Benchmarking Data Efficiency and Computational Efficiency of Temporal Action Localization Models

Jan Warchocki* Teodor Oprescu* Yunhan Wang* Alexandru Dămăcuş Paul Misterka
Robert-Jan Bruintjes Attila Lengyel Ombretta Strafforello Jan van Gemert
Computer Vision Lab, Delft University of Technology
Delft, the Netherlands

Abstract

In temporal action localization, given an input video, the goal is to predict which actions it contains, where they begin, and where they end. Training and testing current state-of-the-art deep learning models requires access to large amounts of data and computational power. However, gathering such data is challenging and computational resources might be limited. This work explores and measures how current deep temporal action localization models perform in settings constrained by the amount of data or computational power. We measure data efficiency by training each model on a subset of the training set. We find that TemporalMaxer outperforms other models in data-limited settings. Furthermore, we recommend TriDet when training time is limited. To test the efficiency of the models during inference, we pass videos of different lengths through each model. We find that TemporalMaxer requires the least computational resources, likely due to its simple architecture.

1. Introduction

Temporal action localization (TAL) is concerned with automatically recognizing an action and its start and end in a video [29]. TAL has found potential use in domains such as video summarization [14] and public video surveillance [27, 29]. Various algorithms are proposed for TAL, and deep learning models such as such as TriDet [23], TemporalMaxer [24], and ActionFormer [33] outperform models based on hand-crafted features [29]. These deep learning models require large datasets to train on, such as THUMOS'14 [11] or ActivityNet [9]. However, curating, annotating and storing datasets of such scale is difficult, expenisve, and time-consuming [20, 29, 30]. To save data, in this work we explore data efficiency of deep learning-based TAL models.

*Equal contribution

In addition to data efficiency, we also evaluate compute efficiency. Compute efficiency is particularly relevant when the success of Transformers [26] in natural language processing (NLP) [12, 26], is employed in TAL [18, 33]. Transformers are known to be computationally expensive [13, 25]. To save computing resources, in this work, we explore how computationally efficient deep learning-based TAL methods are.

Our analysis of data- and compute-efficiency focuses on ActionFormer [33], STALE [19], TemporalMaxer [24], and TriDet [23], as they represent the current state-of-the-art in temporal action localization. The contributions of this paper are four-fold, as detailed below.

First, we test the data efficiency of the TAL models. Inspired by Ding *et al*. [5] and Henaff [10], we train each model multiple times on a percentage of the training set and report the average mean average precision (mAP). By applying this method on both THUMOS'14 [11] and ActivityNet [9] datasets, we find that the TemporalMaxer [24] performs the best in a data-limited setting.

Second, we evaluate the effect of score fusion [28,32,33] on data efficiency. Score fusion combines the outputs of an evaluated model with the outputs of an auxiliary model, often UntrimmetNet [28, 33]. We find that score fusion can significantly increase the performances of the models. We thus recommend that when choosing a model for a custom dataset, the options both with and without score fusion should be considered.

Third, we test the computational efficiency of each model during training. We measure training performance by analyzing the trade-off between training time and obtained average mAP. We find that the TriDet model [23] is the best choice in training time-limited settings, because it requires the least amount of training time but still obtains the best average mAP.

Fourth, we test the computational efficiency of each model during inference. We expand on the approach of measuring the computational complexity of the model by passing to it a video of a specific size [23, 24, 33]. We

evaluate each model on videos of increasing lengths and report the number of floating point operations, the memory consumed, and the inference time. We find that TemporalMaxer requires the least computational resources, while STALE [19] requires the most.

2. Related work

Action recognition. The survey by Xia and Zhan [29] identifies five different tasks in video understanding: untrimmed video classification, trimmed action recognition, temporal action proposals, temporal action localization, and dense-captioning events in videos. This work focuses on temporal action localization (TAL) for its potential uses in video summarization [14] and public surveillance [27]. In TAL, the goal is to predict which actions happen in a video stream, where they begin, and where they end. The deep learning models created for this problem can be divided into two categories [29]: two-stage and one-stage. Two-stage models [7, 16, 17] attempt to first locate the actions and then classify them. One-stage models [18, 23, 24, 33] locate and classify the actions at the same time. This work analyzes ActionFormer [33], STALE [19], TemporalMaxer [24], and TriDet [23], all of which are one-stage models.

Testing for data efficiency. This problem involves assessing how well a given model performs with limited training data available. A common approach is to use n-shot learning [20, 30], which involves training the model on only n samples per class. However, since a single class can be represented multiple times in a single video [9, 11], it is unclear whether n should refer to the number of videos the given class appears in or whether it is the total number of instances of the class. Furthermore, representing each class equally would be difficult as the number of instances of a class per video varies. An alternative approach involves training on a given percentage p of the samples from the training dataset [5, 10]. In this work, we use this approach.

Optimizing for data efficiency. As collecting and annotating datasets is expensive [29], related works have proposed few-shot TAL methods [20, 30]. These models use meta-learning and require all of the support videos to be input into the model at once. This makes their architecture incompatible with the architecture of current state-of-the-art models, which only expect a single video as input [23, 24, 33]. This work, therefore, analyzes the data efficiency of some of the current state-of-the-art models.

