TCN also outperforms the original SpatialCNN + ED-TCN on both reported metrics: improving by 3.19% and 1.34%, in terms of F1@10 and accuracy.

We further show segmentation results of test videos from 50 Salads (on *mid*-level granularity) (Fig. 5a) and GTEA datasets (Fig. 5b). In the figures, the first row is the ground-truth segmentation. The next four rows are results from different long-temporal models using SpatialCNN features: SVM, ST-CNN, DilatedTCN, and ED-TCN. All of these segmentation results are directly retrieved from the provided features in [15], without any further training. The last row shows the segmentation results of our LCDC + ED-TCN. Each row also comes with its respective accuracy on the right. On 50 Salads dataset, Fig. 5a shows that LCDC + ED-TCN achieves a 4.8% improvement over original ED-TCN. On GTEA dataset, Fig. 5b shows a strong improvement of LCDC over ED-TCN, being 9.2% in terms of accuracy. We also achieve a higher accuracy on the temporal boundaries, *i.e.* the beginning and the end of an action instance is close to that of ground-truth.

4.5. Ablation study

We performed an ablation study (Tab. 3) on Split 1 and *mid*-level granularity of 50 Salads dataset to compare LCDC with SpatialCNN and a two-stream framework. For each setup (each row in the table), we show the inputs for spatial and short-term temporal components, its fusion scheme, frame-wise accuracy, the total number of parameters of the model, and the number of parameters related to deformable convolutions (wherever applicable). Since this experiment focuses on comparing short-term features, accuracy metric is more suitable. We also report whether a component requires single or multiple frames as input.

We evaluate on the following setups: **(1) SpatialCNN:** The features from [16] described in Section 4.3. Its inputs are stacked RGB frame and MHI. **(2) NaiveAppear:** Frame-wise class prediction using ResNet50 (no temporal information involved in this setup). **(3) NaiveTempAppear:** Appearance stream from conventional two-stream frameworks uses a single frame input and VGG backbone. Therefore, comparing LCDC with the above is not straightforward. We created an appearance stream with multiple input frames and ResNet50 backbone for better comparison with LCDC. Temporal component was modeled by aver-



(a) 50 Salads dataset (*mid*-level). (b) GTEA dataset.

Figure 5: Comparison of segmentation results across different methods on two test videos (one each for 50 Salads and GTEA dataset). SVM, ST-CNN, DilatedTCN, and ED-TCN are original results with SpatialCNN features. LCDC features are used in conjunction with ED-TCN long-temporal model in the last row. Framewise accuracy is reported for each setup.

Model	Spatial comp	Temporal comp (short)	Fusion scheme	Acc	Total params	Deform params
SpatialCNN	RGB (single)	MHI (multi)	Stacked inputs	60.99	-	-
NaiveAppear	RGB (single)	-	-	68.45	38.9M	-
NaiveTempAppear	RGB (multi)	Avg feat frames (multi)	-	71.52	38.9M	-
OptFlowMotion	-	OptFlow (multi)	-	25.67	134.1M	-
TwoStreamNet	RGB (multi)	OptFlow (multi)	Avg scores	71.82	173.0M	-
DC	RGB (multi)	Learned deformation (w/o local coherency) (multi)	3D-Conv	72.25	45.7M	995.5K
LCDC	RGB (multi)	Learned deformation (multi)	3D-Conv	**73.77**	**42.7M**	**27.7K**

Table 3: Ablation study on 50 Salads dataset (Split 1, *mid*-level). "Single" and "multi" indicate the amount of input frames for spatial/temporal components.

aging feature frames (before feeding to two fc layers with ReLU). This model is the same as *NaiveAppear*, except that we have multiple frames per video snippet. **(4) OptFlowMotion:** Motion stream that models temporal component using VGG-16 (with stacked dense optical flows as input). This is similar to the motion component of conventional two-stream networks. **(5) TwoStreamNet:** The two-stream framework obtained by averaging scores from *NaiveTempAppear* and *OptFlowMotion*. We follow the fusion scheme used in conventional two-stream network [21]. **(6) DC:** Receptive fields of deformable convolution network (with backbone ResNet50) are used to model motion, but without local coherency constraint. **(7) LCDC:** The proposed LCDC model which additionally enforces local coherency constraint on receptive fields.

