Efficient Temporal Action Localization model development practices

A review and analysis of models and a guide of best methods

Paul Misterka
Supervisor(s): Jan van Gemert, Robert-Jan Bruintjes
Attila Lengyel, Ombretta Strafforello
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: Paul Misterka
Final project course: CSE3000 Research Project
Thesis committee: Dr. Jan van Gemert, Robert-Jan Bruintjes, Attila Lengyel, Ombretta Strafforello, Dr.-Ing. Petr Kellnhofer

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

Abstract

Temporal Action Localization (TAL) is an important problem in computer vision with uses in video surveillance and recommendation, healthcare, entertainment, and human-computer interaction. Being an inherently data-heavy process, TAL has been bound by the availability of computing power, resulting in its slow pace of innovation. This work aims to accelerate the development of TAL models by conducting a short review of TAL's state-of-the-art, and providing extensive data about the latest models' data and compute efficiency. By researching how TAL models perform in limited data and compute settings, we find that using less data than available is often beneficial to iterating a model quickly, while in some cases, TAL is constrained by the limited amount of data. Finally, we provide general guidelines that create a simple framework for efficient TAL model development.

1. Introduction

Temporal Action Localization (TAL) algorithms and ML models provide insight into actions' type and temporal location in arbitrary videos [5], as shown in figure 1. The popularization of Deep Neural Networks (DNNs) [12], transformers [15], and advances in semiconductor miniaturization have recently paved the way for groundbreaking models like TriDet [13] and ActionFormer [22].

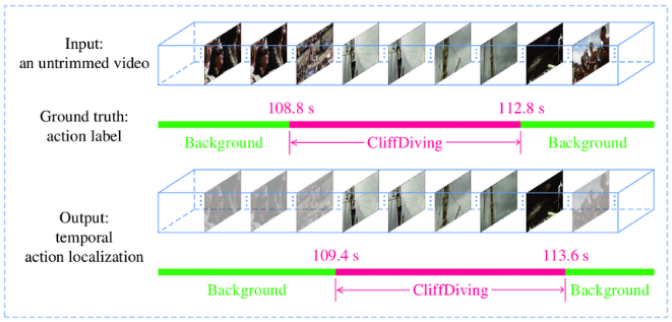
Unfortunately, TAL research is still heavily limited by access to computing power, with the state-of-the-art VideoMaeV2 [18] requiring 300 hours of training on a cluster of 64 cutting-edge A100 GPUs [2]. As the vast majority of researchers don't have access to such resources, they are disadvantaged in contributing to TAL. Furthermore, TAL inference is also very costly, often requiring TMACs (10^{12} multiply-accumulate operations) per action detection and limiting TAL's usefulness despite a wide range of possible applications [20].

To help alleviate these issues, this work explores efficient model development practices for TAL. By comparing past models, we establish a baseline to estimate the performance in limited data and compute settings. Researchers can use our findings to quickly understand if a particular approach is likely to be performant without full training and testing.

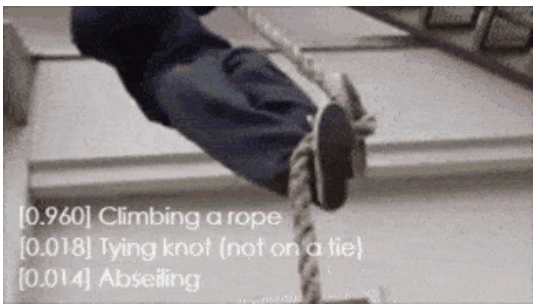
In section 2, we perform a short overview of novel TAL approaches to understand what tools could help estimate compute performance. Subsequently, we take TadTR [7], a modern and lightweight TAL pipeline, as a study case. Section 3 describes how we conduct various experiments on data and compute efficiency. Using that methodology, in section 4, we replicate TadTR results and extensively report on its performance in varying conditions. Section 5 discusses the ethical background of our research and provides an overview of the steps we've taken to ensure fairness, transparency, and reproducibility. Finally, in section 6, we provide an overview of the results and recommendations for efficient TAL model development.

