Efficient Video Action Recognition

How well does TriDet perform and generalize in a limited compute power and data setting?

Alexandru Damacus

Supervisors: Dr. Jan van Gemert[1], Ombretta Strafforello[1], Robert-Jan Bruintjes[1], Attila Lengyel[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: Alexandru Damacus
Final project course: CSE3000 Research Project
Thesis committee: Dr. Jan van Gemert, Ombretta Strafforello, Robert-Jan Bruintjes, Attila Lengyel, Dr.-Ing. Petr Kellnhofer

An electronic version of this thesis is available at http://repository.tudelft.nl/.

Abstract

In temporal action localization, given an input video, the goal is to predict the action that is present in the video, along with its temporal boundaries. Several powerful models have been proposed throughout the years, with transformer-based models achieving state-of-the-art performance in the recent months. Although novel models are becoming more and more accurate, authors rarely study how limited training data or computation power environments affect the performance of their model. This study is carried out on TriDet, a transformer-based temporal action localization model that achieves state-of-the-art performance on two different benchmarks. It evaluates the model's behavior in a limited training data and computation power environment. It is found that TriDet achieves close to state-of-the-art performance when only 60% of the training data or approximately 90 action instances per class are used. It is also notable that inference time, memory usage, multiply-accumulate operations and GPU utilization scale linearly along with the length of the tensor that is passed to the model. These findings, combined with TriDet's mean training time of 11 minutes on the THUMOS'14 dataset can be used to determine the model's hypothetical behavior when run in lower computation power environments.

1. Introduction

Temporal action localization (TAL) refers to the process of understanding actions that are happening in an input video. TAL has two main parts, namely predicting the temporal boundaries of the action, and recognizing the action present in the interval denoted by those boundaries. It has a large number of applications in action detection, surveillance, video summarization, and more.

Temporal action localization has seen more and more powerful models proposed for all of the popular benchmarks [5, 7] in recent years, to some extent due to the advancements made in the field of deep learning [21,25]. Ever since the emergence of the transformer architecture [18], the state-of-the-art for those popular benchmarks seems to be dominated by transformer-based TAL models. However, their power partly comes from the immense amounts of training data and computational power that is needed to train them.

This study aims at assessing the data and compute efficiency of popular TAL models. Although novel models achieving state-of-the-art performance are produced at a very accelerated rate, only a few authors study their models in terms of data and compute efficiency. These studies are needed because new models require immense amounts of training data and computation power, which are difficult to obtain. Having sufficient knowledge in this direction could help further researchers develop and test their models on partial datasets or using limited computation power, and infer their maximum potential from similar models' learning curves. This would also lead to an acceleration in the development of new and more powerful methods, thus leading to further advancements in the field of TAL.

The study is carried out in a group of five students, each conducting the proposed experiments on a different TAL model. In this paper I conduct a study on TriDet [16], a very recent model that achieves state-of-the-art performance on two of the most popular benchmarks, HACS [24] and EPIC-KITCHEN 100 [3], while being more efficient compared to other competing models [16].

2. Related Work

This section provides a description of my chosen model, TriDet [16]. Several methods for computing data efficiency of models in Computer Vision are described, along with an overview of popular methods tailored specifically for TAL, and the method that was chosen for this study. Additionally, popular methods for computing a model's compute efficiency are presented, along with the chosen methods for this study.

TriDet [16] is a one-stage framework for temporal action detection, meaning that it predicts action boundaries without explicitly generating temporal proposals or segments. It employs a transofmer-based architecture, consisting of three main parts: a video feature backbone, a Scalable-Granularity Perception (SGP) feature pyramid, and a boundary-oriented Trident-head. A visualization of the model's architecture is presented in Figure 1, taken from [16]. First, the video features are extracted using a pre-trained action classification network. Then, a SGP feature pyramid is built to handle actions with various temporal lengths. Each scale level is processed with a proposed SGP layer to enhance the interaction between features with different temporal scopes. Finally, the boundary-oriented Trident-head models the action boundary via an estimated relative probability distribution around the boundary to address the imprecise boundary predictions problem. TriDet [16] is the model chosen for conducting this study. It was chosen due to its impressive performance on some of the popular benchmarks, such as THUMOS'14 [7], HACS [24] and EPIC-KITCHEN 100 [3], along with its improved efficiency compared to other transformer-based models that achieve state-of-the-art performance.

