Approximated Bilinear Modules for Temporal Modeling

Xinqi Zhu¹, Chang Xu¹, Langwen Hui², Cewu Lu², and Dacheng Tao¹
¹UBTECH Sydney AI Centre, School of Computer Science, Faculty of Engineering, The University of Sydney, Darlington, NSW 2008, Australia
²Shanghai Jiao Tong University, Shanghai, China
xzhu7491@uni.sydney.edu.au, {c.xu, dacheng.tao}@sydney.edu.au, {kunosarges, lucewu}@sjtu.edu.cn

Abstract

We consider two less-emphasized temporal properties of video: 1. Temporal cues are fine-grained; 2. Temporal modeling needs reasoning. To tackle both problems at once, we exploit approximated bilinear modules (ABMs) for temporal modeling. There are two main points making the modules effective: two-layer MLPs can be seen as a constraint approximation of bilinear operations, thus can be used to construct deep ABMs in existing CNNs while reusing pretrained parameters; frame features can be divided into static and dynamic parts because of visual repetition in adjacent frames, which enables temporal modeling to be more efficient. Multiple ABM variants and implementations are investigated, from high performance to high efficiency. Specifically, we show how two-layer subnets in CNNs can be converted to temporal bilinear modules by adding an auxiliary-branch. Besides, we introduce snippet sampling and shifting inference to boost sparse-frame video classification performance. Extensive ablation studies are conducted to show the effectiveness of proposed techniques. Our models can outperform most state-of-the-art methods on Something-Something v1 and v2 datasets without Kinetics pretraining, and are also competitive on other YouTube-like action recognition datasets. Our code is available on https://github.com/zhuxinqimac/abm-pytorch.

1. Introduction

Video action recognition has been one of the most fundamental problems in computer vision for decades. Since CNNs achieved great success in image classification [23, 36, 40, 16, 17], deep models have been introduced to video domain for action recognition [18, 35, 6, 42, 47, 38, 8, 48]. Different from image classification, video action recognition requires effort for temporal modeling, which is still an open problem in this field.

Up to now there have been three promising ways for temporal modeling in action recognition. The first one is two-stream architecture [35, 9, 47] where the temporal information is captured by optical flow (can cost over 90% of the run time [39]). The second one is 3D CNN [18, 42, 43, 2]. This method has the problem of high pretraining cost because 3D CNNs are hard to be directly used for small datasets due to overfitting. This pretraining can be very expensive, e.g. 64 GPUs used in [2] and 56 GPUs used in [51]. A late fusion step is usually used along with the above two methods for long-term prediction, which slowdowns their inference again. We refer the above two methods as heavy methods. On the contrary, the third way is a light method which conducts temporal modeling based on 2D backbones where input frames are usually sparsely sampled [6, 32, 31, 34, 55, 54, 27]. Without expensive late fusion, preprocessing and postprocessing computational overheads are eliminated. We value these merits of light architectures, and discover a very powerful module which works harmoniously and effectively with them. Additionally, the module we propose is very flexible and can also work with heavy methods to get an evident performance boost.

This paper is based on two discoveries about videos. The first one is: Temporal cues are fine-grained. Here we refer fine-grained since the temporal information (motion or state changes) could be easily dominated by spatial information (color blobs). This property can explain the usage of optical flow [35] or dense trajectory [45] which magnify the impact of temporal features by extracting them explicitly. As bilinear models have been shown effective for fine-grained classification [29, 28], it motivates us to bring bilinear operation to video temporal modeling. The second discovery is: Temporal modeling needs reasoning. Unlike image processing where low-level features like texture or color blobs are crucial for classification, the key features in time could be more high-level and reasoning-required, e.g. basic physics, causality, and human’s intention. As the state-of-the-art technique for VQA problem which requires
textual and visual reasoning is bilinear model \cite{10, 21, 53}, it again inspires us to use bilinear model to do reasoning for temporal sequences.

Based on the discoveries above, we introduce our Approximated Bilinear Modules (ABMs) for temporal modeling. There are two insights that make ABMs effective. The first is that two-layer MLPs can be seen as a constrained approximation of bilinear operations, which enables us to flexibly construct ABMs inside existing deep networks while reusing pretrained parameters. The second is that adjacent frames are likely to be repetitive, so we propose to represent a frame feature with static and dynamic parts to achieve a more efficient computation. We investigate the module’s multiple temporal variants, and how they can work with CNNs smoothly. Particularly, we introduce how ABMs can be carefully initialized so that they can be implanted into deep architectures while keeping pretrained parameters valid. In this paper, our proposed modules are instantiated with two backbones (2D-ResNet-34 and I3D) to show 1. the pure power of ABMs for temporal modeling, and 2. the complementarity with 3D networks. Besides, we also present a flow-inspired snippet sampling to bring short-term dynamics to light models, and also introduce a shifting inference protocol to further boost performance. Our models can outperform most previous state-of-the-art methods on Something-Something v1 and v2 datasets without large video dataset pretraining such as Kinetics, while keeping a decent accuracy-speed tradeoff.

