Benchmarking Data and Computational Efficiency of ActionFormer on Temporal Action Localization Tasks

Analysing the Performance and Generalizability of ActionFormer in Resource-constrained Environments

Jan Warchocki[1]

Supervisors: Dr. Jan van Gemert[1]**, Robert-Jan Bruintjes**[1]**,
Attila Lengyel**[1]**, Ombretta Strafforello**[1]

[1]EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 25, 2023

Name of the student: Jan Warchocki
Final project course: CSE3000 Research Project
Thesis committee: Dr. Jan van Gemert, Robert-Jan Bruintjes, Attila Lengyel, Ombretta Strafforello, Dr.-Ing. Petr Kellnhofer

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 is done assuming access to large amounts of data and computational power. Gathering such data is however a challenging task and access to computational resources might be limited. This work thus explores and measures how well one of such deep learning models, ActionFormer, performs in settings constrained by the amount of data or computational power. Data efficiency was measured by training the model on a subset of the training set and testing on the test set. Although ActionFormer showed promising results on both THUMOS'14 and ActivityNet datasets, TriDet and TemporalMaxer models should likely be chosen in favor of ActionFormer in limited data settings as they exhibit better data efficiency. Similarly, the TriDet model should be chosen in favor of ActionFormer in cases where the training time is limited, as it showed better computational efficiency during training. To test the efficiency of the model during inference, videos of different lengths were passed through the model. Most importantly, we find that both the inference time and the memory usage of the model scale linearly with input video length, as predicted by the authors of the ActionFormer.

1. Introduction

Temporal action localization (TAL) is the process of identifying actions in a video stream. This task has found use in domains such as video summarization [19] and public video surveillance [35, 37]. More precisely, temporal action localization consists of predicting the start and end of an action, and recognizing the action [37]. Different algorithms have been proposed for this task. In recent years, however, models based on deep learning have been found to work better than models based on hand-crafted features [37]. The current state-of-the-art is constituted by models such as TriDet [30], TemporalMaxer [31], and ActionFormer [39].

These deep-learning models are often trained on extensive datasets, such as THUMOS'14 [13] or ActivityNet [15]. However, creating such datasets is difficult and time-consuming [27, 37, 38]. Following the success of Transformers [33] in natural language processing (NLP) tasks [9, 33], some models [24, 39] employ them in temporal action localization. Transformers are, however, known to be computationally expensive [18, 32]. It would thus be desirable to explore how deep learning temporal action localization methods perform in limited data or compute settings.

Special models have been designed for the problem of few-shot learning in the task of temporal action localization, where only a few training videos are available per action class [27, 38]. These proposed models use however an architecture that is incompatible with current state-of-the-art TAL models. Moreover, the authors of some of the current state-of-the-art methods [30, 31, 39] provide a computational complexity analysis by evaluating the model on fixed-size videos. An experimental analysis, that would, for example, show how these models scale with an increase in input video length, is however not provided. It thus remains unanswered how the current state-of-the-art models perform in limited data or compute settings.

Therefore, in this project, we will attempt to answer the question of how one of such state-of-the-art models, ActionFormer [39], performs and generalizes under limited computing power and data settings. ActionFormer, visualized in Figure 1 taken from [39], is of interest, as newer state-of-the-art models TriDet [30] and TemporalMaxer [31] are based on its architecture. Additionally, ActionFormer was one of the first algorithms to successfully demonstrate the use of Transformers [33] in the task of temporal action localization. Answering the aforementioned research question will thus help in understanding the minimum requirements needed by ActionFormer. It will also help in developing future, data or computationally efficient temporal action recognition models.

The contributions of this paper are two-fold. First, a method is designed to test the data efficiency of ActionFormer. Inspired by Ding *et al*. [10] and Henaff [16], the method trains the model on a given percentage p of the training set and tests it on the test set. The procedure is then repeated multiple times and the mean average precisions (mAP) are reported. Through this method, it was found that the model could be trained with only around half of the training data while still maintaining state-of-the-art performance. Second, a method is designed to test the computational efficiency of the model during both training and inference. Training performance is measured by reporting the training time and obtained average mAP. It is found that the ActionFormer model should not be the model of choice in settings where the training time is limited, as it offers worse computational efficiency than TriDet [30]. To test the inference performance, the approach of measuring the computational complexity of the model by passing to it a video of a specific size [30, 31, 39] is expanded upon. The method thus includes evaluating the model on videos of different lengths and reporting the number of floating point operations made, the memory consumed, and the inference time. It is found that the model scales linearly with the length of the input video, as predicted by the authors of [39].

2. Related work

Action recognition. Survey by Xia and Zhan [37] 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 due to its uses in video summarization [19] and public surveillance [35]. In this task, 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 [37]: two-stage and one-stage. Two-stage