Evaluating the data efficiency of a model in Computer Vision involves assessing the model's performance and robustness using limited training data. Several popular techniques for studying a model's data efficiency are used in practice. One common approach is *few-shot learning* [13, 22], where models are trained on a small number of samples per class. Few-shot learning tests the model's ability to generalize from limited data. Additionally, *zero-shot learning* [12] assesses the model's ability to recognize unseen classes. By evaluating the model's performance on unseen classes, one can grasp an understanding of the model's capabilities to generalize and infer information. *Learning curves* show the model's accuracy against the amount of training data used. By observing the learning curve, one

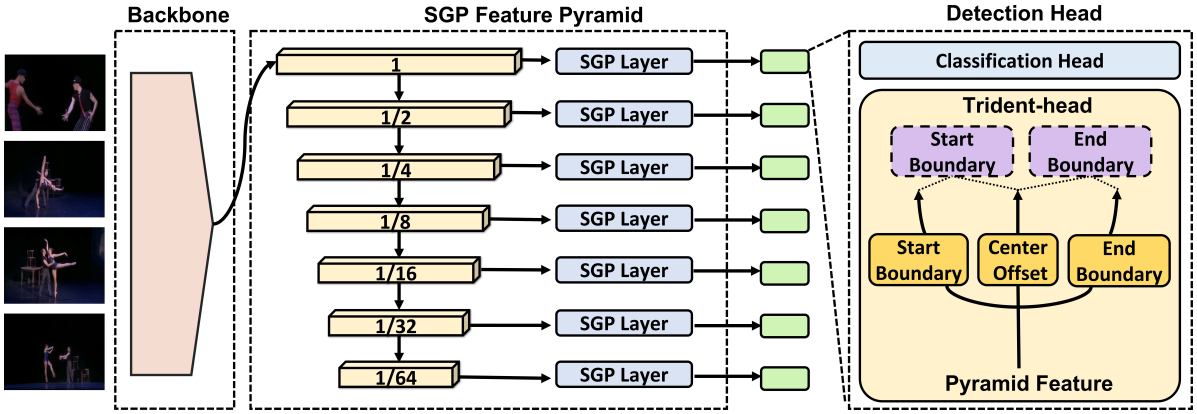

Figure 1. Overview of the architecture of TriDet, taken from [16].

can understand how the model's performance changes as more data is provided. Steep learning curves indicate that the model benefits significantly from additional data, while flat curves indicate the model's maximum capacity on the available data. Another common approach is *transfer learning* [6], which involves using pre-trained models on large datasets and fine-tuning them for different datasets. By assessing the model's performance with varying amounts of training data, one can evaluate how effectively the pre-trained knowledge transfers to the new task and quantify the data efficiency.

For this study, *two methods* of evaluating data efficiency have been discussed among the research group; *few-shot learning* and *sampling percentages of the training data*. Because of class imbalances and multiple classes appearing in a single video in the chosen dataset, *sampling percentages of the training data* was deemed as the appropriate procedure to follow. Additionally, a few-shot learning approach was deemed more difficult to produce in the given timeframe of the project.

Evaluating the compute efficiency of a model involves assessing how effectively it utilizes computational resources. In practice, various methods of studying a model's compute efficiency are used. One of the common methods is measuring a model's number of multiply–accumulate operations (MACs) or floating-point operations per second (FLOPs) [16, 17, 23]. Some other commonly used methods include measuring the memory used by the model [8], the training time [9] or the inference time [16, 17, 23]. Both TriDet [16] and ActionFormer [23] report the number of MACs and the model's run-time by passing a fixed length video of 2304 samples (5 minutes) through the network. However, it would be interesting to study how those metrics change when the video length varies.

The compute efficiency of TriDet [16] is studied using several different methods. The *training time* is computed over 25 different runs, each batch of five runs on a different node on DelftBlue [4] to remove biases that might be caused by different jobs running on the shared node. The *number of MACs* is computed for 15 different tensor (video) lengths, ranging from 200 to 3000 in increments of 200. We also calculate the *GPU utilization* of TriDet [16] during inference. Finally, the *memory consumption* and the *inference time* are computed 25 times for each of the 15 different tensor lengths, each batch of five being run on a different node on DelftBlue [4], in order the decrease the influence carried by other jobs running at the same time as ours.

3. Methodology

This section discusses the methodology followed for carrying out the study, both for data efficiency and compute efficiency. For both studies, each subsection presents the algorithm and the metrics that were used.

