Multi-modal Circulant Fusion for Video-to-Language and Backward

Aming Wu and Yahong Han
School of Computer Science and Technology, Tianjin University, Tianjin, China
{tjwam, yahong}@tju.edu.cn

Abstract
Multi-modal fusion has been widely involved in focuses on the modern artificial intelligence research, e.g., from visual content to languages and backward. Common-used multi-modal fusion methods mainly include element-wise product, element-wise sum, or even simply concatenation between different types of features, which are somewhat straightforward but lack in-depth analysis. Recent studies have shown fully exploiting interactions among elements of multi-modal features will lead to a further performance gain. In this paper, we put forward a new approach of multi-modal fusion, namely Multi-modal Circulant Fusion (MCF). Particularly, after reshaping feature vectors into circulant matrices, we define two types of interaction operations between vectors and matrices. As each row of the circulant matrix shifts one element, with newly-defined interaction operations, we almost explore all possible interactions between vectors of different modalities. Moreover, as only regular operations are involved and defined a priori, MCF avoids increasing parameters or computational costs for multi-modal fusion. We evaluate MCF with tasks of video captioning and temporal activity localization via language (TALL). Experiments on MSVD and MSRVTT show our method obtains the state-of-the-art performance for video captioning. For TALL, by plugging into MCF, we achieve a performance gain of roughly 4.2% on TACoS.

1 Introduction
Multi-modal data are widely involved in recent focuses on artificial intelligence research, e.g., visual to languages [Yao et al., 2015] and the backward of visual localization via language queries [Gao et al., 2017]. Though many efforts have been made towards feature learning of images and texts, e.g., via convolutional neural networks (CNN) [He et al., 2016] and recurrent neural networks (RNN) [Kiros et al., 2015], respectively, in-depth analysis of multi-modal fusion has been unintentionally ignored. Common-used or somewhat straightforward multi-modal fusion methods mainly include element-wise sum (Fig. 1 (a)), element-wise product (Fig. 1 (b)), or even simply concatenation (Fig. 1 (c)) between different types of features [Fukui et al., 2016; Yu et al., 2017]. As feature vectors of different modalities lie in different feature spaces, interactions or correlations might not just exist among corresponding dimensions of multi-modal vectors. Besides no interactions with concatenation, element-wise sum or product only partially explores interactions or correlations among multi-modal features, which may burden the fusion performance. Recent studies [Fukui et al., 2016; Yu et al., 2017] have shown fully exploiting interactions among elements of multi-modal features will lead to a further performance gain. Authors in [Fukui et al., 2016; Yu et al., 2017] develop multi-modal bilinear pooling to capture pairwise interactions between multi-modal feature dimensions. As bilinear pooling defines p parameterized projection matrices, where p is the dimension of the output fused features, additional huge number of parameters are introduced into the model. Although count sketch [Fukui et al., 2016] or matrix factorization [Yu et al., 2017] were employed to shrink projection matrices, more computational costs were introduced again. Moreover, empirical studies in [Fukui et al., 2016] show that the performance gain from multi-modal fusion is guaranteed only when
the dimension $p$ of the output fused features is high, which means there still needs a large number of parameterized projection matrices. Therefore, if we skipped the parameterized projection matrices and defined a fully-interacted module of multi-modal fusion a priori, e.g., constituted with only regular operations like element-wise sum or product, we expect to get a good fusion performance with fewer parameters or lower computational costs.

In this paper, we propose a new module of Multi-modal Circulant Fusion (MCF) to fully exploit interactions among multi-modal features. In Fig. 1 (d) we show the main idea of MCF by taking the fusion of visual and textual vectors as an example. In particular, after reshaping visual or textual vectors into circulant matrices, respectively, we define two types of interaction operations between original feature vectors and the reshaped circulant matrices. Finally, we use element-wise sum to obtain the joint representation of these two cross-fused vectors. As each row of the circulant matrix shifts one elements, with newly-defined interaction operations, we almost explore all possible interactions between vectors of different modalities. Note that, as only regular operations are involved in the MCF and the proposed MCF is defined a priori, we avoid introducing new parameters or increasing computational costs for multi-modal fusion.