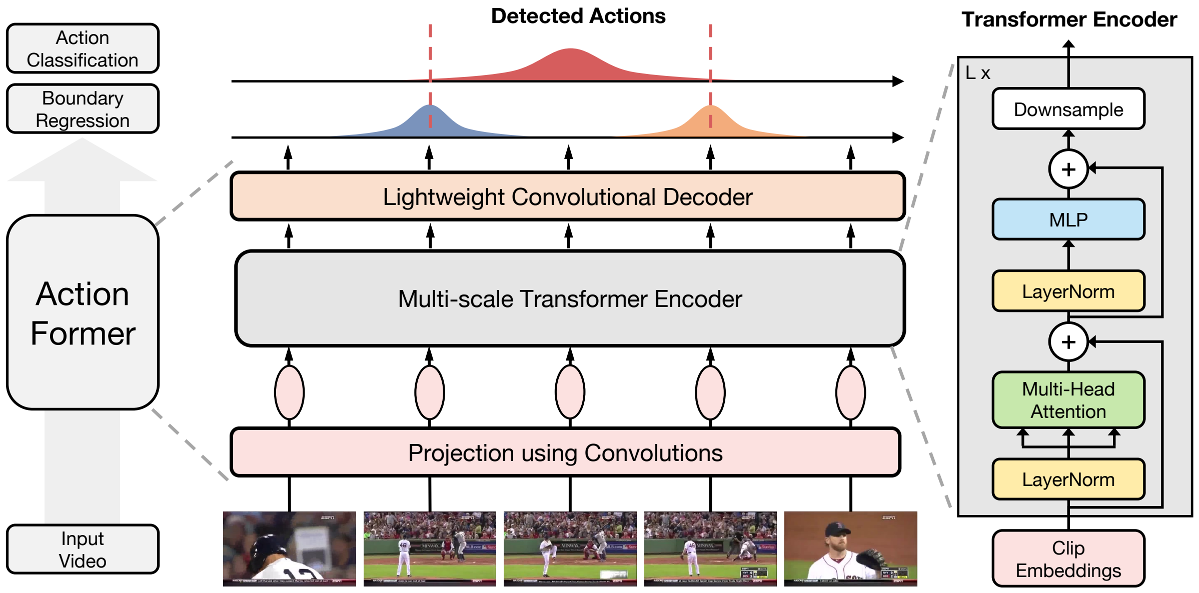


Figure 1. Visualization of the architecture of ActionFormer, taken from [39]. The model first projects input features into D-dimensional embeddings. Using repeating Transformer blocks with downsampling between them, the embeddings are encoded into a feature pyramid. The feature pyramid is then decoded using a convolutional decoder. The decoded features are used to perform action classification and regression for the onset and offset times of the action.

models [12, 22, 23] attempt to first locate the actions and then classify them. One-stage models [24, 30, 31, 39] locate and classify the actions at the same time. This work explores the ActionFormer model [39], which is a one-stage model. Although this work focuses on TAL, insights from trimmed action recognition are used. Namely, features extracted from the Inflated 3D (I3D) model [4] trained on the Kinetics dataset [17] are used as input to the model.

Learning curves. Learning curves provide a visual insight into the generalizability performance of a model with different amounts of training data available [34]. In this work, a learning curve is used when performing experiments regarding data efficiency. The performance of the model, expressed with mean average precision, is plotted against the amount of training data the model was trained on. The resulting line helps in understanding the amount of training data required by the model.

Testing for data efficiency. This problem involves assessing how well a given model performs with limited training data available. Two common approaches have been identified for this task. First, n-shot learning [27, 38], involves training the model on only n samples per each class. The second approach involves training on a given percentage p of the samples from the training dataset [10, 16]. We have however identified several problems with the first approach in the task of temporal action localization. Since a single class can be represented multiple times in a single video [13, 15], 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. In this work, it was thus decided to use the approach where the model is trained on a percentage p of the training set and tested on the test set.

Optimizing for data efficiency. As collecting and annotating datasets is expensive [27, 37, 38], methods have been developed to allow for few-shot learning in the task of temporal action localization. A weakly-supervised algorithm, that uses meta-learning with query and trimmed support videos on input has been proposed [38]. This model was found to generalize well with as little as one training example per class. An extension of that model in a fully supervised setting was proposed [27]. This model uses untrimmed support videos, which are easier to obtain than trimmed videos [27]. This method was also found to work well with little training samples. It is to be noted however, that the models proposed in [38] and [27] require as input all of the support videos at once. This thus makes their architecture incompatible with the architecture of current state-of-the-art models, which only expect a single video as input [30, 31, 39]. Hence, it remains unanswered how current state-of-the-art models perform with limited training data. This work analyzes this problem for ActionFormer [39], which constitutes one of the state-of-the-art models.

Testing for computational efficiency. The term 'computational efficiency' is often used to mean the number of floating point operations made [14, 30–32, 39], the memory used [18, 32], or the training [20] or inference time [30, 31, 39]. In the task of temporal action localization, the TriDet [30], TemporalMaxer [31], and ActionFormer [39] models all report the number of floating point operations expressed through 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. In the case of ActionFormer, testing for different input video lengths would be especially insightful, as the authors of the model claim the model's complexity and memory requirements increase linearly with input video length [39]. Performing such experiments would thus substantiate or invalidate that claim. Thus, in this work, the inference performance of ActionFormer is measured on videos of increasing lengths. Reported are the inference time, video RAM (VRAM) used, and the number of MAC operations executed. Furthermore, motivated by [20], this work reports the training time and the achieved mean average precision of ActionFormer. This is done to better understand the suitability of the model for settings where the training time is limited.

Optimizing for computational efficiency. Both TriDet [30] and TemporalMaxer [31] have been designed to require a lower computational cost than ActionFormer [39], while still maintaining state-of-the-art TAL performance. In TriDet, this was achieved by replacing the multi-head self-attention module with an efficient Scalable-Granularity Perception layer [30]. TemporalMaxer, on the other hand, replaces the entire Transformer module with a max-pooling block [31]. This work compares data and computational efficiencies of ActionFormer against other models, including TriDet [30] and TemporalMaxer [31].

3. Methodology

The ActionFormer model. The architecture of the model is illustrated in Figure 1, sourced from Zhang *et al.* [39]. Most importantly, the model consists of a Transformer [33] encoder, whose structure can be seen on the right of the figure. The encoder accepts embeddings on input and produces encoded features on output. Transformer-based methods are, however, known to be computationally expensive in both training and inference [18, 32]. Furthermore, multi-head self-attention modules, used in Transformers [33], are known to be difficult to parallelize on graphical units [5, 7, 39]. Both of these factors may influence the data or computational efficiency of the model.

3.1. Data efficiency

Evaluation metrics. As is common practice [2, 30, 31, 37, 39], the model was 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 durations 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 is the model performing.

