Re²TAL: Rewiring Pretrained Video Backbones for Reversible Temporal Action Localization

Chen Zhao¹ Shuming Liu¹ Karttikeya Mangalam² Bernard Ghanem¹
¹King Abdullah University of Science and Technology (KAUST), Saudi Arabia ²UC Berkeley, US
{chen.zhao, shuming.liu, bernard.ghanem}@kaust.edu.sa mangalam@berkeley.edu

Abstract

Temporal action localization (TAL) requires long-form reasoning to predict actions of various durations and complex content. Given limited GPU memory, training TAL end to end (i.e., from videos to predictions) on long videos is a significant challenge. Most methods can only train on pre-extracted features without optimizing them for the localization problem, consequently limiting localization performance. In this work, to extend the potential in TAL networks, we propose a novel end-to-end method Re²TAL which rewires pretrained video backbones for reversible TAL. Re²TAL builds a backbone with reversible modules, where the input can be recovered from the output such that the bulky intermediate activations can be cleared from memory during training. Instead of designing one single type of reversible module, we propose a network rewiring mechanism, to transform any module with a residual connection to a reversible module without changing any parameters. This provides two benefits: (1) a large variety of reversible networks are easily obtained from existing and even future model designs, and (2) the reversible models require much less training effort as they reuse the pre-trained parameters of their original non-reversible versions. Re²TAL, only using the RGB modality, reaches 37.01% average mAP on ActivityNet-v1.3, a new state-of-the-art record, and mAP 64.9% at tIoU=0.5 on THUMOS-14, outperforming all other RGB-only methods. Code is available at https://github.com/coolbay/Re2TAL.

1. Introduction

Temporal Action Localization (TAL) [36,53,73] is a fundamental problem of practical importance in video understanding. It aims to bound semantic actions within start and end timestamps. Localizing such video segments is very useful for a variety of tasks such as video-language grounding [23,56], moment retrieval [9,21], video captioning [30,50]. Since video actions have a large variety of temporal durations and content, to produce high-fidelity localization, TAL approaches need to learn from a long temporal scope of the video, which contains a large number of frames. To accommodate all these frames along with their network activations in GPU memory is extremely challenging, given the current GPU memory size (e.g. the commodity GPU GTX1080Ti only has 11GB). Often, it is impossible to train one video sequence on a GPU without substantially downgrading the video spatial/temporal resolutions.

To circumvent the GPU-memory bottleneck, most TAL methods deal with videos in two isolated steps (e.g. [4,67,69,71–73]). First is a snippet-level feature extraction step, which simply extracts snippet representations using a pretrained video network (backbone) in inference mode. The backbone is usually a large neural network trained for an auxiliary task on a large dataset of trimmed video clips (e.g., action recognition on Kinetics-400 [28]). The second step trains a localizer on the pre-extracted features. In this way, only the activations of the TAL head need to be stored in memory, which is tiny compared to those of the backbone (see the illustration of the activation contrast between backbone and localizer in Fig. 1). However, this two-step strategy comes at a steep price. The pre-extracted features can suffer from domain shift from the auxiliary pre-training
task/data to TAL, and do not necessarily align with the representation needs of TAL. This is because they cannot be finetuned and must be used as-is in their misaligned state for TAL. A better alternative is to jointly train the backbone and localizer end to end. But as mentioned earlier, the enormous memory footprint of video activations in the backbone makes it extremely challenging. Is there a way for end-to-end training without compromising data dimensionality?

Reversible networks [20, 25, 31, 48] provide an elegant solution to drastically reduce the feature activation memory during training. Their input can be recovered from the output via a reverse computation. Therefore, the intermediate activation maps, which are used for back propagation, do not need to be cached during the forward pass (as illustrated in Fig. 1). This offers a promising approach to enable memory-efficient end-to-end TAL training, and various reversible architectures have been proposed, such as RevNet [20], and RevViT [48]. However, these works design a specific reversible architecture and train for a particular dataset. Due to their new architecture, they also need to train the networks from scratch, requiring a significant amount of compute resources.

Conversely, it would be beneficial to be able to convert existing non-reversible video backbones to reversible ones, which would (1) avail a large variety of architectures and (2) allow us to reuse the large compute resources that had already been invested in training the non-reversible video backbones. Since pre-trained video backbones are a crucial part of TAL, the ability to convert off-the-shelf backbones to reversible ones is a key to unleash their power in this task.