2. Related Work

Deep Learning for Action Recognition. Nowadays deep neural networks have been popular for video action recognition \cite{19, 55, 18, 43, 2, 47, 55}. Karpathy et al. investigated deep models with various temporal fusion strategies on Sports-1M dataset \cite{19}. Ji et al. proposed 3D CNNs for end-to-end action recognition \cite{18}, and this idea has been extended to more general feature representation learning by C3D \cite{42}. Later, more powerful and deeper 3D CNNs with variations have been introduced, such as Res3D \cite{43}, I3D \cite{4} (using inflated ImageNet-pretrained parameters), S3D \cite{51} (looking for cheaper 3D convolutions), and ARTNet \cite{46}. Usually 3D architectures are heavy and require expensive pretraining. Two-Stream architecture \cite{55} utilizes pre-extracted optical flow to capture temporal information. Feichtenhofer et al. investigated different fusion methods to more efficiently conduct two-stream processing \cite{9}. For long-term temporal modeling, Donahue et al. \cite{6}, Ng et al. \cite{32} and Shi et al. \cite{54} adopted LSTMs in video action recognition. Later, Ballas et al. \cite{11} proposed a ConvGRU for video understanding using multi-layer feature maps as inputs. Later Wang et al. \cite{47} proposed TSN architecture for long-term modeling, which is popular in 3D-based methods. Recently some works focus on light-weight temporal modeling. Zhou et al. \cite{54} do to late reasoning in action recognition. Zolfaghari et al. \cite{55} introduced ECO, a hybrid architecture of BN-Inception and 3D-ResNet-18 for fast action recognition. By shifting part of feature vectors, Lin et al. \cite{27} proposed TSM to do temporal modeling without extra parameters. Our work is to implant the bilinear operations into normal convolutions, with the goal of exploiting the fine-grained nature of temporal dynamics, while reusing the normal pretrained parameters as well.

Bilinear Models. Bilinear pooling has been promising in many computer vision tasks \cite{29, 11, 10, 21, 53, 52, 12}. Lin et al. utilized two streams of CNNs for fine-grained classification by extracting two branches of features and fusing them with outer product \cite{29}. To address the high-dimension problem of bilinear pooling, Gao et al. introduced compact bilinear pooling \cite{11} where the projected low-dimensional feature’s kernel approximates the original polynomial kernel. In VQA tasks where inputs are naturally bi-modal, bilinear models are shown very effective \cite{10, 21, 53}. Kim et al. \cite{21} brought bilinear low-rank approximation \cite{33} to VQA, and Yu et al. \cite{53} proposed a rank-n variant. The bilinear operation has also been shown effective to pool over hierarchical layers in CNNs \cite{52}.

There have been some attempts to apply bilinear approaches to video action recognition \cite{5, 50, 13}. Diba et al. \cite{5} proposed Temporal Linear Encoding (TLE) to encode a video into a compact representation using compact bilinear pooling \cite{11}. Wang et al. \cite{50} introduced a spatiotemporal pyramid architecture using compact bilinear pooling to fuse temporal and spatial information with attention. In \cite{13}, a top-down attention model has been developed based on the rank-1 approximation of bilinear pooling operation. However, these attempts either apply the bilinear operations to spatial features or fuse multiple branches of modalities, but none show its potential for temporal modeling.

3. Approach

We first give definitions of ABM with variants, then introduce how they can work with deep architectures smoothly. Later we present two instantiations of our modules, snippet sampling, and implementation details.

3.1. General Approximated Bilinear Module

Definition. A bilinear pooling module \cite{41, 29} calculates the Gram matrix of two global descriptors to learn a pair-wise relation feature. In this paper, we ignore the pool-over-location operation in bilinear pooling, but focus on the simpler bilinear module taking two vectors as inputs:

\[
    z = WVec(xy^T),
\]

where \( z \in \mathbb{R}^D \) is the output vector, and function \( Vec(\cdot) \) vectorizes a matrix. \( x \in \mathbb{R}^C \) and \( y \in \mathbb{R}^C \) denotes two
input vectors each containing $C$ channels. $W \in \mathbb{R}^{D \times CC'}$ is the learnable parameters.

As the number of parameters of this naive bilinear module is too large for widely usage [11][21][53], we factorize each element $w_{kij}$ in weight $W \in \mathbb{R}^{D \times C \times C'}$ by three smaller matrices: $w_{kij} = \sum_{r=1}^{R} u_{kr} a_{ir} b_{jr}$, where $(u_{kr}) = u \in \mathbb{R}^{D \times R}$, $(a_{ir}) = a \in \mathbb{R}^{C \times R}$, $(b_{jr}) = b \in \mathbb{R}^{C' \times R}$ are factorized parameters. Then the General Approximated Bilinear Module (ABM-G, Fig. 1(a)) can be defined as:

$$z = ABM_g(x, y) = u \cdot (a^T x \circ b^T y),$$

where $\circ$ denotes element-wise product. There are many variants of this form exploited in various applications [21][53][50][46], and all of them can be derived from this general form by substituting some specific elements.