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.* [10] and Henaff [16], a percentage p of the training set $\mathcal{D}_{\text{train}}$ was randomly and uniformly sampled to create a set $\mathcal{D}_{\text{s}}$. The model was 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 [3, 10] 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:

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

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 ← 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 ← $\mathrm{calculate\text{-}mAP}(\mathcal{D}_{\text{test}})$
        mAPs.append(mAP)
    Report $\mathrm{avg}(\mathrm{mAPs})$ and $\mathrm{std}(\mathrm{mAPs})$
```

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 the 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:

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

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

3.2. Computational efficiency

3.2.1 Training performance

Inspired by Li *et al.* [20], the training time of the ActionFormer model 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, randomly-selected seeds, each time measuring the time spent and the average mAP achieved. The exact values of the used seeds can be found in the source code of this paper.

3.2.2 Inference performance

Evaluation metrics. Following the works on TriDet [30], TemporalMaxer [31], and ActionFormer [39] the model was evaluated by reporting the total number of multiply-accumulate (MAC) floating point operations and the inference time when fed an input video. Furthermore, the number of MACs was divided by the inference time to obtain the number of MACs/s, which was used to calculate hardware utilization information. Hardware utilization was computed by dividing the observed number of floating point operations per second (MACs/s) by the theoretical performance obtained from hardware documentation. As the theoretical maximum performance is often expressed in terms of FLOPS/s, it should be noted at this point that a single multiply-accumulate operation corresponds to two floating point operations (add and multiply). Finally, as Transformer-based methods are known to require large amounts of memory [18,32], the total amount of space used by the algorithm was measured.

To count the number of multiply-accumulate operations, the `fvcore` library [29] was used. Inference timing was done using Python built-in `time.time` method. Using the `time.get_clock_info` method, the resolution of that clock was found to be $\pm$ 1 ns on the used hardware, which was precise enough for this task. As ActionFormer runs on a GPU, it was the video memory (VRAM) that had been reported. This was done using the `max_memory_allocated` method from PyTorch.

Testing procedure. 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 [39]. 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[1].

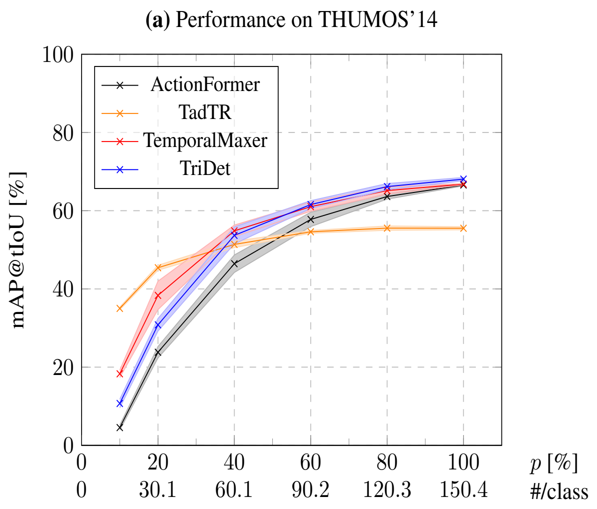
The model was evaluated on randomly-generated tensors, whose shapes correspond to videos of differing lengths. To guarantee the independence of results, the experiments for inference time, memory consumption, and number of MACs were run separately for each such tensor as separate operating system processes. Before each inference time measurement, the random tensor was passed through the model once as a warm-up procedure. This was found to help in investigating the dependency of the model on the input size. Without this procedure, the inference time would be constant for all input sizes, most likely due to memory allocations happening as the tensor was being forwarded through the network. Furthermore, the experiments for inference time were repeated 5 times [30]. Each such repetition would use a different, randomly-generated but fixed seed, whose exact value can be found in the source code of the project.

4. Experiments

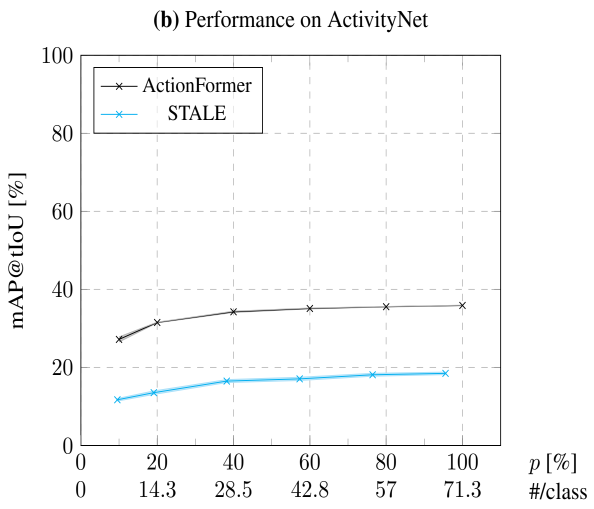
Datasets. The model was evaluated on two datasets, commonly used to assess temporal action localization algorithms [30, 39]: THUMOS'14 [13] and ActivityNet [15]. THUMOS'14 contains 413 untrimmed videos and 20 action classes. This dataset is further split into a validation set, containing 213 videos and a test set containing 200 videos. In total, the validation set contains 3,007 action instances. Following the work by Zhang *et al.* [39], the model is trained on the validation set and tested on the test set. 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 [39], the model is trained on the training set and evaluated on the validation set. Furthermore, it is to be noted, that some of the videos from the ActivityNet dataset have become unavailable over time, resulting in 9,251 videos in the training set and 4,555 videos in the validation set. This training set contains a total of 14,253 action instances.

Similarly to the paper by Zhang *et al.* [39], in both of these datasets, the model uses Inflated 3D (I3D) features [4] pre-trained on the Kinetics dataset [17]. Although ActionFormer achieved 1 percentage point higher average mAP when using TSP features [2] on the ActivityNet dataset [15, 39], in this work, I3D features are used for consistency

[1]This was verified with one of the authors of ActionFormer in the original repository: https://github.com/happyharrycn/actionformer_release/issues/109



(a) Achieved average mAP@tIoU[0.3:0.1:0.7] for the THUMOS'14 dataset [13] of the compared models. It should be noted that the #/class x-axis is not applicable to the TadTR model [24] as the model employed video splitting. Importantly, the ActionFormer model beats the previous best average mAP [21, 39] with only around 40-50% of the dataset.