In this work, for end-to-end TAL, we propose a principled approach to Rewire the architectural connections of a pre-trained non-reversible backbone to make it Reversible, dubbed Re2TAL. Network modules with a residual connection (res-module for short), such as a Resnet block [22] or a Transformer MLP/attention layer [14], are the most popular design recently. **Given any network composed of residual modules, we can apply our rewiring technique to convert it to a corresponding reversible network** without introducing or removing any trainable parameters. Instead of training from scratch, our reversible network can reuse the non-reversible network’s parameters and only needs a small number of epochs for finetuning to reach similar performance. We summarize our contributions as follows.

1. We propose a novel approach to construct and train reversible video backbones parsimoniously by architectural rewiring from an off-the-shelf pre-trained video backbone. This not only provides a large collection of reversible candidates, but also allows reusing the large compute resources invested in pre-training these models. We apply our rewiring technique to various kinds of representative video backbones, including transformer-based Video Swin and ConvNet-based Slowfast, and demonstrate that our reversible networks can reach the same performance of their non-reversible counterparts with only minimum finetuning effort (as low as 10 epochs compared to 300 epochs for training from scratch).

2. We propose a novel approach for end-to-end TAL training using reversible video networks. Without sacrificing spatial/temporal resolutions or network capability, our proposed approach dramatically reduces GPU memory usage, thus enabling end-to-end training on one 11GB GPU. We demonstrate on different localizers and different backbone architectures that we significantly boost TAL performance with our end-to-end training compared to traditional feature-based approaches.

3. With our proposed Re2TAL, we use recent localizers in the literature to achieve a new state-of-the-art performance, 37.01% average mAP on ActivityNet-v1.3. We also reach the highest mAP among all methods that only use the RGB modality on THUMOS-14, 64.9% at tIoU= 0.5, outperforming concurrent work TALLFormer [10].

2. Related Work

**Reversible Networks** are a family of neural models based on the reversible real-valued non-volume preserving (real NVP) transformation introduced in [12, 13]. The transformation has been originally applied widely for image generation using generative flows [24, 29], and later used for various applications, such as compression [38], denoising [43], and steganography recovery [49]. Furthermore, it has also been repurposed for memory-efficient neural network training for a variety of architectures such as ConvNets [20, 25], Masked ConvNet [57], RNNs [47], Graph Networks [31] and more recently, Vision Transformers [48]. However, each of these methods only focuses on one or several specific architectures, and trains their newly proposed reversible architectures from scratch, thereby requiring large compute resources. In this work, we propose a rewiring scheme to adapt off-the-shelf models to reversible architectures which only utilize a small amount of compute for finetuning. This allows democratizing the reach of reversible architecture to arbitrary models and dataset settings by leveraging the architecture design effort and computational cost already spent in vanilla off-the-shelf models.

**Video Recognition Backbones.** ConvNets have had a long and illustrious history of improving video understanding performance [7, 16–19, 26, 33, 52, 55, 61, 64, 65, 74]. Among those, Slowfast [17], which uses a slow pathway and a fast pathway to capture spatial semantics and temporal motion respectively, has been widely adopted for various tasks such as action detection [17], untrimmed video classification [44] and moment/natural language retrieval [21] for its high efficiency and efficacy. Recently, video transformers have ushered in a new life in the field of video repre-
presentation learning. Leveraging long-term temporal connection via the attention mechanism, transformers have quickly gain popularity as the backbone of choice for several video recognition workloads [3, 5, 15, 32, 45, 70]. Among those, Video Swin Transformer [45] introduces an inductive bias of locality to video transformers, leading to an outstanding speed-accuracy trade-off. However, most of the models are vanilla non-reversible architectures. In this work, we unleash the potential of reversible architectures for these models such that the good properties of reversible models, e.g., memory efficiency, can be infused into them. In particular, we show that the pretrained Video Swin Transformer [45] and SlowFast [17] models can be rewired to be reversible, and finetuned cheaply to improve temporal action localization performance with end-to-end training.

