TimeSformer: Is Space-Time Attention All You Need for Video Understanding?

Wonseok Lee · Lunit Team Blog on Medium · October 12, 2021

Reader-formatted course copy of the public article. Original: https://medium.com/lunit/timesformer-is-space-time-attention-all-you-need-for-video-understanding-5668e84162f4

Oct 12, 2021

In this article, we discuss a paper about a novel video classification method that is formed only of self-attention block (as in ViT). This post explains the paper with both equations and **code snippets**. By looking into the implementation, we can clearly understand how the designed module actually operates. Also, the code snippets in this article can be used as building blocks for designing your own custom block, instead of implementing it from scratch. Please note that the authors have made the paper and code available.

$$\boldsymbol{\alpha}_{(p,t)}^{(\ell,a)} = \mathrm{SM}\left(\frac{\mathbf{q}_{(p,t)}^{(\ell,a)\top}}{\sqrt{D_h}} \cdot \left[\mathbf{k}_{(0,0)}^{(\ell,a)} \left\{\mathbf{k}_{(p',t')}^{(\ell,a)}\right\}_{\substack{p'=1,\ldots,N \\ t'=1,\ldots,F}}\right]\right) \tag{5}$$

Complicated mathematical equations have quite simple representation in code.

attn = (q @ k.transpose(-2, -1)) * self.scale

attn = attn.softmax(dim=-1)

Abstract

In this paper, the authors presented:

- A convolution-free approach to video classification built exclusively on self-attention over space and time
- An experimental study that compares different self-attention schemes, and suggests that “divided attention”, where temporal attention and spatial attention separately applied within each block, leads to the best video classification accuracy

- State-of-the-art (at the time of presentation) on the action recognition benchmarks Kinetics-400 and Kinetics-600
- A comparison to 3D convolutional networks; the model is faster to train, it can achieve dramatically higher test efficiency, and it can also be applied to much longer clips

In the following section, the video classification model is explained. In short, their architecture follows the ViT, but it's extended to the time dimension. The described method doesn't restrict its self-attention scheme to a single method. They show various self-attention schemes which include spatial attention(2D) to the proposed divided space-time attention. In the experiment section, those self-attention schemes are compared in two different datasets.

Method

Input clip. TimeSformer takes as input a clip X of size of $H \times W \times 3 \times F$ consisting of F RGB frames of size $H \times W$ sampled from the original video.

Decomposition into patches. Following the ViT, each frame is decomposed into N non-overlapping patches of size $P \times P$.

Linear embedding. Each patch $x(p,t)$ is linearly mapped into an embedding vector $z(0)$ of size D by means of a learnable matrix E of size $D \times 3P^2$:

$$\mathbf{z}_{(p,t)}^{(0)} = E\mathbf{x}_{(p,t)} + \mathbf{e}_{(p,t)}^{pos}$$

Here, the subscription (p, t) represents the spatial and time position of each patch ($p=1,2,...,N$ and $t=1,2,...,F$). The superscription (0) means that it is the first embedding token. Also note that the subscription $(0, 0)$ is allocated for theclass token. $e\_pos(p, t)$ is a learnable positional embedding to encode the spatiotemporal position of each patch. The embeddings z become the input to the transformer.

In the author's official implementation, *patch decomposition* and *linear embedding* is implemented as a 2d convolution and the positional embedding is a learnable parameter that is added after the patch embedding.

```
# Definition
```

class PatchEmbed(nn.Module):

...

self.proj = nn.Conv2d(in_chans, embed_dim, kernel_size=patch_size, stride=patch_size)

...self.patch_embed = PatchEmbed(...)

self.pos_embed = nn.Parameter(torch.zeros(1, num_patches+1, embed_dim))...# Forward

x, T, W = self.patch_embed(x)

x = x + self.pos_embed where *Ex(p, t) = self.patch_embed(x)*, *e_pos(p,t) = self.pos_embed*.

Query-Key-Value computation.**The transformer consists of *L* encoding blocks. At each block *l*, a query/key/value vector is computed for each patch from the representation *z(l−1)* encoded by the preceding block:

$$\mathbf{q}_{(p,t)}^{(\ell,a)} = W_Q^{(\ell,a)} \mathrm{LN}\left(\mathbf{z}_{(p,t)}^{(\ell-1)}\right) \in \mathbb{R}^{D_h}$$

$$\mathbf{k}_{(p,t)}^{(\ell,a)} = W_K^{(\ell,a)} \mathrm{LN}\left(\mathbf{z}_{(p,t)}^{(\ell-1)}\right) \in \mathbb{R}^{D_h}$$

$$\mathbf{v}_{(p,t)}^{(\ell,a)} = W_V^{(\ell,a)} \mathrm{LN}\left(\mathbf{z}_{(p,t)}^{(\ell-1)}\right) \in \mathbb{R}^{D_h}$$

where LN is LayerNorm. In the official implementation, this qkv embedding can be simply implemented as a linear layer.

```
# Definition
```

class Attention(nn.Module):

...

self.qkv = nn.Linear(dim, dim * 3)

...

self.norm1 = norm_layer(dim)

self.attn = Attention(...)# Forward

x = self.attn(self.norm1(x)) where LN = *self.norm1* and *W_Q, W_K, and W_V* are implemented as a single nn.Linear layer *self.qkv*.