(b) Achieved average mAP@tIoU[0.5:0.05:0.95] for the ActivityNet dataset [15] of the compared models. The x-axis for the STALE model [26] is shifted to the left due to fewer training samples being available. Importantly, the ActionFormer model saturates around the 20-40% mark and does not gain from additional data.

Figure 2. Reported average mAP@tIoU for the tested models on the THUMOS'14 [13] and ActivityNet [15] 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. On both of the graphs, the performance of ActionFormer increases with an increase in p. Furthermore, for $p = 100\%$ the results of the original paper by Zhang *et al.* [39] were reproduced.

in comparison across datasets. The exact features used in experiments are those extracted by the authors of [39].

Comparison with other models. This paper compares the ActionFormer model against other temporal action localization models. To this end, the paper combines the results obtained for TriDet [30] by Dămăcuş [11], TadTR [24] by Misterka [25], TemporalMaxer [31] by Oprescu [28], and STALE [26] by Wang [36]. Although the same procedures for testing data and computational efficiencies were attempted to be followed, some small deviations might be present between the works. This might also be a result of the different setups required by different models.

Experimental setup. The model was trained using a single Nvidia Tesla V100S 32GB located on the DelftBlue cluster [8]. This was chosen as the hardware setup due to its availability to all students of the Delft University of Technology. All of the training and testing hyperparameters were left unchanged from the work by Zhang *et al.* [39] unless specified otherwise in Section 4.1 or Section 4.2. This, therefore, also includes the score fusion used by the authors of ActionFormer on the ActivityNet dataset [39].

4.1. Data efficiency

Results on THUMOS'14. Recalling Algorithm 1, datasets $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{test}}$ are required. Following [39], in this setup, the validation set containing 213 videos is used as $\mathcal{D}_{\text{train}}$, and the test set containing 200 videos is used as $\mathcal{D}_{\text{test}}$. The model is then trained on 6 different percentages p of the training set. Similarly to the paper by Zhang *et al.* [39], the trained model is then evaluated on the test set by reporting the mAP at tIoU thresholds from 0.3 to 0.7 in 0.1 increments. Additionally, the average mAP over the 5 thresholds is reported. Figure 2a presents the results, where the width of each line represents two standard deviations obtained by repeating each sampling, training, and testing procedure 5 times.

As can be observed from Figure 2a, the performance of the algorithm increases with an increasing percentage of the training set p. At $p = 100\%$, the model achieves an average mAP of 66.57 ± 0.22 [%], which is almost within one standard deviation away from the average mAP of 66.8% from the original paper [39]. Furthermore, it is to be noted that at the time of the proposal of ActionFormer, state-of-the-art average mAP for the THUMOS'14 dataset was achieved by the AFSD model [21, 39] and was equal to 52%. As can be observed from Figure 2a, ActionFormer achieves this performance with already around 40-50% of the dataset. These percentages correspond to about 60-75 action instances per class.

Results on ActivityNet. In this setup, following the work by Zhang *et al.* [39], the training set is used as $\mathcal{D}_{\text{train}}$ and the validation set is used as $\mathcal{D}_{\text{test}}$. The model is once again trained on 6 different percentages p of the training set. Also following the work by Zhang *et al.* [39], the model is evaluated on the test set by reporting mAP at tIoU thresholds from 0.5 to 0.95 in 0.05 increments. Additionally, the average mAP over all 11 thresholds is reported. The results of the experiment can be found in Figure 2b. The width of the line corresponds to two standard deviations obtained by repeating the procedure 5 times.

Firstly, it can be seen that the performance of the model steadily increases with an increase in p. Secondly, for $p = 100\%$, the average mAP was found to be 35.89 ± 0.06 [%] compared to the original 35.6% [39]. Authors of the ActionFormer [39] observed however that the performance of the model was different when using different hardware. This could explain the difference seen in this experiment. Furthermore, a plateau can be observed around the 40% mark. This shows, that the model achieves its full performance with only around 40% of the training data, which corresponds to around 28 action instances per class.

Comparison with other models. As can be observed from Figure 2a, TriDet [30], TemporalMaxer [31], and ActionFormer [39] all follow a similar learning curve. The difference between the models can be most clearly seen at $p = 10\%$, where the difference between ActionFormer and TemporalMaxer is equal to 13.75 percentage points in average mAP. Interestingly, the TadTR [24] model follows a different curve. This is caused by video slicing employed by the model, which increases the amount of training data available [24]. From Figure 2b, it can be observed that the ActionFormer model outperforms the STALE model [26, 36] on all tested percentages p.