In the experiments, we extensively evaluate the proposed MCF with tasks of video captioning and temporal activity localization via language (TALL) [Gao et al., 2017]. Experimental results on MSVD [Chen and Dolan, 2011] and MSRVTT [Chen and Dolan, 2011] show that our method could obtain the state-of-the-art performance of video captioning. For TALL, by plugging into the MCF, we achieve a performance improvement of roughly 4.2% on TACoS [Regneri et al., 2013].

## 3 Multi-modal Circulant Fusion

The detailed procedures of MCF are illustrated in Fig. 2. Given two feature vectors in different modalities, e.g., the visual features $x \in \mathbb{R}^o$ and the textual features $y \in \mathbb{R}^n$, to reduce computational cost, we first utilize two projection matrices $W_1 \in \mathbb{R}^{d \times o}$ and $W_2 \in \mathbb{R}^{d \times n}$ ($d \leq \min(o, n)$) to project the two input features to a lower dimensional space.

$$V = x W_1^T$$

$$C = y W_2^T$$

where $W_1^T$ and $W_2^T$ are the transpose of $W_1$ and $W_2$.

Then we use the projection vector $V \in \mathbb{R}^d$ and $C \in \mathbb{R}^d$ to construct circulant matrix $A \in \mathbb{R}^{d \times d}$ and $B \in \mathbb{R}^{d \times d}$.

$$A = \text{circ}(V)$$

$$B = \text{circ}(C)$$

where $\text{circ}(b)$ denotes converting $b$ to a circulant matrix.

In order to make elements in projection vector and circulant matrix fully interact, we explore two different multiplication operations. The first is shown in Eq. (3) and uses matrix multiplication between circulant matrix and projection vector.

$$F = CA$$

$$G = VB$$

The second is to have projection vector and each row vector of circulant matrix do element-wise product. The procedures are shown as follows:

$$F = \frac{1}{d} \sum_{i=1}^{d} a_i \odot C$$

$$G = \frac{1}{d} \sum_{i=1}^{d} b_i \odot V$$
where \( a_i \in \mathbb{R}^d \) and \( b_i \in \mathbb{R}^d \) are row vector of circulant matrix \( A \) and \( B \). \( \odot \) denotes operation of element-wise product. It is noted that we do not introduce new parameters in the multiplication operation.

Finally, through a projection matrix \( W_3 \in \mathbb{R}^{d \times k} \), we convert the element-wise sum vector of \( F \in \mathbb{R}^d \) and \( G \in \mathbb{R}^d \) to target vector \( M \in \mathbb{R}^k \).

4 MCF for Video Captioning

In this section, we develop a new framework for video captioning, in which we construct convolutional encoder and decoder for video-to-language translation. Note that, in the decoder, we take MCF as a base layer for coarse decoding, stacked on which with layered dilations for refined and final decoding. Thus we construct a Multi-stage Decoder with MCF.

4.1 Convolutional Encoding Network

Feature Extraction. We use pre-trained convolutional networks to extract feature for each of the \( m \) video frames, which results in a vector \( X_i \in \mathbb{R}^d \) for the \( i \)-th frame.

Discriminative Enhancing. Given two consecutive frame features \( X_i \) and \( X_{i+1} \) \((i = 0, 2, \cdots)\), we first compute inter-frame differences \( diff \). Then by a ReLU operation, we add the positive values of \( diff \) to \( X_{i+1} \) and the absolute value of negative values in \( diff \) to \( X_i \). Thus, we enlarge the discrimination gap between \( X_i \) and \( X_{i+1} \).

\[
diff = X_{i+1} - X_i, \quad V_{i+1} = X_{i+1} + \text{ReLU}(diff), \quad V_i = X_i + \text{ReLU}(-diff),
\]

\[
\text{ReLU}(x) = \begin{cases} x, & \text{if } x > 0; \\ 0, & \text{else} \end{cases}
\]

where \( i = 0, 2, \cdots \), \( V_i \in \mathbb{R}^d \) is the enhancing result of \( X_i \).

