

23 Jun 2023

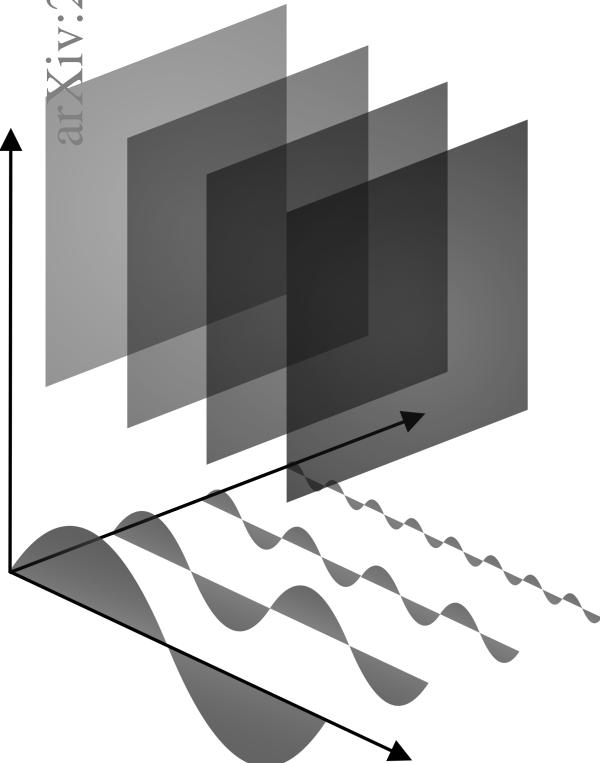


AARHUS
UNIVERSITY

PHD DISSERTATION

Efficient Online Processing with Deep Neural Networks

by Lukas Hedegaard



arXiv:2306.13474v1 [cs.LG]

PHD DISSERTATION

Efficient Online Processing with Deep Neural Networks

Author:
Lukas Hedegaard

Supervisor:
Alexandros Iosifidis



*A thesis submitted in fulfillment of the requirements
for the degree of Doctor of Philosophy*

in the faculty of

Technical Sciences

Signal Processing and Machine Learning
Department of Electrical and Computer Engineering
Aarhus University, Denmark

June 2, 2023

Abstract

The capabilities and adoption of deep neural networks (DNNs) grow at an exhilarating pace: Vision models accurately classify human actions in videos and identify cancerous tissue in medical scans as precisely than human experts; generative models create beautiful artistic renderings based on simple textual prompts; large language models answer wide-ranging questions, generate code, and write prose, becoming the topic of everyday dinner-table conversations. Even though their uses are exhilarating, the continually increasing model sizes and computational complexities have a dark side. The economic cost and negative environmental externalities of training and serving models is in evident disharmony with financial viability and climate action goals.

Instead of pursuing yet another increase in predictive or generative performance, this dissertation is dedicated to the improvement of neural network efficiency. Specifically, a core contribution addresses the efficiency aspects during online inference, where the system must make an updated prediction swiftly after new input information is received, without waiting for subsequent data. Here, the concept of *Continual Inference Networks* (CINs) is proposed and explored across four publications. CINs extend prior state-of-the-art methods developed for offline processing of spatio-temporal data and reuse their pre-trained weights, improving their online processing efficiency by an order of magnitude. These advances are attained through a bottom-up reorganization of computational components and judicious architectural modifications. The benefit to online inference is demonstrated by reformulating several widely used network architectures into CINs, including 3D Convolutional Neral Networks, Spatio-temporal Graph Convolutional Networks, and Transformer Encoders. An orthogonal contribution tackles the concurrent adaptation and computational acceleration of a large source model into multiple lightweight derived models. Drawing on fusible adapter networks and structured pruning, *Structured Pruning Adapters* achieve superior predictive accuracy under aggressive pruning using significantly fewer learned weights compared to fine-tuning with pruning.

This dissertation and the contributions outlined herein constitute necessary steps towards more efficient use of DNNs in scenarios of task specialization as well as online operation. All research-code and supporting Python libraries have been open-sourced at <https://github.com/lukashedegaard> in the hope that they will be put to good use and spur additional research into the efficiency aspects of DNNs.

Resumé

Dybe neurale netværksmodellers formåen og udbredelse gror med hastigige skridt: Visuelle modeller kan klassificere menneskelige handlinger på videoer og identificerer kraftramt væv på medicinske scanninger lige så præcist som menneskelige eksperter; generative modeller skaber kunstfærdige visuelle renderinger ud fra simple tekstuelle beskrivelser; store sprogmodeller besvarer vidtfavnende spørgsmål, genererer kode, skriver prosa, og er blevet et alment samtaaleemne omkring middagsbordet. Selvom anvendelsesmulighederne er utroligt spændende, har de stødt stigende modelstørrelser og -beregningskompleksiteter en skyggeside. De voksende økonomiske udgifter og negative klimaeksternaliteter ved træning og anvendelse af store neurale modeller ligger i tydelig disharmoni til finansiell rentabilitet og globale klimamål.

I stedet for at forfølge endnu en forbedring i prædiktiv eller generativ ydeevne, er denne afhandling dedikeret til effektiviseringen af neurale netværk. Et kernebidrag i afhandlingen omhandler effektivitetsaspekter i online anvendelsesscenarier, hvor et system forventes af give opdaterede prædiktioner umiddelbart efter ny input information er modtaget, uden at vente på yderligere data. Konceptet *Continual Inference Network* (CIN) blev udviklet til denne kontekst på tværs af fire publikationer. CIN'er genbruger netværksarkitekturen og -vægte, der originalt blev trænede til offline bearbejdning af flerdimensionelle tidsvarierende data, og accelererer deres online effektivitet mangefold via reorganiseringer af grundlæggende beregningssekvenser samt minutiøse netværksarkitektoniske justeringer. Fordelene under online processering blev demonstreret ved konvertering af adskillige vidt anvendte netværksarkitekturen til CIN'er, herunder 3D CNN'er, ST-GCN'er og Transformer Encodere.

Et yderlige bidrag takler kombinationen af netværksacceleration og -tilpasning af store basismodeller til flere små specialiserede netværk, der udfører afledte prædiktive opgaver. Ved at trække på adapternetværk, hvis vægte kan fusioneres med hovednetværket, samt strukturerede pruning metoder, opnår *Structured Pruning Adapters* overlegen prædiktiv performance med væsentligt færre lærte vægte end fine-tuning under aggressiv pruning.

Denne afhandling og dens associerede videnskabelige udgivelser udgør nødvendige skridt i retning af mere effektive dybe neurale netværk i scenarier med opgavespecialisering og online processering. Al forskningskode og supporterende Python biblioteker er frit tilgængelige på <https://github.com/lukashedegaard> i håbet om, at de vil finde god anvendelse og inspirere til fremtidig forskning inden for dybe neurale netværks effektivitetsaspekter.

Acknowledgments

First and foremost, I would like to thank my supervisor, Professor Alexandros Iosifidis, for his mentorship and trust throughout my studies, as well as for suggesting that I pursue a PhD degree in the first place. I would also like to thank my co-authors and colleagues at Aarhus University and the Machine Learning and Computational Intelligence group for shaping the positive, informal, and inclusive atmosphere that made the work so enjoyable.

We and our work are the product of our circumstances. I have always had the unwavering support of my father, Birger, and late mother, Wioletta, with whom I spent countless hours doing homework at the kitchen table while my father tended to the farm. Any words I can muster are insufficient in describing the gratitude I feel for their unconditional love and belief in me. Thanks also to my brother, Adam, the extended family on my wife's side, as well as my close friends, Morten, Kristian, Omar, and Alexander, for their support, interest, and the talks we have had these past years. To the love of my life, Ninna, and our son, Arthur, I give thanks for constantly reminding me that life has more to offer than work and the pursuit of abstract goals. As this treatise builds on prior work, so the people listed above have built me up. Thank you all.

*Lukas Hedegaard,
Aarhus, June 26, 2023.*

Preface

This PhD dissertation is written in accordance with the rules and regulations of the Graduate School of Technical Sciences [138]. Following §11.1, this dissertation is a composition of publications and manuscripts relating to the PhD project alongside sections introducing these publications. Accordingly, the introduction are given in Part I and the publications are re-printed in Part II. Except for custom color identification on the header, the publications are identical to those published in respective journals, conferences, books, and as pre-prints.

The works comprised within the dissertation received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 871449 (OpenDR).

The dissertation was written without the use of large language models and typeset in L^AT_EX.

List of publications

The dissertation is based on the following publications:

- [1] **Lukas Hedegaard** and Alexandros Iosifidis. “Continual 3D Convolutional Neural Networks for Real-time Processing of Videos”. In: Avidan, S., Brostow, G., Cissé, M., Farinella, G.M., Hassner, T. (eds) Computer Vision – ECCV 2022. ECCV 2022. Lecture Notes in Computer Science, vol 13664. Springer, Cham. https://doi.org/10.1007/978-3-031-19772-7_22 [72].
- [2] **Lukas Hedegaard**, Negar Heidari, and Alexandros Iosifidis. “Continual Spatio-Temporal Graph Convolutional Networks”. Pattern Recognition, Volume 140, 2023, 109528, ISSN 0031-3203 [79].
- [3] **Lukas Hedegaard**, Arian Bakhtiarnia, and Alexandros Iosifidis. “Continual Transformers: Redundancy-Free Attention for Online Inference”. In International Conference on Learning Representations (ICLR), 2023 [78].
- [4] **Lukas Hedegaard** and Alexandros Iosifidis. “Continual Inference: A Library for Efficient Online Inference with Deep Neural Networks in PyTorch”. In Computer Vision – ECCV 2022 Workshops. Lecture Notes in Computer Science, vol 13803. Springer. ISBN: 978-3-031-25066-8 [73].
- [5] **Lukas Hedegaard**, Negar Heidari, and Alexandros Iosifidis. “Human Activity Recognition” in Deep Learning for Robot Perception and Cognition, A. Iosifidis and A. Tefas, Eds., 1st ed., Academic Press, Jan. 2022, ch. 14, isbn: 9780323857871 [76].
- [6] Negar Heidari, **Lukas Hedegaard**, and Alexandros Iosifidis. “Graph Convolutional Networks”. In Deep Learning for Robot Perception and Cognition, A. Iosifidis and A. Tefas, Eds., 1st ed., Academic Press, Jan. 2022, ch. 4, isbn: 9780323857871 [82].
- [7] **Lukas Hedegaard**, Aman Alok, Juby Jose, and Alexandros Iosifidis “Structured Pruning Adapters”. ArXiv preprint arXiv:2211.10155, 2022 [75].

The following publication was published during the Ph.D. but is not part of the dissertation:

- **Lukas Hedegaard**, Omar Ali Sheikh-Omar, and Alexandros Iosifidis. "Supervised Domain Adaptation: A Graph Embedding Perspective and a Rectified Experimental Protocol," in IEEE Transactions on Image Processing, vol. 30, pp. 8619-8631, 2021, doi: 10.1109/TIP.2021.3118978 [74].