Discussion. In both the THUMOS'14 dataset and the ActivityNet dataset, the size of the training set could be almost halved while still maintaining a good average mAP. This is indicated by the stagnation in performance around the 50-60% mark. Additionally, on the THUMOS'14 dataset, the model would still achieve state-of-the-art performance with access to only 50% of the training data [21, 39]. Based on these results, it is difficult to put an exact bound on the required number of instances per class required for the model. This value is, however, in the same order of magnitude for both datasets. On both datasets, the model reaches state-of-the-art performance with less than 100 instances per class. This performance is, however, still far from one that would enable few-shot learning, where only a few instances per class are available [27, 38]. Following the data efficiency results in this paper, it can be observed that with little training data available, the TadTR [24] model would work the best. Furthermore, all tested values of p suggest that the TriDet [30] and TemporalMaxer [31] models should be chosen over the ActionFormer [39] model. ActionFormer is, however, likely to be selected over STALE [26] in limited data settings.

Known limitations. It should be noted that the method used to compare the models in this work is limited as ActionFormer is the only model that is evaluated on both datasets. A more extensive study evaluating the models on different datasets is required to provide more insight into the comparison between the techniques. Furthermore, the tests on ActivityNet presented in Figure 2b could be extended with lower percentages p to test if the STALE model [26] could outperform ActionFormer in an extremely data-sparse setting.

4.2. Computational efficiency

Concurrent jobs on DelftBlue. By default, the GPU nodes are shared between users in the DelftBlue cluster [8]. This setup could lead to a dependence of the training or inference time on the other jobs running on the cluster. As an attempt to alleviate this issue, the training and inference time experiments were performed 5 times sequentially, such that the experiment jobs did not overlap. Each of these repetitions would use the same random seeds, such that the exact same experiment was repeated. In this work, an assumption is further made, that if the observed variance in training or inference times is low, the other, concurrent jobs likely did not interfere with the experimental job.

4.2.1 Training performance

Results. The results obtained by ActionFormer are compared to the results of TriDet [11, 30], TemporalMaxer [28, 31], and TadTR [24, 25] on THUMOS'14 dataset [13], and STALE [26, 36] on ActivityNet [15]. The training time is reported alongside the average mAP obtained by the model on the test set. Table 1 presents the results on the THUMOS'14 dataset. As can be observed, the average training time is the second largest for the ActionFormer model. Despite the long training time, the model does not present the best average mAP. The TriDet model marks the best performance and outperforms ActionFormer by 2.18 percentage points in average mAP. Table 2 shows the results on ActivityNet. It can be seen that the ActionFormer requires almost five times more training time compared to STALE, but also achieves better results.

Table 1. Training performance of ActionFormer and other compared models on the THUMOS'14 dataset [13]. Both average training time and obtained average mAP@tIoU[0.3:0.1:0.7] are reported. As can be observed, the ActionFormer model takes the longest amount to train while not achieving the best performance.

Model	Time [s]	Avg. mAP [%]
ActionFormer	886.8 ± 54.3	65.89 ± 0.09
TadTR	425.7 ± 3.5	55.3 ± 0.63
TemporalMaxer	2956.6 ± 1660	66.96 ± 0.37
TriDet	646.2 ± 26.1	68.07 ± 0.42

Table 2. Training performance of ActionFormer and STALE [26] on the ActivityNet dataset [15]. Both average training time and obtained average mAP@tIoU[0.5:0.05:0.95] are reported. As can be observed, the ActionFormer model takes longer to train but also achieves better performance.

Model	Time [s]	Avg. mAP [%]
ActionFormer	1944.9 ± 60.6	35.9 ± 0.14
STALE	400.7 ± 5.8	19.37 ± 0.16

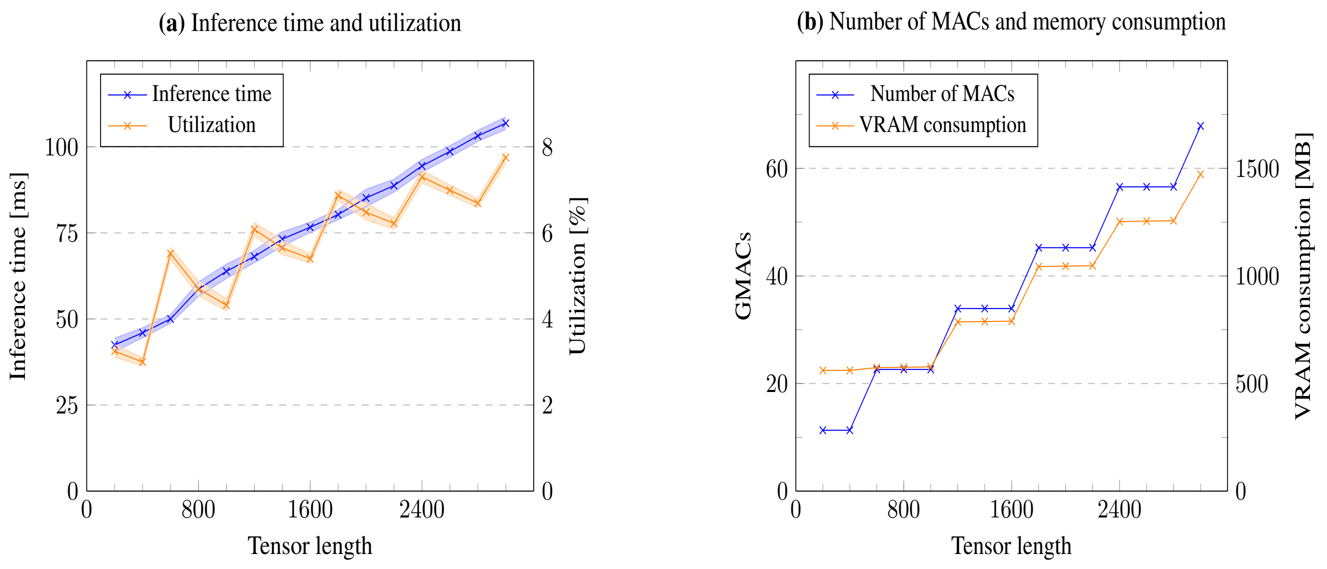


Figure 3. Results for testing the inference performance of ActionFormer [39]. Randomly generated tensors of lengths from 200 to 3000 in 200 increments were generated and passed through the model. The number of multiply-accumulate operations was calculated using the `fvcore` library [29]. The video memory consumption was measured using the `max_memory_allocated` method from PyTorch. Utilization information was measured by calculating GMACs/s and dividing that number by the official Nvidia Tesla V100S 32GB performance of 8.2 TMACs/s [6]. The width of each line in Figure 3a corresponds to two standard deviations obtained by repeating the experiment 25 times. As the 'steps' in memory consumption and the number of MACs are the same length and increase equally in height, the model scales linearly with respect to input size. This can also be observed with inference time.

Discussion. The ActionFormer is unlikely to be chosen in settings where the training time is limited. In those scenarios, TadTR [24] would be more applicable as it was found to offer a 2x training speed-up compared to ActionFormer. Choosing TadTR would however likely come with a decrease in temporal action localization performance, as was observed in Table 1. Alternatively, the TriDet [30] model could be used, which offers a speed-up in training time while still outperforming the ActionFormer model in mean average precision. From Table 2 it can be observed that the STALE model could be selected in a limited training time scenario over ActionFormer, although sacrificing in TAL performance.