Temporal Action Localization (TAL). Due to the conflict between large video data and the GPU memory limit, most TAL methods using deep networks are two-step methods [4, 8, 34, 36, 37, 41, 46, 51, 54, 69, 71–73]. For example, using pre-extracted features, G-TAD [69] and VSGN [73] utilize graph convolutions to model temporal relations between snippets, and ActionFormer [71] leverages transformers to capture long-range context. To mitigate the performance gap between the two-step mechanism and real end-to-end training, some methods explore post-pretraining to enhance the feature representation for TAL [2, 66, 68]. In the meanwhile, researchers also attempt to perform real end-to-end training by reducing network/data complexity [10, 35, 39, 40, 42, 63, 66]. R-C3D [66] is the end-to-end pioneer, but it uses a shallow network C3D [60], thus restricting the performance. PBRNet [39] and ASFD [35] downscale the frame resolution to 96×96. DaoTAD [63] makes RGB-only enough by using end-to-end training with various data augmentations. TALLFormer [10] proposes to use a feature bank strategy and only updates a portion of features during end-to-end training. Compared to these approaches, our method doesn’t sacrifice any data dimensionality or data samples in training, and it supports very deep networks. We significantly reduce memory consumption while preserving the full data fidelity. Moreover, our work is complementary to these methods, and can also be used jointly to further reduce memory cost.

3. Method: Re²TAL

3.1. TAL Formulation and Architecture

Temporal action localization (TAL) predicts the timestamps of actions from a video sequence, which is formulated as follows. Given a video sequence \( V \) of \( T \) frames \( \{I_t \in \mathbb{R}^{3 \times H \times W}\}_{t=1}^{T} \), TAL predicts a set \( m \) of actions \( \Phi = \{\phi_m = (t_{m,s}, t_{m,e}, c_m, s_m)\}_{m=1}^{M} \), where \( t_{m,s} \) and \( t_{m,e} \) are action start and end time respectively, \( c_m \) is action label, and \( s_m \) is prediction confidence. To achieve this, the following two steps are required.

In the first step, the videos \( V \) are encoded as \( N \) features vectors \( \{x_n \in \mathbb{R}^{C}\}_{n=1}^{N} \) via a backbone. This backbone aggregates spatial information within video frames, as well as temporal information across frames. It is usually designed for an auxiliary task such as action recognition. Popular backbones can be categorized into Resnet-based architectures, such as I3D [7], R2+1D [62], SlowFast [17], and Transformer based architectures, such as ViViT [3], Video Swin Transformers [45].

In the second step, a localizer, uses a ‘neck’ to further aggregate the video features \( \{x_n\}_{n=1}^{N} \) in the temporal domain, and a ‘head’, to make predictions of action boundaries, i.e., start and end timestamps \( (t_{m,s}, t_{m,e}) \) and categories. The neck contains layers of networks, e.g., 1D convolutional networks in BMN [36], Graph networks in G-TAD [69] and VSGN [73], and Transformers in RTD-Net [59] and ActionFormer [71].

As mentioned in Sec. 1, an optimal way to train the entire TAL network is to jointly train the backbone and the localizer end to end. However, it is substantially challenging to fit all activation maps of the long video sequence into limited GPU memory. In the following sections, we provide end-to-end TAL solution Re²TAL.

3.2. Rewire for Reversibility, and Reuse

Let’s first analyze what in the GPU memory precludes end-to-end training of TAL. As mentioned in Sec. 1, the activation maps in the backbone are the major occupant in the memory. Concretely, given a batch of a video sequence of \( T \) frames, each with resolution \( S \times S \), to train them on a backbone of \( L \) layers each with \( C \) channels, the training memory complexity is \( O(LCTS^2) \). Downgrading any data dimensionality or the backbone capacity can reduce the memory complexity, but at the same time, may harm the prediction accuracy. We aim to reduce memory consumption without sacrificing the data dimensionality or the model capacity.