2. Related Work

To understand important TAL nomenclature and previously used techniques to limit data and compute complexities, we take a look at works related to TAL. Subsection 2.1, briefly reviews trends relevant to data and compute efficiency in novel TAL models. Using that knowledge, in subsection 2.2, we summarize previously used techniques that limit the quantity or dimensionality of input data. Lastly, in subsection 2.3, we describe some of the limitations of the most popular TAL benchmark THUMOS14 [5] that impact the data-limiting techniques.



(a) Given an input video, a TAL model labels and locates the action of *cliff diving* and marks its beginning and end [17]. To calculate performance, TAL benchmarks compare the prediction with the ground truth.



(b) TAL makes multiple predictions for the action in a video frame (confidence, action type), and recognizes it as *climbing a rope* with a high degree of confidence (0.960) [8].

Figure 1. The principle of Temporal Action Localization (TAL).

2.1. Relevant TAL trends

Temporal Action Localization models can be split into multistage and single-stage models [22] [7]. In the former, action classification follows temporal classification, producing action predictions based on proposals for the occurrence of actions. In comparison, single-stage models attempt to combine the two, using anchoring windows over the image to identify possible actions freely. While these lack the flexibility of multistage models, the added power of including action detection makes them favored in recent works.

Furthermore, the recent introduction of Vision Transformers (VITs) [13] [22] that rely on a single-shot approach further solidifies the advantages of single-stage methods. In contrast to methods like E2E-TAD [6] or BasicTAD [21], transformer-based models separate the backbone from the rest of the model. Rather than performing training end-to-end (E2E), VITs reuse pre-trained features like I3D [1] or SlowFast [4]. This simplifies the model architecture and accelerates training but slightly reduces accuracy. As approaches based on pre-trained features can be iterated much more quickly, breakthrough TAL methods tend to use them, with E2E improvements following later.

2.2. Data limiting techniques

As TAL is a compute-heavy workload, many approaches limit the amount of data processed by the model, both during training and inference [21] [7]. We can divide these techniques into ones that directly reduce the number of samples and transform the input data to decrease its dimensionality.

One of the most important techniques reducing the number of samples is few-shot learning [10]. Rather than training on the whole dataset, few-shot learning provides a predetermined number of samples for each classification class. As these samples are likely to provide a satisfactory representation of a class, many models achieve comparable performance with few-shot learning while significantly reducing training times. Another important method is anchoring [23], which outputs a set of bounding boxes for predicting the occurrence of actions. Determining the density of anchors is an important parameter that heavily influences raw required power and constraints on hardware specifications (e.g., VRAM).

TAL methods also utilize different ways to compress the input data, reducing its dimensionality and, as a result, the input size. The most prevalent are the pretrained backbones that efficiently store information about action features and convolutional layers that downscale the input videos. However, some also compress videos directly before feeding them into the network or sampling them at different framerates. Finally, few models replace videos with their action detection features completely.

2.3. Dataset limitations

Previously, we mentioned that few-shot learning is an important technique for testing model behavior in limited data settings. However, due to the limitations of the THUMOS14 dataset, we found it impractical to use few-shot learning. This is because, for many classes, the class samples are aggregated into a few videos. As a result, we could only train the model on none or a large number of these samples, preventing gradual increments to the number of *shots*.

We considered splitting videos into multiple videos representing individual occurrences of particular classes. However, we decided against it, as it is essentially a modification to the dataset, making comparisons with previous work problematic. Instead, we settle on a simple percentage-wise division. We sample a percentage of the dataset for certain experiments and train the model on that subset without looking at the underlying class distribution. While less ideal, we predict that averaging these experiments over multiple divisions minimizes uncertainty and proves to have minimal performance variance.