3.1. Data efficiency

The pseudocode describing the algorithm used for evaluating TriDet's [16] data efficiency can be found in Algorithm 1. The model is trained on subsets of $\mathcal{D}_{\text{train}}$ and it is evaluated on $\mathcal{D}_{\text{test}}$, which represents the original training/evaluation split for the THUMOS'14 dataset as proposed in [7]. The model is trained on different percentages p of the dataset, with $p \in \{10\%, 20\%, 40\%, 60\%, 80\%, 100\%\}$. For each p, the training and evaluation loop is executed 5 times to remove most of the randomness that may occur. For each training/evaluation loop and each p, $\mathcal{D}_{\text{train}}$ is sampled according to the value of p, such that each class in the THUMOS'14 dataset [7] is represented at least once in the sampled dataset $\mathcal{D}_{\text{s}}$. The model is then trained on the sampled dataset $\mathcal{D}_{\text{s}}$ and evaluated on the original $\mathcal{D}_{\text{test}}$. The metric computed during the evaluation step is mean average precision (mAP), calculated for five different temporal intersection-over-union (tIoU) thresholds $tIoU \in \{0.3, 0.4, 0.5, 0.6, 0.7\}$. We report the average mAP, along with the standard deviation for each percentage of the dataset p.

To facilitate extrapolating our results on different datasets, we also report the average number of action instances for each percentage of the dataset p. Given a training set D_{train}, having C distinct classes, N samples and M

Algorithm 1 Data efficiency procedure

$\mathcal{D} = \{(\mathbf{V_i}, \mathbf{y_i})\}_{i=1}^{N}$ ▷ $\mathcal{D}$ is the whole dataset
$\mathcal{D}_{\text{train}}, \mathcal{D}_{\text{test}}$ ▷ Splits taken from THUMOS'14
for $p = 10\%, ..., 100\%$ **do**
  $mAPs \leftarrow$ empty list
  for $i = 1, ..., 5$ **do**
    $\mathcal{D}_{\text{s}} \leftarrow \texttt{sample}(\mathcal{D}_{\text{train}}, p)$ ▷ Samples s.t. $\frac{|\mathcal{D}_{\text{s}}|}{|\mathcal{D}_{\text{train}}|} \cdot 100\% = p$
    Train on $\mathcal{D}_{\text{s}}$
    $mAP \leftarrow \texttt{calculate-mAP}(\mathcal{D}_{\text{test}})$ ▷ Calculate mAP@tIoU[0.3:0.1:0.7]
    Append mAP to $mAPs$
  Report $\texttt{avg}(mAPs)$ and $\texttt{std}(mAPs)$

Algorithm 2 Training performance procedure

$\mathcal{D}_{\text{train}}, \mathcal{D}_{\text{test}}$ ▷ Splits taken from THUMOS'14
$times \leftarrow$ empty list
$mAPs \leftarrow$ empty list
for $i = 1, ..., 5$ **do** ▷ Each iteration is a separate job
  for $i = 1, ..., 5$ **do**
    $time \leftarrow$ Train on $\mathcal{D}_{\text{train}}$
    $mAP \leftarrow$ Evaluate on $\mathcal{D}_{\text{test}}$
    Append $time$ to $times$
    Append mAP to $mAPs$
Report $\texttt{avg}(times)$, $\texttt{std}(times)$, $\texttt{avg}(mAPs)$, $\texttt{std}(mAPs)$

total action instances, we estimate the number of instances per class for each percentage p as shown in Equation 1.

$$\#/\text{class} = \frac{p}{100\%} \cdot \frac{M}{C} \tag{1}$$

We report an estimated number of features per class because the exact number would be influenced strongly by the specific splits D_{s} used during the data efficiency experiment.

3.2. Compute efficiency

To study the compute efficiency of TriDet [16], we measure both the training performance and the inference performance, which are presented in separate subsubsections.

3.2.1 Training performance

The pseudocode describing the algorithm used for measuring TriDet's [16] training time can be found in Algorithm 2. We run five separate jobs on DelftBlue [4], each job training and evaluating the model on the THUMOS'14 [7] dataset five times. Each job is run at a different time of the day on a different node to account for all possible utilization patterns on the cluster. Thus, one job was run in the morning, one in the evening and three in the afternoon.