Reconstruction Network. In this work, we construct a reconstruction network to learn a compact representation \( Z_i \) for each video frame. The procedures are shown as follows:

\[
Z_i = \text{ReLU}(W_E * V_i + b_E) \quad R_i = W_D * Z_i + b_D \quad L_{\text{recon}} = \|X_i - R_i\|_2^2
\]

where \( i = 0, 1, \cdots \), \( W_E \in \mathbb{R}^{1 \times q \times r} \) denotes convolutional filter and \( r \leq q \), \( b_E \in \mathbb{R}^r \) and \( b_D \in \mathbb{R}^q \) are the bias parameters. \( Z_i \in \mathbb{R}^r \) is the learned compact representation. \( W_D \in \mathbb{R}^{1 \times r \times q} \) is the reconstruction filter and \( R_i \in \mathbb{R}^q \) denotes reconstruction result. \( L_{\text{recon}} \) denotes reconstruction loss between \( R_i \) and \( X_i \) and \( \| \cdot \|_2 \) denotes an \( \ell_2 \)-norm. \( \ast \) denotes a convolutional operator.

4.2 Multi-stage Convolutional Decoder with MCF

As introduced above, we stack many dilated convolutional layers [Chen et al., 2017] to form a coarse-to-fine multi-stage decoder (Fig. 3). In the following, we denote by \( \hat{Y}^j = \{\hat{Y}_0^j, \cdots, \hat{Y}_{T-j}^j\} \) the predicted word sequence of the \( j \)-th stage of decoder, where \( j \in \{0, \cdots, N_j\} \) and \((N_j + 1)\) is the number of stages. We denote by \( Y = \{Y_0, \cdots, Y_{T-1}\} \) the target word sequence, where \( T \) denotes sequence length. Note that we treat the initial stage \( j = 0 \) as coarse decoder and above stages as refined decoders. \( [a, b] \) represents the concatenation of \( a \) and \( b \). And \( \odot \) is an element-wise product operation, \( \ast \) denotes a convolutional operator. \( Z = \{Z_0, Z_1, \cdots\} \). The \( \sigma(\cdot) \) is a sigmoid function.

MCF as a Coarse Decoder. In the bottom stage \( (j = 0) \), we use one dilated convolutional layer to learn a coarse decoder. At each time step \( t \in [0, T - 1] \), the input to coarse decoder consists of previous target word \( Y_{t-1} \) and mean vector \( Z_{\text{mean}} \) of encoding output \( Z \). As the inputs are two different modalities, we first use MCF to obtain the joint representation \( M_i^0 \).

The operation of coarse decoder can be described as follows:

\[
M_i^0 = MCF(Z_{\text{mean}}, Y_{i-1}) \quad h_i^0 = \tanh(w_i^0 \ast H_i^0 + b_i^0) \odot \sigma(w_i^0 \ast H_i^0 + b_i^0)
\]

where \( H_i^0 = [M_i^{0-1}, M_i^0] \). MCF(\( a, b \)) represents using MCF to fuse \( a \) and \( b \). \( w_i^0 \) and \( w_i^0 \) denote convolutional filters on the \( k \)th layer. \( b_i^0 \) and \( b_i^0 \) denote bias on the \( k \)th layer.

Refined Decoder. In this paper, our refined decoder consists of two stages. The first stage contains three dilated convolutional layers. And the second stage only includes one dilated convolutional layer which stacks on top of first stage. We take prediction of the second stage as final description.

For the first refined decoder, we first use the output \( h_i^0 \) of coarse decoder to compute visual attention \( \phi_i^0(Z) \). The operation of first layer in this refined decoder is shown as follows:

\[
M_i^1 = w_i^1 \ast [h_i^0 \ast \phi_i^0(Z)] + b_1 \quad H_i^1 = [M_i^{1-1}, M_i^1] \quad h_i^1 = \tanh(w_i^1 \ast H_i^1 + b_i^1) \odot \sigma(w_i^1 \ast H_i^1 + b_i^1)
\]

where \( w_i \) is learnable filter to convert the channel of concatenated representation. \( w_i^1 \) and \( b_i^1 \) denote convolutional filters on the \( k \)th layer. \( b_i^1 \) and \( b_i^1 \) denote bias on the \( k \)th layer.
Then, operations of the next two layers in first refined decoder are shown as follows:

\[
H_l^t = [h_l^{t-1}, h_{l-1}^{t-1}]
\]

\[
h_l^t = \tanh(w_f^l * H_l^t + b_f^l) \odot \sigma(w_g^l * H_l^t + b_g^l)
\]

(12)

where \( r^l \) represents dilated rate of the layer \( l \). \( h_l^{t-1} \) denotes the output of \((l - 1)\)-th layer at time step \( t \). \( w_f^l \) and \( w_g^l \) denote convolutional filters on the layer \( l \). \( b_f^l \) and \( b_g^l \) are bias.