2.4. Study case

As we don't aim to develop a new approach but rather analyze existing ones and provide insights into their efficient development, we choose one of the existing models as a study case and perform experiments on it. Our choice is motivated by accuracy, performance, availability of the source code, reproducibility, quality of documentation, and recency (in the context of trends from subsection 2.1). Analyzing top papers[1], we find TadTR [7] to be the most complete, transparent, and easy to experiment with from the recent, highly-performant TAL models.

TadTR is a transformed-based model with end-to-end training and uses the I3D [1] backbone. It achieves a 56.7% mAP (mean average precision) on the THUMOS14 dataset, lagging behind ActionFormer and autoencoder-based approaches. The model uses anchoring and splits videos from the original dataset into smaller slices. Its behavior is easily configurable through code, and the authors provide a relatively well-documented codebase that allows for timely modifications. Together, these features make TadTR a great study case for accuracy and performance analysis.

3. Method

To ensure the reliability of our results, we establish a carefully crafted methodology for conducting experiments in limited data and compute settings. First, we describe how we reproduce TadTR's authors' results in subsection 3.1. Subsection 3.2 highlights considerations for experiments that research TadTR's training performance for partial training

[1] Using https://paperswithcode.com/dataset/thumos14-1, accessed 2023-06-21

sets. Subsequently, subsection 3.3 describes the methodology and metrics relevant for determining the compute efficiency of TadTR.

3.1. Reproducing TadTR

To keep our result comparable with the base TadTR paper, we use the pretrained backbone, training, and evaluation scripts provided by the authors. Unless explicitly stated in the description of an experiment, we use the default hyperparameters as in table 1, training the model for 15 epochs.

Hyperparam	Value
# of epochs	15
Base learning rate (LR)	0.0002
Weight decay	0.0001
LR decay epoch	14
Batch size	16
Slice overlap	75%

Table 1. List of default hyperparameter values

Furthermore, all experiments are conducted on an A10 LambdaLabs cluster. This virtualized, single-tenancy (i.e. exclusive access) instance provides a 26 vCPU Intel Platinum 8358 processor and a high-performance SSD drive alongside the Nvidia A10 accelerator. The well-documented approach and a stable compute environment guarantee that our results are highly consistent and can be easily reproduced.

3.2. Data efficiency

Once we verify our replication procedure's correctness, we look at TAL models' data efficiency. We establish whether we can extrapolate models' performance using only a significantly smaller subset of the full dataset. Moreover, if one can train the model multiple times on a fraction of a dataset and further project performance based on that data, they can decrease the training time substantially, saving development time and cutting training costs. Additionally, this approach provides a way to predict how much TAL models would benefit from additional training data beyond the available means.

Our approach to testing TadTR's data efficiency relies on training the model on slices of the dataset. As mentioned in subsection 2.3, we take a percentage p ranging from 1% to 100% and randomly pick p% training dataset videos, i.e., slices. For each p, we train TadTR on t=5 random slices and evaluate its *mAP* (mean average precision) for IoU (intersection over union) between 0.3 and 0.7 with 0.1 increments on the THUMOS14 dataset, averaging the precision from each training. Finally, we plot a *p% vs mAP* graph and fit explore the fits of different curves to understand what function shape works best.

3.3. Compute efficiency

Furthermore, we look at the compute efficiency of TadTR. By running such experiments, we hope to identify trends and relations between different compute metrics (time, MACs, utilization) across different ViT TAL models, provide expected compute resource requirements, and establish development practices for classes of models with varying compute requirements. Similarly to subsection 3.2, we provide baseline results for training time, inference time, and theoretical complexity for randomized input tensors with shape [1, 2048, `length`] using Python's `timeit` and Meta's `fvcore` [11] library (specifically the `FlopCountAnalysis` function). Here, the `length` varies between 100 and 30000. Finally, we plot the results and compare them against the different TAL models.

Reusing the training and inference experiment setups, we can easily profile the performance of TAL models. To understand TadTR's bottlenecks, we analyze its CPU (singlethreaded, multithreaded) and GPU performance by sampling the averaged use over the past 0.5s using Python's `psutil.cpu_percent` and Nvidia's `nvidia-smi`. Unfortunately, we cannot profile the memory access (memory bandwidth, cache misses) due to limitations posed by the compute instance's virtualization layer. We also do not measure memory consumption, as the model pre-allocated necessary memory, with over 90% available at all times. Note that in an idle state, the compute instance is extremely efficient, with CPU and GPU usage below 0.1%, and the impact of our usage measurement script is negligible.