Compared to *SpatialCNN*, *NaiveAppear* has a higher accuracy because the *SpatialCNN* features are extracted using VGG-like model while *NaiveAppear* uses ResNet50. The accuracy is further improved by 3.07% by averaging multiple feature frames in *NaiveTempAppear*. Notice that the number of parameters of *NaiveAppear* and *NaiveTempAppear* are the same because the only difference is the number of frames being used as input (averaging requires no parameters). Accuracy from *OptFlowMotion* is lower than other models because the motion in 50Salads is hard to capture using optical flow. This is consistent with the observation in [15, 16] that optical flow is inefficient for the dataset. Combining *OptFlowMotion* with *NaiveTempAppear* in *TwoStreamNet* slightly improves the performance. However, the number of parameters is significantly increased because of complexity of *OptFlowMotion*. This prevented us from having a larger batch size or training the two streams together.

Both of our DC and LCDC frameworks, which model temporal components as difference of receptive fields, outperform the two-stream approach *TwoStreamNet* with significantly lower model complexities. *DC*, which directly uses adaptive receptive fields from the original deformable convolution, increases the accuracy to 72.25%. LCDC further improves accuracy to 73.77% and with even fewer parameters. This complexity reduction is because LCDC uses fewer parameters for deformation offsets. It means the extra parameters of DC are not necessary to model spatio-temporal features, and thus can be removed. Moreover, if we consider only the parameters related to deformable convolutions, *DC* would require 36x more parameters than *LCDC*. The reduction of 36x matches our derivation in Sec 3.3, where K_h=K_w=3 and G=4. The number of reduced parameters is proportional to the number of deformable convolution layers.

5. Conclusion

We introduced locally-consistent deformable convolution (LCDC) and created a single-stream network that can jointly learn spatio-temporal features by exploiting motion in adaptive receptive fields. The framework is significantly more compact and can produce robust spatio-temporal features without using conventional motion extraction methods, *e.g.* optical flow. LCDC features, when incorporated into several long-temporal networks, outperformed their original implementations. For future work, we plan to unify long-temporal modeling directly into the framework.

Acknowledgments: This material is based upon work supported in part by C3SR. Rogerio Feris is partly supported by

IARPA via DOI/IBC contract number D17PC00341. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. Disclaimer: The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of IARPA, DOI/IBC, or the U.S. Government)

References

[1] Joao Carreira and Andrew Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 2

[2] Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. Deformable convolutional networks. In *International Conference on Computer Vision (ICCV)*, 2017. 2, 3, 4, 6

[3] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2005. 2

[4] Navneet Dalal, Bill Triggs, and Cordelia Schmid. Human detection using oriented histograms of flow and appearance. In *European Conference on Computer Vision (ECCV)*, 2006. 2

[5] James W. Davis and Aaron F. Bobick. The representation and recognition of action using temporal templates. *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2001. 2

[6] Yazan Abu Farha and Jurgen Gall. MS-TCN: Multi-stage temporal convolutional network for action segmentation. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. 3

[7] Alireza Fathi, Xiaofeng Ren, and James M. Rehg. Learning to recognize objects in egocentric activities. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2011. 2, 6

[8] Christoph Feichtenhofer, Axel Pinz, and Richard Wildes. Spatiotemporal residual networks for video action recognition. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2016. 2

[9] Christoph Feichtenhofer, Axel Pinz, and Andrew Zisserman. Convolutional two-stream network fusion for video action recognition. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 2

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 3

[11] Matthias Holschneider, Richard Kronland-Martinet, Jean Morlet, and Philippe Tchamitchian. A real-time algorithm for signal analysis with the help of the Wavelet transform. In *Wavelets*, 1990. 2, 3

[12] Kai Kang, Hongsheng Li, Junjie Yan, Xingyu Zeng, Bin Yang, Tong Xiao, Cong Zhang, Zhe Wang, Ruohui Wang, Xiaogang Wang, and Wanli Ouyang. T-CNN: Tubelets with convolutional neural networks for object detection from videos. *Transactions on Circuits and Systems for Video Technology (TCSVT)*, 2018. 2

[13] Soo Min Kang and Richard P. Wildes. Review of action recognition and detection methods. *arXiv preprint arXiv:1610.06906*, 2016. 1

[14] Colin Lea. Temporal convolutional networks. https://github.com/colincsl/TemporalConvolutionalNetworks. Accessed: 2019-03-20. 6

[15] Colin Lea, Michael D. Flynn, René Vidal, Austin Reiter, and Gregory D. Hager. Temporal convolutional networks for action segmentation and detection. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 2, 3, 5, 6, 7, 8

[16] Colin Lea, Austin Reiter, René Vidal, and Gregory D. Hager. Segmental spatiotemporal CNNs for fine-grained action segmentation. In *European Conference on Computer Vision (ECCV)*, 2016. 2, 5, 6, 7, 8

[17] Peng Lei and Sinisa Todorovic. Temporal deformable residual networks for action segmentation in videos. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 2, 3, 6, 7