Testing for computational efficiency. The term 'computational efficiency' is often used to mean the number of floating point operations [8, 23–25, 33], the memory used [13, 25], or the training [15] or inference time [23, 24, 33]. In the task of temporal action localization, TriDet [23], TemporalMaxer [24], and ActionFormer [33] all report the number of floating point operations as the amount of multiply-accumulate (MAC) operations and the time it takes to forward a single video of a fixed length through the model. However, no experiments have been performed that would show how these models scale with an increase in video length. This is relevant, as models that scale linearly, will asymptotically outperform models that scale *e.g.* quadratically. Hence, even if a quadratic model outperforms a linear model on short videos, it will perform worse on longer videos. Thus, in this work, the inference performance of each of the tested models is measured on videos of increasing lengths.

Furthermore, motivated by [15], this work reports the training time and the achieved mean average precision of each of the TAL models. This is done to better understand the suitability of each model for settings where the training time is limited.

Optimizing for computational efficiency. Both TriDet [23] and TemporalMaxer [24] aim to lower the required computational cost of ActionFormer [33]. In TriDet, this is achieved by replacing the multi-head self-attention module with an efficient Scalable-Granularity Perception layer [23]. TemporalMaxer, on the other hand, replaces the entire transformer module with a max-pooling block [24]. This work compares the computational efficiencies of ActionFormer, STALE [19], TemporalMaxer, and TriDet.

3. Models

ActionFormer. ActionFormer [33] was one of the first models that showed a successful use of Transformers [26] in temporal action localization. The model uses an encoder-decoder architecture with a Transformer encoder and a convolutional decoder. At the time of its proposal, the model reached state-of-the-art performance on the THUMOS'14 dataset obtaining an average mAP of 66.8%. The model also showed promising results on both the ActivityNet [9] and EPIC-Kitchens 100 [4] datasets. We also selected this model for evaluation, as the architectures of newer models, TriDet [23] and TemporalMaxer [24], are inspired by the architecture of the ActionFormer.

STALE. *Zero-Shot Temporal Action Detection via Vision-Language Prompting* (STALE) [19] is the most recent and state-of-the-art method in zero-shot temporal action localization. Inspired by CLIP [21], STALE uses a temporal vision transformer [6] to encode videos into video embeddings and a text transformer [26] to encode class prompts into text embeddings. STALE attempts to learn an inter-relationship of vision-language via cross attention [26]. The model achieved average mAP of 52.9% and 36.4% on the THUMOS'14 and ActivityNet datasets respectively, outperforming similar models. We selected this model for evaluation, to compare it against methods that were not designed for a zero-shot learning scenario.

TemporalMaxer. The TemporalMaxer [24] model was constructed to require a low computational cost without

sacrificing localization performance. Instead of employing a computationally-heavy backbone, such as a Transformer [26, 33], the model uses a basic, parameter-free max pooling block on top of a pre-trained 3D CNN. This model currently represents the state-of-the-art on the MultiTHUMOS dataset [31] obtaining an average mAP of 29.9%. Importantly, the model also has a lower computational complexity compared to other models. On a video of a length of around 5 minutes from the THUMOS'14 dataset, the inference time of the TemporalMaxer was observed to be 3x shorter than that of the ActionFormer.

TriDet. The TriDet model [23] bases its architecture on the ActionFormer. Instead of using a multi-head self-attention mechanism, the model replaces it with an efficient Scalable-Granularity Perception (SGP) layer. The resulting model improves on the performance of the ActionFormer, obtaining an average mAP of 69.3% on the THUMOS'14 dataset. Furthermore, the TriDet model represents the current state-of-the-art for the EPIC-Kitchens 100 dataset. Finally, the model was also shown to require less time and fewer floating point operations than the ActionFormer when performing inference on a 5 minute video from the THUMOS'14 dataset.

4. Evaluation setup

4.1. Data efficiency

Evaluation metrics. Following common practice [1, 23, 24, 29, 33], the models were evaluated by reporting the achieved mean average precision (mAP) on different tIoU thresholds. Intersection over union (tIoU) is a 1-dimensional temporal Jaccard similarity metric and is thus computed as the ratio of the intersection of the predicted and actual duration of an action to their union. Given a tIoU threshold μ and a class c, correct predictions are those, whose tIoU $\geq \mu$ and the predicted class is the class c. Precision is then the ratio of the number of correct predictions to the total number of made predictions for the class c. As there can be multiple videos for each class c, average precision is the average of the precisions obtained in each of those videos. Finally, mean average precision is the average AP over all of the classes c. Thus, in general, given a fixed tIoU threshold μ, the higher the mAP, the better the model performs.

Testing procedure. In this setup, it is assumed that a dataset $\mathcal{D}$ has a predefined split into a training set $\mathcal{D}_{\text{train}}$ and a testing set $\mathcal{D}_{\text{test}}$. Following works by Ding *et al.* [5] and Henaff [10], a percentage p of the training set $\mathcal{D}_{\text{train}}$ was randomly and uniformly sampled to create a subset $\mathcal{D}_{\text{s}}$. The models were then trained on the set $\mathcal{D}_{\text{s}}$ and evaluated on the set $\mathcal{D}_{\text{test}}$. During the evaluation, mean average precision was calculated at different tIoU thresholds. The sampling, training, and testing procedure was repeated 5 times [2, 5] with different random splits. The mAP for each threshold was then averaged and the standard deviation was reported. The entire procedure was repeated for multiple percentages p. Algorithm 1 describes the exact testing procedure in the form of pseudocode.