**Relation to Two-Layer MLP.** For Eq. [3] if we fix $b^T y = 1$ and add a nonlinear layer in the middle [21], the ABM-G becomes a two-layer MLP. From this viewpoint, a two-layer MLP can be seen as a constrained approximation of the bilinear module, whose bilinear weights are factorized in a constrained way: $w_{kij} = \sum_{r=1}^{R} u_{kr} a_{ir} b_{jr}$, where $\sum_{j=1}^{C'} b_{jr} y_j = 1$, with an additional activation layer for keeping nonlinearity. This constraint just ignores information from $y$ thus no bilinear features are learned. Because of this negative effect, we refer this branch outputting 1 as constrained-branch. Reversely, if we free a two-layer MLP from this constraint by making the weights in the constrained-branch tunable, then the freed MLP, which is now ABM-G with an additional nonlinear layer, can learn a bilinear feature rather than the original linear feature, and we name this tunable branch auxiliary-branch. This transformation enables a pathway to enhance the traditional two-layer MLPs to be more discriminative, which is also the key technique how we implant the ABMs into CNNs’ intermediate layers (Sec. 3.3), while reusing the pretrained weights.

### 3.2. Temporal ABMs

Unlike other bilinear applications where inputs usually comes in dual forms, e.g. two branches [29][11], two modalities [10][21][53][50], it is not very straightforward to apply bilinears to temporal problems. We consider several variants for temporal modeling. The frame features along time are denoted as $\{x_1, x_2, ..., x_t, ...\}$.

**ABM-S.** First we consider simply feeding adjacent frame features to ABM-G’s two entries separately: $z = ABM_g(x_t, x_{t+1})$. This is the most straightforward way and easy to implement. We name it ABM-S (see Fig. 1(b)). The potential drawbacks of this variant are: 1. its temporal receptive field is limited since it can only perceive two frames at once; 2. it lacks self-bilinear capability which is shown effective for classification in some cases [22][25][5]. We propose ABM-C to solve these problems.

**ABM-C.** We consider a second way to feed a concatenation of multiple frames into both ABM entries:

$$z = ABM_c(x_t - \lfloor m/2 \rfloor, ..., x_t, ..., x_t + \lfloor m/2 \rfloor) = ABM_g(x'_t, x'_t),$$

where $m$ denotes the number of concatenated frames. We name this module ABM-C (see Fig. 1(c)). This variant can perceive more frames at once and it is similar to naive convolution so it is easier to work with existing CNNs. In this paper, we fix $m = 3$. We show that ABM-C is more effective than ABM-S in the experiments (Sec. 4.2). A potential problem of this module is its massive parameters since its parameter number grows linearly with the perceived frames. To diminish this problem, we propose ABM-A.

**ABM-A.** We consider an intrinsic property of videos: repetition. For most time, adjacent frames come with duplicated visual information, and the dynamics in them that defines the motion of a video is very subtle and fine-grained. Therefore we propose to divide a single frame descriptor into two parts:

$$x_t = Concat(v^*_t, v^d_t),$$

where $v^*_t$ is the part containing information that looks static to the adjacent frames, and $v^d_t$ containing dynamic information. In other words, we hypothesize that the static part is shared in the short local snippet and it mainly contains static visual information: $v^*_t \approx v^*_t \approx v^*_t$, while the

1Here we only show a snippet of 3 frames.
denote pretrained parameters.

(a) A normal two network with same configurations to fairly show our mod- poral pooling methods [5, 47, 54] using a same backbone compare this implementation with various post-CNN tem- can be initialized randomly which is easy to implement. We
ral modeling at the high-semantic level which can lead to
plex temporal nonlinearity. This model conducts all tempo-
crue temporal fields and capture more com-
be intact. We stack multiple layers of ABM modules to in-
bone is not interfered and all pretrained parameters could
reusing the pretrained parameters.
ABM modules can work with existing deep CNNs, while
eral and flexible enough so that it not only can work in a
plug-and-play manner, but also can enjoy the pretraining of
is newly initialized. (c), (d): Network instantiations. Red arrows are
as a hyper-parameter. When
\( \beta = 0 \), the module becomes purely frame-level and con-
ducts no temporal modeling; when \( \beta = 1 \), the module is
equal to ABM-C for full temporal modeling. We investi-
gate \( \beta = \frac{1}{2} \) and \( \frac{1}{4} \) in the experiments (Sec. 4.2).

3.3. Exploiting Deep Architectures

We are committed to develop our modules to be gen-
eral and flexible enough so that it not only can work in a
plug-and-play manner, but also can enjoy the pretraining of
depth architecture. In this subsection we introduce how our
ABM modules can work with existing deep CNNs, while
reusing the pretrained parameters.

On Top of CNNs. The most straightforward way is to
put our modules on top of deep CNNs so that the back-
hone is not interfered and all pretrained parameters could
be intact. We stack multiple layers of ABM modules to in-
crease the temporal receptive fields and capture more com-
plex temporal nonlinearity. This model conducts all tempo-
ral modeling at the high-semantic level which can lead to
a very high-speed inference. In this case, ABM parameters
can be initialized randomly which is easy to implement. We
compare this implementation with various post-CNN tem-
poral pooling methods [5] [47] [54] using a same backbone
network with same configurations to fairly show our mod-
els’ effectiveness. However this implementation has some
problems: 1. since the ABMs are all initialized randomly
and contain element-wise multiplications, it could not go
too deep or may lead to convergence difficulty and slow
down the training; 2. because of the first problem, it could
not have a large temporal receptive field, leading to limited
temporal modeling capability.

Implanted into CNNs. Additionally, we consider a
more flexible way: implanting ABMs into deep architec-
ture’s intermediate layers. As we show in Sec. 3.1 that
MLPs could be seen as a constrained approximation of bi-
linear operations, we could also reversely transform two-
layer convolutions of CNNs into ABMs by constructing an
auxiliary-branch with tunable weights.