[18] Bruce D. Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 1981. 2

[19] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural Networks*, 1999. 6

[20] Marcus Rohrbach, Sikandar Amin, Mykhaylo Andriluka, and Bernt Schiele. A database for fine grained activity detection of cooking activities. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012. 1

[21] Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2014. 2, 8

[22] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 3

[23] Bharat Singh, Tim K. Marks, Michael Jones, Oncel Tuzel, and Ming Shao. A multi-stream bi-directional recurrent neural network for fine-grained action detection. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 1, 2, 3

[24] Suriya Singh, Chetan Arora, and C. V. Jawahar. First person action recognition using deep learned descriptors. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 2

[25] Sebastian Stein and Stephen J. McKenna. Combining embedded accelerometers with computer vision for recognizing food preparation activities. In *International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp)*, 2013. 2, 6

[26] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. 3

[27] Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. Learning spatiotemporal features with

3D convolutional networks. In *International Conference on Computer Vision (ICCV)*, 2015. 2

[28] Heng Wang and Cordelia Schmid. Action recognition with improved trajectories. In *International Conference on Computer Vision (ICCV)*, 2013. 2

[29] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. In *International Conference on Learning Representations (ICLR)*, 2016. 2, 3

[30] Fisher Yu, Vladlen Koltun, and Thomas Funkhouser. Dilated residual networks. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 2, 3

Supplementary Material for Learning Motion in Feature Space: Locally-Consistent Deformable Convolution Networks for Fine-Grained Action Detection

A. Full formulation for convolutions with multiple output channels

Suppose that the input $\mathbf{x}$ of a convolution has I channels and the output has O channels, *i.e.* $\mathbf{x} \in R^{N \times I}, \mathbf{y} \in \mathbb{R}^{M \times O}$, we can write the standard convolution as:

$$\mathbf{y}_j[n] = \sum_i \sum_k \mathbf{w}_{j,i}[-k]\mathbf{x}_i\left[n+k\right], \tag{11}$$

where $i \in \{1, \ldots, I\}, j \in \{1, \ldots, O\}$, and $\mathbf{w} \in \mathbb{R}^{N \times K \times I \times O}$. The original deformable convolution, therefore, is written as:

$$\mathbf{y}_j[n] = \sum_i \sum_k \mathbf{w}_{j,i}[-k]\mathbf{x}_i\left(n+k+\ddot{\Delta}_{n,k}\right), \tag{12}$$

In reality, there are multiple deformable groups ($G > 1$), meaning that different input channels can have different deformation offsets. Specifically, a multi-channel deformable convolution with multiple deformable group can be written as:

$$\mathbf{y}_j[n] = \sum_i \sum_k \mathbf{w}_{j,i}[-k]\mathbf{x}_i\left(n+k+\ddot{\Delta}_{g_i,n,k}\right), \tag{13}$$

where g_i is the deformable group that the input channel i belongs to. We keep the deformable group as $G = 1$ and drop the notation g_i for the sake of simplicity.

We write the multi-channel LCDC as:

$$\mathbf{y}_j[n] = \sum_i \sum_k \mathbf{w}_{j,i}[-k]\mathbf{x}_i\left(n+k+\dot{\Delta}_{n+k}\right). \tag{14}$$

It is equivalent to

$$\mathbf{y}_j[n] = \sum_i \sum_k \mathbf{w}_{j,i}[-k]\tilde{\mathbf{x}}_i[n+k] = (\tilde{\mathbf{x}} * \mathbf{w}_j)[n], \tag{15}$$

where

$$\tilde{\mathbf{x}}_i[n] = (D_{\dot{\Delta}}\{\mathbf{x}_i\})[n] = \mathbf{x}_i\left(n+\dot{\Delta}_n\right). \tag{16}$$

B. More reasoning on the difference of receptive fields

$\ddot{\mathbf{r}}^{(t)}$ and $\dot{\mathbf{r}}^{(t)}$ of deformable convolution and locally-consistent deformable convolution *carries temporal information* because the offsets are constructed from inputs at different time frames. This property is not valid in other types of convolutions. We can write standard convolutions and dilated convolutions as special cases of deformable convolutions, *i.e.* $\ddot{\Delta} = 0$ in standard convolution and $\ddot{\Delta}^{(t)} = const, \forall t$. Hence,