Self-attention computation. Self-attention weights are computed via dot-product. The self-attention weights for query patch *(p, t)* are given by:

$$\boldsymbol{\alpha}_{(p,t)}^{(\ell,a)} = \mathrm{SM}\left(\frac{\mathbf{q}_{(p,t)}^{(\ell,a)\top}}{\sqrt{D_h}} \cdot \left[\mathbf{k}_{(0,0)}^{(\ell,a)} \left\{\mathbf{k}_{(p',t')}^{(\ell,a)}\right\}_{\substack{p'=1,\ldots,N \\ t'=1,\ldots,F}}\right]\right) \tag{5}$$

where *SM* is *softmax*. In the official implementation, it is simply implemented as a batch matrix multiplication.

self.scale = qk_scale or head_dim ** -0.5...attn = (q @ k.transpose(-2, -1)) * self.scale

attn = attn.softmax(dim=-1) The typical method of computing self-attention weights results in quadratic complexity (O((H*W*T)²)) where H and W are the height and width of the feature maps, and T is the sequence length). The authors instead propose to reduce computations by computing over the spatial or temporal dimension separately (similar to separable convolutions). For example, if we compute temporal attention, *h* and *w* goes to batch dimension and matrix multiplication complexity reduced to *T*.

```
# CODE IS SIMPLIFIED FOR PRESENTATION
```

```
# EXAMPLE OF DIVIDED SPACE-TIME ATTENTIONfrom einops import rearrange## Temporal attention
```

xt = x[:,1:,:]

xt = rearrange(xt, 'b (h w t) m -> (b h w) t m',b=B,h=H,w=W,t=T)

res_temporal = self.temporal_attn(xt)

res_temporal = self.temporal_fc(res_temporal)

xt = x[:,1:,:] + res_temporal## Spatial

init_cls_token = x[:,0,:].unsqueeze(1)

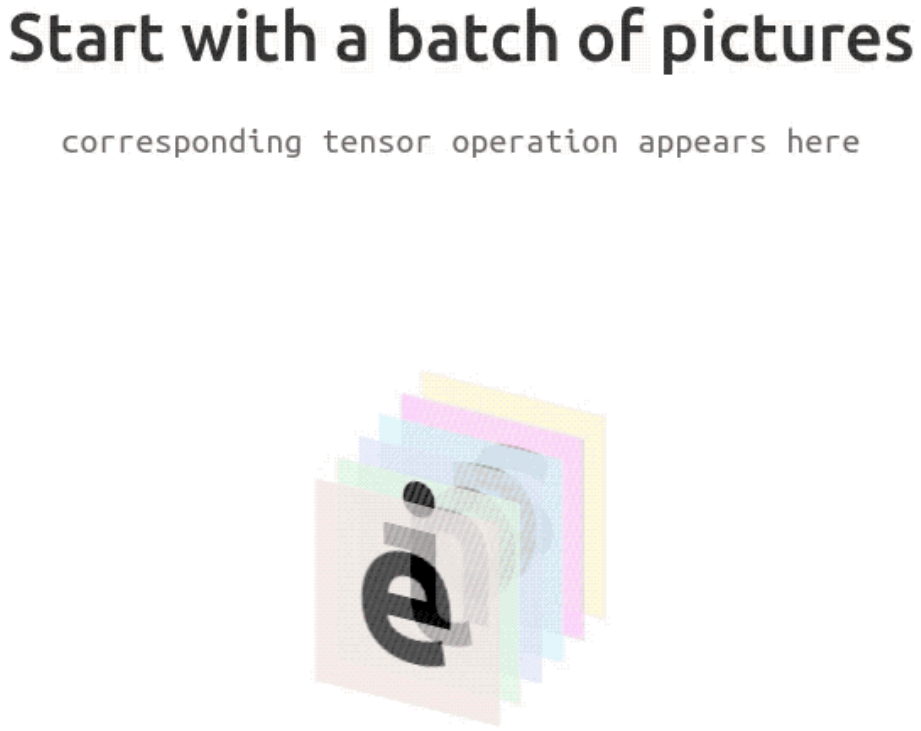
cls_token = init_cls_token.repeat(1, T, 1)

xs = xt

xs = rearrange(xs, 'b (h w t) m -> (b t) (h w) m',b=B,h=H,w=W,t=T)

xs = torch.cat((cls_token, xs), 1)

res_spatial = self.attn(xs) Above implementation uses einops, which supports flexible and powerful tensor operations for readable and reliable code.