In the pseudocode, the function sample randomly samples videos from the training set, such that:

$$|\mathcal{D}_{\text{s}}| = \text{round}\left(|\mathcal{D}_{\text{train}}| \cdot \frac{p}{100\%}\right) \tag{1}$$

with round rounding the value to the nearest integer. Additionally, the function sample needs to ensure that each action class is represented at least once in the resulting set $\mathcal{D}_{\text{s}}$. In practice, this was realized by repeatedly sampling from the set $\mathcal{D}_{\text{train}}$ until a split, where all classes are represented, was found. The function $\mathrm{calculate\text{-}mAP}$ evaluates the model, that is, it calculates the mean average precision at different tIoU thresholds the model achieved on the test set $\mathcal{D}_{\text{test}}$.

Algorithm 1 Data efficiency testing procedure

$\mathcal{D}_{\text{train}} = \{(\mathbf{X_i}, \hat{\mathbf{Y}}_\mathbf{i})\}_{i=1}^{N}$
$\mathcal{D}_{\text{test}} = \{(\mathbf{X_i}, \hat{\mathbf{Y}}_\mathbf{i})\}_{i=1}^{M}$
for $p = 10\%, \ldots, 100\%$ **do**
  mAPs $\leftarrow$ empty list
  for $i = 1, \ldots, 5$ **do**
    $\mathcal{D}_{\text{s}} \leftarrow \mathrm{sample}(\mathcal{D}_{\text{train}}, p)$
    Train on $\mathcal{D}_{\text{s}}$
    mAP $\leftarrow \mathrm{calculate\text{-}mAP}(\mathcal{D}_{\text{test}})$
    mAPs.append(mAP)
  Report avg(mAPs) and std(mAPs)

Algorithm 1. The data efficiency testing procedure. Assuming $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{test}}$ are given, $\mathcal{D}_{\text{train}}$ is repeatedly subsampled with percentage p to create the set $\mathcal{D}_{\text{s}}$. The model is then trained on $\mathcal{D}_{\text{s}}$ and evaluated on $\mathcal{D}_{\text{test}}$. The procedure is repeated 5 times for each percentage p, at each time reporting the averages of the mAPs and their standard deviation.

To understand the results between different datasets, for each percentage p the expected number of instances per class is reported. This will help in investigating how many instances per class each model requires. Given a dataset $\mathcal{D}_{\text{train}}$ containing N samples, having M action instances in total, and C action classes, the expected number of instances per class for each percentage p is calculated as:

$$\#/\text{class} = \frac{p}{100\%} \cdot \frac{N}{C} \cdot \frac{M}{N} = \frac{p}{100\%} \cdot \frac{M}{C} \tag{2}$$

It should be noted that the value computed in Equation (2) is an estimate. The exact values would depend on the splits $\mathcal{D}_{\text{s}}$ used in the testing procedure. Nonetheless, this approximation was found to be useful in practice when comparing the models on different datasets.

Score fusion. Score fusion is a commonly used technique in TAL [19, 23, 33] to improve the performance of a model. Although the exact implementations vary between models, the general rule is that the final predictions made by a model are combined with the output of UntrimmedNet [19, 23, 28, 33]. UntrimmedNet [28] is a weakly-supervised action recognition model which only predicts video-level classes without temporal localization. It should be noted that UntrimmedNet is trained on the full ActivityNet and THUMOS datasets, respectively, while in practice limited training data would also apply to UntrimmedNet. Thus, in this work, the setups that use score fusion by default are also evaluated without score fusion.

4.2. Computational efficiency

4.2.1 Training performance

Inspired by Li *et al*. [15], the training time of each of the models is reported alongside the average mAP achieved on the test set. The training time is measured without initialization, that is, only the time spent in the training loop is measured. In this way, only the model performance is measured, without the time taken by external methods such as PyTorch data loaders. The training and testing procedure is repeated 5 times using different random seeds, each time measuring the time spent and the average mAP achieved.

4.2.2 Inference performance

Evaluation metrics. Following the works on TriDet [23], TemporalMaxer [24], and ActionFormer [33] the models were evaluated by reporting the total number of multiply-accumulate (MAC) floating point operations, memory consumed, and the inference time when fed an input video. To count the number of multiply-accumulate operations, the `fvcore` library [22] was used. As Transformer-based methods are known to require large amounts of memory [13, 25], we additionally report the total GPU memory (VRAM) footprint of each model, which is measured using the `max_memory_allocated` method from PyTorch.

Testing procedure. The models were evaluated on randomly generated tensors, whose shapes correspond to videos of differing lengths. To guarantee the independence of results, the experiments for inference time, memory consumption, and number of MACs were run independently. Before each inference time measurement, the random tensor was passed through the model once as a warm-up procedure. Without this procedure, it was observed for the ActionFormer model that the inference time would be constant for all lengths of the input tensor. This was most likely caused by memory allocations happening as the tensor was being passed through the model. As such, the warm-up procedure was applied to all models to ensure a fair evaluation for all input sizes. Furthermore, the experiments for inference time were repeated 5 times [23] with different random tensors.