- Standard convolution:
$$\ddot{\Delta}^{(t)} = 0, \forall t \Rightarrow \ddot{\mathbf{r}}^{(t)} = 0, \forall t,$$
- Dilated convolution:
$$\ddot{\Delta}^{(t)} = \ddot{\Delta}^{(t-1)}, \forall t \Rightarrow \ddot{\mathbf{r}}^{(t)} = 0, \forall t.$$
- Deformable convolution:
$$\ddot{\Delta}^{(t)} \neq \ddot{\Delta}^{(t-1)} \Rightarrow \ddot{\mathbf{r}}^{(t)} \neq 0.$$
- Locally-consistent deformable convolution:
$$\dot{\Delta}^{(t)} \neq \dot{\Delta}^{(t-1)} \Rightarrow \dot{\mathbf{r}}^{(t)} \neq 0.$$

C. In-detail architecture of LCDC

Tab. 4 shows the detailed architecture implementation of LCDC.

Layer	Input(s)	Output size	Kernel size	Comments
conv1	data	(112,112,64)	(7,7,64), stride2	with 3,3 maxpool, stride2 frames of all snippets are unrolled
conv2x	bn_conv1	(56,56,256)	$\begin{bmatrix} 1,1,64 \\ 3,3,64 \\ 1,1,256 \end{bmatrix} \times 3$	input is output of conv1
conv3x	res2c_relu	(28,28,512)	$\begin{bmatrix} 1,1,128 \\ 3,3,128 \\ 1,1,512 \end{bmatrix} \times 4$	input is output of conv2x
conv4x	res3d_relu	(14,14,1024)	$\begin{bmatrix} 1,1,256 \\ 3,3,256 \\ 1,1,1024 \end{bmatrix} \times 6$	input is output of conv3x
res5a_branch1	res4f_relu	(14,14,2048)	(1,1,2048)	input is output of conv4x
bn5a_branch1	(prev)	(14,14,2048)	-	batch normalization
res5a_branch2a	(prev)	(14,14,512)	(1,1,512)	convolution
bn5a_branch2a	(prev)	(14,14,512)	-	batch normalization
res5a_branch2a_relu	(prev)	(14,14,512)	-	ReLU
res5a_branch2b_offset	(prev)	(14,14,2)	(3,3,2)	offset learner
res5a_branch2b_offset_expand	(prev)	(14,14,18)	-	expand by replication
res5a_branch2b	res5a_branch2a_relu res5a_branch2b_offset_expand	(14,14,512)	(3,3,512)	deformable convolution
bn5a_branch2b	(prev)	(14,14,512)	-	batch normalization
res5a_branch2b_relu	(prev)	(14,14,512)	-	ReLU
res5a_branch2c	(prev)	(14,14,2048)	(1,1,2048)	convolution
bn5a_branch2c	(prev)	(14,14,2048)	-	batch normalization
res5a	bn5a_branch1 bn5a_branch2c	(14,14,2048)	-	addition
res5a_relu	(prev)	(14,14,2048)	-	ReLU
res5b_branch2a	(prev)	(14,14,512)	(1,1,512)	convolution
bn5b_branch2a	(prev)	(14,14,512)	-	batch normalization
res5b_branch2a_relu	(prev)	(14,14,512)	-	ReLU
res5b_branch2b_offset	(prev)	(14,14,2)	(3,3,2)	offset learner
res5b_branch2b_offset_expand	(prev)	(14,14,18)	-	expand by replication
res5b_branch2b	res5b_branch2a_relu res5b_branch2b_offset_expand	(14,14,512)	(3,3,512)	deformable convolution
bn5b_branch2b	(prev)	(14,14,512)	-	batch normalization
res5b_branch2b_relu	(prev)	(14,14,512)	-	ReLU
res5b_branch2c	(prev)	(14,14,2048)	(1,1,2048)	convolution
bn5b_branch2c	(prev)	(14,14,2048)	-	batch normalization
res5b	res5a_relu bn5b_branch2c	(14,14,2048)	-	addition
res5b_relu	(prev)	(14,14,2048)	-	ReLU
res5c_branch2a	(prev)	(14,14,512)	(1,1,512)	convolution
bn5c_branch2a	(prev)	(14,14,512)	-	batch normalization
res5c_branch2a_relu	(prev)	(14,14,512)	-	ReLU
res5c_branch2b_offset	(prev)	(14,14,2)	(3,3,2)	offset learner
res5c_branch2b_offset_expand	(prev)	(14,14,18)	-	expand by replication
res5c_branch2b	res5c_branch2a_relu res5c_branch2b_offset_expand	(14,14,512)	(3,3,512)	deformable convolution
bn5c_branch2b	(prev)	(14,14,512)	-	batch normalization
res5c_branch2b_relu	(prev)	(14,14,512)	-	ReLU
res5c_branch2c	(prev)	(14,14,2048)	(1,1,2048)	convolution
bn5c_branch2c	(prev)	(14,14,2048)	-	batch normalization
res5c	res5b_relu bn5c_branch2c	(14,14,2048)	-	addition
res5c_relu	(prev)	(14,14,2048)	-	ReLU
conv_new_1	(prev)	(14,14,256)	(1,1,256)	convolution
conv_new_1_relu	(prev)	(14,14,256)	-	ReLU
spacetime_fusion	conv_new_1_relu res5a_branch2b_offset res5b_branch2b_offset res5c_branch2b_offset	(L-1,14,14,262)	-	reshape all frames back into snippets, then concatenate difference of all offset layers with conv_new_1_relu
spacetime_conv1	(prev)	(L-1,14,14,256)	(4,3,3,256)	3Dconv with window size for temporal dimension of 4
spacetime_bn1	(prev)	(L-1,14,14,256)	-	batch normalization
spacetime_relu1	(prev)		-	ReLU