How einops works, clipped from the source: official github

In the above implementation, `rearrange(xt, 'b (h w t) m -> (b h w) t m')` means that the input tensor is 3-dimensional *(b, hwt, m)* and it will be reshaped into *(bhw, t, m)*(To do that, it requires the actual values of *b, h, w* and *t*). That means, it moves spatial dimension to the batch dimension to process temporal attention. Likewise, `rearrange(xs, 'b (h w t) m -> (b t) (h w) m')` moves temporal dimension *t* to the batch dimension to process spatial attention.

Encoding. The encoding at block *l* is obtained by first computing the weighted sum of value vectors using self-attention coefficients from each attention head:

$$\mathbf{s}_{(p,t)}^{(\ell,a)} = \alpha_{(p,t),(0,0)}^{(\ell,a)} \mathbf{v}_{(0,0)}^{(\ell,a)} + \sum_{p'=1}^{N} \sum_{t'=1}^{F} \alpha_{(p,t),(p',t')}^{(\ell,a)} \mathbf{v}_{(p',t')}^{(\ell,a)}. \tag{7}$$

In the official implementation, (the class token handling part = the equation related to index (0,0), is omitted)

s = (attn @ v).transpose(1, 2).reshape(B, N, C) Then, the concatenation of these vectors from all heads is projected and passed through an MLP using residual connection:

$$\mathbf{z'}^{(\ell)}_{(p,t)} = W_O \begin{bmatrix} \mathbf{s}^{(\ell,1)}_{(p,t)} \\ \vdots \\ \mathbf{s}^{(\ell,\mathcal{A})}_{(p,t)} \end{bmatrix} + \mathbf{z}^{(\ell-1)}_{(p,t)}$$

$$\mathbf{z}^{(\ell)}_{(p,t)} = \mathrm{MLP}\left(\mathrm{LN}\left(\mathbf{z'}^{(\ell)}_{(p,t)}\right)\right) + \mathbf{z'}^{(\ell)}_{(p,t)}.$$

In the official implementation,

```
## Mlp
```

```
# shape of cls_token: b 1 m
```

```
# shape of x, res: b (h w t) mres_temporal = self.temporal_fc(res_temporal)
```

...

x = torch.cat((init_cls_token, x), 1) + torch.cat((cls_token, res), 1)

x = x + self.mlp(self.norm2(x)) where *W_O* = *self.temporal_fc*, MLP and LN corresponds to *self.mlp* and *self.norm2*.

Finally, the classification embedding is obtained from the final block for the classification token with LayerNorm. On top of it, 1-hidden-layer MLP is used to predict final video classes.

Various self-attention schemes.