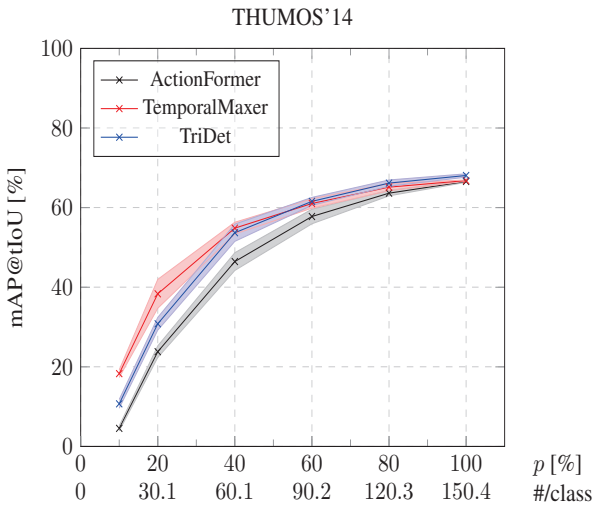
Additional setup was also required by the ActionFormer model. Given a dataset $\mathcal{D}$, the ActionFormer model is parameterized by a value `max_seq_len` indicating the maximum length of a video in $\mathcal{D}$ expressed in the number of features [33]. During inference, all videos are padded with zeros to the `max_seq_len` length, which results in the same amount of computation done regardless of video length. To alleviate this issue, the value `max_seq_len` was configured to the lowest allowable value, which would be found through an inspection of the code. It should be noted that the value `max_seq_len` is only used during training and changing it during inference does not influence the output of the model, which was verified with one of the authors of the ActionFormer.

5. Experiments

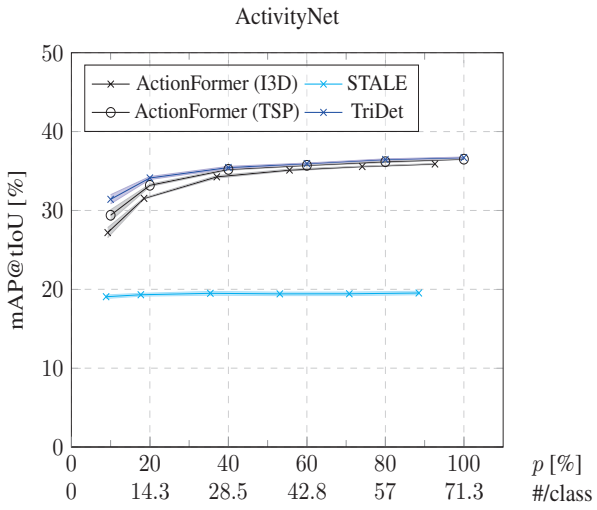
Datasets. The models were evaluated on two datasets, commonly used to assess temporal action localization algorithms [19, 23, 33]: THUMOS'14 [11] and ActivityNet [9]. THUMOS'14 contains 413 untrimmed videos and 20 action classes. This dataset is further split into a validation set, containing 213 videos and a test set containing 200 videos. In total, the validation set contains 3,007 action instances. We follow the configuration from the authors of the tested models and hence train the models on the validation set and test on the test set [19, 23, 24, 33]. ActivityNet contains around 20,000 videos with 200 action classes. The dataset is further split into a training set (10,024 videos), a validation set (4,926 videos), and a test set (5,044 videos). Using the approach from [19, 23, 33], the models are trained on the training set and evaluated on the validation set. As some of the videos from the ActivityNet dataset have become unavailable over time, it should be noted that the exact size of the dataset varies when using different models or features.

Features. We take into consideration all features that were made available by the authors of a model for the given dataset. Hence, on the THUMOS'14 dataset, ActionFormer [33], TemporalMaxer [24], and TriDet [23] are all evaluated using the Inflated 3D (I3D) features [3]. On the ActivityNet dataset, the ActionFormer was evaluated using both I3D and TSP [1] features. STALE [19] was tested with the CLIP [21] features. The STALE model was not evaluated using I3D features due to limited compute availability. Finally, the performance of the TriDet model on the ActivityNet dataset was measured using the TSP features.

Experimental setup. All of the models were trained and tested using a single NVIDIA Tesla V100S 32GB located on an HPC cluster. All of the training and testing



(a) Performance of the compared models on the THUMOS'14 dataset [11] in terms of average mAP@tIoU[0.3:0.1:0.7]. The TemporalMaxer model [24] performs the best with little training data available, likely due to a simpler architecture. The TriDet model outperforms TemporalMaxer when the average number of instances per class is > 100.



(b) Performance of the compared models on ActivityNet [9] in terms of average mAP@tIoU[0.5:0.05:0.95]. The x-axis is shifted to the left for some setups due to fewer training samples being available. Most importantly, the ActionFormer and TriDet models can be seen to outperform the STALE model on all tested values of p.