spacetime_pool1	(prev)	((L-1)/2,14,14,256)	-	temporal max pooling of size 2
spacetime_conv2	(prev)	((L-1)/2,14,14,256)	(4,3,3,256)	3Dconv with window size for temporal dimension of 4
spacetime_bn2	(prev)	((L-1)/2,14,14,256)	-	batch normalization
spacetime_relu2	(prev)		-	ReLU
spacetime_pool2	(prev)	((L-1)/4,14,14,256)	-	temporal max pooling of size 2
spacetime_reduce	(prev)	(14,14,256)	-	averaging across time domain
pool_new	(prev)	(7,7,256)	-	max pooling, stride 2
fc_new_1	(prev)	(1024)	-	fully connected with ReLU
fc_new_2	(prev)	(1024)	-	fully connected with ReLU

Table 4: LCDC architecture in detail. The groups conv1, conv2x, conv3x, and conv4x are the same as the original ResNet50. The convention of kernel size: (kernel_height, kernel_width, number_of_output_channels) for 2D convolution and (kernel_time, kernel_height, kernel_width, number_of_output_channels) for 3D convolution. Size of input data is (224, 224, 3). L is the number of frames per video snippet (we choose $L = 16$). If the input is annotated as *(prev)*, it means it uses the output from the previous layer.

D. In-detail figures

We provide higher-resolution versions of Fig. 2, Fig. 4, and Fig. 3 in Fig. 6, Fig. 7, and Fig. 8 respectively. Fig. 9 and Fig. 10 also show higher-resolution versions of Fig. 5 with annotation of color-code. We also provide the groundtruth action sequence of the two videos. Readers can view the videos corresponding to Fig. 9 and Fig. 10 in other additional supplementary materials (*50salads.mp4* and *gtea.mp4*).