Algorithm 3 Inference performance procedure

$MACs \leftarrow$ empty list
$inf\_times \leftarrow$ empty list
$memory \leftarrow$ empty list
for $l = [200 : 200 : 3000]$ **do**
  $video \leftarrow$ random tensor with length l
  $macs\_current \leftarrow Compute\ MACs\ for\ video$
  Append $macs\_current$ to $MACs$
for $iter = 1, ..., 5$ **do**
  for $repetition = 1, ..., 5$ **do**
    for $l = [200 : 200 : 3000]$ **do**
      $time \leftarrow \texttt{inference\_time}(l)$
      Append $time$ to inf_times
      $current\_memory \leftarrow \texttt{calc\_memory}(l)$
      Append $current\_memory$ to $memory$
Report MACs for each tensor length
Report $\texttt{avg}(inf\_times)$, $\texttt{std}(inf\_times)$, $\texttt{avg}(memory)$, $\texttt{std}(memory)$ *for each tensor length*

3.2.2 Inference performance

The pseudocode describing the algorithm used for measuring TriDet's [16] inference performance can be found in Algorithm 3. We run one job on DelftBlue [4] for measuring GMACs, and one job for each iteration of the outer for loop for measuring inference time and memory usage. Additionally, TriDet's [16] GPU utilization during inference is measured.

We make use of the *fvcore* [15] library's *FlopCountAnalysis* to measure GMACs. The inference time is measured using Python's *time.time* function, subtracting the starting time from the end time when passing a random tensor to the model. The memory consumption is computed using PyTorch's $max\_memory\_allocated$ function, which returns the maximum memory allocated since starting the program until the call to $max\_memory\_allocated$. Because of this behavior, the counter needs to be reset using $reset\_peak\_memory\_stats$ after each measurement is taken to achieve reliable results. The GPU utilization is measured by *calculating its GMACs/s and divinding it by NVIDIA V100S 32GB's official performance of 8.2 TMACs/s* [2].

4. Experiments

This section presents and discusses the results obtained from the experiments for both data and compute efficiency.

The experiments for this study, as well as my research group's studies, are carried out on the THUMOS'14 dataset [7], by making use of pre-processed I3D [1] features. All of the experiments are carried out on TU Delft's HPC cluster DelftBlue [4], on a single NVIDIA Tesla V100S GPU.

4.1. Data efficiency

The experimental findings are illustrated in Figure 2 and Figure 3. In Figure 3, it is evident that consistent patterns

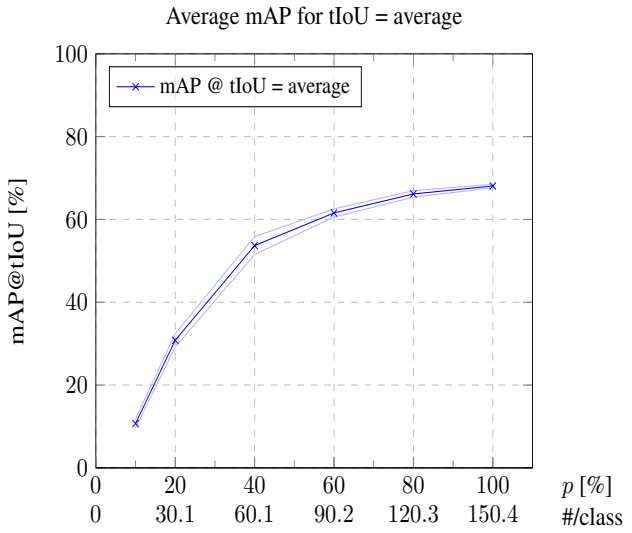

Figure 2. Data efficiency results obtained by TriDet [16] on the THUMOS'14 dataset [7], reported at average tIoU.

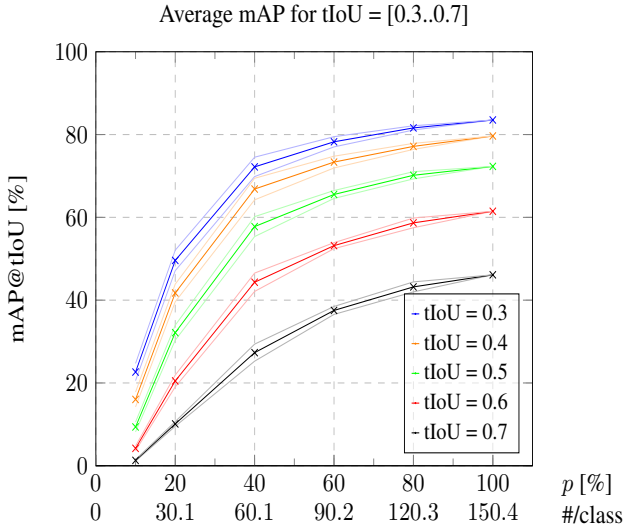

Figure 3. Data efficiency results obtained by TriDet [16] on the THUMOS'14 dataset [7], reported at each tIoU threshold from [0.3 .. 0.7].

emerge across all tIoU values. Notably, these results indicate a decrease in the learning rate when training the model on approximately 60% of the dataset. Moreover, the outcomes suggest the potential for fitting a curve, as detailed in [19], which could facilitate and expedite further research in the field of TAL.