4. Experiments

Using the methodology described in section 3, we perform various experiments on TadTR, and compare the results against two other TAL models. First, we provide baseline experiment results for various TAL models in subsection 4.1 to verify the validity of our experimental setup. Then, we perform data efficiency experiments in subsection 4.2 and compute efficiency experiments in subsection 4.3. Together, these provide insight into the behavior of TAL models in limited data and compute settings.

4.1. General experiment results

We replicate TadTR's results using the codebase provided by its authors[2] and achieve an average 55.3% mAP (max 56.7%). This, while ∼1.3% lower than the reported 56.7%, is within the expected range, as TadTR authors report their best result, with our best model achieving 56.9% mAP.

[2] https://github.com/xlliu7/TadTR, accessed 21-06-2023

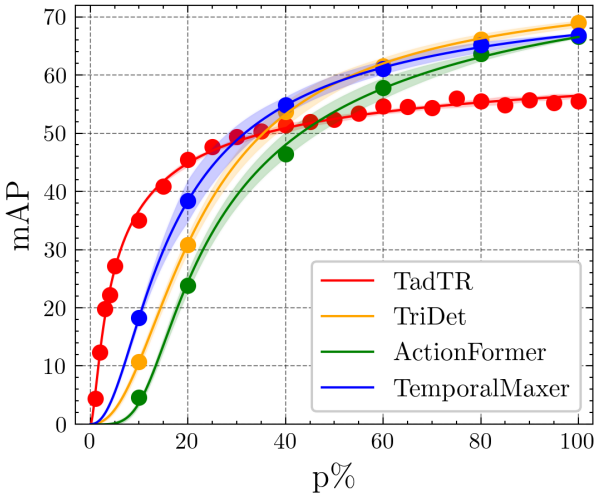


Figure 2. mAP of different ViT TAL models on parts of the THUMOS14 dataset, where p% represents the percentage of the dataset used; shadowed area represents the standard deviation for each datapoint's 3 experiments. As TriDet and TemporalMaxer are based on ActionFormer, their scaling behavior is similar.

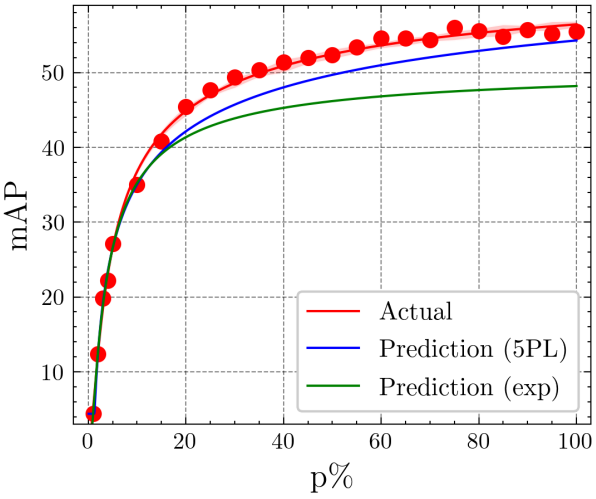


Figure 3. Predicting accuracy with partial data. Training the model on up to 10% of the dataset and extrapolating results gives a good prediction for the actual accuracy.

Model	Training (s)	mAP
TadTR [7]	426 ± 4	$55.3\% \pm 0.6\%$
TriDet [13]	646 ± 26	$68.1\% \pm 0.4\%$
ActionFormer [22]	866 ± 27	$66.5\% \pm 0\%$
TemporalMaxer [14]	2956 ± 1660	$67.0\% \pm 0.4\%$

Table 2. Training times[3] and mAPs for different TAL models on THUMOS14. Each model has been trained 5 times, with the mean values reported (and the standard deviation after $\pm$ sign).