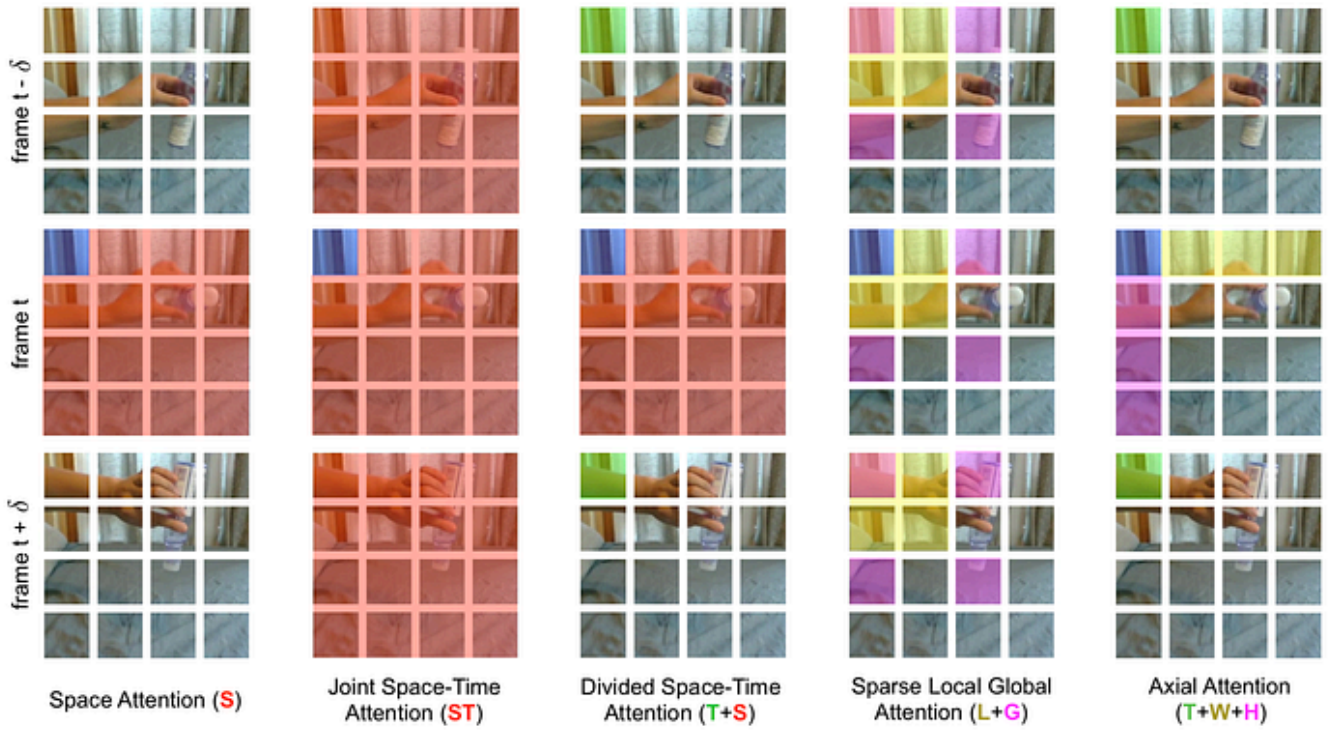


Figure 1. Visualization of the five space-time self-attention schemes studied in this work. Note that self-attention is computed for every single patch in the video clip, i.e., every patch serves as a query. We also note that although the attention pattern is shown for only two adjacent frames, it extends in the same fashion to all frames of the clip.

Five space-time attention schemes studied in this paper are visualized in Fig 1. In this visualization, each video clip is viewed as a sequence of frame-level patches with a size of 16 x 16 pixels. Patches are colored:

- Blue: query patch
- No color: not used for the self-attention computation of the blue patch
- Other (multiple) colors within a scheme denote attentions seprately applied along different dimension (e.g. space and time for (T + S))

Spatio-temporal attention is a straightforward way to extend self-attention to 3D, but it requires high computational cost. Thus, it should be replaced with another self-attention scheme to reduce the computational complexity. The authors investigate the following five schemes:

- Spatial Attenion (S): Query-key relationship is built only within each frame. Such a model neglects to capture temporal dependencies across frames.
- Joint Space-Time Attention (ST): Relationship is built across all location, all frames. However, it costs exhaustive computation.
- Divided Space-Time Attention (T+S): Temporal attention and spatial attention are separately applied one after the other.
- Sparse Local Global Attention (L+G): First computes the local attention by considering the neighboring F x H/2 x W/2 patches and then calculates a sparse global attention over the entire clip using a stride of 2 patches along the temporal and spatial dimension

- Axial Attention (T+W+H): The attention computation decomposes into three distinct steps; over time, width and height.

Datasets

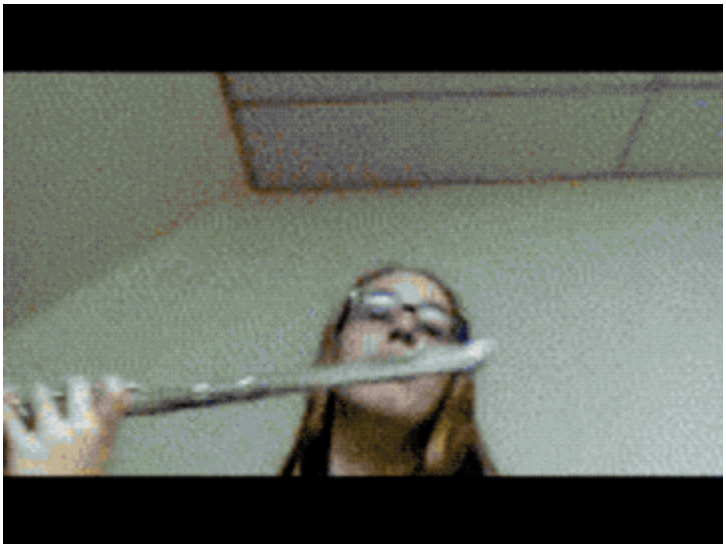
The authors test their method on four common benchmark datasets: Kinetics (400, 600 and 700), Something-Something V2, Diving-48 and HowTo100M.

Get Wonseok Lee's stories in your inbox

Kinetics (400, 600 and 700). According to the official homepage,

> A collection of large-scale, high-quality datasets of URL links of up to 650,000 video clips that cover 400/600/700 human action classes, depending on the dataset version. The videos include human-object interactions such as playing instruments, as well as human-human interactions such as shaking hands and hugging. Each action class has at least 400/600/700 video clips. Each clip is human annotated with a single action class and lasts around 10 seconds.

Here are some samples from the dataset,



Three Kinetics samples: Playing Flute, Feeding Birds and Baking Cookies

Image source: http://kinetics-explorer.com/#/

In a related paper, that uses the same dataset (Only Time Can Tell: Discovering Temporal Data for Temporal Modeling), the authors mentioned that a part of Kinetics dataset can be easily guessed from a single(or static) frame. It turns out to be an important characteristic of the dataset when the authors investigate the effect of different self-attention schemes.