Let’s assume we want to build an ABM-A module out of
two convolutional layers (Fig. 2(a)), which should be quite
common in deep CNN architectures. Firstly we retain the
first convolution \( \text{Conv}1 \) (see \( x_t \rightarrow a_t \) in Fig. 2(a) and (b)).
Secondly we construct its sibling operation \( \text{Conv}2 \) (see
\( x_t \rightarrow a_t' \) in Fig. 2(b)), containing the same input and output
dimensions as \( \text{Conv}1 \). Then we initialize all its weights
to be \( 0 \), and the corresponding bias to be 1 so that initially
whatever the input is, the output of \( \text{Conv}2 \) is 1. By taking
the above two steps, we have manually constructed the
auxiliary-branch (\( b^T y = 1 \), see Sec. 3.1 for explanation).
This guarantees that: \( \text{Conv}1(x) \circ \text{Conv}2(y) = \text{Conv}1(x) \),
meaning the original pathway in the CNN is intact, there-
fore the pretrained parameters are still valid. By freeing the
weights in the auxiliary-branch, we get an ABM-G whose
initial power is the same as the original two-layer network.

If the original CNN is a 2D network (e.g. pretrained on
ImageNet), we also need to adapt it for temporal modeling.
In this case, the input of \( \text{Conv}1 \) and \( \text{Conv}2 \) is expanded
to perceive surrounding frame features. How the expan-
sion happens depends on the type of ABM in used here,
e.g. for ABM-A/C, we need to incorporate the dynamic fea-
ture parts of adjacent frames; for ABM-S, two branches take

| Image Width | Image Height | Image Pixel Depth | Image Format | Image Colorspace | Image Type | Image Orientation | Image Size | Image Resolution | Image Description |
|-------------|--------------|-------------------|--------------|------------------|-------------|------------------|------------|-----------------|------------------|
| 612.0       | 792.0        | 4                 | 10           |                   |             |                  |            |                 |                  |
two neighbored frames as inputs separately. However only the weights corresponding to the current frame are initialized with pretrained parameters while others are set to be zeros (cyan areas in Fig. 2 (b)). This modification also does not change the behavior of the original 2D network so the pretrained parameters are still valid (red-stripe areas in Fig. 2 (b) denote the original CNN pathway with pretrained parameters). Together with the second convolution layer (see \( o'_b \rightarrow z_t \) in Fig. 2(b), which is also \( o_t \rightarrow z_t \) in Fig. 2(a)), we built an ABM-A out of pretrained two-layer convolutions with all parameters preserved, based on the idea that freeing the auxiliary-branch to be tunable. If the original CNN is already 3D, we can construct ABM-C modules by just adding the auxiliary-branch with initialization of \( W = 0, b = 1 \). By implanting ABMs into CNNs with careful initializations, the ABMs can be stacked very deeply with temporal receptive fields becoming very large.

**Network Instantiations.** We instantiate our proposed modules with two architectures: 2D-ResNet-34 [16] and I3D [2]. The 2D-ResNet-34 backbone is used to show the true power of ABMs for temporal modeling. This backbone is pure 2D and pretrained on ImageNet dataset [4] for image classification without any prior knowledge about temporal information. The I3D backbone is used to show the complementarity between our ABMs and the state-of-the-art 3D architecture. It is pretrained on Kinetics dataset [20] for action recognition.

For 2D-ResNet-34, we implant ABMs into each non-down-sampling residual blocks for block-layer 2, 3, and 4. In each block, there are two convolutional layers which is perfect for ABMs’ construction. Following last section, we can add a tunable auxiliary-branch and expand the temporal receptive field to build ABM modules (see Fig. 2 (c)). We add a kernel-2 stride-2 temporal maxpooling layer after block-layer 2 for more efficient computation.

For I3D, we build ABMs aside the larger \( 3 \times 3 \times 3 \) convolution in each Inception block after layer-3c (see Fig. 2 (d)). Though there is no appended layer to form a complete ABM in a single Inception block, by taking into account the next Inception block the ABM is still complete.

### 3.4. Snippet Sampling

In [54][55][27] sparse sampling is used for efficient video processing. Specifically, they divide each video into \( N \) segments and sample a single frame from each. We generally follow this strategy, but also argue that only sampling a single frame per segment discards too much useful short-term information in consecutive frames. Instead, we borrow a strategy from optical flow sampling [35], which each time samples a short snippet (containing \( K \) frames) rather than only a single frame. To keep efficiency of sparse sampling, only the weights in the first convolutional layer are duplicated to perceive the snippet, while rest of the network still feels it is processing \( N \) single frames. In this paper, we choose \( N = 8 \) or 16, and fix \( K = 3 \) after ablation study. Snippet sampling is represented by \( N \times K \) in tables.