4.2. Data efficiency experiments

Figure 2 shows TadTR's accuracy when trained on parts of the THUMOS14 dataset as described in subsection 3.2. The averaged mean standard deviation of the results from the different experiments for the same p% is ~0.6 mAP, or about 1.1% the respective mAP. Moreover, the curve appears to flatten out at $p = 70\%$, with additional data having minimal effect. Indeed, in this instance, we achieve the best result at $p = 75\%$ of 55.97 mAP. We also successfully replicate the results of three other models, TriDet [3], ActionFormer [19], and TemporalMaxer [9], and compare them against each other.

4.2.1 Extrapolating results

As the dataset scaling of the four compared models has a similar shape, we attempt to fit a curve against the p% vs mAP relationship. Following a recent review of ML learning curves [16], we attempt to find an exponential relationship for log-log normalized data, i.e., where both axes have a logarithmic base.

Unfortunately, the larger the accuracy variance, the less accurate this prediction will be. For TemporalMaxer the mAP variance reaches 10.6% at $p = 20\%$, but for TadTR it is relatively low at an average of 1.1%, and therefore we attempt the fit on TadTR. While the exponential fit on all datapoints is accurate, extrapolating from the first 6 datapoints

$$mAP = 2^{5.67 - 3.546/2^{log2(p)/1.214}}$$

yields a mean mAP difference of 6.05 ± 1.77 (figure 2). At $p = 100\%$, the exponential fit predicts mAP of 48.07, off by 7.13 (13%). With such a large difference, this method is unlikely to be helpful, especially for models that show a higher mAP variance.

More interestingly, a closer inspection of ActionFormer's data efficiency shows that the relationship can't be purely exponential. Rather, at p% below 15%, the mAP gain accelerates, only slowing down at higher p%. We theorize that this behavior is due to the information added by the appearance of new action class types. At very low p% only a few classes are visible, allowing for few correct detections, but as the number of the seen classes increases, the information carried by each new action class type provides more information about the remaining unseen classes. Once the majority of the classes are seen, this effect diminishes, as there are increasingly fewer unseen classes. Incidentally, as TadTR splits videos from the original dataset into subsamples, it will see more action classes at lower p%, making its curve

[3] TriDet, ActionFormer, and TemporalMaxer were trained on a shared-tenancy AMD EPYC 7402 and NVIDIA Tesla V100S 32GB cluster.

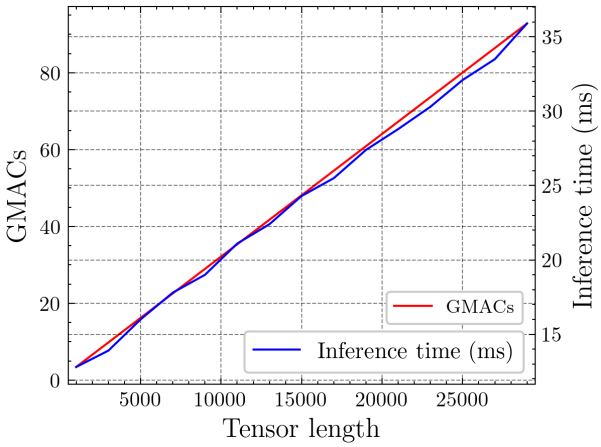


Figure 4. TadTR's inference performance for varying random input tensor lengths. The red line shows theoretical giga-MAC complexity, while the blue one shows inference time (in milliseconds).