Something-Something v2

> Something-Something is a large scale video initiative for teaching machines common sense of the physical world

This dataset contains 220,847 videos, each containing an action that can be described by a verb. The provided per-video label is typically as such: “Putting [something] onto [something]”. (link to SSv2 is not available currently)

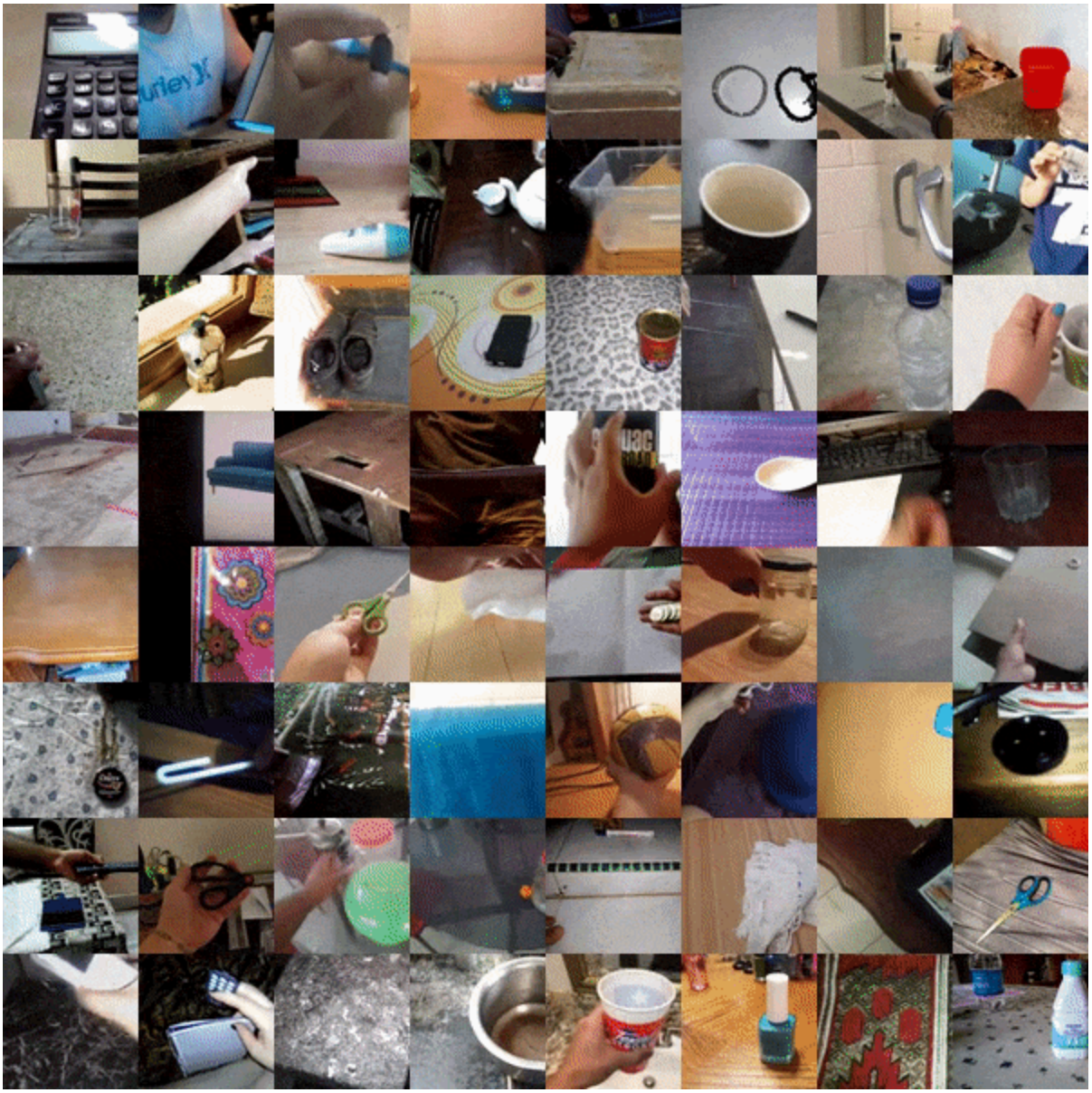


Image source: https://medium.com/twentybn/something-something-v2-release-9107b4a8ce99

The above two datasets are the main datasets used in most experiments, however, the model is also validated on the other datasets to show that the model can work in long video.

Diving-48

> Diving48 is a fine-grained video dataset of competitive diving, consisting of ~18k trimmed video clips of 48 unambiguous dive sequences (standardized by FINA). This proves to be a challenging task for modern action recognition systems as dives may differ in three stages (takeoff, flight, entry) and thus require modeling of long-term temporal dynamics.
>
> **Description**
>
> Each of the 48 dive sequences are defined by a combination of takeoff (dive groups), movements in flight (somersaults and/or twists), and entry (dive positions). The prefix tree below summarizes all the dive classes present in the dataset.

HowTo100M

> **HowTo100M** is a large-scale dataset of narrated videos with an emphasis on instructional videos where content creators teach complex tasks with an explicit intention of explaining the visual content on screen. HowTo100M features a total of:
>
> **- 136M video clips with captions** sourced from 1.2M Youtube videos (15 years of video)
>
> **- 23k activities** from domains such as cooking, hand crafting, personal care, gardening or fitness
>
> Each video is associated with a narration available as subtitles automatically downloaded from Youtube.