### 3.5. Implementation Details

**Training.** For inputs, we randomly sample a snippet in each segment for 2D-ResNet-34 models, and densely sample 64 frames for I3D models. The input frames are scaled to \( 256 \times 256 \) and randomly cropped to \( 224 \times 224 \). For Something-Something v1 and v2 datasets, we use 2D-ResNet-34 backbone pretrained on ImageNet to show the pure effectiveness of our modules’ temporal modeling; for other datasets, we use I3D backbone pretrained on Kinetics. We train all models using SGD with momentum of 0.9. All models are trained or fine-tuned with 0.001 initial learning rate and decayed by 10 twice. For 2D-ResNet-34-top models, lr decays at epoch 30 and 40 (total 50 epochs). For 2D-ResNet-34-implemented models, lr decays at epoch 15 and 20 (total 25 epochs). Because of limited GPU resources, I3D-based models are first trained on Kinetics for 8 epochs, and fine-tuned for 20 epochs on smaller datasets. The lr decays every 8 epochs during fine-tuning. All experiments are conducted on 4 GeForce GTX TITAN X gpus.

**Testing.** During testing of 2D-ResNet-34-based models, we sample the center snippet for each segment to do the inference. We also introduce a new testing protocol by shifting the snippets so that more frames are used in a segment. We define shifting-time \( ST \), so shifted samples of a video will be used for inference, and the output of a video will be the averaged output of each shifted sample. To calculate the shifting-offset, we divide each segment by \( ST \). In this paper, we fix \( ST = 3 \), and we will specify if shifting inference is used in the experiments. For I3D-based models, center 150 frames are used for inference. During validation and testing, the video frames are scaled to \( 224 \times 224 \) then fed to models. No other cropping strategies are used.

### 4. Experiments

We perform comprehensive ablation studies on Something-v1 dataset [14]. Then we compare our models with state-of-the-art methods on various datasets, while showing our models’ generality to optical flow modality. Efficiency analysis and visualization are also provided.

#### 4.1. Datasets

Something-Something v1 [14] and v2 [31] are crowdsourced datasets focusing on temporal modeling, containing fine-grained human motions and human-object interactions. There are 108,499 videos in v1 and 220,847 videos in v2, with 174 categories in each dataset. Besides, other YouTube-like datasets, Kinetics [20], UCF101 [37], and HMDB51 [24], are also used to validate our models on action recognition. These three datasets contain more static-
To calculate VPS (version of TSN [47]; 2. TRN [54]; 3. Compact Bilinear Pooling [11]).

Table 1. Results of On Top of CNNs implementation and some other temporal models on Something-v1 dataset. L denotes the number of ABM-C layers. VPS means video per second.

| Model                        | #Frame | Top-1  | Top-5  | VPS  |
|------------------------------|--------|--------|--------|------|
| Avg Pooling                  | 8      | 18.09  | 43.67  | 85.81|
| TRN [54]                     | 8      | 31.68  | 60.61  | 84.22|
| CBP [11]                     | 8      | 34.40  | 60.70  | 65.50|
| ABM-S-top L=1               | 8      | 29.94  | 57.83  | 84.84|
| ABM-S-top L=3               | 8      | 30.31  | 57.56  | 80.56|
| ABM-C-top L=1               | 8      | 35.49  | 64.11  | 84.82|
| ABM-C-top L=3               | 8      | 38.32  | 66.15  | 83.66|
| ABM-C-top L=3 8x2           | 8x2    | 41.01  | 68.46  | 82.07|
| ABM-C-top L=3 8x3           | 8x3    | 42.35  | 71.82  | 80.38|

Table 2. Comparison among different ABM variants. #Frame is shown in $N \times K$ to indicate the snippet sampling.

| Model                        | #Frame | Top-1  | Top-5  | VPS  |
|------------------------------|--------|--------|--------|------|
| ABM-C-top                    | 8x3    | 42.35  | 71.82  | 80.38|
| ABM-C-in                     | 8x3    | 44.14  | 74.16  | 28.02|
| ABM-A-in $\beta=1/4$         | 16x3   | 44.89  | 74.62  | 40.19|
| ABM-A-in $\beta=1/2$         | 16x3   | 45.67  | 74.80  | 36.65|
| ABM-A-in $\beta=1$ (C)       | 16x3   | 46.08  | 74.32  | 20.12|

Table 3. Shifting Inference. The shifting-time is fixed to 3.

| Inference | ABM-A $\beta=1/4$ | ABM-A $\beta=1/2$ | ABM-C |
|-----------|-------------------|-------------------|-------|
| w/o Shift | 44.89             | 45.67             | 46.08 |
| w/ Shift  | 45.56             | 46.16             | 46.81 |

Does Shifting Inference Work? We show the effect of snippet sampling by increasing snippet length from 1 to 3 on ABM-C-top L=3. Results are in Table 1 bottom. There is an obvious gain when snippet length is increased by just a small number, e.g. 4% boost from K = 1 to K = 3 while the inference time is almost not affected. This is because the additional calculation only comes from the first convolutional layer. However if a snippet is longer than 3, the data loading time will surpass the inference time, slowing down the training, so the snippet length is not further increased.

How to work with CNNs? In Table 2 first half, we compare On Top of CNNs (denoted by top) and Implanted into CNNs (denoted by in) frameworks. We see the top model can run at a very high inference speed, while the in model can achieve higher performance. But in Fig. 3 we can see the top model converges much slower. This is due to the better initialization of in models. Also there is a little training difficulty at the beginning of ABM-C-top, which is more obvious when layers are over 4. Therefore we prefer to use ABMs by implanting them into CNNs, but the top models are more useful when inference speed is a main concern.