For the second refined decoder, the procedures can be described as follows:

\[
M_l^{t+1} = h_l^t + \varphi_l^t(Z)
\]

\[
H_l^{t+1} = [M_l^{t+1}, M_{l-1}^{t+1}]
\]

\[
Mid1 = \tanh(w_j^{l+1} * H_l^{t+1} + b_j^{l+1})
\]

\[
Mid2 = \sigma(w_j^{l+1} * H_l^{t+1} + b_j^{t+1})
\]

\[
h_l^{t+1} = Mid1 \odot Mid2
\]

(13)

where \( L \) denotes the number of layers of first refined stage. \( h_l^t \) denotes the output of layer \( L \) at time step \( t \). \( w_j^{l+1} \) and \( w_j^{t+1} \) denote convolutional filters of layer \( L + 1 \). \( b_j^{l+1} \) and \( b_j^{t+1} \) are bias. \( \varphi_l^t(Z) \) represents attention computed by \( h_l^t \).

In this section, we use the method [Yao et al., 2015] to compute attention (Fig. 3). Besides, based on different size of filter and dilated rate, we use different number of zero vectors to pad the input of each layer.

Finally, the \( t \)-th generated word \( \hat{Y}_t^1 \) of the first refined stage and \( \hat{Y}_t^2 \) of the second refined stage are computed as follows:

\[
\hat{Y}_t^1 \sim \text{softmax}(w_p(h_l^t + \varphi_l^t(Z)) + b_p)
\]

\[
\hat{Y}_t^2 \sim \text{softmax}(w_p(h_l^{t+1} + \varphi_l^t(Z)) + b_p)
\]

(14)

where \( w_p \) and \( b_p \) are learnable projection matrix and bias.

**Training Loss.** For each stage \( j \), we employ a cross-entropy (XE) loss.

\[
L_{XE}^j(\theta_{0:j}) = - \sum_{t=0}^{T-1} \log(p_{\theta_{0:j}}(Y_t|Y_{0:t-1}, Z))
\]

(15)

where \( Y_t \) is the ground-truth word at time \( t \). \( \theta_{0:j} \) are parameters up to the \( j \)-th stage decoder, and \( p_{\theta_{0:j}}(Y_t|Y_{0:t-1}, Z) \) is the output probability of word \( Y_t \) given by the previous word \( Y_{0:t-1} \) and encoding output \( Z \).

The training loss \( L_{train} \) is computed as follows:

\[
L_{train} = \beta_2 L_{recon} + \beta_1 \sum_{j=0}^{N_f} \lambda_j L_{XE}^j(\theta_{0:j})
\]

(16)

where \( \lambda_j \), \( \beta_t \), and \( \beta_2 \) are hyper-parameters.

5 MCF for TALL

For temporal activity localization via language query, we plug MCF into the architecture of work [Gao et al., 2017]. Concretely, we first feed visual feature \( f_v \) and textual feature \( f_s \) into MCF (Fig. 4). For MCF, after obtaining \( F \in \mathbb{R}^d \) and \( G \in \mathbb{R}^d \) by Eq. (4), we use a projection matrix \( W_M \in \mathbb{R}^{d \times k} \) to convert \( F \) and \( G \) to \( F_s \in \mathbb{R}^k \) and \( G_v \in \mathbb{R}^k \).

\[
F_s = f_s W_M
\]

\[
G_v = f_v W_M
\]

(17)

The following operations are same as that of the work [Gao et al., 2017].

6 Evaluation on Video Captioning

We evaluate the benefit of MCF on two video captioning datasets. All results all evaluated by metrics of BLEU, METEOR, and CIDEr [Yao et al., 2015].

6.1 Dataset and Implementation Details

**Datasets.** MSVD [Chen and Dolan, 2011] contains 1,970 video clips. We use 1,200 clips for training, 100 clips for validation, and 670 clips for testing. MSRVTT [Xu et al., 2016] contains 10,000 video clips. We use 6,513 clips for training, 497 clips for validation, and 2,990 clips for testing.