Figure 1: Reported average mAP@tIoU for the tested models on the THUMOS'14 [11] and ActivityNet [9] datasets. For each model, only the average mAP is shown. The width of each line corresponds to two standard deviations obtained by repeating the procedure 5 times for each p. Additionally, the expected average number of instances per class (#/class) is reported as a secondary x-axis. We find that all of the models reach their near-best performance with less than or around 100 action instances per class.

hyperparameters were left unchanged for the models unless otherwise stated in the subsequent sections. During data efficiency experiments, we therefore also use score fusion implemented by the ActionFormer, STALE, and TriDet models on the ActivityNet dataset [19, 23, 33]. We reflect on the impact of the score fusion on the performance of the models in Section 5.1.1.

5.1. Data efficiency

Results on THUMOS'14. The results on the THUMOS'14 dataset can be seen in Figure 1a. Firstly, we note that at $p = 100\%$, the average performance for each of the models is slightly lower than in the original works. We find an average mAP of 66.57 ± 0.22 [%] compared to 66.8% for the ActionFormer, 66.79 ± 0.16 [%] contrary to 67.7% for TemporalMaxer, and 68.07 ± 0.42 [%] instead of 69.3% for TriDet. As noticed by [33], however, different hardware setups may lead to different results, which might explain the differences observed in this work. Furthermore, we see that all models follow a similar learning curve. This is most likely caused by the fact that the models share a similar architecture, inspired by the architecture of the ActionFormer [23, 24, 33]. Moreover, as can be observed, at the low percentages p, the TemporalMaxer [24] model performs the best. This can be explained by the simpler architecture employed by the model, which would require less training data than the other models. We also note that for all models, the incline in performance noticeably slows down above $p = 60\%$, which corresponds to around 90-100 action instances per class. We can thus see that each model saturates at around 100 action instances per class and does not gain much from additional data. We also see that the TriDet model begins to outperform the TemporalMaxer around the same mark.

Results on ActivityNet. As can be seen in Figure 1b, both the ActionFormer and the TriDet models outperform the STALE model on all tested percentages p. Furthermore, we observe that ActionFormer and TriDet saturate around the 40-60% mark and do not gain from additional training data. This corresponds to around 30-40 action instances per class. We also notice that the STALE model does not visibly gain from an increase in training data. The model achieves an average mAP of 19.06 ± 0.22 [%] at $p = 10\%$ compared to 19.53 ± 0.22 [%] at $p = 100\%$. This flat learning curve is caused by the score enhancement, as is shown experimentally in Section 5.1.1.

Discussion. From Figure 1a, we can observe that the TemporalMaxer should likely be chosen in settings where

the amount of data is limited. The simple architecture of that model allows it to show the best data efficiency out of the tested models. Figure 1b suggests that the ActionFormer or TriDet models should be chosen in favor of STALE. Based on the combined results in Figure 1, it is difficult to put an exact bound on the number of action instances per class required by the models. On both of the datasets, however, we can observe that the models reach their near-best performance with less than or around 100 action instances per class. This suggests making datasets larger will not further improve the performance of the tested models.

5.1.1 Score fusion

In the default configuration, the score fusion techniques are used by the ActionFormer [33], STALE [19], and TriDet [23] on the ActivityNet dataset [9]. We repeat the data experiments without score fusion for these models on the ActivityNet dataset and report the results. We use the default features for these models for these experiments, hence, ActionFormer and TriDet use the TSP features [1], and STALE uses CLIP [21]. The results can be seen in Figure 2. Score fusion improves the performance of the models for all tested values of p. The largest impact can be seen at low percentages p in the small data regime.

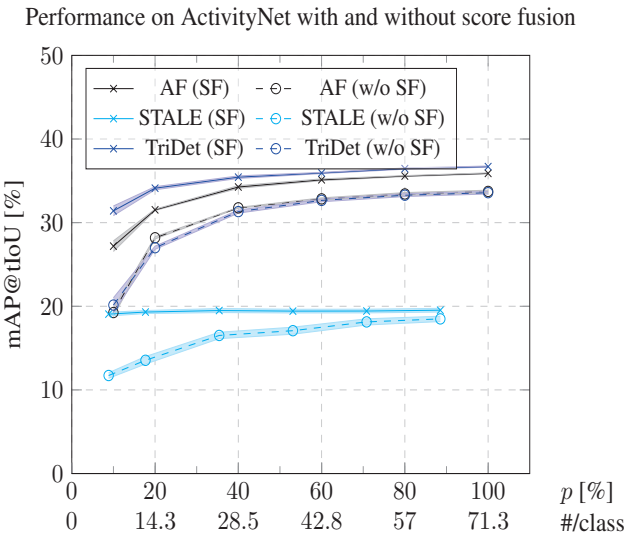


Figure 2: Performance of ActionFormer (AF), STALE, and TriDet with score fusion (SF) and without score fusion (w/o SF). As we can observe, the performance of the models drops when score fusion is not used.

Discussion. Unsurprisingly, score fusion has a significant influence on the performances of the models. It should be noted that score fusion is based on the assumption that UntrimmedNet class predictions are readily available, which in practice may not be the case. One should therefore be aware that performance on ActivityNet or Thumos does not always directly translate to true performance on a custom dataset, which may be lower. Alternatively, employing score fusion on custom data requires additional compute for retrieving the UntrimmedNet predictions. We argue that when choosing a model on a custom dataset, it is important to decide on the applicability of score fusion and evaluate the model both with and without score fusion.