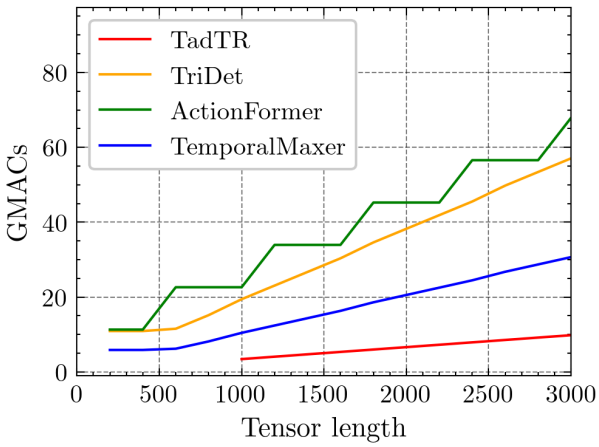


Figure 5. Theoretical inference compute intensity of the different TAL models. The required computer power scales linearly for all models, with TadTR being the most efficient.

closer to a pure exponential form.

4.2.2 Five parameter logistic fit

Instead of an exponential fit, we researched other types of function shapes that could correctly represent the data efficiency curve. While much less popular, we found that the sigmoidal shape of the 5PL (5-parameter logistic regression) fit works well. As seen in figure 2, for TadTR we derive

$$mAP = 76.75 - 72.38/(1 + (x/1.2845)^{49.405})^{0.005437}$$

with a surprisingly low mean mAP difference of 0.566 ± 0.30. Having effectively used just a fifth of the dataset, we can use this method to decrease computational costs for developing and training complex models.

We also perform a 5PL fit against the other researched models on all datapoints with R^2 of 0.9999, 0.9991, and 0.9999 for TriDet, ActionFormer and TemporalMaxer, respectively. As these models scale with additional data well, they would benefit from training on a larger dataset. For example, assuming that TriDet doesn't overfit, the 5PL fit indicates the model would achieve mAP of 74.72% (+7.8%) with a dataset 3 times the size. Despite being anecdotal, this indicates that 5PL could be a good fit for learning curves in ML, and more research into its usefulness could prove invaluable in estimating datasets' robustness.

4.3. Compute efficiency experiments

As seen in figure 4, TadTR's inference performance scales linearly with the size of the input, which is only limited by available VRAM. For a tensor of size $[1, 2048, 30000]$, the inference takes just 35ms, compared to 1s for ActionFormer. In fact, TadTR is so performant the runtime is largely

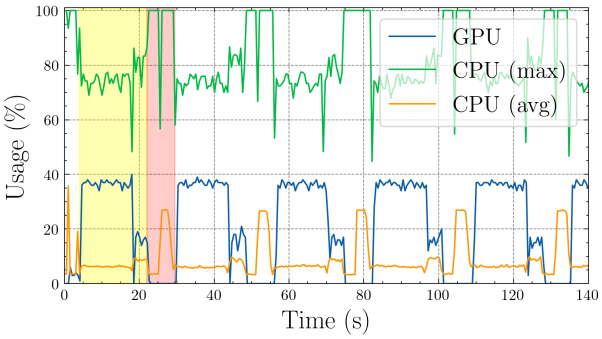


Figure 6. Hardware usage during TadTR's training; the highlighted region represents one epoch. The performance is heavily bottlenecked by the single-threaded CPU performance and memory bandwidth rather than the GPU.

thwarted by the CUDA garbage collector ($\sim 300ms$), and the model can be run on a CPU without dedicated accelerators with similar performance.

Creating the model is also very fast at $\sim$2.5s. Together, the training and inference performance make us determine TadTR a simple model, rather than a complex one like VideoMAEv2. This implies a need for a different set of accuracy and performance optimizations that focus on hyper-optimization through a large number of experiment trials due to the relative ease of such an approach.

Hardware bottlenecks

To understand how the hardware impacts TadTR's performance, we conduct a hardware performance study, looking at the CPU and GPU usage during training. Given its speed and low theoretical GPU intensity, we suspect the training is bottlenecked by CPU performance and memory speed

rather than the GPU. Incidentally, we also evaluate the inference performance, as the model is evaluated (tested) after every epoch. Figure 6 shows the first 140s of the test, with repeating usage patterns for individual epochs - the region highlighted in yellow represents the learning stage for one epoch, while the red region represents the evaluation stage.