Likewise, the following software libraries were developed during the Ph.D. but will not be covered as part of the dissertation:

- **Ride**: Training wheels, side rails, and helicopter parent for your Deep Learning projects in PyTorch [70].
<https://github.com/lukashedegaard/ride>
- **Co-Rider**: Tiny configuration library tailored for the Ride ecosystem [69].
<https://github.com/lukashedegaard/co-rider>
- **OpenDR**: A modular, open and non-proprietary toolkit for core robotic functionalities by harnessing deep learning [144].
<https://github.com/opendr-eu/opendr>
- **DatasetOps**: Fluent dataset operations, compatible with your favorite libraries [77].
<https://github.com/lukashedegaard/datasetops>
- **PyTorch Benchmark**: Easily benchmark PyTorch model FLOPs, latency, throughput, allocated gpu memory and energy consumption [71].
<https://github.com/lukashedegaard/pytorch-benchmark>
- **Supers**: Call a function in all superclasses using supers(self).foo(42) [68].
<https://github.com/lukashedegaard/supers>

Contents

Abstract	i
Resumé	ii
Acknowledgments	iii
List of publications	v
Contents	vii
I Overview	1
1 Introduction	2
1.1 The triumph of Deep Learning	2
1.2 The burden of Deep Neural Networks	3
1.3 A pursuit of efficiency	5
2 Quantifying efficiency	7
2.1 Task efficiency	7
2.2 Storage efficiency	7
2.3 Computational efficiency	8
2.4 Efficiency is a trade-off	9
3 Approaches for enhancing inference efficiency	10
3.1 Pruning methods	10
3.2 Knowledge distillation	11
3.3 Quantization	12
3.4 Low-rank factorization of weights	13
3.5 Efficient architectures	13
3.6 Caching and temporal redistribution of features	14
3.7 Efficient run-times	14
4 Redistributing computations through time with Continual Inference Networks	15

4.1	Introduction	15
4.2	Related works	17
4.3	Continual Inference Networks	18
4.4	Continual 3D Convolutional Neural Networks	22
4.5	Continual Spatio-temporal Graph Convolutional Networks	26
4.6	Continual Transformers	30
4.7	A Python library to support implementation	37
4.8	Conclusion	39
5	Compressing and accelerating derived networks with Structured Pruning Adapters	40
5.1	Introduction	40
5.2	Related works	41
5.3	Structured Pruning Adapters	42
5.4	Experiments	43
5.5	Conclusion	44
6	Conclusions and future work	46
	Bibliography	48
	II Publications	1
1	Continual 3D Convolutional Neural Networks for Real-time Processing of Videos	1
2	Continual Spatio-Temporal Graph Convolutional Networks	2
3	Continual Transformers: Redundancy-Free Attention for Online Inference	3
4	Continual Inference: A Library for Efficient Online Inference with Deep Neural Networks in PyTorch	4
5	Human Activity Recognition	5
6	Graph Neural Networks	6
7	Structured Pruning Adapters	7

Part I

Overview

Chapter 1

Introduction

1.1 The triumph of Deep Learning

Neural networks have come a long way since McCulloch and Pitts conceived the perceptron in 1943 [125] and Rosenblatt subsequently implemented the first prototype in 1958 [167]. While Rosenblatt had a clear vision that computers would one day see and understand language, the artificial intelligence (AI) field has undergone multiple upswings interspersed with troughs of disillusionment (*AI winters*) [169]. Fast-forward to the 2010’s and 2020’s, deep neural networks (DNNs) are broadly studied and used in universities and companies worldwide, performing many cognitive tasks exclusively reserved for humans but a few years earlier.

In computer vision, a research group at the University of Toronto developed AlexNet [108], a convolutional neural network (CNN), that achieved a 15.3% top-5 error rate in the challenging ILSVRC-2012 competition (also known as ImageNet-1k), wherein the algorithm must predict among 1,000 object categories on a test set of 100,000 yet unseen images. Only four years later, ResNets [65] achieved human-level performance of five percent top-5 error on the benchmark by scaling the network to hundreds of layers. DNNs are not limited to visual perception but can also be trained to generate images of faces and natural-world objects as showcased by Generative Adversarial Nets in 2014 [59]. Later, DALL-E [161], Latent Diffusion Models [157], and Imagen [170] brought textual prompt-based photo-realistic scene generation into the mainstream.

In 2017, Transformers [199] ushered in a similar architectural revolution in the domain of natural language processing (NLP) as AlexNet did in computer vision. The BERT model [39], which utilized the Transformer architecture, achieved large-margin improvements on multiple NLP tasks through the use of masked language modeling. Large language models (LLMs) were subsequently scaled to over one billion (GPT-2 [156]) and one trillion (Switch Transformer [51]) parameters. In 2022, ChatGPT [139] made LLMs part of the dinner table conversion by providing a chat experience with a seemingly all-knowing conversational partner who could also assist in editing, summarizing, and translating text. Despite occasionally fabricating

answers and lacking proper disclosure of informational sources [2], some speculate it a possible contender to the PageRank-based search engine `google.com` [140].

Deep learning also made headlines after AlphaGo [183], a DNN trained with a mix of human supervision and reinforcement learning from games with self-play, beat Go champion Leo Sedoll in 2016. The next year, AlphaZero [184] attained superhuman performance in the games of chess, shogi, and Go within only 24 hours of training using tabula rasa reinforcement learning with self-play. DeepMind, the creators of AlphaGo and AlphaZero, also attracted attention when AlphaFold [96] achieved a quantum leap in the speed and precision of protein structure prediction in the CASP14 challenge in 2020. Touted as the possibly most important scientific achievement of AI in history [192], AlphaFold has generated a database of over 200 million protein structure predictions that will empower future research in medicine and bioengineering.

1.2 The burden of Deep Neural Networks

The larger the capacity of a Deep Neural Network (DNN), the more abstract and fine-grained distinctions and reasoning processes can be modeled. *Overparameterization* of DNNs, *i.e.*, the use of significantly more network parameters than training data to fit, has been repeatedly demonstrated to possess favorable learning characteristics such as lower generalization error and robustness to over-training despite perfectly fitting the training data [5, 135, 186]. By this logic, networks are continually scaled up in pursuit of higher accuracy and modeling capability. However, this often happens with a disregard to the expense and environmental impact of doing so. Moreover, the continual up-scaling provides diminishing returns with an evident limit in sight [191]. Consider, for instance, an over-parameterized network optimized to reduce the root mean squared (RMS) prediction error of unseen data given n data points for training. The training cost of a network scales multiplicatively with the data in the training set and the parameter count of the model. If we keep the over-parameterization factor o constant¹, and assume the cost of processing a single data-point c scales $c \propto o \cdot n$, we can provide a rough estimate of the cost of training a model:

$$\text{cost} \propto \underbrace{o \cdot n \cdot n \cdot e \cdot h}_c, \quad (1.1)$$

where e denotes the number of training epochs and h is the number of tested hyper-parameter configurations. Following statistical learning theory, the RMS prediction error cannot not drop more than $1/\sqrt{n}$ [118, 191]. Equivalently, at least a quadratic increase in available data, $n \rightarrow n^2$, is required to reduce the error by half. Accordingly, the training cost increases from $n^2 \cdot o \cdot e \cdot h \rightarrow n^4 \cdot o \cdot e \cdot h$. Note that this cost estimate is simplified and does not take into account learning strategies such as early stopping nor the dynamics of the double descent phenomenon and the complex interactions

¹E.g., $o = 250\times$ as found for the 296M parameter VOLO-D5 model [222] compared to 1.2M data points in the ILSVRC dataset [168].

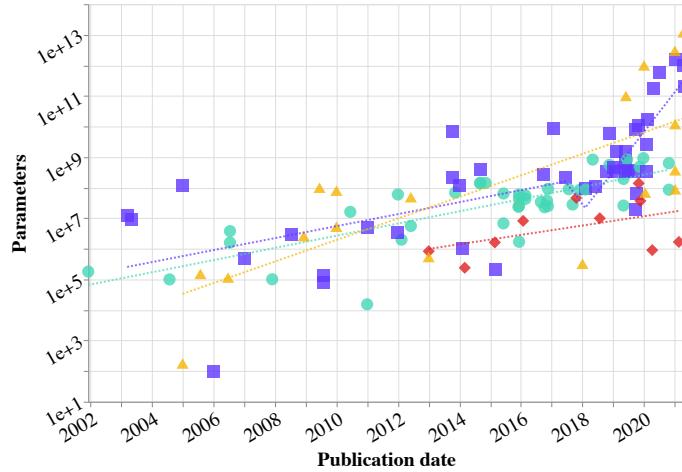


Figure 1.1: Parameter count of prominent deep learning DNNs from 2002 to 2022 for vision (●), language (■), game (◆), and other (▲) models. Data source: [175].

between training data volume, model capacity, and training time [5, 10, 56, 135]. Still, the outlook to a such a scaling of training cost should be a concern. Judging by the parameter count of the largest DNNs from the year 2002 to 2022, visualized in Fig. 1.1, we follow the up-scaling trend with a rapid, even increasing pace, especially in the language-domain.

Continuing this trend would not only be economically infeasible for the organizations training and utilizing the models, but additionally comes with the externality of increasing load on our collective computational infrastructure and energy supply. By extension, the trend comes at a cost to our environment and climate. For instance, the training of BLOOM was reported to use more than one million NVIDIA A100 GPU hours, and GPT-3 incurred an estimated 502 tons of CO₂ equivalent emissions [173], the equivalent of driving three million kilometers in a diesel car. Furthermore, this cost creates a democratic divide between those who cannot afford the expense and the largest commercial actors who are increasingly choosing not to open-source their work.

Are these immense trained models worth the computational cost? As described in Section 1.1, DNNs can achieve astounding performance on tasks that were previously reserved for humans and even go beyond human capability. Once trained, the model can be widely deployed to create value for the users. For instance, ChatGPT reached 100 million users within two months of its launch [1]. However, using approximately 190,000 GPUs hours on NVIDIA A100 GPUs to serve users each day [153], even training cost becomes less important than increased inference efficiency. Moreover, AlphaGo may have bested Lee Sedol in Go, but while AlphaGo required an estimated one megawatt of power, Lee Sedol's human brain uses only 20 watts - that is 50,000× fewer watts [124]. Clearly, nature shows that there is room for improvement.

1.3 A pursuit of efficiency

The encompassing theme of *efficient* deep neural networks contains a plethora of potential research directions and improvement opportunities. These include efficient training methods [120, 127], hyperparameter selection approaches [12], transfer learning [74, 89], and efficient hardware [23, 217]. Within this PhD dissertation, the work towards efficiency is anchored in the algorithmic and architectural aspects of DNNs with the goal of improving the computational and memory efficiency during online processing.

Fully autonomous cars and robots interacting with humans have been frequently imagined in popular culture movies for many years. An ongoing challenge in realizing these visions is the required speed and computational capacity of the computing devices empowering such entities. To get there, it takes significant research efforts in making visual perception algorithms more computationally efficient and capable of real-time processing under recourse-constrained settings. As an effort in this direction, Chapter 4 presents a line of works on online processing of continual input streams. It is based on the observation that many prior state-of-the-art architectures for time-series processing, such as 3D Convolutional Neural Networks [21, 52], Spatio-temporal Graph Neural Networks [179, 213], and Transformers [199], exhibit significant computational redundancy during online inference. The associated research question is:

RQ1 *How can we accelerate state-of-the-art deep neural networks, which were tailored for offline batch-processing of time-series data, to perform online stream processing efficiently?*

The concept of *Continual Inference Networks*, proposed and refined in papers 1 to 4, is introduced as a solution to this. These works feature bottom-up reformulations of the computational sequence of multiple DNN building blocks, alongside guidelines and best practices for implementing architectures that operate efficiently in the stream-processing setting. An overview of these works is given in Chapter 4.

A second line of work lies in the intersection of computational and storage efficiency. The disk-space associated with the execution and training of DNNs is often overlooked in favor of ever-more exhilarating network capabilities and predictive performance. Yet, a possible scenario for the future is a convergence to a smaller set of large *foundation models* forming the base for a large number of derived networks. If each derived model has the same number of parameters as its base model, it poses both an issue during long-term storage as well as during model transfer and serving: The electricity cost of transferring data is easily overlooked despite posing a non-trivial burden on our infrastructure and environment. Arguably, derived tasks should not impose the same run-time cost as the foundation model, either. Since the solved task is most likely simpler than that of the base model, should it not be able to solve it using less computational resources? In particular, the studied research question is:

RQ2 *How can we make parameter-efficient adaptations to pre-trained models while improving their computational efficiency on derived tasks?*

Structured Pruning Adapters (SPAs) were proposed in paper 7 to improve the attainable trade-offs between storage and computational efficiency of derived models. Specifically, SPAs combine parallel fusible Adapters [163] with structured pruning, which is described in Section 3.1, to provide derived models with an order of magnitude fewer learned parameters than fine-tuning with pruning while achieving significantly accelerated inference compared to the base models. The work on Structured Pruning Adapters is described in Chapter 5.