Actually, the intermediate activations are stored in memory during training for the purpose of gradient computation in back propagation. If we can reconstruct them during back propagation, then there is no need for their memory occupation. Reversible networks [20, 25, 48, 49] is a superb solution for reconstructing the input from output, and various reversible architectures have been proposed recently (e.g., Revnet [20], RevViT [48]). However, the performance of TAL heavily depends on the backbone that is pre-trained on one or even multiple large-scale datasets. For example, the popular I3D [7] backbone is trained on Kinetics-400 [28] with its parameters initialized from Resnet [22] trained on ImageNet [11]. If we make up a new reversible network for TAL as is the case with previous reversible networks (e.g., [20, 25]), we need to go through all the expensive and time-consuming training stages to reach an equivalently
In Transformers, if $F$ normalization, and ReLU can be one Resnet, the building block consisting of convolution, batch normalizations, and can contain any computations. For example, in any row). The blocks $F$... connections, but make it end after two blocks, as shown in the second row of Fig. 2. As a result, two consecutive residual connections in concurrent neural network architectures, such as Resnet [22] and Transformers [14]. Given any residual modules, we can rewire them into reversible modules. Fig. 2 illustrates this process with an example of 3 consecutive residual modules that are in the same stage, i.e., there are no downsampling operations in between. Each of the original residual modules contains a block $F_i$ (blue boxes) where $i = 1, 2, 3, \ldots$ and a residual connection (orange arrow). The blocks $F_i$ have the same input and output dimensions, and can contain any computations. For example, in Resnet, the building block consisting of convolution, batch normalization, and ReLU can be one $F_i$ block (e.g. Fig. 3 (a)). In Transformers, if $F_i$ represents an attention layer, then $F_{i+1}$ is an MLP layer (e.g. Fig. 3 (b) and (c)). The residual connection in each module only skips one $F_i$ block in the same module.

To rewire, we simply let the residual connection skip the next block $F_{i+1}$ as well as the current one $F_i$. To be more specific, we keep the starting point of each residual connection, but make it end after two blocks, as shown in the second row of Fig. 2. As a result, two consecutive residual connections become overlapped, and there are always two pathways of activations (gray and green in Fig. 2) throughout the modules, as shown in the third row of Fig. 2.

To prepare a second pathway of input to the first module, we duplicate the input $x_0$ to obtain $y_0$. To combine the results $x_3$ and $y_3$ of the two pathways, we simply average them at the end of the modules. This design guarantees that the dimensions of the input and output of every block $F_i$ are identical before and after the rewiring. Therefore, the structure of $F_i$ stays exactly the same.

**Reversibility.** In the following, we mathematically formulate the rewired modules to show their reversibility. To be concise, we use the first two modules as an example. Given the input activations $x_0$ and $y_0$, and the blocks $F_1$ and $F_2$, the output of the two blocks are $x_1, y_1$ and $x_2, y_2$ respectively, computed as follows

$$
\begin{align*}
    y_1 &= x_0 \\
    x_1 &= F_1(x_0) + y_0,
\end{align*}
\quad
\begin{align*}
    y_2 &= x_1 \\
    x_2 &= F_2(x_1) + y_1.
\end{align*}
$$

The above equations are reversible, which means that we can recover the input $x_1, y_1$ from $x_2, y_2$, and then $x_0, y_0$ from $x_1, y_1$. The reverse computation is formulated in the following equation

$$
\begin{align*}
    x_0 &= y_1 \\
    y_0 &= x_1 - F_1(x_0),
\end{align*}
\quad
\begin{align*}
    x_1 &= y_2 \\
    y_1 &= x_2 - F_2(x_1).
\end{align*}
$$

A network with more modules follows the same strategy as in Eq. 1 and Eq. 2. When we stack multiple consecutive reversible modules as one network (as the example in Fig. 2), the entire network is reversible. In this case, we can start from the last module and sequentially reconstruct the input of each module with Eq. 2.

**Reuse.** As the structure and parameter dimensions of the $F_i$ block in our reversible modules after rewiring stay exactly the same as the $F_i$ block in the corresponding residual module, we can directly reuse the pre-trained parameters in the residual modules to initialize our reversible modules.

Our rewiring mechanism provides a large collection of reversible candidates. We can easily convert a network into a reversible one without designing a new architecture from scratch. Not only can we use the existing architectures, but we can also benefit from future even better models to obtain better reversible models. The reuse strategy...
3.3. Reversible Temporal Action Localization

Considering that the backbone is the heaviest part in the TAL network in terms of memory usage, as illustrated in Fig. 1, we target the backbone and apply our Re² technique described above to rewire it to obtain a reversible network.