A comparison between the results obtained by our research group can be visualized in Figure 4. Notably, both TriDet [16] and ActionFormer [23] exhibit remarkably similar patterns. TadTR [10] shows a more unique learning pattern, which originates from the model's preprocessing and its ability to use video segments in the training process. This unique approach allows TadTR [10] to achieve results that closely resemble those obtained when utilizing the entire dataset for training, even when employing just 20% of the available training data.

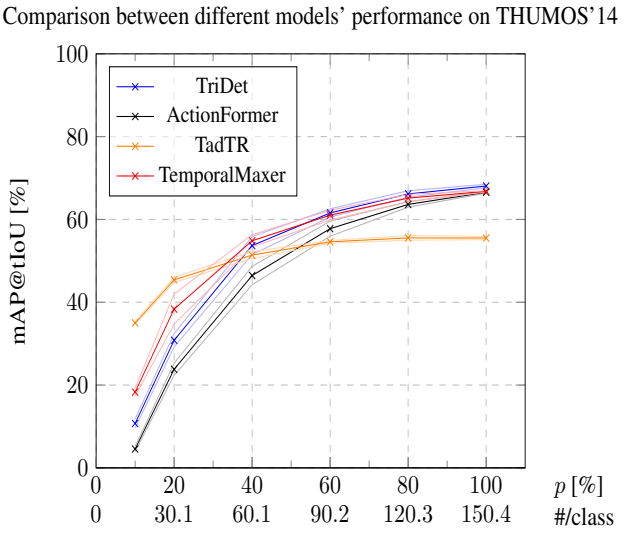

Figure 4. Data efficiency results obtained by the research group's chosen models on the THUMOS'14 dataset [7], reported at average tIoU [11, 14, 20]. Plot taken from [20].

4.2. Compute efficiency

This subsection discusses the results obtained from running the experiments outlined in Algorithm 2 and Algorithm 3. The experimental findings for both training performance and inference performance are discussed in separate subsubsections.

4.2.1 Training performance

Table 1. Training performance of TriDet and other compared models on the THUMOS'14 dataset [7]. Both average training time and obtained average mAP@tIoU=average are reported.

Model	Time [s]	Avg. mAP [%]
TriDet [16]	646.17 ± 26.12	68.07 ± 0.42
ActionFormer [23]	866.22 ± 26.97	66.5 ± 0.31
TadTR [10]	425.72 ± 3.469	55.3 ± 0.63
TemporalMaxer [17]	2955.64 ± 1659.98	66.96 ± 0.37

The results obtained after running the experiment outlined in Algorithm 2 are presented in Table 1. TriDet [16] is compared with ActionFormer [20, 23], TadTR [10, 11] and TemporalMaxer [14, 17], achieving the highest mAP, with a mean training time of roughly 11 minutes.

TriDet [16], ActionFormer [23] and TemporalMaxer [17] were trained and evaluated 25 times, in batches of five,

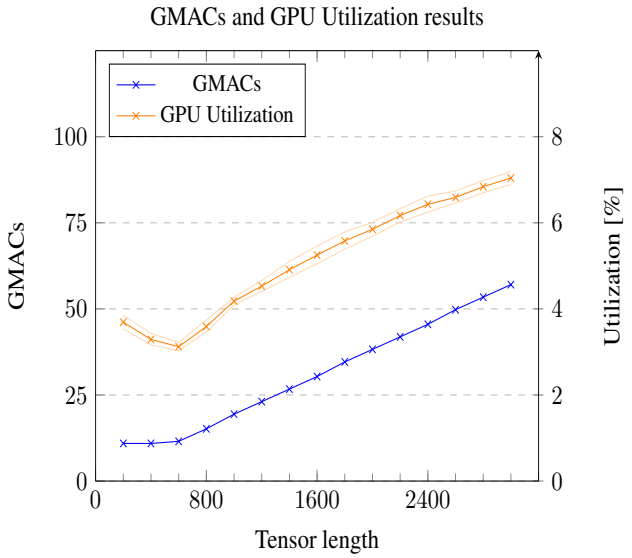

Figure 5. TriDet's [16] number of GMACs produced by running the experiment from Algorithm 3 over random tensors with lengths from 200 to 3000 in increments of 200. Additionally, TriDet's [16] GPU utilization during inference is presented.