Description from the official homepage

Results

The default clip size is 8 x 224 x 224, frames sampled at a rate of 1/32. A patch size(P) of 16 x 16 is used by default. During the inference, 3 spatial crops(top-left, center, bottom-right) were used.

Analysis of self-attention schemes. Different self-attention schemes are compared in two datasets, Kinetics 400 and Something-Something v2. In this experiment, the proposed divided space-time attention showed the best result among the others.

Attention	Params	K400	SSv2
Space	85.9M	76.9	36.6
Joint Space-Time	85.9M	77.4	58.5
Divided Space-Time	121.4M	**78.0**	**59.5**
Sparse Local Global	121.4M	75.9	56.3
Axial	156.8M	73.5	56.2

Table 1. Video-level accuracy for different space-time attention schemes in TimeSformer. We evaluate the models on the validation sets of Kinetics-400 (K400), and Something-Something-V2 (SSv2). We observe that divided space-time attention achieves the best results on both datasets.

Recall from section 'various self-attention schemes', the space-only attention only builds an attention within each frame and neglects temporal dependencies across frames.

An interesting result is that space-only attention(S) performs well on Kinetics-400, but not on SSv2. The reason might be speculated form the related work that has found that the spatial cues are more important than temporal information in order to achieve strong accuracy in K400. On contrary, space-only attention(S) performs poorly on SSv2, since the dataset requires complex temporal reasoning.

Comparison to 3D CNNs. The authors performed an empirical study aimed at understanding the distinguishing properties of TimeSformer compared to 3D convolutional architectures, which have been the prominent approach to video understanding in recent years. Two 3D CNN models are compared with the transformer: 1) SlowFast (Feichtenhofer et al., 2019b), which was the state-of-the-art(at the time of presentation) in video classification, and 2) I3D (Carreira & Zisserman, 2017), which has been shown to benefit from image-based pretraining.

Model	Pretrain	K400 Training Time (hours)	K400 Acc.	Inference TFLOPs	Params
I3D 8x8 R50	ImageNet-1K	444	71.0	1.11	28.0M
I3D 8x8 R50	ImageNet-1K	1440	73.4	1.11	28.0M
SlowFast R50	ImageNet-1K	448	70.0	1.97	34.6M
SlowFast R50	ImageNet-1K	3840	75.6	1.97	34.6M
SlowFast R50	N/A	6336	76.4	1.97	34.6M
TimeSformer	ImageNet-1K	**416**	75.8	**0.59**	121.4M
TimeSformer	ImageNet-21K	**416**	**78.0**	**0.59**	121.4M

Table 2. Comparing TimeSformer to SlowFast and I3D. We observe that TimeSformer has lower inference cost despite having a larger number of parameters. Furthermore, the cost of training TimeSformer on video data is much lower compared to SlowFast and I3D, even when all models are pretrained on ImageNet-1K.

- Although transformer has a large number of parameters, its inference cost is lower than the 3D CNNs. It is because while 3D CNNs reduce their number of parameters by sharing the kernel, they still have to compute over both space and time. On the other hand, the transformer reduces the input into patches and performs efficient operations.
- Training time is also shorter. If the same budget is used, then the performance of I3D and SlowFast degrades

Pre-training and variants in resolution and clip length. Due to the large number of parameters, it is beneficial to use ImageNet pre-trained model. Two ImageNet-pretrainings are compared. Also, a model that uses high resolution image and a model that uses a longer clip are also compared.

Method	Pretraining	K400	SSv2
TimeSformer	ImageNet-1K	75.8	**59.5**
TimeSformer	ImageNet-21K	**78.0**	**59.5**
TimeSformer-HR	ImageNet-1K	77.8	62.2
TimeSformer-HR	ImageNet-21K	**79.7**	**62.5**
TimeSformer-L	ImageNet-1K	78.1	**62.4**
TimeSformer-L	ImageNet-21K	**80.7**	62.3

Table 3. Comparing the ImageNet-21K pretraining on Kinetics-400 (K400) and Something- Something-V2 (SSv2). On K400, ImageNet-21K pretraining leads consistently to a better performance compared to ImageNet-1K pre- training. On SSv2, ImageNet-1K and ImageNet-21K pretrainings lead to similar accuracy.

- The results are not on the table, but the authors mentioned that using an ImageNet pre-trained model is always better than training from scratch in every variant. For example, the model can be trained from scratch, but it results in much lower accuracy 64.8% on Kinetics-400.
- (1) **TimeSformer**, which is the default version of our model operating on 8 × 224 × 224 video clips, (2) **TimeSformer-HR**, a high spatial resolution variant that operates on 16 × 448 × 448 video clips, and lastly (3) **TimeSformer-L**, a long-range configuration of our model that operates on 96 × 224 × 224 video clips with frames sampled at a rate of 1/4.
- Using ImageNet-21K benefits in K400, but results in similar accuracy in SSv2. That might be reasonable since SSv2 requires more complex spatio-temporal reasoning, while K400 is biased more towards spatial scene information

The Impact of video-data scale. To understand the effects of video-data scale on performance, TimeSformer is trained on different subsets of K400 and SSv2: {25%, 50%, 75%, 100%} of the full datasets.