Analyzing the usage performance, we find that TadTR indeed is limited mainly by the memory bandwidth rather than the GPU compute capacity itself, which peaks at around 40%. This indicates that a significant quantity or complexity of operations per network stage would not lead to a reduction in performance with GPU idling while waiting for the cache. More interestingly, the CPU usage is also high at around 80%, hitting 100% for most of the inference. The peaks in average use at around 30% occur when 8 of the threads hit 100% usage simultaneously.

The 26 vCPU processor doesn't perform well, with the single-threaded performance limiting the algorithm. Rather than using a server-grade CPU meant for highly parallel workloads, TadTR would benefit from a modern, more lightweight CPU with better single-threaded performance and memory interconnect. We estimate that an Intel i9-13900k or a similar model would slash the training times by 30-40%, allowing for faster development. Additionally, while TadTR would benefit from a more modern GPU architecture (e.g. Nvidia's Hopper), it would not utilize its full potential.

5. Responsible Research

Machine learning models for TAL play a crucial role in various applications, such as video analysis, surveillance, and human activity recognition. Therefore, it is essential to ensure that these models are developed responsibly to mitigate potential risks and biases. In this section, we outline how our paper prioritizes transparency and fairness, and describe ethical considerations.

Reproducability

We emphasize the importance of reproducibility in our approach to machine learning model development for TAL. As some papers do not make it a priority to make results easily reproducible, or include doubtful techniques touted as *score enhancements*, we base our work on a simple, quick, and transparent TadTR. To make our results easily comparable with TadTR, we strive to reuse their experimental setup as much as possible, and fill in blanks in their documentation. Finally, we release our source code and provide instructions on how to run our experiments.

Ethical Considerations

TAL has a number of potentially malicious applications, particularly in mass surveillance. While we understand that our research can accelerate harmful practices, we believe that the potential benefits (such as public safety, healthcare, and video analysis) outweigh the harm. Moreover, we are dedicated to evangelizing the responsible use of TAL in real-world applications and strongly encourage anyone using these technologies to adhere to ethical guidelines and consider the potential negative consequences.

6. Conclusion

Lastly, we offer recommendations for efficient TAL model development practices. While these guidelines are meant for researchers working on improving TAL's state-of-the-art, most of them are transferable to other types of ML problems, especially in the field of computer vision. Altogether, we present five main points describing how to make informed decisions on data, compute, and software engineering issues.

Take the right amount of data

The data efficiency experiments showed that different models scale with data differently. In TadTR's case, training on just 50% of the dataset gives a good insight into its potential on the full dataset. While the model trains quickly, researchers should consider training large, expensive models on a fraction of the dataset when consistently improving your method. This can save both money and development time without significantly impairing the research efforts.

Predict trends

Extrapolating data efficiency is a great tool in determining if a model needs more data and the type of data it works with the best. Researchers can also use it to estimate compute performance on large amounts of data and find reasonable use cases. While we saw that understanding trends can be difficult in cases where additional data is costly to acquire or experimenting is expensive, predicting the model's behavior serves as a good indicator of the best course of action.

Understand your model's performance class

Different ML models require vastly varying amounts of compute power, impacting not only their usefulness but also the speed of development. Having a clear picture of the performance class allows researchers to prioritize their development efforts; for example, while hyperparameter optimization studies might be nearly impossible for a costly model like VideoMAEv2, it can be useful in improving the performance of models that can be trained hundreds of times a day on a single machine. Moreover, sharing the performance specifics of your model with the open-source community is crucial to well-formed collaboration. Consider documenting the training time and inference speed to entice

other researchers to experiment with your models.

Use the right hardware

Machine learning is well-known to be constrained by the availability of computing power, in particular, the GPU. However, researchers can't forget that other parts of the system also have a huge impact on a model's performance, with TadTR largely bottlenecked by the CPU. We urge to consider what hardware fits the model best to improve development efficiency, maximize performance in real-world applications, and help other researchers estimate the feasibility of running a particular model.