**Video Processing.** For the MSVD dataset, we select 40 frames from each video and feed them into GoogleLeNet [Szegedy et al., 2015] to extract a 1,024 dimensional representation. For the MSRVTT dataset, we select 20 frames from each video and feed them into GoogLeNet and ResNet-152 [He et al., 2016] to extract 1,024 and 2,048 dimensional representation, respectively.

**Encoding Network.** In the encoder, we set the channel \( r \) (in Eq. (9)) of the encoding temporal output \( Z_t \) to 512.

**Decoding Network.** For the multi-stage decoder, we use five dilated layers with dilated rate 1, 1, 2, 4 and 2. The number of filter channel is set to 512, 256, 256, 512 and 512, respectively. The width of filter is set to 2. For MCF, we set \( W_1 \in \mathbb{R}^{256 \times 512} \), \( W_2 \in \mathbb{R}^{256 \times 512} \) (in Eq. (1)) and \( W_3 \in \mathbb{R}^{256 \times 512} \).

**Training Details.** The vocabulary size is 12,596 for MSVD and 23,308 for MSRVTT, respectively. We use Adam optimizer with an initial learning rate of \( 1 \times 10^{-3} \). We empirically set \( \beta_1 \) and \( \beta_2 \) to 0.9 and 0.1, respectively. And \( \lambda_0 \), \( \lambda_1 \), \( \lambda_2 \) are bias.
and $\lambda_2$ are set to 0.2, 0.2, and 0.6, respectively. Note that we do not conduct beam search in testing.

6.2 Experimental Results

**MSVD Dataset.** On MSVD dataset, we compare our method with other methods. The results are shown in Table 1.

It can be seen that our method outperforms all above methods on the metric of METEOR and CIDEr. Particularly, for the work [Song et al., 2017], they use multi-layer LSTM as the decoder. Our convolutional sequential decoder outperforms the performance of hLSTMmat [Song et al., 2017]. This shows that our method is effective.

<table>
<thead>
<tr>
<th>Method</th>
<th>BLEU@4</th>
<th>METEOR</th>
<th>CIDEr</th>
</tr>
</thead>
<tbody>
<tr>
<td>MA-LSTM [Xu et al., 2017]</td>
<td>36.5</td>
<td>26.5</td>
<td>41.0</td>
</tr>
<tr>
<td>G+LSTM [Venugopalan et al., 2014]</td>
<td>34.6</td>
<td>25.6</td>
<td>-</td>
</tr>
<tr>
<td>C3D+SA [Yao et al., 2015]</td>
<td>36.1</td>
<td>25.7</td>
<td>35.2</td>
</tr>
<tr>
<td>R+MA-LSTM [Song et al., 2017]</td>
<td>38.3</td>
<td>26.3</td>
<td>-</td>
</tr>
<tr>
<td>G+MCNN+MCF-matrix multiply</td>
<td>36.1</td>
<td>26.4</td>
<td>39.2</td>
</tr>
<tr>
<td>R+MCNN+MCF-matrix multiply</td>
<td>38.1</td>
<td>27.2</td>
<td>42.1</td>
</tr>
<tr>
<td>R+MCNN+MCF-element-wise product</td>
<td>37.7</td>
<td>27.1</td>
<td>41.3</td>
</tr>
</tbody>
</table>

Table 1: Comparison with other models on MSVD. Here ‘G’ denotes GoogLeNet. ‘MCNN’ represents our multi-stage CNN. ‘MCF-element-wise product’ and ‘MCF-matrix multiply’ mean we respectively use element-wise product and matrix multiplication in our MCF. All values are measured by percentage (%).

**MSRVTT dataset.** On MSRVTT dataset, we compare our method with representative methods. Results are shown in Table 2. Compared with all above methods which use a single kind of visual feature as input, our method obtains the best performance on METEOR and CIDEr metric. Besides, compared with the work [Xu et al., 2017] which uses many kinds of features as input, our method outperforms its performance. This shows that the performance of our method on a single kind of visual feature is valid.