5.2. Computational efficiency

Concurrent jobs on the HPC cluster. By default, the GPU nodes are shared between users in our compute cluster. This setup could lead to a dependence of the training or inference time on the other jobs running on the cluster. To alleviate this issue, the training and inference time experiments were performed five times sequentially, such that the experiment jobs did not overlap. Therefore, the results for training and inference times are averaged over a total of 25 runs. We measure the remaining variance in training time, assuming that a low variance means that there are no important unmeasured confounding factors stemming from the concurrent use of the cluster.

5.2.1 Training efficiency

Results. We present the results in Table 1. On THUMOS'14, TriDet achieves the best performance while requiring the least amount of training time on average. Interestingly, we find that the training time of the TemporalMaxer varies greatly between runs: from 1216.56 to 6829.95 seconds. This variance might come from the early stopping criterion employed in the training script of the model. Nonetheless, even in its fastest training run, TemporalMaxer is still the slowest of the tested models. On ActivityNet, ActionFormer and TriDet train for around five times as long as STALE, but also achieve much better performance. Finally, we note that the variance in training times was low for all models, except for the TemporalMaxer, thus the concurrent jobs on the cluster likely did not interfere with the experiment jobs.

Discussion. From the results obtained in Table 1 we find that the TriDet model should be chosen in settings where the training time is limited. This is due to the fact that the model was found to not only train for the least amount of time on THUMOS'14 but also achieve the best performance on both datasets. If the choice of the model is between ActionFormer and STALE, we find that the latter could be used in limited training time settings. Choosing STALE over the ActionFormer would, however, likely come with a decrease in TAL performance.

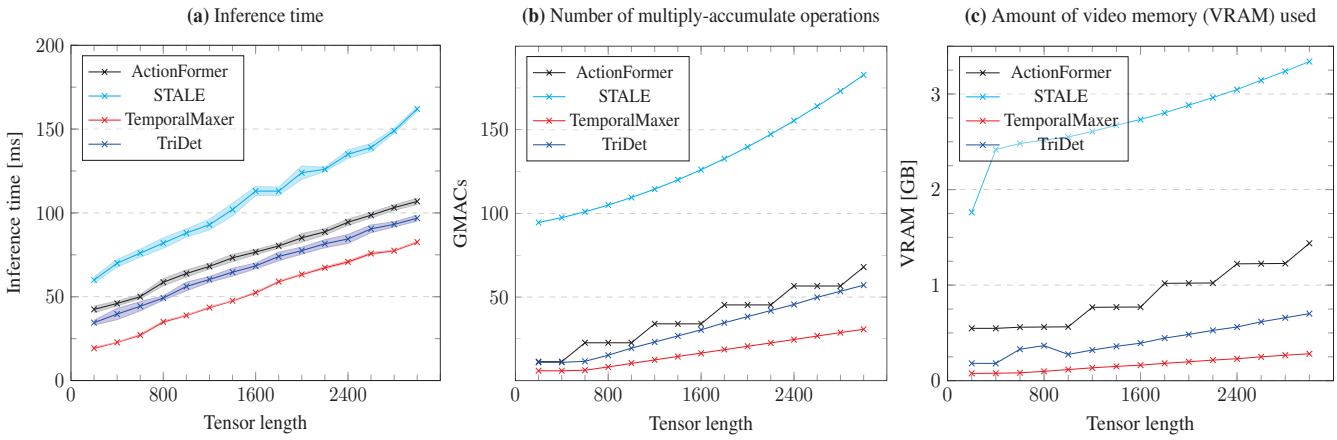


Figure 3: Inference time, number of floating point operations, and memory consumption for ActionFormer [33], TemporalMaxer [33], and STALE [19]. Most notably, we find that the TemporalMaxer model requires the least computational resources during inference, while STALE requires the most.

Table 1: Training performance of the compared models on the ActivityNet [9] and THUMOS'14 [11] datasets. Both average training time and obtained average mAP are reported. On THUMOS'14, TriDet is the fastest and performs the best. On ActivityNet, the ActionFormer and TriDet models take longer to train than STALE but also achieve better performance.

Model	Time [s]	Avg. mAP [%]
THUMOS'14		
AF	887 ± 54	65.89 ± 0.09
TemporalMaxer	2957 ± 1660	66.96 ± 0.37
TriDet	646 ± 26	68.07 ± 0.42
ActivityNet		
ActionFormer (I3D)	1945 ± 61	35.9 ± 0.14
ActionFormer (TSP)	1932 ± 232	36.4 ± 0.05
STALE	401 ± 6	19.37 ± 0.16
TriDet	2236 ± 224	36.57 ± 0.18

5.2.2 Inference efficiency

Additional experimental setup. For the ActionFormer [33], TemporalMaxer [24], and TriDet [23] models, we obtained inference efficiency results by creating random tensors corresponding to I3D features [3] extracted from videos from the THUMOS'14 dataset [11]. We obtained results for the STALE model by creating random tensors corresponding to CLIP features [21] on the ActivityNet dataset [9]. The lengths of the tensors vary from 200 to 3000 in 200 increments. This range is dictated by the ActionFormer model, where the lowest allowable value of `max_seq_len` is 576, so videos of lengths longer than 3456 cannot be passed through the model without further changes to the configuration.