Known limitations. It should be noted that the above procedure for testing training performance has 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, it was assumed that if the variance in training times is low, the other concurrent jobs on DelftBlue did not interfere with the experiment. This assumption would not however hold if all of the concurrent jobs would utilize hardware equally. Finally, the TadTR model was evaluated using a different experimental setup [25], thus the timings might not reflect the difference between models fully accurately.

4.2.2 Inference performance

Additional experimental setup. The results were obtained by creating random tensors corresponding to the Inflated 3D (I3D) [4] features extracted from the THUMOS'14 dataset [13] by the authors of ActionFormer [39]. In this setup, the minimum allowable value of `max_seq_len` was found to be 576, which thus constitutes the value of `max_seq_len` that was used in the configuration file of the model. Due to this change, the maximum tensor size that can be passed through the model without any further changes to the configuration is 3456. Therefore, in this work, the sizes of the tensor vary from 200 to 3000 in 200 increments. This exact configuration is also used because it was found to visualize the dependence of the model on the input size well.

Results. Figure 3 presents the results, where the means over the total 25 runs for inference time are reported along with the obtained standard deviation. Utilization information was calculated based on the official floating point performance of the Nvidia V100S 32GB GPU of 16.4 TFLOPS/s [6,8] or 8.2 TMAC/s. First, as seen in Figure 3a, the relatively small standard deviations in inference time indicate that other DelftBlue jobs did not interfere with the experiment. Moreover, as can be observed from Figure 3, all the measured statistics increase with an increase in the tensor size. It can be noticed, that the statistics for the number of GMACs and memory consumption in Figure 3b increase in 'steps'. This is caused by ActionFormer padding input videos to sizes that are multiples of 576, which is required by the architecture of the model [39]. The inference time, on the other hand, increases smoothly. This is caused by the fact that even though videos are padded, the longer the video the more predictions the model will make. Consequently, this slows down the postprocessing step, where the predictions have to be analyzed for, for example, overlaps. Furthermore, as can be noticed, the hardware is not fully utilized. This could be caused by the fact that self-attention modules are known to be IO-bound and hence difficult to efficiently parallelize on graphical units [5,7,39]. Finally, it can be seen that inference time, memory consumption, and the number of GMACs all increase linearly with respect to the size of the input. This thus validates the claim made by the authors of ActionFormer, that the model scales linearly with an increase in input video length [39].

Comparison with other models. From the works by Dămăcuş [11] and Oprescu [28], it can be seen that both the TriDet [30] and the TemporalMaxer [31] models are more efficient during inference than ActionFormer. The inference time, as well as the number of MACs, are consistently lower for different tensor sizes for these models. This difference is best visualized in Fig. 4 and Fig. 5 in Appendix A. Similarly, in the work by Misterka [25], it can be seen that the TadTR model [24] takes around 15ms for tensor size 3000, greatly outperforming ActionFormer, which requires over 100ms for the same video length.

Discussion. The compute efficiency results obtained in this work can be used to provide lower bounds for hardware needed to run ActionFormer [39]. It was, for example, observed that to forward a video of a length of 5 minutes (tensor length 2247) from the THUMOS'14 dataset [13], the model requires a minimum of around 1,100 megabytes of VRAM. Furthermore, given these hardware utilization results, the inference time could be approximated for setups different than the one used in these experiments. This thus also includes hardware more limited than what is available on the DelftBlue cluster. Following the comparison with other models, it can be deduced that either the TriDet model [30] or the TemporalMaxer [31] model should be chosen in favor of ActionFormer if the computational resources during inference are limited.

Known limitations. It should be noted that this method for testing inference performance is limited, as an assumption was made that if the variance in inference times is small, the concurrent DelftBlue jobs did not interfere with the experiment. It is possible, however, that all of the jobs interfered equally. Furthermore, RAM usage could be measured alongside VRAM to provide more insight into the amount of memory used. Finally, it should be noted that the results for the TadTR model were obtained using a different compute instance [25], thus the results might not be fully comparable.

5. Responsible research

Various steps have been done to ensure the reproducibility of the research. First and foremost, the code as well as the obtained results in raw format are made publicly available at `https://github.com/Jaswar/bachelor-thesis`. The code is licensed with an MIT license as required by the ActionFormer repository. Secondly, the results that could differ when reproduced, are reported with a mean as well as a standard deviation. This includes metrics such as mean average precision, training time, and inference time. Furthermore, the experiments for compute efficiency have been constructed in such a way as to use a fixed random seed, whose exact value can be found in the source code. Thus, the randomly generated tensors will always be the same. Finally, the exact splits $\mathcal{D}_s$ used in the data efficiency experiments are published in the repository.