In Table 2 second half, we compare ABM-A models and ABM-C models. An ABM-A becomes ABM-C when $\beta = 1$. We can see $\beta$ controls the speed-accuracy tradeoff when it is shifting, but accuracy seems to be very similar between $\beta = 1/2$ and $\beta = 1$. We consider ABM-A with $\beta = 1/2$ as a model with balanced performance and efficiency.

Does Snippet Sampling Work? We show the effect of snippet sampling by increasing snippet length from 1 to 3 on ABM-C-top L=3. Results are in Table 1 bottom. There is an obvious gain when snippet length is increased by just a small number, e.g. 4% boost from K = 1 to K = 3 while the inference time is almost not affected. This is because the additional calculation only comes from the first convolutional layer. However if a snippet is longer than 3, the data loading time will surpass the inference time, slowing down the training, so the snippet length is not further increased.

4.2. Ablation Study on Something-v1

Top-of-CNN Effectiveness. We first conduct experiments with On Top of CNNs structure using 8-frame sampling. No shifting inference is used. Results are shown in Table 1. For comparison, we reimplement several existing post-CNN temporal models: 1. Avg Pooling or naive version of TSN [47]; 2. TRN [54]; 3. Compact Bilinear Pooling [11].

4.3. Comparison with State-of-the-Art

We compare our ABM models with other state-of-the-art methods on Something-v1 and v2 datasets since these two datasets focus on temporal modeling. Our ImageNet-pretrained ABM models can outperform most other methods, showing high effectiveness for temporal modeling.

RGB Models. In the first half section of Table 4 we compare state-of-the-art RGB models. The only models that have the same pretraining setting as ours are Multi-Scale TRN [54] and a baseline model 3D-VGG-LSTM [31]. Under this setting, the two methods can only achieve very limited performance, outperformed by ours by about 10% of accuracy on both sets. With the pretraining of Kinet-
**Table 4. State-of-the-art comparison on Something-v1 and v2 datasets.** En means an ensemble model, ImgN means pretrained on ImageNet, and Kin means pretrained on Kinetics. $N \times K$ means snippet sampling (see Sec. 3.4). Grouped by input modalities.

| Model                | Pretrain | Modality       | #Frame | Backbone          | v1-Val | v1-Test | v2-Val | v2-Test |
|----------------------|----------|----------------|--------|-------------------|--------|---------|--------|---------|
| Multi-Scale TRN \[54\] | ImgN     | RGB            | 8      | BN-Inception      | 34.44  | 33.60   | 48.80  | 50.85   |
| 3D-VGG-LSTM \[51\]   | ImgN     | RGB            | 48     | 3D-VGG            | -      | -       | 51.96  | 51.15   |
| ECO-Lite$_{En}$ \[55\] | Kin      | RGB            | 92     | 2D-Inc+3D-Res     | 46.4   | 42.3    | -      | -       |
| NL 13D \[49\]        | Kin      | RGB            | 64     | 3D-ResNet-50      | 44.4   | -       | -      | -       |
| NL 13D+GCN \[49\]    | Kin      | RGB            | 64     | 3D-ResNet-50      | 46.1   | 45.0    | -      | -       |
| TSM \[27\]           | Kin      | RGB            | 16     | 2D-ResNet-50      | 44.8   | -       | 58.7   | 59.9    |
| TSM$_{En}$ \[27\]    | Kin      | RGB            | 24     | 2D-ResNet-50      | 46.8   | -       | -      | -       |
| ABM-C-in             | ImgN     | RGB            | 16     | 2D-ResNet-50      | 47.45  | -       | -      | -       |
| ABM-C-in             | ImgN     | RGB            | 16×3   | 2D-ResNet-50      | 49.83  | -       | -      | -       |
| ABM-A-in $\beta=1/2$ | ImgN     | RGB            | 16×3   | 2D-ResNet-34      | 46.16  | -       | -      | -       |
| ABM-C-in             | ImgN     | RGB            | 16×3   | 2D-ResNet-34      | 46.81  | 61.25   | 60.13  | 60.13   |
| ABM-AC-in$_{En}$     | ImgN     | RGB            | 32×3   | 2D-ResNet-34      | 49.02  | 42.66   | -      | -       |
| Multi-Scale TRN \[54\] | ImgN     | RGB+Flow       | 8×8   | BN-Inception      | 40.01  | 40.71   | 55.52  | 56.24   |
| ECO-Lite$_{En}$ \[55\] | Kin      | RGB+Flow       | 92×92  | 2D-Inc+3D-Res     | 49.5   | 43.9    | -      | -       |
| TSM \[27\]           | Kin      | RGB+Flow       | 16×8   | 2D-ResNet-34      | 49.6   | 46.1    | 63.5   | 63.7    |
| ABM-C-in             | ImgN     | RGB+Flow       | (16+16)×3 | 2D-ResNet-34 | 50.09  | -       | 63.90  | 62.18   |
| ABM-AC-in$_{En}$     | ImgN     | RGB+Flow       | (32+16)×3 | 2D-ResNet-34 | 51.77  | 45.66   | -      | -       |

**4.4. Efficiency Analysis**

We compare models’ FLOPs and inference speed (by video per second) in Table 5 together with 3D \[49\], ECO \[55\], and TSM \[27\]. VPS of TSM is obtained by reimplementation using the official code. The measurement is conducted on a same single GeForce GTX TITAN X with batch size set to be 8 (2 for I3D because of memory limitation). As we see our heaviest model ABM-A-in $\beta=1$ (is also ABM-C-in) has 1/3 of FLOPs and 5 times faster than I3D while achieving higher accuracy. Compared with efficiency-oriented ECO, two of our models have smaller FLOPs, but can achieve higher accuracy and perceive more frames. Compared with TSM, four of our models inference faster, and three perform better.