The computational efficiency aspects of DNNs could be studied alongside any task and task performance metric of interest. In this dissertation, the tasks of *human activity recognition* and *image classification* are used to gauge algorithmic efficiency improvements in Chapters 4 and 5, respectively. Publication 5 covers human activity recognition and reviews prior works on both video-based and skeleton-based recognition. Publication 6 covers graph theory and graph convolutional networks, which form the basis of skeleton-based recognition methods.

Before delving into the scientific contributions in Chapter 4 and 5, an introduction to the metrics that quantify efficiency is provided in Chapter 2, followed by a brief overview of prior approaches for enhancing the inference efficiency of DNNs in Chapter 3.

Chapter 2

Quantifying efficiency

2.1 Task efficiency

The majority of prior research on DNNs focuses near unidirectionally on improving predictive performance on tasks of interest, be it the classification accuracy on images [42, 65, 108, 115, 222] or of human action recognition on video-clips [21, 53, 194, 197] or skeleton-sequences [152, 179, 214]; the mean average precision (mAP) of object detectors [58, 116, 164]; the CLEAR and IDF1 scores of trackers [13, 166]; the mean intersection over union (mIoU) of semantic segmentation networks [66, 119, 132]; the Inception score or Fréchet inception distance (FID) of visual generative models [84, 86]; or the F1, exact match (EM) [159], perplexity (PPL), Spearman correlation [200], BLEU-scores [141] of language models [16, 39, 148, 158, 199].

Since overparameterization is known to improve the generalization capabilities of a network [5, 135, 186], the simple act of increasing network parameters improves task-related performance metrics in most cases. The trend of growing model size is illustrated in Fig. 1.1. However, the financial realities and associated environmental externalities of training larger and larger models necessitate a multi-faceted view of a model’s favorability. Moreover, some use-cases require faster model inference to avoid outdated information corrupting the decision processes, for instance real-time monitoring and perception with application in robotics and autonomous vehicles.

2.2 Storage efficiency

There are multiple metrics, which can aid us in quantifying the storage efficiency of a DNN. The *learned parameter count* of a model is reported in most works. The numeric format of parameters (*e.g.*, float64 or int8) has a significant impact on both predictive performance and on actual *disk space consumption*. The model size in bytes, however, is seldom reported if numeric quantization is not of primary focus. Instead, most works implicitly assume a float32 format, which is the default datatype for popular deep learning frameworks such as PyTorch [145] and TensorFlow [4].

While the *in-memory size* of a model during execution is proportional to the learned parameter count, the model architecture and shape of input data have an equally important effect that should be considered in unison. Although model parameters can serve as a proxy for computational efficiency as well, they can be misleading. This is for instance the case in sparsely activated models [43, 51], where only a fraction of model weights are active during inference, or autoregressive models [39, 86], where parameters are iterative applied many times.

2.3 Computational efficiency

DNNs produce predictions via complex structures of multiplications and additions between the input data and learned model parameters. Therefore, the metric *floating-point operations* (FLOPs¹), *i.e.*, the total number of multiplications and additions during the forward pass of a model, is commonly used as a measure of computational efficiency. Since most modern hardware uses a fused multiply-add (FMA) instruction set, where an input is multiplied with- and added to a floating-point number in one operation, the *multiply-accumulate operation* (MAC)² metric is commonly used as well. FLOPs and MACs are tightly related with approximately two FLOPs per MAC.

FLOPs and MACs are hardware-agnostic³ measures of computational complexity and can be derived by counting the operations of a network during inference. However, they are not always representative of actual on-hardware performance. For instance, a modified ResNet was trained [11] to achieve half the *latency* on both a graphical processing unit (GPU) and tensor processing unit (TPU) despite measuring double the FLOPs of an EfficientNet [188] with similar accuracy. This discrepancy can be explained by the heavy use of depth-wise convolutions in EfficientNets, which have a higher latency per floating-point operation on GPUs and TPUs compared to regular convolutions.

While FLOPs can give a good indication of theoretical computational efficiency, on-hardware tests are necessary for true use-case specific comparisons. Commonly, either the *speed* (*i.e.*, inverse latency) or the *throughput* (*i.e.*, number of predictions per second) during inference is noted. Speed measurements use a batch size of one and throughput measurement may use higher batch sizes. This makes a considerable difference on massively parallel hardware such as GPUs or TPUs but has limited effect on CPUs that predominantly conduct sequential processing. Similarly, the *energy* used (in joules) to process an input or the *power* consumption (in watts) are interesting metrics for battery-powered devices.

A final note on bench-marking is that GPUs and TPUs do not perform computations in isolation but require another processor (usually a CPU) to transfer data. The

¹FLOPs in this context should not be confused with floating-point operations per second, which is a common metric of computational performance for hardware such as GPUs.

²Occasionally referred to as multiply-adds (mult-adds).

³Disregarding the fact, that the MACs definition is based on a hardware-implementation detail.

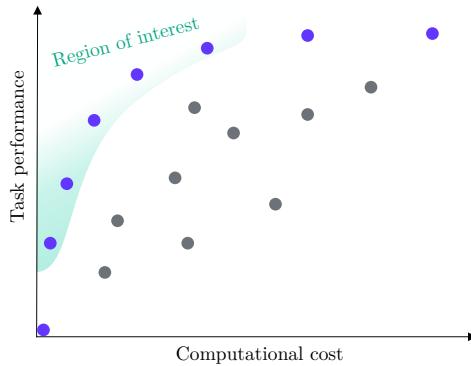


Figure 2.1: Pareto frontier of models •. Models that strike a useful balance between task performance and computational cost, *i.e.*, those within the green region, are of primary interest.

measurement of computational latency should thus include the data transmission to and from GPU/TPU in addition to the latency of computation⁴.

2.4 Efficiency is a trade-off

The story so far has been restrictive: It makes little sense to judge a model solely on its parameter count, FLOPs, MACs, inference latency, or throughput. A one-parameter algorithm, which always produces a constant-valued output, could be considered as ideal by these measures, even though its usability on any metric of predictive performance is naught. Hence, any practical measure of efficiency requires trading off predictive and computational capability. *Pareto efficiency* [142] captures this notion: A solution is optimal if there is no other solution where one metric improves without deterioration in another. In that sense, both a single parameter “constant producing” algorithm and a 1.6 trillion parameter colossal Switch Transformer [51]⁵ are Pareto efficient, but in each their end of the *Pareto frontier*. While either has limited practical utility due to either unusable predictions or prohibitive computational expense, the Pareto frontier covers a multitude of solutions which provide useful trade-offs. It is exactly these possible solutions that are the focus of this dissertation (see Fig. 2.1).

⁴Data transfer times, on-GPU processing time, maximum allocated GPU memory and energy consumption on NVIDIA Jetson devices can be easily measured using the Python library *PyTorch-Benchmark*, <https://github.com/lukashedegaard/pytorch-benchmark>.

⁵Switch-C is among the world’s largest language models at the time of writing (Feb, 2023).

Chapter 3

Approaches for enhancing inference efficiency

The pursuit of efficient deep neural networks has resulted in a multitude of techniques, which have produced models that exhibit greatly improved inference characteristics compared to non-optimized DNNs. This chapter provides an overview of available approaches, specifying how the contributions outlined in this dissertation complements and extends prior literature.

3.1 Pruning methods

An intuitive and effective approach to creating models with good efficiency trade-offs is to take an existing DNN that performs well on the task-metric of interest and *prune* (*i.e.*, remove) its least important weights. In the simplest version, every parameter in the network is considered in its own right and ranked either *globally* (for the whole network) or *locally* (within a network layer) to determine which weights to discard. This is referred to as *unstructured* pruning (Fig. 3.1a). Other methods prune multiple structurally connected weights in unison. This is called *structured* pruning. Owing to the flexibility in weight selection, unstructured pruning generally entails less deterioration of predictive performance when the model is pruned than

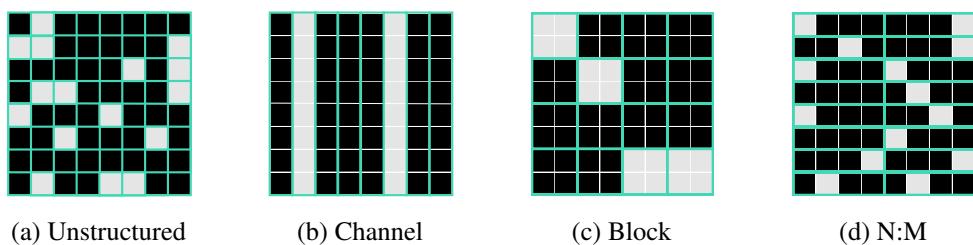


Figure 3.1: Different pruning approaches at 25% pruning. White blocks are pruned and green contours indicate pruning structure.

structured pruning. However, unstructured pruning results in sparse weight matrices at a computational disadvantage on GPUs and TPUs, but it has the potential to greatly reduce the computational run-time on CPUs [3].

Structured pruning methods are generally capable of speeding up models across CPUs as well as GPUs and TPUs, though some restrictions may apply. Pruning of entire network *channels* is a simple structured approach, which always comes with computational benefits. For fully connected layers, pruning of a unit corresponds to removing an entire column/row of weights (Fig. 3.1b) before/after the feature-map whose channel is pruned. For convolutional layers, a channel plane in the convolutional kernel is removed during channel-pruning.

Block pruning [110] structures a matrix into a grid of $p \times q$ weights where each weight block is bundled as one structural unit (Fig. 3.1c). This corresponds to the manner in which GPUs divide and process large matrix multiplications. GPU acceleration can subsequently be attained using a block-sparse GPU kernel [61].

In $N:M$ structured pruning [155, 229], weights are removed in such a way that N out of M consecutive elements in a matrix row become zero (Fig. 3.1d). By permuting the weight matrix channels and surrounding layers appropriately, findings suggest that a 2:4 sparsity pattern could achieve a 2 \times speedups on NVIDIA Ampere GPUs without sacrificing accuracy [131].

The literature on pruning has many other interesting directions. These include the design on ranking criteria (*e.g.*, based on weight magnitude [111], 1st and 2nd order gradients [133, 187], or explainability [218]), and when to prune (*e.g.*, iteratively [133] and with the option to recover from pruning [231]). Finally, *knowledge distillation*, which we will describe in the next section, is a popular technique for retaining model performance during pruning [171, 172, 190].

Paper 7, which is summarized in Chapter 5, proposes the use of “fusible” adapters [162] alongside channel-based structured pruning to create light-weight add-on networks that specialize a base-network with extremely few new learned parameters.

3.2 Knowledge distillation

Originally proposed for compressing the knowledge of an ensemble of models into a single model [85], *knowledge distillation* (KD) is a widely applied model compression technique [18] for transferring the knowledge of a cumbersome teacher model to an efficient student model. Specifically, a student model is supervised by the soft target distributions produced by a teacher model using a high temperature coefficient for the softmax operation. If ground truth labels (hard targets) are available, a weighted sum of objectives for the soft and hard targets is employed.

There are multiple variants of KD, including *deep mutual learning* [227], where an ensemble of students teach each other concurrently during the training process; *assistant teaching* [129], where models of intermediary size (teacher assistants) are employed to bridge the supervision from teacher models with too large capacity to effectively teach low-capacity student networks; *lifelong learning* [224] uses KD to

avoid catastrophic forgetting while learning new tasks by employing the prior model as teacher alongside an auxiliary set of data from previous tasks; *teacher-free KD* [221] uses a pre-trained student model to regularize itself.