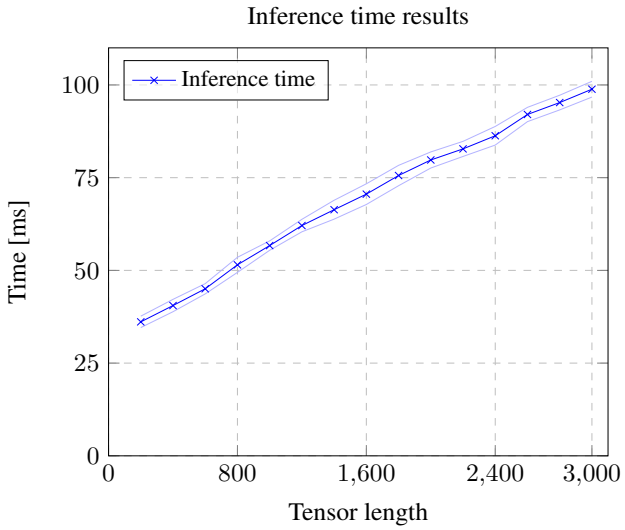

Figure 6. TriDet's [16] inference time when run on random tensors with lengths from 200 to 3000 in increments of 200.

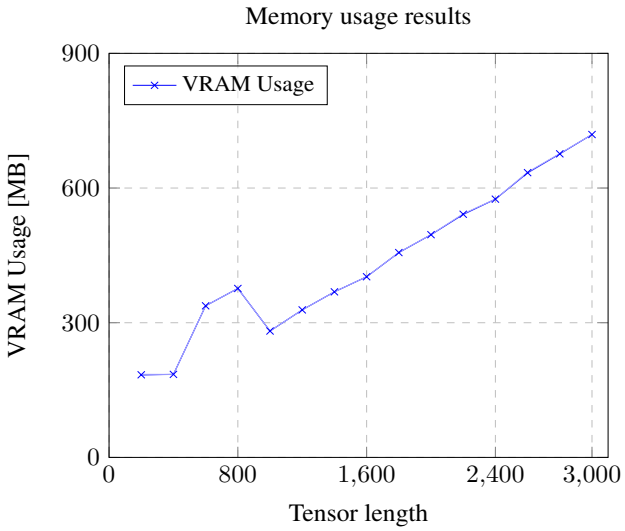

Figure 7. TriDet's [16] memory usage when run on random tensors with lengths from 200 to 3000 in increments of 200.

each batch in a different job submitted on TU Delft's HPC cluster DelftBlue [4], while TadTR [10] was studied on a different GPU, namely an NVIDIA A10. When running the experiment for TriDet [16], the five batches of jobs were run at different times of the day, in order to account for all usage patterns on DelftBlue [4]. One of the batches was run in the morning, one slightly past midnight, and 3 in the afternoon, all in the same day. Additionally, the nodes that they were running on were under different loads, with different jobs running at the same time. It is notable that none of these measures had an influence on the training time for TriDet [16], this being roughly eleven minutes at most times.

It is important to note that while TadTR [10] achieved the lowest mean training time, along with an impressive standard deviation of only 3.4s accross 25 runs in different times and GPU loads [11], the experiments for it were carried out on a different GPU, namely an NVIDIA A10. In practice, in settings where the training time takes priority over acccuracy, TadTR [10] would be the suitable model to use. When the model's accuracy is important, TriDet [16] is the obvious choice, achieving state-of-the-art performance at only slightly higher training times.

4.2.2 Inference performance

The results obtained after carrying out the experiment presented in Algorithm 3 are presented in Figure 5 (MACs), Figure 6 (inference time) and Figure 7 (memory usage).

TriDet [16] originally uses a max_seq_len of 2304, which pads all tensors passed to the model to 2304 units of length. Due to this, running the inference experiments would result in mostly constant scores. The minimum allowed max_seq_len was found to be 576, and the experiments were instead run with this small change to the original model.

With this change, TriDet's [16] GMACs, memory usage, inference time and GPU utilization results are increasing linearly along with increasing tensor lengths passed to the model. An exception to this linear trend is the memory usage for tensor lengths between 600 and 1000, which show an abrupt increase in VRAM usage, followed by a decrease

for tensors of length 1200 and a continuation of the linear trend for higher tensor lengths.