By comparing different ABM variants, we can clearly see the accuracy-speed tradeoff controlled by $\beta$: smaller $\beta$ results in lighter model and worse performance. However, the performance drop is not very huge, indicating representing frame features with separated static and dynamic parts is effective. As ABM-C-top works significantly faster than in models, it is an ideal model for online video understanding.

**4.5. Results on Other Datasets**

We validate ABM-C-in module equipped with I3D backbone on Kinetics, UCF101, and HMDB51 datasets using RGB inputs. Because of our limited GPU resources, we only train our model on Kinetics for 8 epochs, with initial learning rate of 0.001 and decay by 0.1 at epoch 4. Thanks to the good initialization method of in models which benefits a lot from the pretrained backbone, it turns out the model works not bad on these datasets. In Table 6 and 7 our model can outperform many other action recognition
methods. Specifically, we outperform I3D [2] on Kinetics, showing our ABM-C module can work complementarily with the 3D architecture. The model fine-tuned on UCF101 and HMDB51 are also competitive, achieving 95.1% and 72.7% accuracy respectively using only RGB inputs.

### 4.6. Keyframe Selection Visualization

We conduct a keyframe selection experiment to qualitatively show that our ABM-C-top model can more effectively capture fine-grained temporal moments than TRN. We compare these two models because they are both post-CNN temporal models and have similar structures. Specifically, on Something-v1 validation set we divide a video into 8 segments and randomly sample one frame per segment to generate a candidate tuple. 200 generated candidate tuples are fed to networks to get predictions, then we select the tuple with the highest top-1 prediction score as the keyframes for this video. In Fig. 4 we show two videos in which two models selected different keyframes (only center 4 frames are shown since the rest selected frames are almost the same for both models). We can see our model performs better on capturing instant key moments in both videos: TRN focused on the falling moment of the objects, while our model captured the instant standing moment of edge supporting. A failure sample of our model is provided in Fig. 5. However the failure is caused by the wiping motion of a hand instead of capturing key moments, and the failure frequency is much lower.

### 5. Conclusion

We brought bilinear modules to temporal modeling, motivated by the temporal reasoning and fine-grained classification. The key points that made ABMs effective and efficient are 1. connection between MLPs and ABMs, and 2. static-plus-dynamic representation of frame features. By exploiting these two ideas, effective ABM temporal variants were proposed, which can work smoothly with existing deep CNNs. We showed in detail how subnets in deep CNNs can be converted to ABMs by adding the auxiliary branch, while keeping the pretrained parameters valid. Our modules were instantiated with 2D-ResNet-34 and I3D backbones. Additionally, we introduced snippet sampling and shifting inference to boost performance of sparse-frame models. It was shown that top models are highly efficient while in models are highly effective. Our models outperformed most state-of-the-art methods on Something-v1 and v2, and were also competitive for traditional action recognition tasks. Though not explored in this paper, our modules should work with other techniques like attention mechanism [7, 26] and non-local module [48], which are remained for future works.

### Acknowledgement

This work is supported by Australian Research Council under Projects FL-170100117, DP-180103424, DE-180101438, and in part by the National Natural Science Foundation of China under Grants 61772332.
References

[1] Nicolas Ballas, Li Yao, Chris Pal, and Aaron Courville. Delving deeper into convolutional networks for learning video representations. In International Conference on Learning Representations (ICLR), 2016.

[2] Joao Carreira and Andrew Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 4724–4733, 2017.

[3] Yunpeng Chen, Yannis Kalantidis, Jianshu Li, Shuicheng Yang, and Jiashi Feng. Multi-fiber networks for video recognition. In ECCV, 2018.

[4] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In CVPR09, 2009.

[5] Ali Diba, Vivek Sharma, and Luc Van Gool. Deep temporal linear encoding networks. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 1541–1550, 2017.

[6] Jeff Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell. Long-term recurrent convolutional networks for visual recognition and description. 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 2625–2634, 2015.

[7] Wenbin Du, Yali Wang, and Yu Qiao. Recurrent spatial-temporal attention network for action recognition in videos. IEEE Transactions on Image Processing, 27:1347–1360, 2017.

[8] Christoph Feichtenhofer, Axel Pinz, and Richard P. Wildes. Spatiotemporal residual networks for video action recognition. In NIPS, 2016.

[9] Christoph Feichtenhofer, Axel Pinz, and Andrew Zisserman. Convolutional two-stream network fusion for video action recognition. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 1933–1941, 2016.

[10] Akira Fukui, Dong Huk Park, Daylen Yang, Anna Rohrbach, Trevor Darrell, and Marcus Rohrbach. Multimodal compact bilinear pooling for visual question answering and visual grounding. In EMNLP, 2016.

[11] Yang Gao, Oscar Beijbom, Ning Zhang, and Trevor Darrell. Compact bilinear pooling. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 317–326, 2016.