A backbone network, e.g., Video Swin Transformer [45], Slowfast [17], is usually comprised of several stages. Within each stage, the activation sizes stay the same. We convert all the modules into reversible ones following the rewiring method proposed in Sec. 3.2. During training, we can clear all the input and intermediate activations from GPU memory, and only store the final output of the stage during the forward pass (as shown in Fig. 1). In the back propagation, we re-compute all the activations based on the reverse process as in Eq. 2. As [48] pointed out, this re-computation doesn’t incur too much more computational time since we can parallelize the process to make use of the spare computation of GPUs.

Across stages, there are downsampling layers, which reduce the activation sizes. We leave the downsampling layers as they are and cache the activations inside to enable back propagation. Considering that there are only several downsampling layers, the memory occupation of these activations is acceptable.

3.4. End-to-End TAL Training

For end-to-end training with our Re²TAL, we just need to do the following: find a well-performing video backbone, rewire it into a reversible one, load the parameters from the original backbone to the reversible one and finetune for several epochs on the pretraining task, and train with a localizer end to end. This reversible TAL network is significantly more efficient in memory usage, enabling end-to-end training a GPU of limited memory (e.g., as small as a 11GB commodity GPU). But is it a wise choice to directly adopt the training strategies from the feature-based methods?

Input frame arrangement. Since most TAL methods are designed and experimented with the pre-extracted, they have predisposed to particular design choices, such as extracting features with overlapped snippets to arrange the input frames. But we find it not ideal for end-to-end training.

Fig. 4 (a) illustrates the framework commonly used in two-step (or feature-based) methods. To extract \( N \) feature vectors as the localizer input, \( N \) snippets need to be processed by the backbone independently. Each snippet contains a sequence of \( D \) frames. The snippets are usually overlapped with one another to maintain temporal consistency, which causes duplicate computation and extra memory occupation. Consequently, at least \( ND \) frames need to go through the backbone.

This framework works fine with the two-step method, since feature extraction is one-off effort and all the snippets can be processed sequentially to circumvent the memory issue. However, end-to-end training cannot bear the extra memory cost. Therefore, we treat the entire video sequence as one single input instead, and the backbone aggregates all frames at one time, as shown in Fig. 4 (b). This mechanism not only reduces the cost incurred by the duplicate computation, but also has the advantage of aggregating long-term temporal information, in contrast to being restricted within the snippet as with the snippet mechanism. In this way, to obtain \( N \) feature vectors, we just need to process \( Ns \) frames, where \( s \) is the overall strides in the backbone (e.g., \( s=2 \) in Video Swin Transformers). Considering that \( s \) is usually much smaller than \( D \), the frame-input mechanism is more efficient in memory and computation cost.

4. Experiments

Datasets and evaluation metrics. We present our experimental results on two representative datasets ActivityNet-v1.3 (ActivityNet for short) [6] and THUMOS-
Table 1. Advantage of end-to-end training. End-to-end training leads to significant mAP (%) boost for TAL with either backbone.

<table>
<thead>
<tr>
<th>Method</th>
<th>RE²Vswin-tiny</th>
<th>RE²Slowfast-101</th>
</tr>
</thead>
<tbody>
<tr>
<td>Features</td>
<td>0.5 0.75 0.95</td>
<td>0.5 0.75 0.95</td>
</tr>
<tr>
<td>End2End</td>
<td>51.18 37.23</td>
<td>9.47 32.53</td>
</tr>
</tbody>
</table>

14 (THUMOS for short) [27]. ActivityNet is a challenging large-scale dataset, with 19994 temporally annotated untrimmed videos in 200 action categories, which are split into training, validation and testing sets by the ratio of 2:1:1. THUMOS contains 413 temporally annotated untrimmed videos with 20 action categories, in which 200 videos are for training and 213 videos for validation. For both datasets, we use mean Average Precision (mAP) at different tIoU thresholds as the evaluation metric. On ActivityNet, we choose 10 values in the range \([0.5, 0.95]\) with a step size 0.05 as tIoU thresholds; on THUMOS, we use tIoU thresholds \([0.3, 0.4, 0.5, 0.6, 0.7]\); following the official evaluation practice.