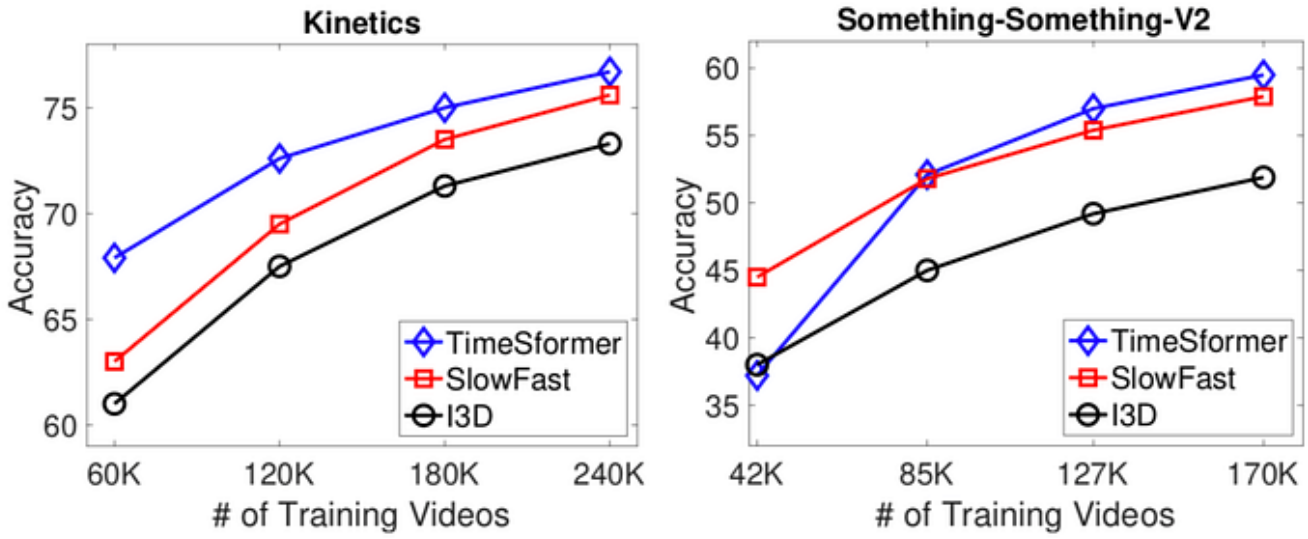


Figure4. Accuracy on Kinetics-400 (K400), and Something- Something-V2 (SSv2) as a function of the number of training videos. On K400, TimeSformer performs best in all cases. On SSv2, which requires more complex temporal reasoning, TimeSformer outperforms the other models only when using enough training videos. All models are pretrained on ImageNet-1K.

- In Kinetics, TimeSformer outperforms the other models regardless of the amount of training videos.
- In SSv2, TimeSformer requires more data to outperform the other models

The importance of positional embedding. To investigate the importance of learned spatiotemporal positional embeddings, the authors also conduct experiments with a few variants of TimeSformer.

Positional Embedding	K400	SSv2
None	75.4	45.8
Space-only	77.8	52.5
Space-Time	**78.0**	**59.5**

Table 4. Ablation on positional embeddings. The version of TimeSformer using space-time positional embeddings yields the highest accuracy on both Kinetics-400 and SSv2.

The result shows that different positional embedding schemes show different behavior in different dataset (consistently, as explained above).

Comparison to the state-of-the-art. In this section, the authors compared the method with the state-of-the-art methods, such as SlowFast, bLVNet, etc.

Method	Top-1	Top-5	TFLOPs
R(2+1)D (Tran et al., 2018)	72.0	90.0	17.5
bLVNet (Fan et al., 2019)	73.5	91.2	0.84
TSM (Lin et al., 2019)	74.7	N/A	N/A
S3D-G (Xie et al., 2018)	74.7	93.4	N/A
Oct-I3D+NL (Chen et al., 2019)	75.7	N/A	0.84
D3D (Stroud et al., 2020)	75.9	N/A	N/A
I3D+NL (Wang et al., 2018b)	77.7	93.3	10.8
ip-CSN-152 (Tran et al., 2019)	77.8	92.8	3.2
CorrNet (Wang et al., 2020a)	79.2	N/A	6.7
LGD-3D-101 (Qiu et al., 2019)	79.4	94.4	N/A
SlowFast (Feichtenhofer et al., 2019b)	79.8	93.9	7.0
X3D-XXL (Feichtenhofer, 2020)	80.4	94.6	5.8
TimeSformer	78.0	93.7	**0.59**
TimeSformer-HR	79.7	94.4	5.11
TimeSformer-L	**80.7**	**94.7**	7.14

Table 5. Video-level accuracy on Kinetics-400.