[12] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 2414–2423, 2016.

[13] Rohit Girdhar and Deva Ramanan. Attentional pooling for action recognition. In NIPS, 2017.

[14] Raghav Goyal, Samira Ebrahimi Kahou, Vincent Michalski, Joanna Materzynska, Susanne Westphal, Heuna Kim, Valentin Haenel, Ingo Fründ, Peter Yianilos, Moritz Mueller-Freitag, Florian Hoppe, Christian Thurau, Ingo Bax, and Roland Memisevic. The something something video database for learning and evaluating visual common sense.

[15] Dongliang He, Zhichao Zhou, Chuang Gan, Fu Li, Xiao Liu, Yandong Li, Limin Wang, and Shilei Wen. Snet: Local and global spatial-temporal modeling for action recognition. In AAAI, 2019.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 770–778, 2016.

[17] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. Densely connected convolutional networks. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 2261–2269, 2017.

[18] Shuicheng Ji, Wei Xu, Ming Yang, and Kai Yu. 3d convolutional neural networks for human action recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence, 35(1):221–231, Jan 2013.

[19] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. 2014 IEEE Conference on Computer Vision and Pattern Recognition, pages 1725–1732, 2014.

[20] Will Kay, Joao Carreira, Karen Simonyan, Brian Zhang, Chloe Hillier, Sudheendra Vijayanarasimhan, Fabio Viola, Tim Green, Trevor Back, Paul Natsev, Mustafa Suleyman, and Andrew Zisserman. The kinetics human action video dataset. CoRR, abs/1705.06950, 2017.

[21] Jin-Hwa Kim, Kyoung Woon On, Woosang Lim, Jeonghee Kim, Jung-Woo Ha, and Byoung-Tak Zhang. Hadamard Product for Low-rank Bilinear Pooling. In The 5th International Conference on Learning Representations, 2017.

[22] Shu Kong and Charless C. Fowlkes. Low-rank bilinear pooling for fine-grained classification. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 7025–7034, 2017.

[23] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In Advances in Neural Information Processing Systems 25, pages 1097–1105. Curran Associates, Inc., 2012.

[24] Hildegard Kuehne, Hueihan Huang, Estibaliz Garrote, Tomas A. Poggio, and Thomas Serre. Hmdb: A large video database for human motion recognition. 2011 International Conference on Computer Vision, pages 2556–2563, 2011.

[25] Yanghao Li, Naiyan Wang, Jiaying Liu, and Xiaodi Hou. Factorized bilinear models for image recognition. 2017 IEEE International Conference on Computer Vision (ICCV), pages 2098–2106, 2017.

[26] Zhenyang Li, Efstratios Gavves, Mihir Jain, and Cees Snoek. Hadamard convolutes, attends and flows for action recognition. Computer Vision and Image Understanding, 166:41–50, 2018.

[27] Tsung-Yu Lin and Subhransu Maji. Improved bilinear pooling with cnns. CoRR, abs/1811.08383, 2018.
Tsung-Yu Lin, Aruni RoyChowdhury, and Subhransu Maji. Bilinear cnn models for fine-grained visual recognition. In International Conference on Computer Vision (ICCV), 2015.

Tsung-Yu Lin, Aruni RoyChowdhury, and Subhransu Maji. Bilinlear cnn models for fine-grained visual recognition. 2015 IEEE International Conference on Computer Vision (ICCV), pages 1449–1457, 2015.

Farzaneh Mahdisoltani, Guillaume Berger, Waseem Gharbieh, David J. Fleet, and Roland Memisevic. Fine-grained video classification and captioning. CoRR, abs/1804.09235, 2018.

Joe Yue-Hei Ng, Matthew J. Hausknecht, Sudheendra Vijayanarasimhan, Oriol Vinyals, Rajat Monga, and George Toderici. Beyond short snippets: Deep networks for video classification. 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 4694–4702, 2015.

Hamed Pirsiavash, Deva Ramanan, and Charless C. Fowlkes. Bilinear classifiers for visual recognition. In Y. Bengio, D. Schuurmans, J. D. Lafferty, C. K. I. Williams, and A. Culotta, editors, Advances in Neural Information Processing Systems 22, pages 1482–1490. Curran Associates, Inc., 2009.

Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang Chun Woo. Convolutional lstm networks: A machine learning approach for precipitation nowcasting. In NIPS, 2015.

Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. In NIPS, 2014.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. CoRR, abs/1409.1556, 2014.

Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild. CoRR, abs/1212.0402, 2012.

Lin Sun, Kui Jia, Dit-Yan Yeung, and Bertram E. Shi. Human action recognition using factorized spatio-temporal convolutional networks. 2015 IEEE International Conference on Computer Vision (ICCV), pages 4597–4605, 2015.

Shuyang Sun, Zhanghui Kuang, Lu Sheng, Wanli Ouyang, and Wei Zhang. Optical flow guided feature: A fast and robust motion representation for video action recognition. In The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2018.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 1–9, 2015.

Joshua B. Tenenbaum and William T. Freeman. Separating style and content with bilinear models. Neural computation, 12 6:1247–83, 2000.

Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. Learning spatiotemporal features with 3d convolutional networks. In Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV), ICCV ’15, pages 4489–4497, Washington, DC, USA, 2015. IEEE Computer Society.