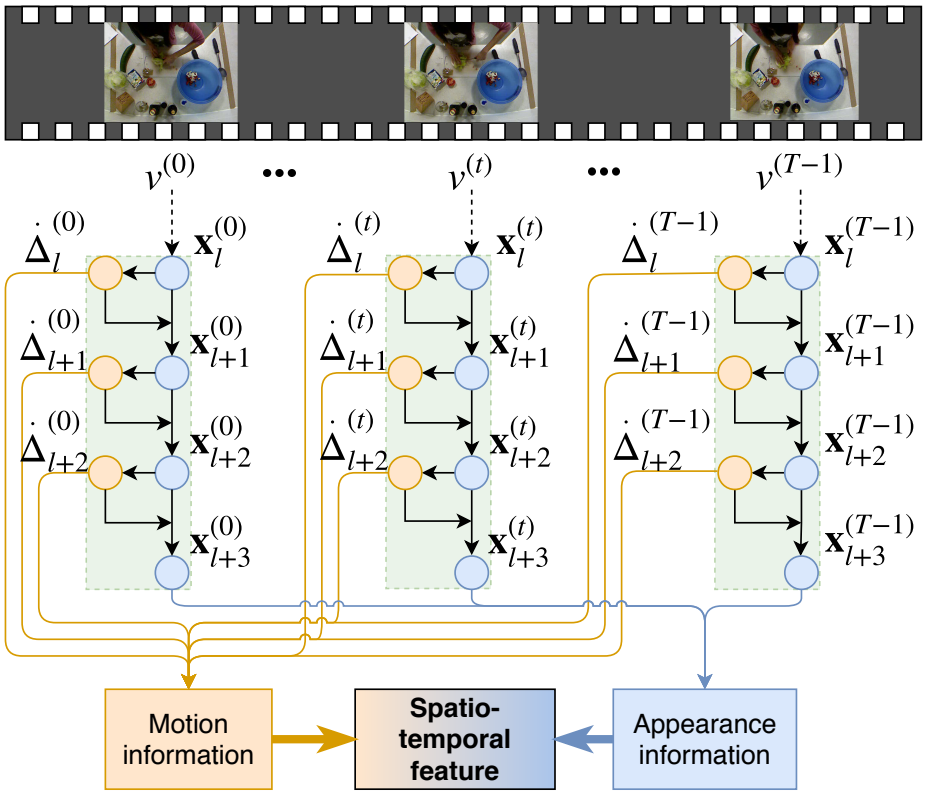


Figure 6: Network architecture of our proposed framework across multiple frames $v^{(t)}$. Appearance information comes from the last layer while motion information is extracted directly from deformation $\dot{\Delta}$ in the feature space instead of from a separate optical flow stream. Weights are shared across frames over time.

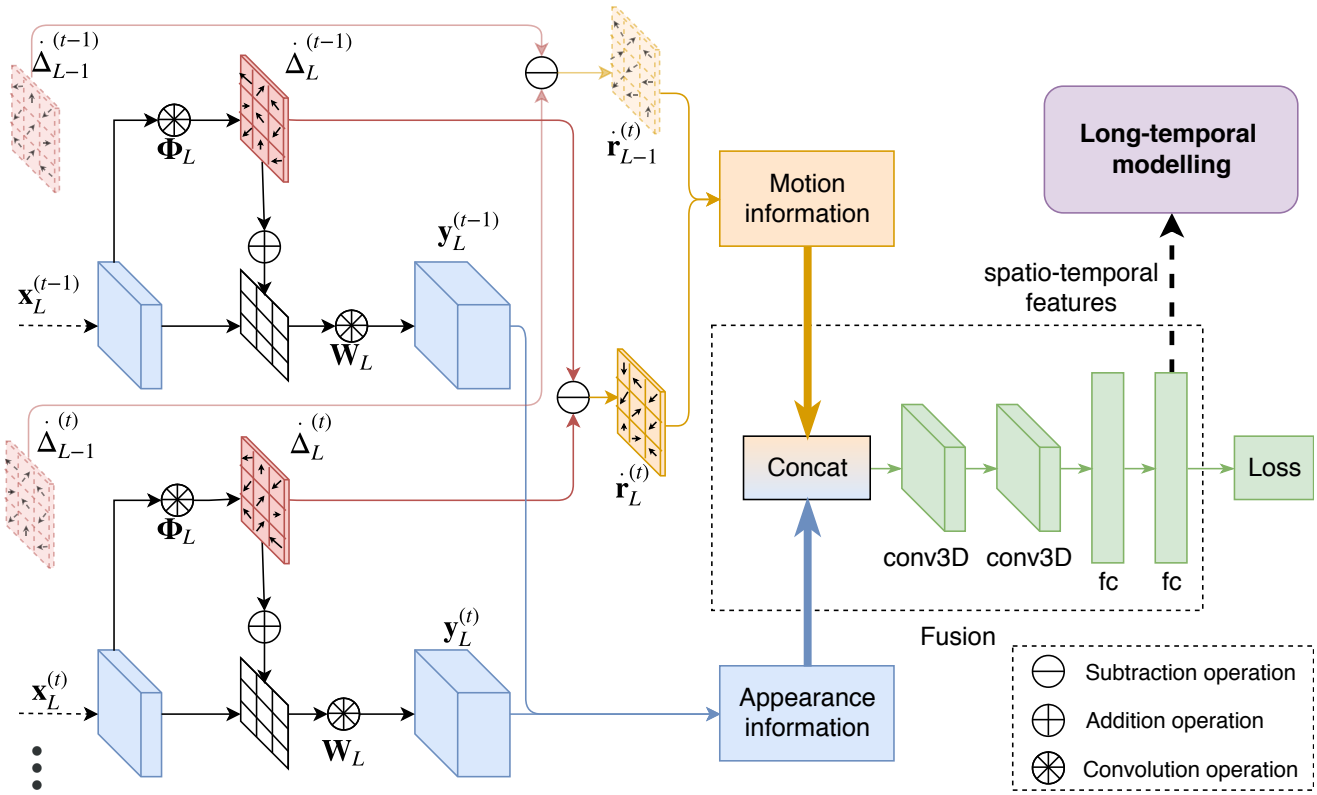


Figure 7: A more detailed view of our network architecture with the fusion module. Appearance information comes from output of the last layer while motion information comes from aggregating $\dot{\mathbf{r}}$ from multiple layers. Outputs of the final fc layer can be flexibly used as the features for any long-temporal modeling networks.