<table>
<thead>
<tr>
<th>Method</th>
<th>BLEU@4</th>
<th>METEOR</th>
<th>CIDEr</th>
</tr>
</thead>
<tbody>
<tr>
<td>MA-LSTM [Xu et al., 2017]</td>
<td>36.5</td>
<td>26.5</td>
<td>41.0</td>
</tr>
<tr>
<td>G+LSTM [Venugopalan et al., 2014]</td>
<td>34.6</td>
<td>25.6</td>
<td>-</td>
</tr>
<tr>
<td>C3D+SA [Yao et al., 2015]</td>
<td>36.1</td>
<td>25.7</td>
<td>35.2</td>
</tr>
<tr>
<td>R+MA-LSTM [Song et al., 2017]</td>
<td>38.3</td>
<td>26.3</td>
<td>-</td>
</tr>
<tr>
<td>G+MCNN+MCF-matrix multiply</td>
<td>36.1</td>
<td>26.4</td>
<td>39.2</td>
</tr>
<tr>
<td>R+MCNN+MCF-matrix multiply</td>
<td>38.1</td>
<td>27.2</td>
<td>42.1</td>
</tr>
<tr>
<td>R+MCNN+MCF-element-wise product</td>
<td>37.7</td>
<td>27.1</td>
<td>41.3</td>
</tr>
</tbody>
</table>

Table 2: Comparison with other models on MSRVTT. Here ‘G’, ‘R’, and ‘C’ denote GoogLeNet, ResNet, and C3D. ‘MCF-matrix multiply’ and ‘MCF-element-wise product’ represent we respectively use matrix multiplication and element-wise product in our MCF. ‘MCNN’ represents our multi-stage CNN for video captioning.

In Fig. 5, we show some video captioning examples generated by multi-stage CNN which uses different fusion method. We can see that the captions generated by multi-stage CNN using MCF are better than that generated by multi-stage CNN using other fusion methods. Particularly, taking the first and third results as examples, our method successfully identifies ‘pasta’ and ‘makeup’, while are better than above fusion method.
methods. This also demonstrates that based on our architecture, MCF is an effective fusion method for video captioning.

<table>
<thead>
<tr>
<th>Fusion Method (dataset)</th>
<th>BLEU@4</th>
<th>METEOR</th>
<th>CIDEr</th>
</tr>
</thead>
<tbody>
<tr>
<td>Element-wise sum (MSVD)+G</td>
<td>47.13</td>
<td>32.71</td>
<td>71.20</td>
</tr>
<tr>
<td>Element-wise product (MSVD)+G</td>
<td>46.21</td>
<td>32.67</td>
<td>71.92</td>
</tr>
<tr>
<td>Concatenation (MSVD)+G</td>
<td>45.25</td>
<td>32.90</td>
<td>70.64</td>
</tr>
<tr>
<td>Bilinear Pooling (MSVD)+G</td>
<td>43.61</td>
<td>32.46</td>
<td>71.11</td>
</tr>
<tr>
<td>MCF-matrix multiply (MSVD)+G</td>
<td>46.46</td>
<td>33.72</td>
<td>75.46</td>
</tr>
<tr>
<td>Element-wise sum (MSRVTT)+R</td>
<td>37.3</td>
<td>26.4</td>
<td>40.9</td>
</tr>
<tr>
<td>Element-wise product (MSRVTT)+R</td>
<td>36.8</td>
<td>26.6</td>
<td>39.3</td>
</tr>
<tr>
<td>Concatenation (MSRVTT)+R</td>
<td>36.2</td>
<td>26.3</td>
<td>39.9</td>
</tr>
<tr>
<td>Bilinear Pooling (MSRVTT)+R</td>
<td>37.5</td>
<td>26.7</td>
<td>40.9</td>
</tr>
<tr>
<td>MCF-matrix multiply (MSRVTT)+R</td>
<td>38.1</td>
<td>27.2</td>
<td>42.1</td>
</tr>
</tbody>
</table>

Table 3: The results using different fusion method in our multi-stage CNN. Here ‘G’ and ‘R’ denote GoogleNet and ResNet feature. ‘MCF-matrix multiply’ denote using matrix multiplication in MCF.