Make it easy to iterate

Lastly, we find it paramount to build a model's codebase in a way that allows for quick iteration. We propose a set of simple rules that TAL models should follow:

- Provide well-documented, top-level functions for the model's training, evaluation, and testing. These functions serve as reusable building blocks of your model and an interface allowing quick experimentation
- Document the setup and environment. As the hardware and virtual environment to run a TAL model are likely to be complex, provide a detailed description of how to reproduce and experiment with your results.
- Embed configuration and logging from the start. As researchers are likely to run dozens of quickly evolving experiments, understanding what happens during experiments and reproducing them saves time.
- Avoid over-documenting. As the TAL approaches evolve quickly, the codebase will likely change rapidly. While top-level documentation is crucial, restrain from perfecting every part of the codebase.

Altogether, these guidelines help avoid and recover from mistakes while making it easy to evolve the researched TAL approach rapidly.

References

[1] João Carreira and Andrew Zisserman. Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset. *CoRR*, abs/1705.07750, 2017. 2

[2] Jack Choquette, Wishwesh Gandhi, Olivier Giroux, Nick Stam, and Ronny Krashinsky. NVIDIA A100 Tensor Core GPU: Performance and Innovation. *IEEE Micro*, 41(2):29–35, 2021. 1

[3] Alex Dămăcuş. *Efficient Video Action Recognition*. Bachelor's thesis, Delft University of Technology, 2023. 4

[4] Haoqi Fan, Yanghao Li, Bo Xiong, Wan-Yen Lo, and Christoph Feichtenhofer. PySlowFast. https://github.com/facebookresearch/slowfast, 2020. 2

[5] 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

[6] Xiaolong Liu, Song Bai, and Xiang Bai. An Empirical Study of End-to-end Temporal Action Detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 20010–20019, 2022. 2

[7] 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 (TIP)*, 2022. 1, 2, 4

[8] MMAction2 Contributors. OpenMMLab's Next Generation Video Understanding Toolbox and Benchmark. https://github.com/open-mmlab/mmaction2, 2020. 1

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

[10] Archit Parnami and Minwoo Lee. Learning from Few Examples: A Summary of Approaches to Few-Shot Learning, 2022. 2

[11] Facebook Research. Fvcore. https://github.com/facebookresearch/fvcore, 2023. (Accessed on 04/06/2023). 3

[12] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, jan 2015. 1

[13] Dingfeng Shi, Yujie Zhong, Qiong Cao, Lin Ma, Jia Li, and Dacheng Tao. TriDet: Temporal Action Detection with Relative Boundary Modeling, 2023. 1, 2, 4

[14] 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. 4

[15] 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

[16] Tom Viering and Marco Loog. The Shape of Learning Curves: a Review, 2022. 4

[17] Le Wang, Xuhuan Duan, Qilin Zhang, Zhenxing Id, Gang Hua, and Nanning Zheng. Segment-Tube: Spatio-Temporal Action Localization in Untrimmed Videos with Per-Frame Segmentation. *Sensors*, 18, 05 2018. 1

[18] Limin Wang, Bingkun Huang, Zhiyu Zhao, Zhan Tong, Yinan He, Yi Wang, Yali Wang, and Yu Qiao. VideoMAE V2: Scaling Video Masked Autoencoders with Dual Masking, 2023. 1

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

[20] Huifen Xia and Yongzhao Zhan. A Survey on Temporal Action Localization. *IEEE Access*, PP:1–1, 04 2020. 1

[21] Min Yang, Guo Chen, Yin-Dong Zheng, Tong Lu, and Limin Wang. BasicTAD: an Astounding RGB-Only Baseline for Temporal Action Detection, 2023. 2

[22] Chenlin Zhang, Jianxin Wu, and Yin Li. ActionFormer: Localizing Moments of Actions with Transformers, 2022. 1, 2, 4

[23] Shifeng Zhang, Cheng Chi, Yongqiang Yao, Zhen Lei, and Stan Z. Li. Bridging the Gap Between Anchor-based and Anchor-free Detection via Adaptive Training Sample Selection, 2020. 2