Method	SSv2	Diving-48**
SlowFast (Feichtenhofer et al., 2019b)	61.7	77.6
TSM (Lin et al., 2019)	63.4	N/A
STM (Jiang et al., 2019)	64.2	N/A
MSNet (Kwon et al., 2020)	64.7	N/A
TEA (Li et al., 2020b)	65.1	N/A
bLVNet (Fan et al., 2019)	**65.2**	N/A
TimeSformer	59.5	74.9
TimeSformer-HR	62.2	78.0
TimeSformer-L	62.4	**81.0**

Table 7. Video-level accuracy on Something-Something-V2 and Diving-48. **Due to an issue with Diving-48 labels used in previously published results, we only compare our method with a reproduced SlowFast 16 × 8 R101 model. All models are pre-trained on ImageNet-1K.

In Kinetics-400, TimeSformer-L outperforms the other models. Also, TimeSformer still has competitive performance compared to convolutional network based architectures such as R(2+1)D, TSM, and I3D.

On the other hand, in Something-Something V2, it didn't show better performance compared to the other models, even though they use high resolution and longer clip . My conjecture to this different result on two datasets is that the target of interest(or objects) are frequently moving in SSv2, and TimeSformer encodes the feature in temporal dimension only within its own spatial location in one step. So it may have difficulty when reasoning about different frame and different location objects.

There are other results in the paper, using HowTo100M dataset to show that the model is very effective in long-term task classification.

Qualitative Visualizations

Visualization of space-time attention

Figure 7. Visualization of space-time attention from the output token to the input space on Something-Something-V2. Our model learns to focus on the relevant parts in the video in order to perform spatiotemporal reasoning.

To visualize this, they used Attention Rollout scheme. Visual result suggests that TimeSformer learns to attend to the relevant regions in the video in order to perform complex spatiotemporal reasoning.

Feature visualization on SSv2, using t-SNE

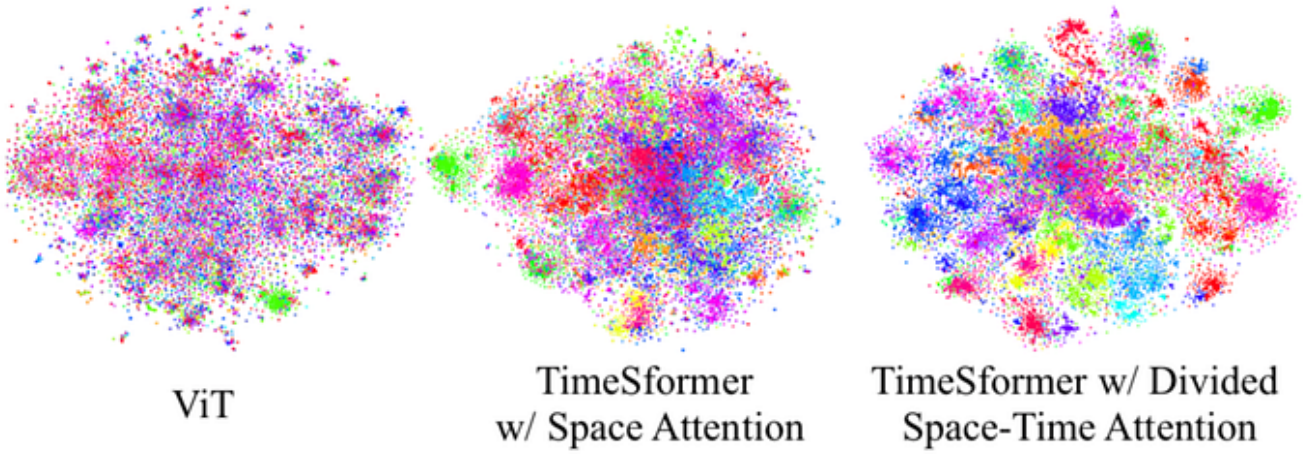


Figure 8. Feature visualization with t-SNE (van der Maaten & Hinton, 2008) on Something-Something-V2. Each video is visualized as a point. Videos belonging to the same action category have the same color. The TimeSformer with divided space-time attention learns semantically more separable features than the TimeSformer with space-only attention or ViT (Dosovitskiy et al., 2020).

Above visualization shows that TimeSformer with divided space-time attention learns semantically more separable features than the TimeSformer with space-only attention or ViT.

Conclusion

In this article, we discussed a paper introducing a novel convolution-free video classification model named TimeSformer and described the method and experimental results with accompanying code. It is conceptually simple, and achieves the state-of-the-art result on major action recognition benchmarks, has low training & inference cost, and can be applied to long videos.

My personal lessons from this paper are...

- The paper presents results on multiple datasets with very different characteristics (K400 and SSv2) which makes drawing a simple conclusion difficult.
- Training efficiency is very important in video recognition tasks. Using high resolution and longer clip almost always gives better result, which means enabling those usage is key to enhance the recognition performance.
- The new library, einops, really helps in simplifying and clarifying tensor operations in code, such as tensor.view() or tensor.transpose().