Backbones and localizers of RE²TAL. For backbones, we choose two representative models from the Resnet and Transformer families respectively for experimental demonstration: Video Swin Transformers (Vswin for short) [45] and Slowfast [17], both well known for their powerful video representation and memory efficiency. For Vswin, we use three variants: tiny, small, and base; for Slowfast, we use 50, 101, and 152. For localizers, we experiment with the recent temporal action localization (TAL) methods VSGN [73] and ActionFormer [71].

Implementation Details of RE²TAL. We initialize all our reversible backbones with their non-reversible counterparts, and finetune for up to 30 epochs on Kinetics-400 with Cosine Annealing learning rate policy and Adamw (for Vswin) and SGD (for Slowfast) optimizers. Actually, in our experiments, we find 10 epochs of finetuning already reaches similarly good performance. With the pre-trained reversible backbone, we train TAL on a single GPU, A100 with batchsize = 1 for Vswin and V100 with batchsize = 2 for Slowfast. This is only possible due to the massive GPU memory savings from the reversible backbones. For the hyper-parameters, we follow the original training recipe of VSGN [73] and ActionFormer [71] for the learning rates and epochs in the localizers. We set the learning rates of the Vswin backbones 1 magnitude lower than the localizer, and those of the Slowfast backbones 2 magnitude lower. The spatial resolution is \(224 \times 224\). For ActivityNet, the number of frame input is \(T = 512\), the number of feature vectors is \(N = 256\); and for THUMOS, the number of frame input is \(T = 1024\), the number of features is \(N = 512\).

1The training and validation sets of THUMOS are temporally annotated videos from the validation and testing sets of UCF101 [58], respectively.
4.1.2 How effective and efficient is Re²TAL?

An alternative way to enable end-to-end training is to reduce data resolutions [39, 63]. In Fig. 6, we downscale the input videos in the spatial or temporal dimensions to adjust the memory requirements, and compare our Re²TAL models to their corresponding original models. We can see that under even smaller memory requirements, our Re²TAL models achieve higher performance than the original models that rely on sacrificing video resolutions to reduce memory.

To further visualize the memory efficiency of our Re²TAL compared to their non-reversible counterparts when trained end to end, we demonstrate the GPU memory consumption of the three Vswin backbones of different depths and widths and the three Slowfast backbones of different depths in Fig. 7. For Vswin backbones, we assume a GPU memory budget of 90GB, which is the size of A100. For Slowfast backbones, we assume our GPU budget is 11GB, which is the size of the commodity GPU such as GTX1080Ti. We can see that for either case, the non-reversible network will go out of memory when the model size reaches a certain level. In contrast, our Re²TAL almost keeps constant memory usage when only the network depth increases (from Vswin-tiny to small, from Slowfast-50 to 101). With the low memory cost, our Re²TAL enables end-to-end training with a deep Slowfast backbone on one single 11GB GPU. Moreover, Re²TAL doesn’t incur much extra training time compared to its non-reversible counterpart. The time to train one epoch is the following, Vswin-tiny (114 mins) vs. Re²Vswin-tiny (135 mins); Slowfast-50 (147 mins) vs. Re²Slowfast-50 (158 mins).

4.1.3 Why rewiring and reuse?

We rewire existing network architectures to make them reversible, and reuse their parameters for initialization. This way we dramatically reduce the effort of training the reversible networks, and still reach the representation capability of the original non-reversible ones. To compare the video representation capabilities of both types of networks, we use them to extract video features and train a localizer (VSGN [73] in this case) with the features. We demonstrate their performance in Tab. 2, showing that our reversible models are comparable to the original models.

If there are two versions of the same non-reversible model trained in different ways and with different performance, will our Re²TAL models benefit more from the higher-performing version? Vswin-base happens to have such two versions: one trained on Kinetics-400 and the other trained on Kinetics-600, the latter with better performance. We initialize our Re²Vswin-base with either version; memory reduction from Slowfast-101 to 152 is because the former uses input configuration (8, 8) while the latter uses (4, 16).

---

Table 2. Comparison of feature representations between the Re²TAL and original models, in terms of average mAP (%) on the dataset ActivityNet. Vw: Vswin; Slowf: Slowfast.