KD has also been successfully used in combination with other efficiency approaches such as pruning [95, 171, 172], efficient architectures [193, 209], and quantization [130, 154]. Many more methods have employed and expanded on KD than can be covered here. A comprehensive survey by Guo et al. [60] explores the field in more detail.

3.3 Quantization

Quantization methods are concerned with the mapping of continuous numbers to discrete numerical representations. Here, the opposing goals of high representation accuracy (for a use-case of interest) and of low representation size (few storage bits) set the stage for a plethora of methods [57]. Although quantization is a long-studied problem [165, 177], the over-parameterization and computational expense of DNNs yields novel opportunities. Particularly, their over-parameterization makes them robust to intense quantization, which can reduce computational expenses significantly. Also, their high degrees of freedom makes it possible to attain good generalization performance on the forward error metric of interest (*e.g.*, accuracy) despite having high error between original and quantized weights.

Quantization formats are either *uniform* or *non-uniform*, *i.e.*, with or without equal spacing between values. A uniform quantization Q for a real valued input r is given by

$$Q(r) = \text{Int}(r/S) - Z, \quad (3.1)$$

where $S \in \mathbb{R}$ is a scaling factor and $Z \in \mathbb{Z}$ denotes the zero point. If $Z = 0$, the quantization is *symmetric*, *i.e.*, zero-centered, which can reduce computational costs [208]. $Z \neq 0$ yields an *asymmetric* quantization, which better represents imbalanced weights and activations such as ReLU. In either case, an important consideration for the quantization is the *calibration* of its *clipping range* $[\alpha, \beta]$, *i.e.*, the minimum and maximum represented values. For a bit width b , the scaling factor is

$$S = \frac{\beta - \alpha}{2^b - 1}. \quad (3.2)$$

Here, $-\alpha = \beta$ is used for symmetric quantization. Commonly, the minimum and maximum observed values are used to define α and β . Percentiles [126] or minimization or the KL divergence between real and quantized values [128] are also usable. While it is straight-forward to determine the clipping range for model weights, which are readily available, the calibration of activations can either be *static* or *dynamic*. In dynamic quantization, signal statistics are determined on the fly. Although this can provide higher predictive performance, it entails a considerable computational overhead. Static quantization uses a set of input data to calculate typical activation ranges *a priori* [93].

Quantization of DNNs is rich field with many research directions. These include quantization granularity (considering layers [106], groups [178], or channels [93]), quantization aware training (QAT) [27, 46, 97], post-training quantization (PTQ) [9, 50, 134], mixed-precision quantization [41, 63], and extreme quantization [27, 91].

3.4 Low-rank factorization of weights

Another way to compress model weights is through low-rank tensor decomposition. Multiple decomposition schemes have been explored, including Singular Value Decomposition [30], CP decomposition [102, 195], and a fast SVD-free “greedy bilateral” scheme called GreBsmo [220, 230]. In addition to the low-rank factorization, which captures global weight trends, some methods [62, 220] add sparse weights to model local variations better. This may, however, prohibit reduction in computational latency. Finally, some methods treat the decomposed weights as efficient architectural structures and train them from random initialization [194, 195].

3.5 Efficient architectures

Fully connected layers, convolutions and scaled dot-product attention [199] operations can be computationally expensive if feature and weight dimensions are not carefully managed.

Depth-wise separable convolutions, *i.e.*, a depth-wise convolution followed by a point-wise convolution, are widely used to reduce the computational complexity of both 2D convolutions [65, 88, 188, 226] and 3D convolutions [52, 104] on a micro-level. In practice, however, their speed can be memory constrained on some computational devices [226]. Multi-linear convolutional filters are explored in [195].

Shift modules, which shift features along a feature-dimension, have been utilized by multiple works [24, 94, 113, 206, 215, 216] as an alternative to spatial or temporal convolutions, with greatly reduced number of floating-point operations. While feature shifts do not add FLOPs, it should be noted that they do impose a computational overhead in practice.

The scaled dot-product attention (SDA) in Transformers [199] has itself received considerable attention in attempts to alleviate its computational complexity scaling quadratically with the sequence length. Particularly, some works [42, 143] group entities into fewer blocks that attend to one another. Other works approximate the self-attention matrix [26, 202, 210]. Paper 3 [78] exploits temporal redundancies observed in the SDA during stream processing to reduce complexity to scale linearly with the sequence length.

On an architectural level, one line of works is concerned with balancing the scaling of input resolutions, network depth, and network channels [19, 52, 188]. Early exit architectures dynamically reduce computational cost based the “hardness” of predictions or to meet computational deadlines [8, 174]. Finally, neural architecture search (NAS) methods automate the network design based on a predefined search space

and performance metric (*e.g.*, an accuracy-latency trade-off on mobile devices [189]). While the architecture search of early approaches was very computationally expensive [233, 234], newer works focus on reducing computational expense, for instance by means of weight sharing [19, 151] or progressive search [196].

3.6 Caching and temporal redistribution of features

Recent developments in time-series processing and specifically in human action recognition have favored convolutional- and Transformer-based models over Recurrent Neural Networks (RNNs) [6, 21, 53, 203, 212]. While this pushed the state-of-the-art in task performance as well as offline processing efficiency (*i.e.*, inference over a spatio-temporal clip), Convolutional Neural Networks (CNNs) and Transformers [199] could not incrementally process individual frames at a time to produce predictions in an online manner, except through the use of input caching and sliding window processing, where a shifted spatio-temporal clip is assembled and processed in each step.

To alleviate this, *Continual Inference Networks*, which are described in the papers 1 to 2 and summarized in Chapter 4, propose a bottom-up restructuring of the CNN, Spatio-temporal Graph Convolutional Network (ST-GCN) [213] and Transformer building blocks, to augment prior models with the ability to incrementally process time-series input efficiently one time-step at a time. The proposed restructured networks can reuse the network weights of pre-trained CNNs, ST-GCNs, and Transformers to provide reductions in computational complexity proportional to the temporal receptive field of the source networks during online processing of a continually arriving stream of data.

Other approaches are not weight compatible with prior model weights and instead utilize specialty architectures that either segregate spatial and temporal convolutions [109, 185] or cache spatial features through time [64, 207].

3.7 Efficient run-times

In addition to the efficiency approaches outlined in the prior sections, there exist a number of *compilers*. These optimize the network structure for training and deployment, *e.g.*, by fusing batch normalization weights with convolutional kernels. While most frameworks widely support PyTorch [145], TensorFlow [4] and ONNX [7] model formats, the support for target hardware varies. Apache TVM [22], ONNX Runtime [35], OpenXLA [37], and BladeDISC [31] support a wide variety of models and platforms; OpenVINO [36] optimizes for Intel hardware; TensorRT [34] optimizes deployments on NVIDIA GPUs; DeepSparse [32] optimizes the runtime of sparse networks on CPUs; and TFLite [38] optimizes TensorFlow for mobile and edge devices. Finally, Speedster [33] is an effort to provide a unified interface that automatically selects the appropriate compiler.

Chapter 4

Redistributing computations through time with Continual Inference Networks

4.1 Introduction

The chase to push predictive performance in tasks such as human activity recognition has led the field through a series of phases, where different architectural designs have been dominant. For instance, Recurrent Neural Networks (RNNs), which can be used to temporally aggregate the spatial (image) features extracted by 2D Convolutional Neural Networks (CNNs), were considered state-of-the-art between 2014 and 2017 [40, 223].

While CNNs with spatio-temporal 3D kernels were not uncommon during this time-period, they initially struggled to beat prior predictive performance, presumably due to the limited dataset-sizes, which were insufficient to effectively optimize the 3D kernels. With the introduction of the Kinetics-400 dataset [98] as well as the pre-training technique of *inflating* 2D kernels of models trained on ImageNet1k [168] to 3D, a new level of predictive performance was achieved with I3D [21]. In the following years, 3D CNN architectures were further refined with methods such as R(2+1)D [194], which partitioned the 3D convolution into a separate spatial 2D convolution and temporal 1D convolution; SlowFast [53], which processed the input data in multiple branches with different capacities; X3D [52], which employed an EfficientNet-inspired [188] design based on depth-wise separable convolutions and proposed a progressive network scaling approach; and MoViNets [103], which used neural architecture search and temporal ensembles to boost performance further.

Inspired by the non-local means method [17] and Transformer self-attention [199], Non-local Neural Networks [204] were an early work to augment 3D CNNs such as I3D with spatio-temporal self-attention. A few years later, the success of ViT [42] in the image domain, spurred multiple methods that adopted transformer architectures in the video-domain, *e.g.*, ViT-B VTN [136], ViViT [6], TimeSFormer [14], MViT-

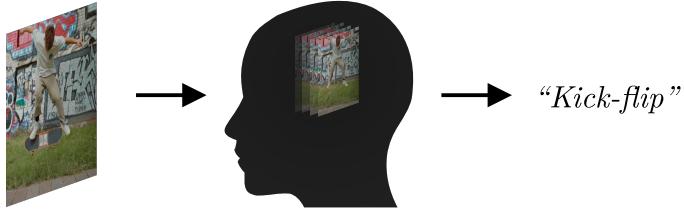


Figure 4.1: Perception in the natural world happens continually, one impression at a time, with temporal context modeled internally.

v1 [49] and -v2 [112]. At the time of writing, the dominant design is still based on Transformers.

The aforementioned methods were all driven by a competition to attain the best performance on *offline* video-clip prediction problems, *i.e.*, the assignment of class-labels to a fixed-size image sequence. However, the natural world is not composed of fixed-size temporal blocks. We experience a continual stream of visual impressions without inherent beginning and end (see Fig. 4.1). In our interaction with the world, on-the-fly classification of events is crucial to our survival. Similarly, online visual recognition is central in many important use-cases such as autonomous vehicles, collaborative robots, and live monitoring systems.

Yet, naive attempts to perform online recognition with the recent dominant designs, 3D CNNs and Transformers, are both inelegant and computationally inefficient. The issue relates to the input format required by 3D CNNs, which expect a spatio-temporal clip as input, and Transformers, which process sets of tokens. Consequently, they are not able to handle single frames but must aggregate data within a sliding window and pass full spatio-temporal clips or token sets to make predictions. This process is illustrated in Fig. 4.2.

Besides the additional complexity of input caching, sliding-window processing entails considerable hidden computational redundancy, due to repeat computations of mutual interactions of weight and input data. While RNN-based methods do not suffer this inefficiency, their predictive capabilities are often inferior, and 3D CNN and Transformer methods may still provide better computational/predictive trade-offs. Even in online action detection [29], *i.e.*, per-frame action recognition where methods do not peek into the future, recent works have favored Transformer-based models [203, 212] despite their inability to process one frame at a time without the use of sliding-window processing.

This leads to the central research questions of this chapter:

RQ1 *How can we accelerate state-of-the-art deep neural networks, which were tailored for offline batch-processing of time-series data, to perform online stream processing efficiently?*

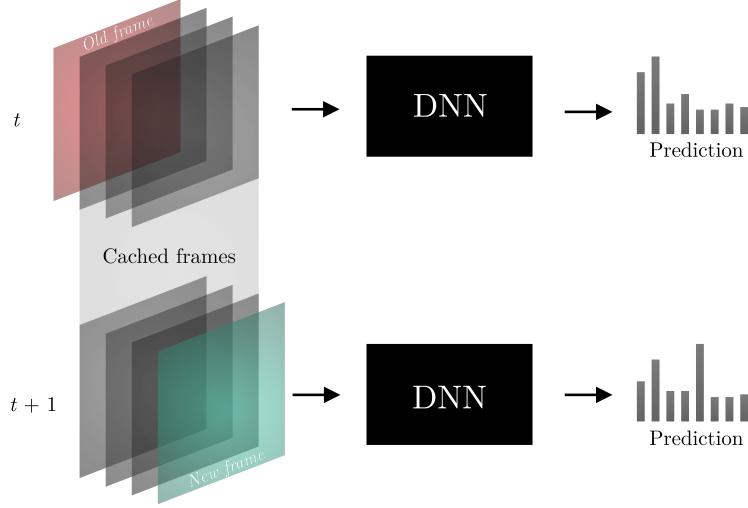


Figure 4.2: Sliding-window processing with DNNs. With each passing time-step, the **old frame** is discarded while intermediary frames are cached and prepended to the **new frame**. To process each time-step, many DNNs (*e.g.*, 3D CNN or Transformer-based networks) require the full frame-sequence as input. This entails considerable computational redundancy.