Nonetheless, some results might still differ when using different hardware. As noted by [39], the performance of the ActionFormer may be different on different setups. Similarly, the training and inference times are going to be different depending on the exact hardware used. On the contrary, if the same hardware is used, the inference and training time results should still be similar, despite this work using the DelftBlue cluster. This is thanks to the extra step taken that repeats the experiments multiple times sequentially.

This work fulfills the five principles of research integrity outlined in [1]: *Honesty*, *Scrupulousness*, *Transparency*, *Independence*, and *Responsibility*. *Honesty* is achieved through acknowledging the limitations of the work, considering alternative solutions to answer the research question, and providing sources for claims made in the work. *Scrupulousness* is realized by carefully designing a scientific method to answer the research question. *Trasparency* is ensured by providing the source code of the work alongside a description of how to reproduce the results, and the results in a raw format. *Independence* holds as the work is not guided by any commercial or political motives. Finally, *Responsibility* is achieved as the research is societally and scientifically relevant and will help in devising data or computationally efficient temporal action localization models.

One of the uses of temporal action localization is in the domain of video surveillance [35]. By providing insight into the computational efficiency of ActionFormer, it will be better understood to what extent this model can be deployed on embedded devices. In the future, this might thus result in increased public video surveillance. In this paper, we acknowledge this problem and suggest more work be done to alleviate this issue. Perhaps laws and regulations could be created to limit or control the use of temporal action localization models in certain tasks. Research could also be guided to examine the sociological impacts of performing research in data or computationally efficient algorithms in different computer vision domains, including temporal action localization.

6. Conclusion

This work aimed to answer the question of how the ActionFormer [39] model performs in a limited data or computational power setting. To this end, a methodology was designed for testing the model with access to only a limited amount of training data. The method consisted of repeatedly sampling a smaller set from the entire training set and training the model on the sampled set. The performance was then measured on the test set. It was found, that on both THUMOS'14 [13] and ActivityNet [15] datasets, the model performs well with only around half the training data available, requiring less than 100 action instances per class. Nonetheless, TriDet [30] and TemporalMaxer [31] should be chosen in favor of ActionFormer when training data is limited. Furthermore, a method for testing computational efficiency during training and inference was designed. The method for measuring training performance consisted of training the model on both THUMOS'14 and ActivityNet and reporting the time it took. The model was then evaluated on the test set and the average mAP was noted. It was found that the ActionFormer model is unlikely to be used in situations where the training time is limited, as the TriDet [30] model outperforms it in such settings. The method for measuring performance during inference consisted of measuring the number of floating point operations, video memory consumed, and the inference time of the model when fed videos of increasing sizes. It was found that the model scales linearly with an increase in video size. The results for hardware utilization during inference obtained with this method can also be used to predict inference times when using different hardware than the one used in this paper.

As this work focused solely on the current performance of ActionFormer in resource-constrained environments, the next step could be to improve its performance in such settings. Perhaps through hyperparameter tuning or by modifying the structure of the model, a better algorithm could be found. Furthermore, the computational efficiency of the model could be further evaluated. This work focused only on the input video length as it is one of the very few parameters that are shared across all temporal action localization models. Nonetheless, a study that modifies parameters specific to ActionFormer could be conducted.

Acknowledgements

I would foremost like to thank the responsible professor Dr. Jan van Gemert, the supervisors, Robert-Jan Bruintjes, Attila Lengyel, and Ombretta Strafforello for all the feedback, input, and help I have received from them. I would also like to thank my teammates and friends for the discussions and feedback they have provided. I am also grateful to Professor Yin Li from the University of Wisconsin-Madison for answering my questions on ActionFormer.

References

[1] Keimpe Algra, Lex Bouter, Antoine Hol, Jan van Kreveld, Daan Andriessen, Catrien Bijleveld, Roberta D'Alessandro, Jenny Dankelman, and Peter Werkhoven. Netherlands code of conduct for research integrity (2018), 2018. 8