7 Evaluation on TALL

7.1 Dataset and Implementation Details

We evaluate the benefit of MCF on TACoS dataset [Regneri et al., 2013]. This dataset contains 127 videos. Each video includes two types of annotations. The first is activity labels with temporal location (start and end frame). The second is language descriptions. In total, there are 17,344 pairs of sentence and video clips. We split it in 50% for training, 25% for validation and 25% for test. In experiment, we set $W_1 \in \mathbb{R}^{96 \times 1024}$, $W_2 \in \mathbb{R}^{96 \times 1024}$ (in Eq. (1)), and $W_M \in \mathbb{R}^{96 \times 1024}$ (in Eq. (17)). All other parameter settings are same as those of the work [Gao et al., 2017].

7.2 Experimental Results

Evaluation Metric. We employ the metric used by [Regneri et al., 2013; Gao et al., 2017] to compute ‘R@n, IoU=m’. Comparison with other methods. We compare our method with other methods on TACoS and report the result for IoU $\in \{0.1, 0.3, 0.5\}$ and Recall@{1, 5}. The results are shown in Table 4. ‘Random’, ‘Verb’ and ‘Verb+Obj’ are used by work [Gao et al., 2017] to make contrast. ‘VSA-RNN’ and ‘VST-STV’ [Karpathy and Fei-Fei, 2015] leverage images and their descriptions to learn about their correspondences. ‘CTRL-p’ and ‘CTRL-np’ denote using parameterized and non-parameterized regression loss to train CTRL [Gao et al., 2017]. And we use element-wise product in MCF.

It can be seen from Table 4 that our method outperforms all above methods on most metrics. Particularly, our method achieves a relative improvement of 4.2% on the metric R@5 with IoU=0.1. This shows that MCF is an effective fusion method for this task.

<table>
<thead>
<tr>
<th>Method</th>
<th>R@1 IoU=0.5</th>
<th>R@1 IoU=0.3</th>
<th>R@1 IoU=0.1</th>
<th>R@5 IoU=0.5</th>
<th>R@5 IoU=0.3</th>
<th>R@5 IoU=0.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>0.83</td>
<td>1.81</td>
<td>3.28</td>
<td>3.57</td>
<td>7.03</td>
<td>15.09</td>
</tr>
<tr>
<td>Verb</td>
<td>1.62</td>
<td>2.62</td>
<td>6.71</td>
<td>3.72</td>
<td>6.36</td>
<td>11.87</td>
</tr>
<tr>
<td>Verb+Obj</td>
<td>8.25</td>
<td>11.24</td>
<td>14.69</td>
<td>16.46</td>
<td>21.50</td>
<td>26.60</td>
</tr>
<tr>
<td>VSA-RNN</td>
<td>4.78</td>
<td>6.91</td>
<td>8.84</td>
<td>9.10</td>
<td>13.90</td>
<td>19.03</td>
</tr>
<tr>
<td>VSA-STV</td>
<td>7.56</td>
<td>10.77</td>
<td>15.01</td>
<td>15.50</td>
<td>23.92</td>
<td>32.82</td>
</tr>
<tr>
<td>CTRL-p</td>
<td>11.85</td>
<td>17.59</td>
<td>23.71</td>
<td>23.05</td>
<td>33.19</td>
<td>47.51</td>
</tr>
<tr>
<td>CTRL-np</td>
<td>13.30</td>
<td>18.32</td>
<td>24.32</td>
<td>25.42</td>
<td>36.69</td>
<td>48.73</td>
</tr>
<tr>
<td>CTRL+MCF-m</td>
<td>13.05</td>
<td>17.08</td>
<td>23.16</td>
<td>25.74</td>
<td>35.62</td>
<td>48.80</td>
</tr>
<tr>
<td>CTRL+MCF-p</td>
<td>12.53</td>
<td>18.64</td>
<td>25.84</td>
<td>24.73</td>
<td>37.13</td>
<td>52.96</td>
</tr>
</tbody>
</table>

Table 4: Comparison of different methods on TACoS. ‘MCF-m’ and ‘MCF-p’ denote using matrix multiplication and element-wise product in MCF. All values are measured by percentage (%).

In Fig. 6, we show some examples about TALL task. Particularly, taking the second result as example, our method accurately determines the start and end time for the query which includes two actions.

8 Conclusion

In this paper, we propose the Multi-modal Circulant Fusion (MCF) to combine visual and text representations. We test the MCF on video captioning and TALL tasks. Experimental results on three datasets demonstrate the effectiveness of our method.

Acknowledgments

This work was supported by the NSFC (under Grant U1509206, 61472276, U1736219).
References