4.2 Related works

Multiple recent works have proposed specialty architectures, which can efficiently process input streams. *Recurrent Convolutional Units* [185] replace 3D kernels with a 2D kernel to model the current image alongside a temporal 1D convolution to aggregate temporal context. The *Temporal Shift Module* [113] augments prior 2D CNNs by shifting a portion of channels along the temporal axis and fusing temporal information within residual connections. *MoviNets* [103] use stream buffers to cache activations from previous frames and aggregate them with causal 3D convolutions to achieve efficient stream processing. While the utilization of stream buffers alongside 3D convolution has similarities to the concurrently published *Continual 3D Convolutions* proposed in paper 1, the stream versions of MoviNets are not weight-compatible with regular 3D CNNs. *Dissected 3D CNNs* [109] use residual connections to pass temporal information to the subsequent time-step and aggregate information over the temporal dimension via 3D convolutional kernels with temporal kernel size of two, fusing information from the prior time-step and the current one.

The *Streaming Transformer* [205] processes segments of an input stream (*i.e.*, multiple time-steps at a once) and augments key and value matrices with the keys and values from prior time-steps without letting gradient flow back into prior processed segments. The tokens in the current segment thus attend mutually to one another, while tokens from prior steps are causally attended to without generating output tokens themselves. A similar approach is used for the training of Transformer-XL [28], where

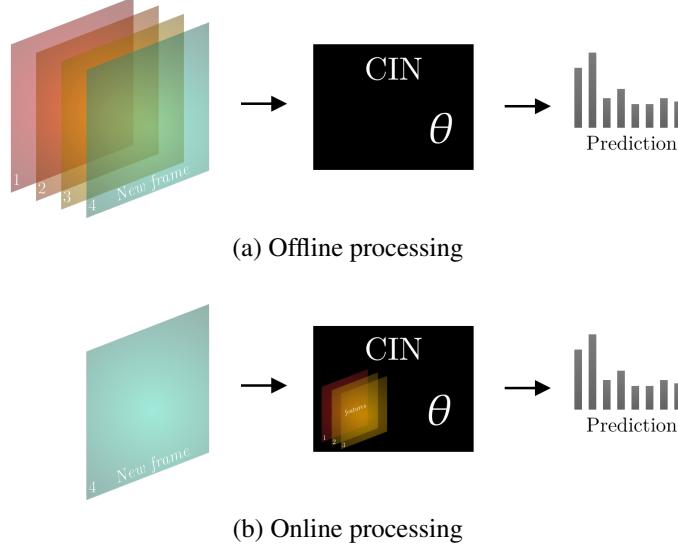


Figure 4.3: Continual Inference Network (CIN) used for (a) offline processing of a spatio-temporal batch and (b) online processing of an input stream, where intermediary frame features are cached internally. Given identical inputs, a CIN can use one set of learned weights θ in either situation and produces identical results.

causal scaled dot-product attention (SDA) is also processed in segments and gradients are restricted to flow in the current segment only. Since the employed self-attention is causal (*i.e.*, only newer tokens attend to older ones), Transformer-XL can process tokens sequentially in a stream if appropriate caching is used between time-steps. Moreover, the SDA operation is modified with a relative positional encoding scheme.

The *Continual Single-output SDA* proposed in paper 3, which is a simplification of the full *Continual Retroactive SDA*, operates in a similar manner to the Transformer-XL attention. However, it retains backward compatibility with regular SDA [199] by utilizing *Recycling Positional Encoding* as its positional encoding scheme.

None of the described related works provide backwards compatibility with basic 3D CNNs and Transformer architectures. In contrast, the *Continual Inference Networks* proposed in papers 1, 2, and 3, and are computational extensions of basic 3D CNNs, Transformers, and ST-GCNs [214], which retain weight-compatibility and enable efficient online stream processing in addition to the traditional offline processing.

4.3 Continual Inference Networks

4.3.1 Definition

Continual Inference Networks (CINs) spring from a desire to continue the fruitful competition on offline prediction benchmarks while retaining the ability to efficiently

process a continual input stream without predefined beginning and end. Formally, we can define CINs to encompass network architectures which uphold the ideal:

Online inference: A CIN can process a continual input stream time-step by time-step, producing a prediction for each step, without computational redundancy (Fig. 4.3b).

Offline inference: A CIN can process a spatio-temporal input similarly to a regular DNN (Fig. 4.3a).

Identical results: A CIN produces identical results during online and offline inference given identical inputs.

Identical weights: A CIN uses one set of trainable parameters for online and offline inference.

4.3.2 Relation to recurrent neural networks

The notion of a Continual Inference Network encompasses all networks, which can operate sequentially, producing outputs for each observed step without peeking into the future. This includes uni-directional recurrent neural networks (RNNs) as well as causal neural networks with directed acyclic graph (DAG) structure (Fig. 4.4a). DAG neural networks include feed-forward neural networks, CNNs, and Transformers, which operate locally within a time-step, as well as *Continual* CNNs (see Sections 4.4 and 4.5) and *Continual* Transformer Encoders (see Section 4.6) with spatio-temporal receptive fields.

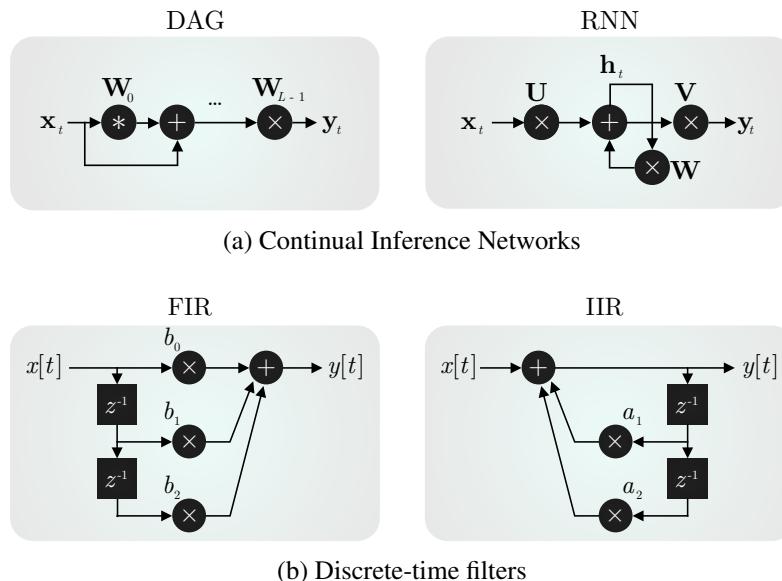


Figure 4.4: Continual Inference Networks are multi-modal discrete-time filters with learned weights. Specifically, directed acyclic graph (DAG) structured networks are finite impulse response (FIR) filters and recurrent neural networks (RNNs) are infinite impulse response (IIR) filters.

4.3.3 Relation to digital signal processing

In signal processing, digital discrete-time filters are widely used to remove unwanted frequency components of an input signal. In particular, the signal processing literature [121] distinguishes between two filter types, categorized by their response to a Kronecker delta: Finite impulse response (FIR) filters have an impulse response with finite duration, after which it settles at zero; infinite impulse response (IIR) filters have an impulse response, which continues indefinitely. Mechanistically, FIR filters consist of multiplicative forward coefficients (b) and tapped delays (noted as z^{-1} using z-domain notation), which are arranged as a DAG. IIR filters, on the other hand, contain feedback cycles with multiplicative coefficients (a). These are illustrated in Fig. 4.4b.

Continual Inference Networks can be viewed as large discrete-time filters with learned coefficients. In particular, RNN-structured CINs are IIR filters while DAG-structured CINs are FIR filters.

4.3.4 Architectural considerations

A wide range of existing network architectures either satisfy or can be modified to satisfy the CIN criteria outlined in Section 4.3.1. In particular, the temporal (re)distribution of computations plays a crucial role. Moreover, zero-padding must be used with care.

4.3.4.1 Add delays to satisfy causality

CINs operate with a linear progression of time and satisfy *causality*, *i.e.*, they do not peek into the future. While many DNNs designed for offline processing do not satisfy this, surgical placement of tapped delays can rectify the issue. The *Continual 3D CNNs* proposed in paper 1 demonstrate this by delaying partially convolved frame-features before summing them at the appropriate time-step. In general, a network module with temporal kernel size K_T , dilation D_T , and zero-padding P_T has a delay of

$$\underbrace{K_T + (K_T - 1)(D_T - 1)}_{\text{Receptive field}} - P_T - 1. \quad (4.1)$$

4.3.4.2 Remove end-padding in the temporal dimension

Many CNNs use zero-padding to avoid shrinking features maps. However, zero padding of the temporal edge in the direction of most recent data corresponds to observing zeros in the future. Consider the situation depicted in Fig. 4.5 where a multi-layer neural network conducting online inference processes an input feature at time t_0 . The t_0 input feature results in one intermediary feature-map after layer f and one output feature after layer g (yellow highlight). If temporal end-padding is used with each layer, additional *acausal* outputs (red dotted boxes) will be computed. To make matters worse, the extra acausal computations accumulate with each zero-padding added throughout the network! When the subsequent time-step arrives, all

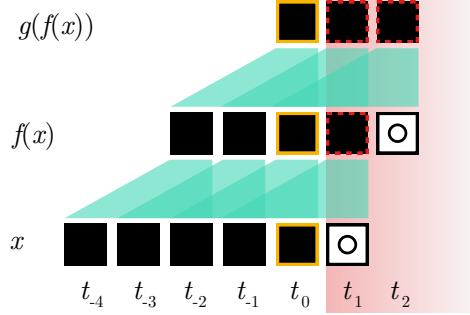


Figure 4.5: Temporal end-padding \square in neural network layers with receptive field of size \triangle results in acausal outputs (red dotted boxes \square). The features corresponding to the current time-step t_0 are highlighted yellow \square .

the acausal features become obsolete and must be computed anew for the next padded zeros. Causal features, on the other hand, can be cached and reused. Accordingly, CINs should not use temporal end-padding.

Empirically, it was observed in paper 1 that weights from networks which were trained with temporal end-padding can be reused for efficient stream processing in corresponding CINs without end-padding. This, however, introduced a *model shift*, *i.e.* a deviation of the model operation, with lead to slight accuracy deterioration.

4.3.4.3 Residual connections

The above-mentioned architectural consideration has particularly interesting implications for the residual connection of temporal CNNs. Commonly, residuals are employed in the context of an “equal padding” strategy, *i.e.*, where padding is used to retain the temporal feature shape after the layer. This is illustrated in Fig. 4.6a. If we remove the end-padded zero, the alignment of the residual shifts: The input of a time-step 1 must be added to the output in time-step 2 (see Fig. 4.6b). This corresponds to delaying the residual connection by the same number of steps as removed zeros. Residual connections should generally match the delay of the network module they wrap, as described by Eq. (4.1).