The experiment was run several times, using a max_seq_len of 576 and 1152, and this memory usage pattern for tensor lengths between 600 and 1000 was consistently observed. We then thought that the pattern could be produced by inconsistencies in GPU memory allocation and deallocation, and the experiment was rerun using $torch.cuda.synchronize()$, which adds memory synchronization points in the execution of the proposed experiment. Using this, we can make sure that all past operations finish and all of the used memory is deallocated before starting our next VRAM usage measurement. This, however, showed minimal improvements in VRAM usage, and the unnatural pattern was still appearing.

The next step that should be pursued in identifying the cause of the pattern is to use PyTorch's $profiler$, to see exactly which parts of the model are using the memory. We do not further pursue this problem in this study, due to the time constraints for our project.

The results from Figure 5, 6 and 7 show stable results across different runs, with very little standard deviations. Due to this, the results that we obtained can be used to impose a minimum requirement on the hardware needed for running TriDet [16].

When compared with ActionFormer [20,23], TriDet [16] achieves lower scores in all metrics, which confirms the original authors' claim, stating that the model is more efficient than its competitors. TemporalMaxer [14, 17], however, proves to be even more lightweight than TriDet [16], with only slightly worse performance. Both the results obtained by ActionFormer [20, 23] and TemporalMaxer [14, 17] show a linear increase along with increasing tensor lengths, with ActionFormer [23] showing increases in steps for GMACs, VRAM usage and GPU utilization [20]. This specific pattern occurs due to the model padding tensors to the most suitable multiple of the chosen max_seq_len.

5. Responsible Research

This section discusses the ethical implications of TAL and of this study, along with the steps taken to ensure reproducibility and transparency throughout our study.

Temporal action localization, while offering significant advancements in video analysis and understanding, brings important ethical implications that must be taken into consideration. The deployment of TAL systems in public spaces can raise concerns about surveillance and the potential for misuse or abuse. Safeguards, regulations, and clear guidelines are essential to achieve a balance between the benefits and risks, promoting transparency and accountability in the use of such technology.

This study was conducted with the mentioned implications in mind. TriDet [16] was trained on pre-trained I3D [1] features from the THUMOS'14 [7] dataset. This ensures that the videos and the annotations were not tampered with in any way. Additionally, to ensure reproducibility and transparency, all of the files, code, and results are made available at TUDelft's GitLab server. The folder named *datasets_for_training* contains all of the sampled datasets that were used to extract the presented results. Additionally, the folder named *results_data* contains the raw results for the data efficiency studies, as returned by the original authors' evaluation script, for each training procedure in this study. The results obtained by running the compute efficiency experiments can be found in the *results_data* folder. The scripts used for running the experiments are also available, with *data_efficiency.py* being the python script used, and *data_ef_thumos.sh* being the bash script used to run the data efficiency experiment on TU Delft's HPC Cluster [4]. For compute efficiency, *measure_training.sh* is the script used for measuring training time, *script_macs.sh* is used for counting GMACs, and *script_memory_inftime.sh* runs the five different jobs calculating memory usage and inference time.

6. Conclusion

In temporal action localization, the goal is to derive the action that is present in an input video, along with its temporal boundaries. Throughout the years, several methods have been proposed for this task, with the state-of-the-art being dominated by deep learning models in the recent years.

Although their performance is impressive, these models often require large datasets or immense computational power to achieve their presented performance. Nowadays, authors rarely study their models' behavior under more limited environments, in terms of the available training data or computation power.

This study was carried out on TriDet, a novel temporal action localization model that achieves state-of-the-art performance on two popular benchmarks, while being more efficient than its competitors. This paper aimed at studying TriDet's behavior under limited data and computation power settings.

In terms of limited training data behavior, results show that TriDet achieves close to state-of-the-art performance on the THUMOS'14 dataset, when using merely 60% of the training data available. It is also notable that TriDet was trained on multiple subsets of the data available, and the accuracies obtained after evaluation form a distinguishable curve. These results can be further studied upon, and using this curve, researchers could infer the maximum potential of their models from training only on a few subsets of the training data.

For TriDet's compute efficiency, its training time was found to be at around eleven minutes. The model's inference performance was measured using inference time, multiply-accumulate operations and memory usage, when random tensors of different set lengths between [200:200:3000] were passed to the model. The model's GPU utilization during inference is also reported. Results show that all those metrics scale linearly along with the length of the tensors. These results, combined with my research group's results can later be used to set a hypothetical boundary on the hardware necessary to run novel models.