Results. As can be seen in Figure 3, the TemporalMaxer model consistently achieves the lowest inference time, number of floating point operations, and memory consumption. This is because of the simple architecture of the model, which contains fewer parameters than other models [24]. Conversely, we find that the STALE model is the most computationally expensive in all three tested metrics and on all tested lengths of the input video. Furthermore, we observe that the number of floating point operations and the memory consumption increase in steps for the ActionFormer model. This is because the model architecture requires padding input videos to multiples of 576. Nonetheless, the model scales linearly with respect to the input size. This thus matches the claims of the original work [33]. We see that TriDet and TemporalMaxer both also scale linearly with respect to the input size. As can be seen in Figure 3b, the computational complexity of the STALE model does not increase linearly. A similar pattern is observable for memory consumption of STALE in Figure 3c. Interestingly, we find a linear pattern in the inference time of STALE in Figure 3a.

Discussion. In case of limited compute resources, TemporalMaxer should be chosen. TemporalMaxer requires the least amount of computational power on all tested video lengths. STALE should not be chosen in such settings, not only due to higher computational complexity but also because it scales non-linearly with respect to input video length. Hence, even if a configuration would be found that causes STALE to be more efficient on short videos, asymptotically it will still be worse than any other linear model.

6. Conclusion

In this work, we ask how well state-of-the-art temporal action localization models perform in settings limited by the amount of training data or computational resources available. We find that in a data deficient setting the TemporalMaxer model [24] works the best, likely due to its simple architecture, which consists of fewer parameters compared to other models and does not use a Transformer backbone. Additionally, we find that performance barely improves when adding data beyond 100 action instances per class. This suggests making datasets larger will not further improve the performance of the tested models. The use of score fusion was shown to improve the performances of the models, hence when training a model on a custom dataset, options with and without score fusion should be considered. Furthermore, we test computational efficiency during training and inference. We find that TriDet [23] offers the lowest training time as well as the best performance. Additionally, we find that TemporalMaxer requires the least computational resources at inference time, again likely due to its simple architecture without a Transformer backbone.

Limitations. It should be noted that the method for measuring data efficiency is limited as ActionFormer and TriDet are the only models that were evaluated on both datasets. Furthermore, the procedures for testing training and inference efficiency have limitations. The models have only been so far evaluated on the THUMOS'14 and ActivityNet datasets. The results on different datasets could lead to different conclusions. Furthermore, the timing experiments have been performed on a shared HPC cluster. It was however observed that the variance in training and inference times was small, which indicates that the concurrent jobs did not interfere with the experimental jobs.

Future work. This work provides insights that will help in developing future data or computationally efficient TAL models. Based on the results of ActionFormer [33] and STALE [19], we see that self-attention should not be the mechanism of choice if the training data or computational resources are limited. We find that replacing such modules with custom layers, such as SGP [23] or replacing transformer modules with max pooling [24] improves the efficiency of the model. Finally, we note that future work in evaluating current models in terms of data or computational efficiency is possible. More models could be evaluated or the models could be evaluated on more datasets.

Acknowledgements. This project is (partly) financed by the Dutch Research Council (NWO) (project VI.Vidi.192.100).

References

[1] Humam Alwassel, Silvio Giancola, and Bernard Ghanem. Tsp: Temporally-sensitive pretraining of video encoders for localization tasks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3173–3183, 2021. 3, 4, 6

[2] David Berthelot, Nicholas Carlini, Ian Goodfellow, Nicolas Papernot, Avital Oliver, and Colin A Raffel. Mixmatch: A holistic approach to semi-supervised learning. *Advances in neural information processing systems*, 32, 2019. 3

[3] Joao Carreira and Andrew Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6299–6308, 2017. 4, 7

[4] Dima Damen, Hazel Doughty, Giovanni Maria Farinella, Antonino Furnari, Evangelos Kazakos, Jian Ma, Davide Moltisanti, Jonathan Munro, Toby Perrett, Will Price, et al. Rescaling egocentric vision: Collection, pipeline and challenges for epic-kitchens-100. *International Journal of Computer Vision*, pages 1–23, 2022. 2

[5] Xinpeng Ding, Nannan Wang, Xinbo Gao, Jie Li, Xiaoyu Wang, and Tongliang Liu. Kfc: An efficient framework for semi-supervised temporal action localization. *IEEE Transactions on Image Processing*, 30:6869–6878, 2021. 1, 2, 3

[6] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020. 2

[7] Guoqiang Gong, Liangfeng Zheng, and Yadong Mu. Scale matters: Temporal scale aggregation network for precise action localization in untrimmed videos. In *2020 IEEE International Conference on Multimedia and Expo (ICME)*, pages 1–6. IEEE, 2020. 2

[8] Kai Han, An Xiao, Enhua Wu, Jianyuan Guo, Chunjing Xu, and Yunhe Wang. Transformer in transformer. *Advances in neural information processing systems*, 34:15908–15919, 2021. 2