<table>
<thead>
<tr>
<th>Model</th>
<th>Tiny</th>
<th>Small</th>
<th>Base</th>
<th>Model</th>
<th>50</th>
<th>101</th>
<th>152</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vswin</td>
<td>34.36</td>
<td>33.86</td>
<td>34.47</td>
<td>Slowfast</td>
<td>34.60</td>
<td>35.04</td>
<td>34.61</td>
</tr>
<tr>
<td>Re²Vw</td>
<td>34.47</td>
<td>34.04</td>
<td>34.09</td>
<td>Re²Slowf</td>
<td>34.93</td>
<td>35.24</td>
<td>34.54</td>
</tr>
</tbody>
</table>

Table 3. Effectiveness of rewiring and reusing pre-trained video models. Reusing pre-trained models leads to significantly better performance than training from scratch (compare Row 1 to the rest). Reusing a better pre-trained model gives even higher performance (compare Row 2 to Row 3). Pret.: pretraining.

<table>
<thead>
<tr>
<th>Pret. Model</th>
<th>Pret. Dataset</th>
<th>0.5</th>
<th>0.75</th>
<th>0.95</th>
<th>Avg.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vswin-base</td>
<td>None</td>
<td>44.58</td>
<td>28.41</td>
<td>7.40</td>
<td>28.68</td>
</tr>
<tr>
<td>Kinetics-400</td>
<td></td>
<td>52.72</td>
<td>36.73</td>
<td>8.88</td>
<td>35.73</td>
</tr>
<tr>
<td>Kinetics-600</td>
<td></td>
<td>52.46</td>
<td>37.37</td>
<td>10.39</td>
<td>36.28</td>
</tr>
</tbody>
</table>

Table 4. Comparison of localization performance with different backbones and localizers, in terms of average mAP (%) on the dataset ActivityNet.

<table>
<thead>
<tr>
<th>Backbone</th>
<th>Localizer</th>
<th>0.5</th>
<th>0.75</th>
<th>0.95</th>
<th>Avg.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Re²Vswin-tiny</td>
<td>VSGN</td>
<td>53.24</td>
<td>37.23</td>
<td>10.49</td>
<td>36.38</td>
</tr>
<tr>
<td></td>
<td>ActionFormer</td>
<td>54.75</td>
<td>37.81</td>
<td>9.03</td>
<td>36.80</td>
</tr>
<tr>
<td>Re²Slowfast-101</td>
<td>VSGN</td>
<td>53.63</td>
<td>37.53</td>
<td>10.67</td>
<td>36.82</td>
</tr>
<tr>
<td></td>
<td>ActionFormer</td>
<td>55.25</td>
<td>37.86</td>
<td>9.05</td>
<td>37.01</td>
</tr>
</tbody>
</table>

---

2Memory increases from Vswin-small to base is due to the channel in-
Table 5. Compared to the state-of-the-art for temporal action localization performance on ActivityNet-v1.3 and THUMOS-14, measured by mAPs (%) at different tIoU thresholds according to their respective official metrics. E2E: end-to-end; Mem: memory (GB).