[2] Humam Alwassel, Silvio Giancola, and Bernard Ghanem. Tsp: Temporally-sensitive pretraining of video encoders for localization tasks, 2020. 3, 4

[3] David Berthelot, Nicholas Carlini, Ian Goodfellow, Nicolas Papernot, Avital Oliver, and Colin Raffel. Mixmatch: A holistic approach to semi-supervised learning, 2019. 3

[4] Joao Carreira and Andrew Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset, 2017. 2, 4, 7

[5] Shiyang Chen, Shaoyi Huang, Santosh Pandey, Bingbing Li, Guang R. Gao, Long Zheng, Caiwen Ding, and Hang Liu. E.t.: re-thinking self-attention for transformer models on gpus, 2021. 3, 8

[6] NVIDIA Corporation. Nvidia v100 tensor core gpu. https://images.nvidia.com/content/technologies/volta/pdf/volta-v100-datasheet-update-us-1165301-r5.pdf, 2020. (Accessed on 05/06/2023). 7

[7] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness, 2022. 3, 8

[8] Delft High Performance Computing Centre (DHPC). DelftBlue Supercomputer (Phase 1). https://www.tudelft.nl/dhpc/ark:/44463/DelftBluePhase1, 2022. (Accessed on 13/06/2023). 5, 6, 7

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018. 1

[10] 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

[11] Alex Dămăcuş. *Efficient Video Action Recognition*. Bachelor's thesis, Delft University of Technology, 2023. 5, 6, 8

[12] Guoqiang Gong, Liangfeng Zheng, Kun Bai, and Yadong Mu. Scale matters: Temporal scale aggregation network for precise action localization in untrimmed videos, 2019. 2

[13] Alex Gorban, Haroon Idrees, Yu-Gang Jiang, Amir Roshan Zamir, Ivan Laptev, Mubarak Shah, and Rahul Sukthankar. Thumos challenge: Action recognition with a large number of classes. http://crcv.ucf.edu/THUMOS14/, 2014. (Accessed on 20/06/2023). 1, 2, 4, 5, 6, 7, 8, 9

[14] Kai Han, An Xiao, Enhua Wu, Jianyuan Guo, Chunjing Xu, and Yunhe Wang. Transformer in transformer, 2021. 2

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

[16] Olivier Henaff. Data-efficient image recognition with contrastive predictive coding, 2020. 1, 2, 3

[17] Will Kay, Joao Carreira, Karen Simonyan, Brian Zhang, Chloe Hillier, Sudheendra Vijayanarasimhan, Fabio Viola, Tim Green, Trevor Back, Paul Natsev, Mustafa Suleyman, and Andrew Zisserman. The kinetics human action video dataset, 2017. 2, 4

[18] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer, 2020. 1, 2, 3, 4

[19] 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

[20] 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, 2020. 2, 3, 4

[21] Chuming Lin, Chengming Xu, Donghao Luo, Yabiao Wang, Ying Tai, Chengjie Wang, Jilin Li, Feiyue Huang, and Yanwei Fu. Learning salient boundary feature for anchor-free temporal action localization, 2021. 5, 6

[22] Tianwei Lin, Xiao Liu, Xin Li, Errui Ding, and Shilei Wen. Bmn: Boundary-matching network for temporal action proposal generation, 2019. 2

[23] Tianwei Lin, Xu Zhao, Haisheng Su, Chongjing Wang, and Ming Yang. Bsn: Boundary sensitive network for temporal action proposal generation, 2018. 2

[24] 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, 5, 6, 7, 8

[25] Paul Misterka. *Efficient Temporal Action Localization model development practices*. Bachelor's thesis, Delft University of Technology, 2023. 5, 6, 7, 8

[26] Sauradip Nag, Xiatian Zhu, Yi-Zhe Song, and Tao Xiang. Zero-shot temporal action detection via vision-language prompting, 2022. 5, 6

[27] Sauradip Nag, Xiatian Zhu, and Tao Xiang. Few-shot temporal action localization with query adaptive transformer, 2021. 1, 2, 6

[28] Teodor Oprescu. *TemporalMaxer Performance in the Face of Constraint: A Study in Temporal Action Localization*. Bachelor's thesis, Delft University of Technology, 2023. 5, 6, 8

[29] Facebook Research. fvcore. https://github.com/facebookresearch/fvcore, 2023. (Accessed on 04/06/2023). 4, 7

[30] Dingfeng Shi, Yujie Zhong, Qiong Cao, Lin Ma, Jia Li, and Dacheng Tao. Tridet: Temporal action detection with relative boundary modeling, 2023. 1, 2, 3, 4, 5, 6, 7, 8, 9

[31] Tuan N. Tang, Kwonyoung Kim, and Kwanghoon Sohn. Temporalmaxer: Maximize temporal context with only max pooling for temporal action localization, 2023. 1, 2, 3, 4, 5, 6, 8, 9

[32] Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. Efficient transformers: A survey. *ACM Comput. Surv.*, 55(6):Article 109, 2022. 1, 2, 3, 4

[33] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017. 1, 3

[34] Tom Viering and Marco Loog. The shape of learning curves: A review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(6):7799–7819, 2023. 2

[35] Sarvesh Vishwakarma and Anupam Agrawal. A survey on activity recognition and behavior understanding in video surveillance. In *Visual Computer*, volume 29, pages 983–1009, 2013. 1, 8

[36] Yunhan Wang. *Efficient Temporal Action Localization via Vision-Language Modelling*. Bachelor's thesis, Delft University of Technology, 2023. 5, 6

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

[38] Pengwan Yang, Vincent Tao Hu, Pascal Mettes, and Cees G. M. Snoek. Localizing the common action among a few videos, 2020. 1, 2, 6

[39] Chenlin Zhang, Jianxin Wu, and Yin Li. Actionformer: Localizing moments of actions with transformers, 2022. 1, 2, 3, 4, 5, 6, 7, 8

A. Auxiliary plots

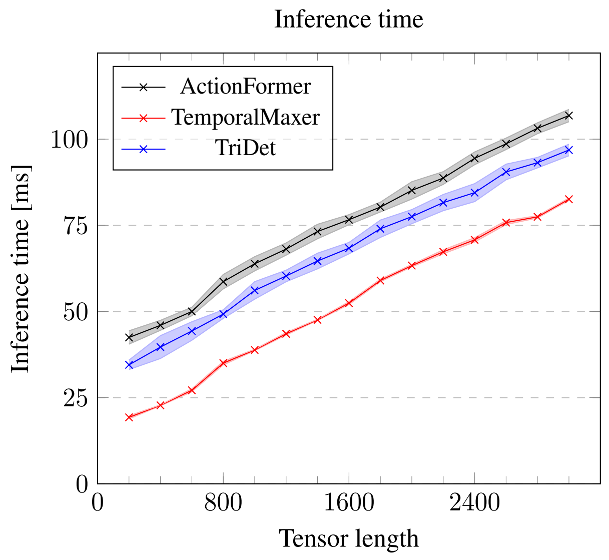


Figure 4. Comparison of the inference time of the different models. As can be seen, TriDet and TemporalMaxer both outperform ActionFormer on all video lengths.

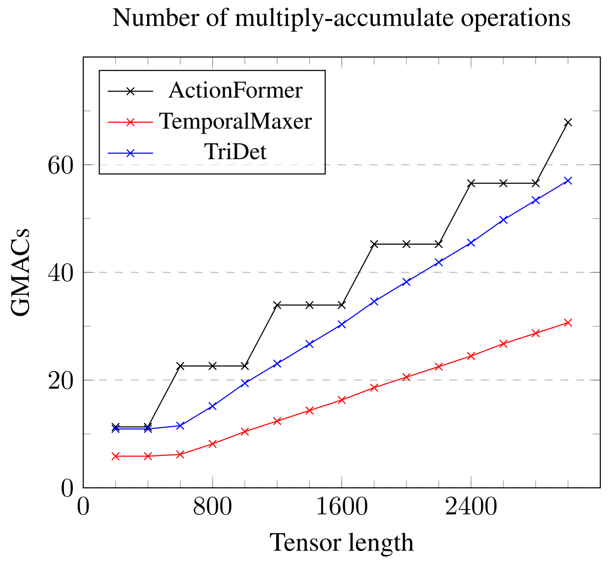


Figure 5. Comparison of the number of GMACs of ActionFormer and TriDet. As can be observed, both TriDet and TemporalMaxer require fewer floating point operations than ActionFormer on all tested values of video length.