4.3.4.4 Temporal stride

CINs are designed to perform online processing on continual streams, where one output is produced for each input step. However, some network layers, *e.g.*, convolution and pooling, are occasionally used with temporal stride larger than one. While this can be used to *reduce* the computational complexity for full-sequence processing in a regular network due to the reduced temporal dimension of subsequent feature-maps, it *increases* the per-prediction complexity of CINs. In fact, each network layer with a stride larger than one corresponds to a temporal down-sampling operation.

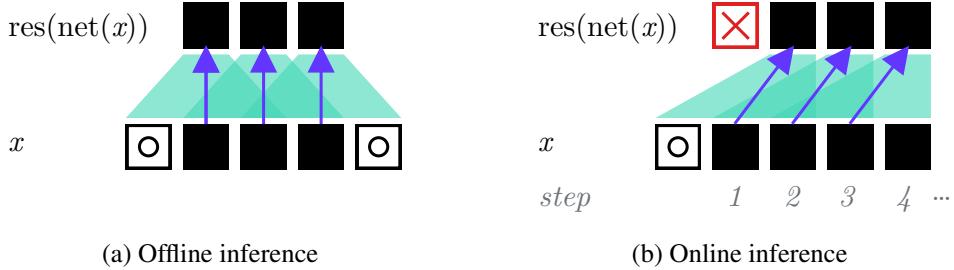


Figure 4.6: Residual connections ↑ over a module with receptive field of size ▲ and padding one (“equal padding”) ◻. ✗ are empty outputs. This illustration first appeared in paper 4.

Generally, the stride of a neural network, S_{NN} , aggregates multiplicatively throughout the L layers of a network:

$$S_{NN} = \prod_{l=1}^L S_l. \quad (4.2)$$

The network prediction rate, R_{NN} , is given by the inverse stride:

$$R_{NN} = 1/S_{NN}. \quad (4.3)$$

4.3.5 Training

Since CINs can operate efficiently on both spatio-temporal inputs and on step sequences, either can be used for training. Most prior methods train on datasets with spatio-temporal inputs. Full sequence training is recommended for CINs as well.

Should step-wise training be desired, nonetheless, two modifications are recommended to match the full sequence training. First, the normalization layer momentum parameters should be adjusted to accommodate the smaller statistical sample provided by a step compared to a full clip. To match the running statistics, the step momentum M_{step} should be updated to

$$M_{\text{step}} = \frac{2}{(L * (2/M_{\text{seq}} - 1) + 1)}, \quad (4.4)$$

where M_{seq} is the momentum during full sequence processing, and L is the length of the sequence observed by the layer. Second, the drop-out mask should be fixed during the steps originating from one sequence to avoid within-sequence changes in network configuration.

4.4 Continual 3D Convolutional Neural Networks

4.4.1 Related works

3D CNNs are a widely used architecture for clip-based spatio-temporal perception tasks such as human activity recognition. Specifically, networks such as I3D [21],

R(2+1)D [194], SlowFast [53], and X3D [52] each pushed the state-of-the-art on the large-scale video-classification dataset Kinetics-400 [98].

Despite efficiency-focused innovations introduced in recent works [52, 53, 194], 3D CNNs remained fundamentally hamstrung in the context of online-recognition, where a prediction is desired for each input frame. This stems from the convolution implementation in deep learning tool-kits such as PyTorch [145] and TensorFlow [4], where the full input sequence is processed at once during the forward pass. On the other hand, convolutions are routinely implemented as FIR filters for digital signal processing of continual, uni-dimensional input streams (see Fig. 4.4b left). A similar idea can be used for multi-dimensional 3D convolutions in the context of DNNs.

4.4.2 Convolutions for processing of continual input streams

Consider an input $\mathbf{X}_t \in \mathbb{R}^{H \times W}$ sampled from a time-series $\mathbf{X} = [\mathbf{X}_t : t \in 0..T - 1] \in \mathbb{R}^{T \times H \times W}$ and a 3D convolutional kernel $\mathbf{W} \in \mathbb{R}^{K_T \times K_H \times K_W}$ with time, height, width, and kernel dimensions K . Without loss of generality, we will omit the channel dimensions of input and convolutional kernel. A three-dimensional convolution between the sequence \mathbf{X} and kernel \mathbf{W} is defined as:

$$(\mathbf{W} * \mathbf{X})^{(t,h,w)} \stackrel{\text{def}}{=} \sum_{k_t=0}^{K_T-1} \sum_{k_h=0}^{K_H-1} \sum_{k_w=0}^{K_W-1} \mathbf{W}^{(k_t, k_h, k_w)} \mathbf{X}^{(t-k_t, h-k_h, w-k_w)}. \quad (4.5)$$

The feature-map $\mathbf{Y}^{(t)}$ at time-step t can be computed by summation of the spatial convolutions:

$$\mathbf{Y}^{(t)} = (\mathbf{W} * \mathbf{X})^{(t)} = \sum_{k_t=0}^{K_T-1} \mathbf{W}^{(k_t)} * \mathbf{X}^{(t-k_t)}. \quad (4.6)$$

The above can be extended to any spatial convolution, including dilated, strided, padded, and grouped convolutions, and holds for continual input streams where $T \rightarrow \infty$. *Continual convolutions* for DNNs, *i.e.*, convolutions for processing of a continual input stream, can be efficiently implemented through the use of appropriate caching, similar to a FIR filter for digital signal processing, but with multi-dimensional signals and spatial convolutions instead of multiplication by simple filter coefficients. There are two implementation options for this.

- (a) *Pre-cached*: Prior input values can be cached first and spatially convolved with the kernel later when all required time-steps have been observed (see Fig. 4.7a). This corresponds to a *direct form* FIR filter implementation.
- (b) *Post-cached*: Individual inputs steps are eagerly convolved with the kernel, cached, and progressively summed up (see Fig. 4.7b). This corresponds to a *transposed form* FIR filter implementation.

The floating-point operations are identical in both implementation options, but the memory requirements can vary. If the convolutional kernel $\mathbf{W} \in \mathbb{R}^{C_{in} \times C_{out} \times K_T \times \dots}$ performs an up-projection (*i.e.*, $C_{in} > C_{out}$), the pre-cached structure gives the smallest

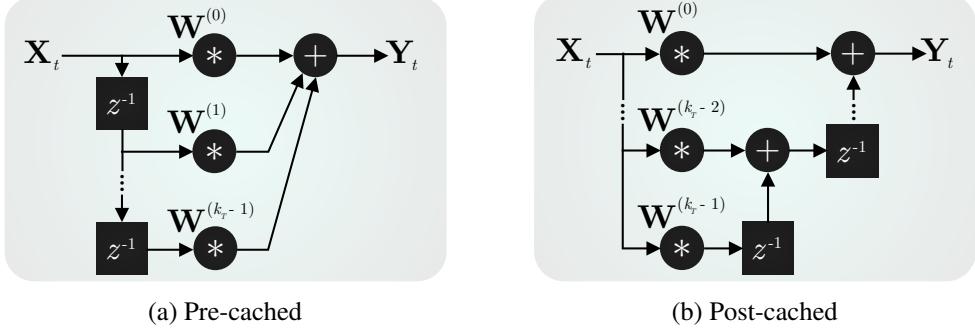


Figure 4.7: Continual convolutions can be modeled as digital finite impulse response filters with temporal caching (a) pre or (b) post spatial convolutions of a temporally sliced kernel $\mathbf{W}^{(i)} \in \mathbb{R}^{K_H \times K_W}, i \in 0..K_T - 1$.

cache size. Similarly, post-cached structures are more memory-efficient for down-projections.

To enable prior state-of-the-art 3D CNNs to operate efficiently during stream processing, the complete architecture requires a modified implementation. Cf. Section 4.3.4, this includes delaying residuals appropriately, removing end-padding, as well as modifying average pooling operations. The latter is easily achieved by pooling each frame, caching the appropriate receptive field, and updating a running average wherein the newest frame is added and oldest the frame is subtracted in each time-step. For additional details, please see paper 1.

4.4.3 Experiments

To validate the efficacy of reusing prior 3D CNNs for efficient stream processing by adapting the principles of Continual Inference Networks (Section 4.3), we implemented *continual* versions of I3D, Slow, and X3D-{S, L, M} (denoted *CoI3D*, etc.) and benchmarked them on Kinetics-400 using pre-trained weights from corresponding regular 3D CNNs. The regular 3D CNNs, which served as our baselines, were evaluated with sliding-window processing, *i.e.* by processing full spatio-temporal clips in the usual manner. The *Continual* 3D CNNs were initialized with frames corresponding to their receptive field minus one frame, and evaluated on the next frame. In addition to accuracy, we report parameter count, GPU memory consumption, FLOPs, as well as throughput on the CPU of a MacBook Pro 16" 2019 2.6 GHz i7 processor, NVIDIA TX2 and Xavier embedded GPUs, and an NVIDIA RTX 2080Ti GPU.

The benchmark results are presented in Table 4.1 and visually depicted for the accuracy/FLOPs trade-off in Fig. 4.8. Here, clip-sizes are illustrated with subscripts and within the illustrated data points. Across all tested CINs, we observed impressive FLOPs reductions and increases in throughput. These were generally proportional to the clip size of models (*e.g.*, $7.95\times$ fewer FLOPs for *CoSlow* compared to *Slow*, which had a clip size of 8, and $15.34\times$ fewer for *CoX3D-L* compared to *X3D-L*,

which had a clip size of 16), but with lower relative throughput increases than FLOPs reductions due to the runtime overhead of reading and writing cached features. While the parameter counts for continual and regular 3D CNNs were identical, the maximum allocated GPU memory was lower for most CINs despite their use of feature-caching. This can be explained by the larger intermediary feature-maps handled by regular networks compared to CINs, which did not include the temporal dimension. Finally, the test accuracies of CIN network versions were slightly reduced due to the model shift incurred by the missing end-padding with zeros. Since the computational cost of increasing the temporal receptive field of a CIN is negligible, the lost accuracy can be regained by increasing the receptive field for the average pooling layer residing as the penultimate network layer in all the considered networks. Networks with larger average pooling size of 64 frames are denoted accordingly in Table 4.1 and Fig. 4.8. Additional details as well as ablation studies of network initialization and global average pooling sizes, as well as results on the Charades dataset [182] are available in paper 1.

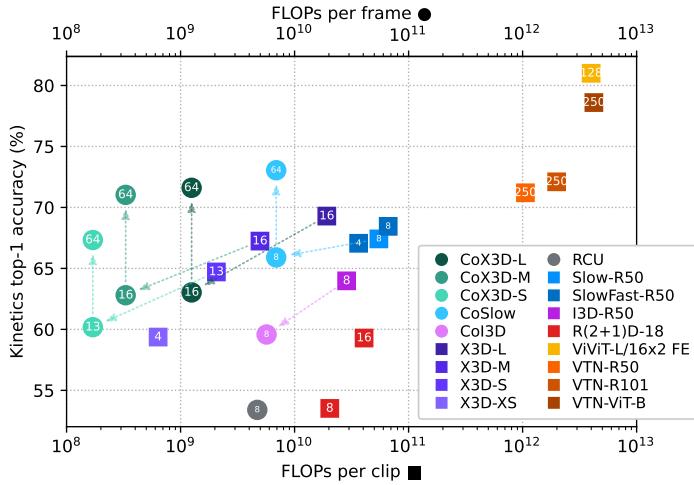


Figure 4.8: Accuracy/complexity trade-off for Continual 3D CNNs and recent state-of-the-art methods on Kinetics-400 using 1-clip/frame testing. ■ FLOPs per *clip* are noted for regular networks, while ● FLOPs per *frame* are shown for the Continual 3D CNNs. Frames per clip / global average pool size is noted in the representative points. Diagonal and vertical arrows indicate a direct weight transfer from regular to Continual 3D CNN and an extension of the receptive field. The figure first appeared in paper 1 [72].