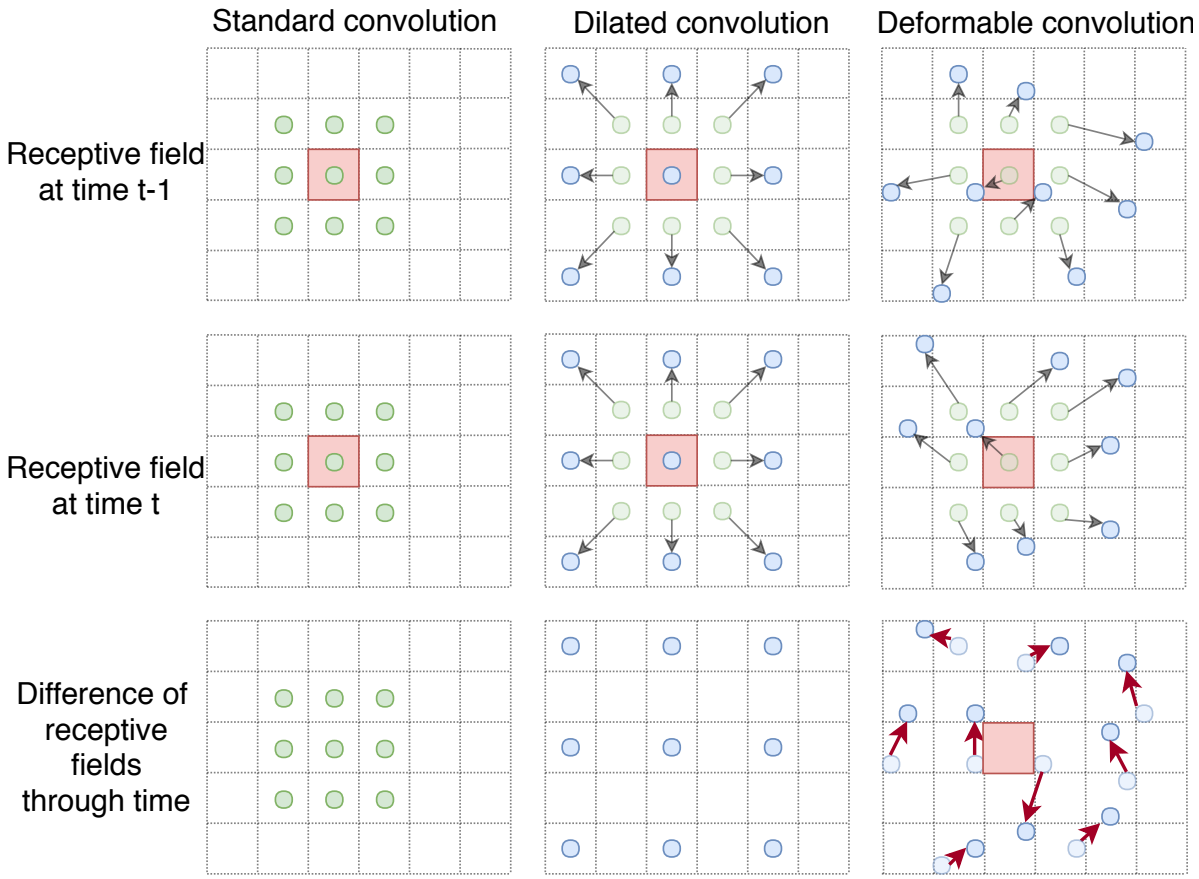


Figure 8: Illustration of temporal information modeled by the difference of receptive fields at a single location in 2D. Only deformable convolution can capture temporal information (shown with red arrows). Related to Eq. (2) and Eq. (3), n is red square, $n+k$ are green dots, $\ddot{\Delta}_{n,k}$ are black arrows, $n+k+\ddot{\Delta}_{n,k}$ are blue dots, and $\ddot{\mathbf{r}}$ are red arrows.