Although the results that were obtained by the research group are impressive, several other metrics for data efficiency and compute efficiency can be further studied to cement our findings. For data efficiency, studies on transfer learning and few-shot learning capabilities of popular models would provide interesting insights. In terms of compute efficiency, evaluating metrics such as model size, energy consumption and latency could provide interesting results.

Acknowledgments

I would like to express my gratitude to our responsible professor, Dr. Jan van Gemert, and our supervisors, Ombretta Strafforello, Robert-Jan Bruintjes and Attila Lengyel for their support, guidance and valuable feedback throughout the course of the past 10 weeks, which have been instrumental in shaping this project. I would also like to thank my colleagues, Jan, Teo, Paul and Yunhan, for the interesting and insightful discussions we've had throughout the project, and for their constructive feedback. I am extremely grateful for the opportunity to collaborate with all of those mentioned, and they hold an important stake in the successful completion of this project.

References

[1] Joao Carreira and Andrew Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset, 2018. 3, 6

[2] NVIDIA Corporation. Nvida v100 tensor core gpu. https://images.nvidia.com/content/technologies/volta/pdf/volta-v100-datasheet-update-us-1165301-r5.pdf, 2020. (Accessed on 23/06/2023). 3

[3] Dima Damen, Hazel Doughty, Giovanni Maria Farinella, Sanja Fidler, Antonino Furnari, Evangelos Kazakos, Davide Moltisanti, Jonathan Munro, Toby Perrett, Will Price, and Michael Wray. The epic-kitchens dataset: Collection, challenges and baselines. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(11):4125–4141, 2021. 1

[4] Delft High Performance Computing Centre (DHPC). DelftBlue Supercomputer (Phase 1). https://www.tudelft.nl/dhpc/ark:/44463/DelftBluePhase1, 2022. 2, 3, 5, 6

[5] Bernard Ghanem Fabian Caba Heilbron, Victor Escorcia 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

[6] Ahsan Iqbal, Alexander Richard, and Juergen Gall. Enhancing temporal action localization with transfer learning from action recognition. In *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, pages 1533–1540, 2019. 2

[7] Y.-G. Jiang, J. Liu, A. Roshan Zamir, G. Toderici, I. Laptev, M. Shah, and R. Sukthankar. THUMOS challenge: Action recognition with a large number of classes. http://crcv.ucf.edu/THUMOS14/, 2014. 1, 2, 3, 4, 6

[8] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer, 2020. 2

[9] Zhuohan Li, Eric Wallace, Sheng Shen, Kevin Lin, Kurt Keutzer, Dan Klein, and Joseph E. Gonzalez. Train large, then compress: Rethinking model size for efficient training and inference of transformers, 2020. 2

[10] 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. 4, 5

[11] Paul Misterka. *Efficient Temporal Action Localization model development practices*. Bachelor's thesis, Delft University of Technology, 2023. 4, 5

[12] Sauradip Nag, Xiatian Zhu, Yi-Zhe Song, and Tao Xiang. Zero-shot temporal action detection via vision-language prompting, 2022. 1

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

[14] Teodor Oprescu. *TemporalMaxer Performance in the Face of Constraint: A Study in Temporal Action Localization*. Bachelor's thesis, Delft University of Technology, 2023. 4, 6

[15] Facebook Research. fvcore. https://github.com/facebookresearch/fvcore, 2023. 3

[16] 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

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

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

[19] Tom Viering and Marco Loog. The shape of learning curves: a review, 2022. 4

[20] Jan Warchocki. *Benchmarking Data and Computational Efficiency of ActionFormer on Temporal Action Localization Tasks*. Bachelor's thesis, Delft University of Technology, 2023. 4, 6

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

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

[23] Chenlin Zhang, Jianxin Wu, and Yin Li. Actionformer: Localizing moments of actions with transformers, 2022. 2, 4, 6

[24] Hang Zhao, Antonio Torralba, Lorenzo Torresani, and Zhicheng Yan. Hacs: Human action clips and segments dataset for recognition and temporal localization, 2019. 1

[25] Yi Zhu, Xinyu Li, Chunhui Liu, Mohammadreza Zolfaghari, Yuanjun Xiong, Chongruo Wu, Zhi Zhang, Joseph Tighe, R. Manmatha, and Mu Li. A comprehensive study of deep video action recognition, 2020. 1