Model	Acc. (%)	Par. (M)	Mem. (MB)	FLOPs (G)	Throughput (preds/s)				
					CPU	TX2	Xavier	2080Ti	
Clip	I3D-R50	63.98	28.04	191.59	28.61	0.93	2.54	9.20	77.15
	R(2+1)D-18 ₈	53.52	31.51	168.87	20.35	1.75	3.19	6.82	130.88
	R(2+1)D-18 ₁₆	59.29	31.51	215.44	40.71	0.83	1.82	3.77	75.81
	Slow-8x8-R50	67.42	32.45	266.04	54.87	0.38	1.34	4.31	61.92
	SlowFast-8x8-R50	68.45	66.25	344.84	66.25	0.34	0.87	2.72	30.72
	SlowFast-4x16-R50	67.06	34.48	260.51	36.46	0.55	1.33	3.43	41.28
	X3D-L	69.29	6.15	240.66	19.17	0.25	0.19	4.78	36.37
	X3D-M	67.24	3.79	126.29	4.97	0.83	1.47	17.47	116.07
Frame	X3D-S	64.71	3.79	61.29	2.06	2.23	2.68	42.02	276.45
	X3D-XS	59.37	3.79	28.79	0.64	8.26	8.20	135.39	819.87
	RCU ₈ [185]	53.40	12.80	-	4.71	-	-	-	-
	CoI3D ₈	59.58	28.04	235.87	5.68	3.00	2.41	14.88	125.59
	CoI3D ₆₄	56.86	28.04	236.08	5.68	3.15	2.41	14.89	126.32
	CoSlow ₈	65.90	32.45	175.98	6.90	2.80	1.60	6.18	113.77
	CoSlow₆₄	73.05	32.45	176.41	6.90	2.92	1.60	6.19	102.00
	CoX3D-L ₁₆	63.03	6.15	184.29	1.25	2.30	0.99	25.17	206.65
	CoX3D-L₆₄	71.61	6.15	184.37	1.25	2.30	0.99	27.56	217.53
	CoX3D-M ₁₆	62.80	3.79	68.88	0.33	7.57	7.26	88.79	844.73
	CoX3D-M₆₄	71.03	3.79	68.96	0.33	7.51	7.04	86.42	796.32
	CoX3D-S ₁₃	60.18	3.79	41.91	0.17	13.16	11.06	219.64	939.72
	CoX3D-S₆₄	67.33	3.79	41.99	0.17	13.19	11.13	213.65	942.97

Table 4.1: Kinetics-400 benchmark results. The noted accuracy is the single clip or frame top-1 score using RGB as the only input-modality. The performance was evaluated using publicly available pre-trained models without any further fine-tuning. For throughput comparison, predictions per second denote frames per second for the *Co* models and clips per second for the remaining models. Throughput results are the mean of 100 measurements. Pareto-optimal models are marked with bold. Mem. is the maximum allocated memory during inference noted in megabytes. The table first appeared in paper 1 [72].

4.5 Continual Spatio-temporal Graph Convolutional Networks

4.5.1 Human actions as a skeleton sequence

Instead of predicting human action categories directly from raw RGB video clips, another approach explicitly models human body poses for each frame in a sequence and utilizes those for prediction of action labels. Specifically, a body pose can be extracted with pose estimation tools such as OpenPose [20] and modeled as a skeleton graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where body joint positions are nodes \mathcal{V} and limbs are edges \mathcal{E} . To construct a spatio-temporal graph, joints are connected across time-steps in the skeleton sequence as well. This is illustrated in Fig. 4.9.

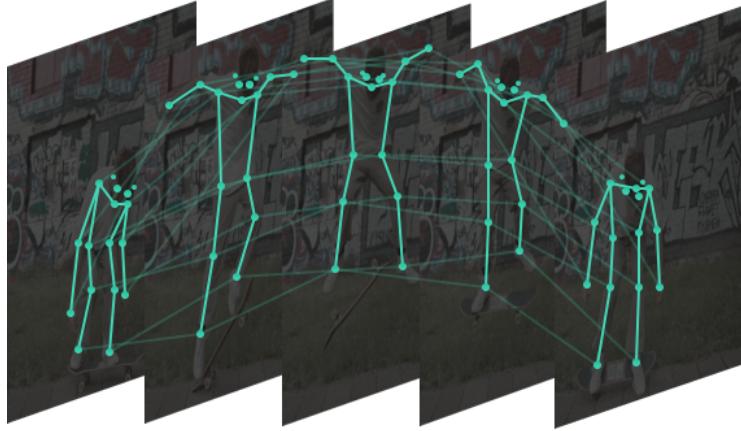


Figure 4.9: Sequence of human body poses modeled as a spatio-temporal graph.

4.5.2 Related works

Early works on skeleton-based action recognition did not use the graph representation, but either made pseudo-images from rearranged body joint positions, which were subsequently processed by a CNN [99], or trained RNN models on feature-vectors comprising concatenated joints [176, 225]. However, both of these representations discarded valuable information on the non-Euclidean skeleton structure.

Graph Convolutional Networks (GCNs), which are described in publication 6, are a recently proposed class of neural networks for processing graph-structured data. The Spatio-temporal Graph Convolutional Network (ST-GCN) [214] was the first method to utilize GCNs for skeleton-based action recognition. Here, graph convolutions were used for spatial processing while regular convolutions aggregated the temporal information. Multiple extensions of ST-GCN have been proposed, including 2s-AGCN [181], which uses two input streams (*i.e.*, one considering joint features and another considering bone features) and which enhances ST-GCN with a graph attention mechanism considering node similarity; DGNN [180] modeled skeleton connections with a directed graph; S-TR [152] used self-attention layers instead of graph convolutions; and MS-G3D [117] extracted long-range features with multi-scale graph convolutions.

The ST-GCN [214] and multiple derived networks [152, 181] employ spatial graph convolution (GC) and temporal convolution (TC) as separate network layers. In practice, GC varies among methods¹, and TC is implemented as a 2D convolution. A spatio-temporal graph convolution block which computes the feature \mathbf{H}_l at layer l takes the form:

$$\mathbf{H}_l = \text{ReLU}(\text{Res}(\mathbf{H}_{l-1}) + \text{BN}(\text{TC}(\text{GC}(\mathbf{H}_{l-1})))), \quad (4.7)$$

where BN denotes Batch Normalization, and a residual connection (Res) performs a point-wise convolution when input and output channel dimensions vary. Multiple

¹See paper 2 for exact formulations of graph convolution (GC) layers.

such blocks are stacked throughout the network, before a global average pooling layer and a prediction head aggregate the final prediction.

Various approaches have been explored to reduce the high computational complexity of ST-GCN-based methods. These include Neural Architecture Search [80, 146], selection of informative frames [81], and the use of channel shift operations in place of temporal convolutions [24, 25]. However, both the ST-GCN and its derived methods suffer the same restriction as 3D CNNs during online processing: They require full skeletons sequences as their input and hence need a sliding window of T cached input skeletons for stream processing.

4.5.3 Continual Spatio-temporal Graph Convolutions

During continual processing of an input stream, the network should produce temporal features for each time step $\mathbf{H}_l^{(t)}$ rather than for the whole sequence \mathbf{H}_l . This behavior can be achieved by reformulating the spatio-temporal graph convolution block (see Eq. (4.7)) to conform with CIN requirements (Section 4.3.1). The separation between the graph and temporal convolutions makes the transition easy: First, the regular temporal convolution is replaced with a continual temporal convolution (*CoTC*), and second, the residual connection is delayed according to Eq. (4.1) based on the parameters of *CoTC*. The continual spatio-temporal graph convolution block is given by

$$\mathbf{H}_l^{(t)} = \text{ReLU} \left(\text{Delay}(\text{Res}(\mathbf{H}_{l-1}^{(t)})) + \text{BN}(\text{CoTC}(\text{GC}(\mathbf{H}_{l-1}^{(t)}))) \right). \quad (4.8)$$

To construct the full *Continual* ST-GCN variant, multiple such blocks are stacked, followed by a running global average pooling layer and a prediction head, which produces a prediction each time-step. The simplicity of the transformation from Eq. (4.7) to Eq. (4.8) exemplifies the utility of the CIN reformulation. While a considerable engineering effort is required to handle network delays and inputs shapes properly², the effect on the computational complexity of per-step prediction is dramatic: Continual inference only requires the processing of a single frame with the network body and prediction head, *i.e.*, $\mathcal{O}(\text{CoNet}) \approx \mathcal{O}(\text{Body}) + \mathcal{O}(\text{Head})$, whereas sliding-window processing requires the body to process the whole sequence of T skeletons for each step, *i.e.*, $\mathcal{O}(\text{Net}) \approx T \cdot \mathcal{O}(\text{Body}) + \mathcal{O}(\text{Head})$. For $\mathcal{O}(\text{Body}) \gg \mathcal{O}(\text{Head})$, which is the case for most neural networks, computational savings are thus proportional to T .

4.5.4 Experiments

In practice, the ST-GCN-based methods use long skeletal sequences as inputs. To reduce the computational complexity, multiple layers in the network use a temporal stride of two. In the context of CINs, however, the temporal stride has the undesired effect of reducing the prediction rate (see Section 4.3.4). In addition to direct CIN

²The *Continual Inference* library described in paper 4 and Section 4.7 alleviates the engineering effort substantially.

Model	Frames/ pred	Accuracy (%)		Par. (M)	Max mem. (MB)	FLOPs/pred (G)	Throughput (preds/s)	
		X-Sub	X-View				CPU	GPU
ST-GCN	300	86.0	93.4	3.14	45.3	16.73	2.3	180.8
ST-GCN*	300	86.3 (+0.3)	93.8 (+0.4)	3.14	72.6 (160%)	36.90 ($\uparrow 2.2\times$)	1.1 ($\downarrow 2.1\times$)	90.4 ($\downarrow 2.0\times$)
CoST-GCN	4	85.3 (-0.7)	93.1 (-0.3)	3.14	36.0 (79%)	0.27 ($\downarrow 63.2\times$)	32.3 ($\uparrow 14.0\times$)	2375.2 ($\uparrow 13.1\times$)
CoST-GCN*	1	86.3 (+0.3)	93.8 (+0.4)	3.14	36.1 (80%)	0.16 ($\downarrow 107.7\times$)	46.1 ($\uparrow 20.0\times$)	4202.2 ($\uparrow 23.2\times$)
AGCN	300	86.4	94.3	3.47	48.4	18.69	2.1	146.2
AGCN*	300	84.1 (-2.3)	92.6 (-1.7)	3.47	76.4 (158%)	40.87 ($\uparrow 2.2\times$)	1.0 ($\downarrow 2.1\times$)	71.2 ($\downarrow 2.0\times$)
CoAGCN	4	86.0 (-0.4)	93.9 (-0.4)	3.47	37.3 (77%)	0.30 ($\downarrow 63.2\times$)	25.0 ($\uparrow 11.9\times$)	2248.4 ($\uparrow 15.4\times$)
CoAGCN*	1	84.1 (-2.3)	92.6 (-1.7)	3.47	37.4 (77%)	0.17 ($\downarrow 108.8\times$)	30.4 ($\uparrow 14.5\times$)	3817.2 ($\uparrow 26.1\times$)
S-TR	300	86.8	93.8	3.09	74.2	16.14	1.7	156.3
S-TR*	300	86.3 (-0.5)	92.4 (-1.4)	3.09	111.5 (150%)	35.65 ($\uparrow 2.2\times$)	0.8 ($\downarrow 2.1\times$)	85.1 ($\downarrow 1.8\times$)
CoS-TR	4	86.5 (-0.3)	93.3 (-0.5)	3.09	35.9 (48%)	0.22 ($\downarrow 63.2\times$)	30.3 ($\uparrow 17.8\times$)	2189.5 ($\uparrow 14.0\times$)
CoS-TR*	1	86.3 (-0.3)	92.4 (-1.4)	3.09	36.1 (49%)	0.15 ($\downarrow 107.6\times$)	43.8 ($\uparrow 25.8\times$)	3775.3 ($\uparrow 24.2\times$)

Table 4.2: NTU RGB+D 60 transfer accuracy and performance benchmarks. Noted is the top-1 validation accuracy using joints as the only modality. Max mem. is the maximum allocated memory on GPU during inference noted in megabytes. Max. mem, FLOPs, and throughput on CPU account for one new prediction with batch size one, while throughput on GPU uses the largest fitting power of two as batch size. Parentheses indicate the improvement / deterioration relative to the baseline models ST-GCN, AGCN, and S-TR. The table first appeared in paper 2.

reformulations of ST-GCN [214], AGCN [179], and S-TR [152], *i.e.* the proposed CoST-GCN, CoAGCN, and CoS-TR models, variants with stride one and without temporal padding are also created. While the weights of regular networks and their CIN counterpart can be shared, the stride modification requires fine-tuning of the model. As described in Section 4.3.4, this can be done either with full-sequence training or with step-wise training. Both options were explored in paper 2 and found to give similar results.