[9] Fabian Caba Heilbron, Victor Escorcia, Bernard Ghanem, and Juan Carlos Niebles. Activitynet: A large-scale video benchmark for human activity understanding. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 961–970, 2015. 1, 2, 4, 5, 6, 7

[10] Olivier Henaff. Data-efficient image recognition with contrastive predictive coding. In *International conference on machine learning*, pages 4182–4192. PMLR, 2020. 1, 2, 3

[11] Haroon Idrees, Amir R Zamir, Yu-Gang Jiang, Alex Gorban, Ivan Laptev, Rahul Sukthankar, and Mubarak Shah. The thumos challenge on action recognition for videos "in the wild". *Computer Vision and Image Understanding*, 155:1–23, 2017. 1, 2, 4, 5, 7

[12] Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of naacL-HLT*, volume 1, page 2, 2019. 1

[13] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*, 2020. 1, 2, 4

[14] Yong Jae Lee, Joydeep Ghosh, and Kristen Grauman. Discovering important people and objects for egocentric video

summarization. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1346–1353, 2012. 1, 2

[15] Zhuohan Li, Eric Wallace, Sheng Shen, Kevin Lin, Kurt Keutzer, Dan Klein, and Joey Gonzalez. Train big, then compress: Rethinking model size for efficient training and inference of transformers. In *International Conference on machine learning*, pages 5958–5968. PMLR, 2020. 2, 4

[16] Tianwei Lin, Xiao Liu, Xin Li, Errui Ding, and Shilei Wen. Bmn: Boundary-matching network for temporal action proposal generation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3889–3898, 2019. 2

[17] Tianwei Lin, Xu Zhao, Haisheng Su, Chongjing Wang, and Ming Yang. Bsn: Boundary sensitive network for temporal action proposal generation. In *European Conference on Computer Vision*, pages 3–19, 2018. 2

[18] Xiaolong Liu, Qimeng Wang, Yao Hu, Xu Tang, Shiwei Zhang, Song Bai, and Xiang Bai. End-to-end temporal action detection with transformer. *IEEE Transactions on Image Processing*, 31:5427–5441, 2022. 1, 2

[19] Sauradip Nag, Xiatian Zhu, Yi-Zhe Song, and Tao Xiang. Zero-shot temporal action detection via vision-language prompting. In *European Conference on Computer Vision*, pages 681–697. Springer, 2022. 1, 2, 4, 5, 6, 7, 8

[20] Sauradip Nag, Xiatian Zhu, and Tao Xiang. Few-shot temporal action localization with query adaptive transformer. *arXiv preprint arXiv:2110.10552*, 2021. 1, 2

[21] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021. 2, 4, 6, 7

[22] Facebook Research. fvcore. https://github.com/facebookresearch/fvcore, 2023. (Accessed on 04/06/2023). 4

[23] Dingfeng Shi, Yujie Zhong, Qiong Cao, Lin Ma, Jia Li, and Dacheng Tao. Tridet: Temporal action detection with relative boundary modeling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 18857–18866, 2023. 1, 2, 3, 4, 5, 6, 7, 8

[24] Tuan N. Tang, Kwonyoung Kim, and Kwanghoon Sohn. Temporalmaxer: Maximize temporal context with only max pooling for temporal action localization. *arXiv preprint arXiv:2303.09055*, 2023. 1, 2, 3, 4, 5, 7, 8

[25] Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. Efficient transformers: A survey. *ACM Comput. Surv.*, 55(6):Article 109, 2022. 1, 2, 4

[26] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017. 1, 2, 3

[27] Sarvesh Vishwakarma and Anupam Agrawal. A survey on activity recognition and behavior understanding in video surveillance. *The Visual Computer*, 29:983–1009, 2013. 1, 2

[28] Limin Wang, Yuanjun Xiong, Dahua Lin, and Luc Van Gool. Untrimmednets for weakly supervised action recognition and detection. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 4325–4334, 2017. 1, 4

[29] Huifen Xia and Yongzhao Zhan. A survey on temporal action localization. *IEEE Access*, 8:70477–70487, 2020. 1, 2, 3

[30] Pengwan Yang, Vincent Tao Hu, Pascal Mettes, and Cees G. M. Snoek. Localizing the common action among a few videos. In *European Conference on Computer Vision*, pages 505–521. Springer, 2020. 1, 2

[31] Serena Yeung, Olga Russakovsky, Ning Jin, Mykhaylo Andriluka, Greg Mori, and Li Fei-Fei. Every moment counts: Dense detailed labeling of actions in complex videos. *International Journal of Computer Vision*, 126:375–389, 2018. 3

[32] Runhao Zeng, Wenbing Huang, Mingkui Tan, Yu Rong, Peilin Zhao, Junzhou Huang, and Chuang Gan. Graph convolutional networks for temporal action localization. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 7094–7103, 2019. 1

[33] Chenlin Zhang, Jianxin Wu, and Yin Li. Actionformer: Localizing moments of actions with transformers. In *European Conference on Computer Vision*, pages 492–510. Springer, 2022. 1, 2, 3, 4, 5, 6, 7, 8