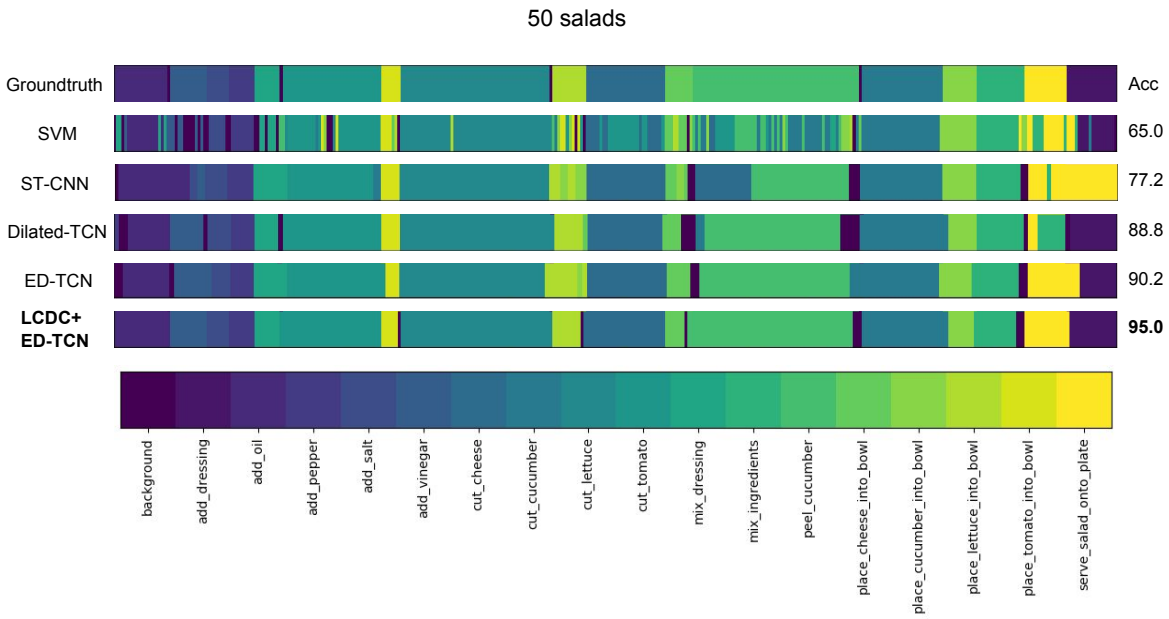


Figure 9: Comparison of segmentation results across different methods on a test video from 50 Salads dataset (*mid*-level). The action sequence is: *add_oil, background, add_vinegar, add_salt, add_pepper, mix_dressing, background, cut_tomato, place_tomato_into_bowl, cut_lettuce, background, place_lettuce_into_bowl, cut_cheese, place_cheese_into_bowl, peel_cucumber, background, cut_cucumber, place_cucumber_into_bowl, mix_ingredients, serve_salad_onto_plate, add_dressing*.

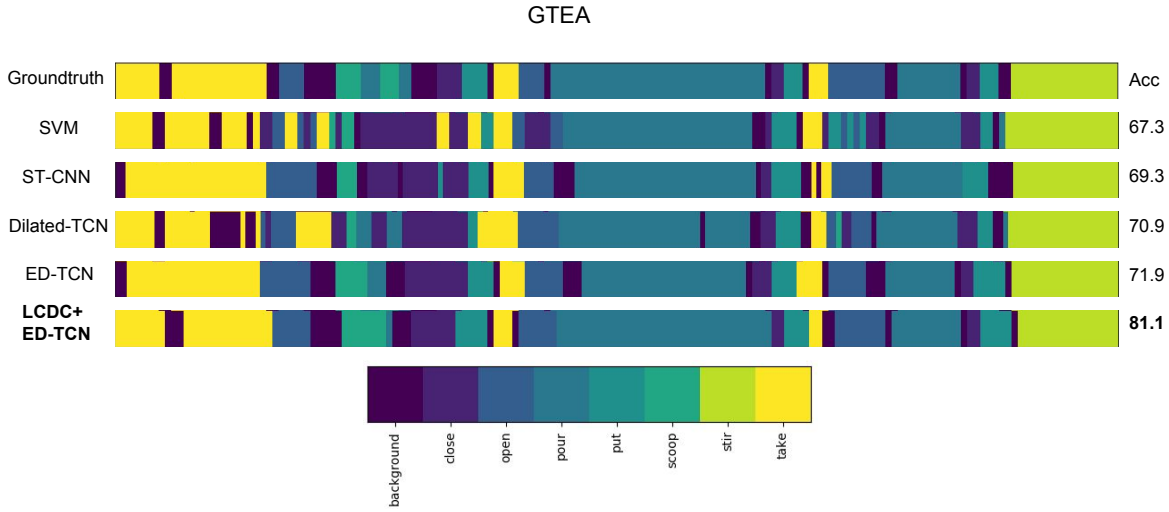


Figure 10: Comparison of segmentation results across different methods on a test video from GTEA dataset. The action sequence is: *take, background, take, background, open, background, scoop, pour, scoop, pour, background, close, put, background, take, open, background, pour, background, close, put, background, take, open, background, pour, background, close, put, background, stir*.