<table>
<thead>
<tr>
<th>Method</th>
<th>Backbone</th>
<th>E2E</th>
<th>Flow</th>
<th>Mem</th>
<th>ActivityNet-v1.3</th>
<th>THUMOS-14</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.5</td>
<td>0.75</td>
</tr>
<tr>
<td>TAL-Net [8]</td>
<td>I3D</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
<td>38.23</td>
<td>18.30</td>
</tr>
<tr>
<td>BMN [36]</td>
<td>TSN</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
<td>50.07</td>
<td>34.78</td>
</tr>
<tr>
<td>G-TAD [69]</td>
<td>TSN</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
<td>50.36</td>
<td>34.60</td>
</tr>
<tr>
<td>TSI [41]</td>
<td>TSN</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
<td>51.18</td>
<td>35.02</td>
</tr>
<tr>
<td>BC-GNN [4]</td>
<td>TSN</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
<td>50.56</td>
<td>34.75</td>
</tr>
<tr>
<td>VSGN [73]</td>
<td>TSN</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
<td>32.38</td>
<td>36.01</td>
</tr>
<tr>
<td>ActionFormer [71]</td>
<td>I3D</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
<td>33.96</td>
<td>34.97</td>
</tr>
<tr>
<td>PBRNet [39]</td>
<td>I3D</td>
<td>✓</td>
<td>✓</td>
<td>12</td>
<td>32.40</td>
<td>35.30</td>
</tr>
<tr>
<td>AFSD [35]</td>
<td>I3D</td>
<td>✓</td>
<td>✓</td>
<td>12</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>R-C3D [66]</td>
<td>C3D</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
<td>26.80</td>
<td>-</td>
</tr>
<tr>
<td>DaoTAD [65]</td>
<td>I3D</td>
<td>✓</td>
<td>✓</td>
<td>11</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>TALLFormer [10]</td>
<td>VSwin-Base</td>
<td>✓</td>
<td>✓</td>
<td>29</td>
<td>54.10</td>
<td>36.20</td>
</tr>
<tr>
<td>ActionFormer [71]</td>
<td>VSwin-Tiny</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
<td>53.83</td>
<td>35.82</td>
</tr>
<tr>
<td>ActionFormer + Re²TAL</td>
<td>Re²VSwin-Tiny</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
<td>54.75</td>
<td>37.81</td>
</tr>
<tr>
<td>ActionFormer + Re²TAL</td>
<td>Slowfast-101</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
<td>33.98</td>
<td>37.00</td>
</tr>
<tr>
<td>ActionFormer + Re²TAL</td>
<td>Re²Slowfast-101</td>
<td>✓</td>
<td>✓</td>
<td>6.8</td>
<td>55.25</td>
<td>37.86</td>
</tr>
</tbody>
</table>

4.1.4 Choice of Backbones and Localizers

In Tab. 4, we demonstrate the results using the two different backbones Re²VSwin-tiny and Re²Slowfast-101, and two different localizers VSGN [73] and ActionFormer [71] on ActivityNet. Since the ActionFormer localizer yields better performance for both Re²VSwin-tiny and Re²Slowfast-101, we compare the ActionFormer results to other methods in the literature, and also apply them to the THUMOS dataset, as shown in Sec. 4.2.

4.2. State-of-the-Art Comparisons

We compare the performance of our Re²TAL to recent state-of-the-art (SOTA) methods in the literature in Tab. 5 on ActivityNet and THUMOS. On ActivityNet, our Re²TAL reaches a new SOTA performance: average mAP 37.01%, outperforming all other methods by significant margins. On THUMOS, ours surpasses the concurrent work TaLLFormer [10] with mAP 64.9% at tIoU=0.5. The Taskonomy models with the highest among all methods that only use the RGB modality.

Furthermore, for an apple-to-apple comparison with the feature-based method ActionFormer, we re-ran ActionFormer using their official code with the RGB features extracted with the VSwin-tiny and Slowfast-101 backbones, corresponding to our Re²VSwin-tiny and Re²Slowfast-101, respectively. We see that our Re²TAL always outperforms vanilla ActionFormer under the same backbone categories (e.g. Re²VSwin-tiny and VSwin-tiny are in the same backbone category; Re²Slowfast-101 and Slowfast-101 are in the same backbone category).

5. Conclusions

In this work, we propose a novel rewiring-to-reversibility (Re²) scheme to convert off-the-shelf models into reversible models while preserving the number of trainable parameters. The procedure allows reusing the compute invested in training large models and adds only a tiny sliver of fine-tuning compute. We apply the procedure to video backbones such as Video Swin (VSwin) and SlowFast to obtain Re²Vwin and Re²Slowfast backbones respectively. Further, we utilize the Re² backbones for memory-efficient end-to-end temporal action localization, reaching mAP 64.9% at tIoU=0.5 on THUMOS-14, and average mAP 37.01% on ActivityNet-v1.3, establishing a new state-of-the-art. We hope that future work in this direction can explore extending the Re² method to other memory-bottlenecked tasks such as dense video captioning, movie summarization etc.

Acknowledgments. This work was supported by the King Abdullah University of Science and Technology (KAUST) Office of Sponsored Research through the Visual Computing Center (VCC) funding and SDAIA-KAUST Center of Excellence in Data Science and Artificial Intelligence.
References


[38] Kang Liu, Dong Liu, Li Li, Ning Yan, and Houqiang Li. Semantics-to-signal scalable image compression with learned reversible representations. International Journal of Computer Vision (IJCV), 2021. 2