We tested each baseline model alongside continual padding and stride-free variants on the NTU RGB+D 60 dataset [176]. In addition to accuracy and parameter count, we benchmarked the maximum allocated GPU memory, FLOPs per prediction, as well as throughput on a MacBook Pro 16" 2019 2.6 GHz i7 processor and an NVIDIA RTX 2080Ti GPU. The results of this benchmark are presented in Table 4.2. In each case, the conversion to CIN comes with the desired reduction in FLOPs and a throughput increase. While direct conversion of the baseline models results in a $63\times$ FLOPs reduction per prediction, the removal of zero-padding and reduction of temporal stride (marked with an asterisk '*' in Table 4.2) reduces the per-prediction complexity further, to approx. $108\times$ less FLOPs. While the on-hardware throughput increases do not reach multiple magnitudes of improvement due to the overhead of repeat memory access, they are increased by an average of $14.4\times$ for direct conversion and $22.3\times$ for stride-modified models (marked with '*'). Moreover, maximum allocated memory is reduced by approx. 22% for CoST-GCN and CoAGCN and by 51% for CoS-TR due to the smaller intermediary feature-maps, which exclude the temporal dimension. The accuracy is similar across all models, except for (Co)AGCN*, which has a slightly

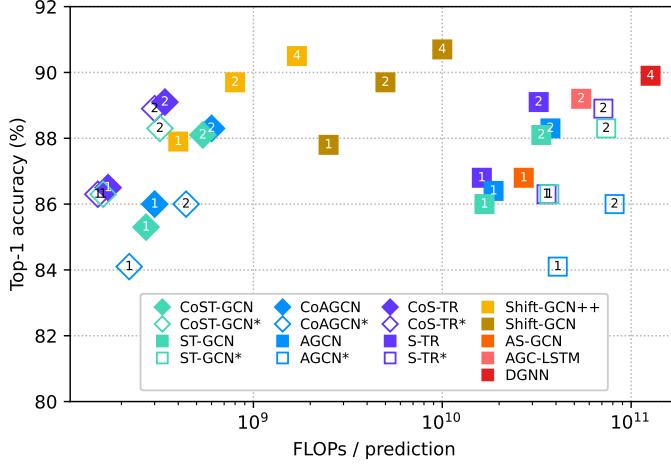


Figure 4.10: Accuracy/complexity trade-off on NTU RGB+D 60 X-Sub for \blacklozenge *Continual* and \blacksquare prior methods during online inference. Numbers denote the number of data streams employed for each evaluation. * Architecture modification with stride one and no padding. The figure first appeared in paper 2.

lower accuracy due to a modification to the self-attention, which restricts it to satisfy causality by only considering the current skeleton rather than all skeletons in the sequence.

Paper 2 also features extensive comparisons with prior works on the NTU RGB+D 60 [176], NTU RGB+D 120 [114], and Kinetics Skeleton 400 [98] benchmarks, where the stride-modified continual models provided the best accuracy/complexity trade-offs for online processing. A visual depiction of the Accuracy/FLOPs trade-off for NTU RGB+D 60 is provided in Fig. 4.10. For additional details, please see paper 2.

4.6 Continual Transformers

4.6.1 Related works

Transformers [199] are among the most influential DNN architecture innovations of the past decade. They are constructed around the core building-block of scaled dot-product attention (SDA), which performs global attention among a set of input tokens. The SDA uses three sets of n tokens denoted query, key, and value, $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{n \times d}$. Given a single input set, $\mathbf{X} \in \mathbb{R}^{n \times d'}$, query, key and value can be retrieved as projections with embedding weights $\mathbf{W}_{\{q,j,v\}} \in \mathbb{R}^{d' \times d}$, i.e., $\mathbf{Q} = \mathbf{X}\mathbf{W}_q$, $\mathbf{K} = \mathbf{X}\mathbf{W}_k$, and $\mathbf{V} = \mathbf{X}\mathbf{W}_v$. The token-wise similarity between key and query matrices is used to aggregate information

from a value matrix as follows:

$$\text{Att}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathbf{D}^{-1} \mathbf{A} \mathbf{V} \quad (4.9)$$

$$\mathbf{A} = \exp\left(\mathbf{Q}\mathbf{K}^\top/\sqrt{d}\right) \quad (4.10)$$

$$\mathbf{D} = \text{diag}\left(\mathbf{A}\mathbb{1}_n^\top\right), \quad (4.11)$$

where $\mathbb{1}_n$ is a row-vector of n ones. Here, the outer product between \mathbf{Q} and \mathbf{K} gives rise to a quadratic computational complexity of $\mathcal{O}(n^2 d)$.

Much work has been dedicated to improving the transformer efficiency. The Image Transformer [143] and Vision Transformer [42] use fixed groups of n_b tokens which attend to one another locally to reduce the complexity to $\mathcal{O}(n_b^2 d)$. As opposed to a fixed grouping scheme, Reformer [101] learns groupings via locality-sensitive hashing to reduce the complexity to $\mathcal{O}(nd \log n)$. Linformer [202], Nyströmformer [210] and Performer [26] approximate the self-attention matrix with a $\mathcal{O}(nd)$ complexity. However, none of the above-mentioned methods optimize for sequential attention, where one new token is considered each time-step during stream processing. During online processing, they are restricted to operate on tokens in a sliding-window, all of which are passed each time a new prediction is desired. Paper 3 presents the first formulation of redundancy-free sequential self-attention, which retains weight-compatibility with the SDA in Eq. (4.11), but is optimized for online processing. Specifically, this work provides a bottom-up reformulation of Transformers as CINs.

4.6.2 Continual Scaled Dot-product Attention

The global interaction between tokens passed to the SDA operation is simultaneously its greatest benefit and the main hindrance to efficient computation. In the continual inference setting, it poses an issue as well: How can we efficiently separate computations across time-steps when the output of the first frame we observed depends on the newest frame and vice versa? As stream processing needs to handle a potentially endless sequence, we have no choice but to restrict the considered sequence to at most n tokens. Furthermore, there are two behavioral options to consider when we process a time-step: Should the *Continual SDA* output (a) n updated tokens retroactively in accordance with a regular SDA used with sliding-window processing, or (b) a single updated token corresponding to a desired token position, *e.g.*, the latest frame or a class token? Paper 3 proposes efficient formulations for both, namely the *Continual Retroactive SDA* and the *Continual Single-output SDA*, which have computational and memory complexities of $\mathcal{O}(nd)$ and produce identical results as Eq. (4.11).

4.6.2.1 Continual Retroactive SDA

Instead of caching \mathbf{A} , which would inflict a memory complexity of $\mathcal{O}(n^2)$, the input sequences \mathbf{Q} , \mathbf{K} , and \mathbf{V} are treated as first-in-first-out queues. Then, the idea is to add contributions from new tokens, $\mathbf{q}_{\text{new}}, \mathbf{k}_{\text{new}}, \mathbf{v}_{\text{new}} \in \mathbb{R}^{1 \times d}$ while subtracting contributions from old tokens, $\mathbf{k}_{\text{old}}, \mathbf{v}_{\text{old}} \in \mathbb{R}^{1 \times d}$, which have slid out of scope. First, we compute

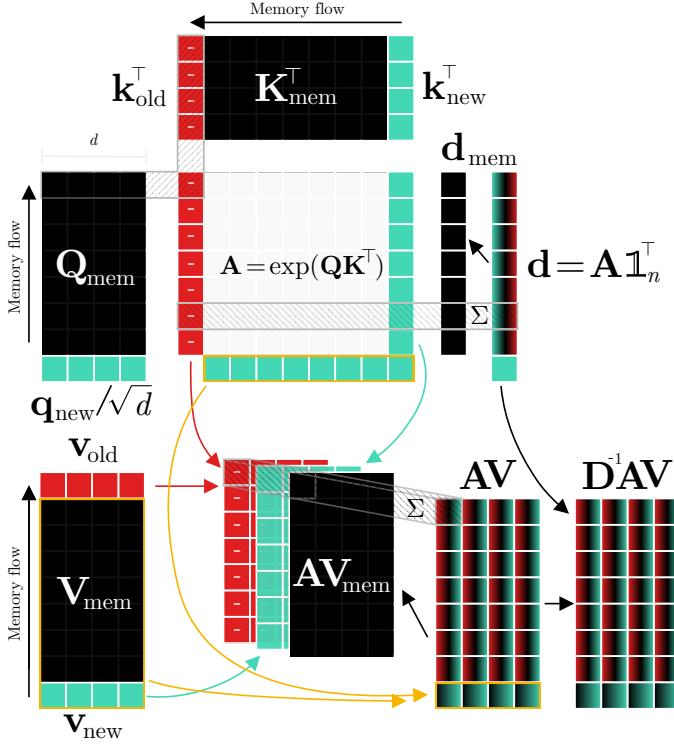


Figure 4.11: Continual Retroactive Dot-Product Attention. The query (\mathbf{Q}), key (\mathbf{K}), and value (\mathbf{V}) matrices are aggregated over time by caching the step vectors \mathbf{q}_{new} , \mathbf{k}_{new} , and \mathbf{v}_{new} in each their FIFO queue (denoted by \square_{mem}). During each step, only the entries of \mathbf{A} associated with \mathbf{q}_{new} , \mathbf{k}_{new} and the oldest \mathbf{K} step, \mathbf{k}_{old} are computed. The diagonal entries of the row-normalization matrix \mathbf{D} as well as the \mathbf{AV} can be updated retroactively by subtracting features corresponding to \mathbf{k}_{old} and adding features related to \mathbf{k}_{new} to the cached outputs of the previous step, \mathbf{D}_{mem} and \mathbf{AV}_{mem} , respectively. The figure first appeared in paper 3.

new entries of \mathbf{A} , namely $\exp(\mathbf{Q}_{\text{mem}} \mathbf{k}_{\text{new}}^{\top})$, as well old entries, $\exp(\mathbf{Q}_{\text{mem}} \mathbf{k}_{\text{old}}^{\top})$. We can use these alongside cached entries of the softmax row-normalization vector, \mathbf{d}_{mem} , to compute and updated row-normalization vector

$$\mathbf{d}^{(-n+1:-1)} = \mathbf{d}_{\text{mem}}^{(-n+2:0)} - \exp(\mathbf{Q}_{\text{mem}} \mathbf{k}_{\text{old}}^{\top}) + \exp(\mathbf{Q}_{\text{mem}} \mathbf{k}_{\text{new}}^{\top}). \quad (4.12)$$

The newest entry is computed from scratch:

$$\mathbf{d}^{(0)} = \exp\left(\frac{\mathbf{q}_{\text{new}}}{\sqrt{d}} (\mathbf{K}_{\text{mem}} \| \mathbf{k}_{\text{new}})^{\top}\right) \mathbb{1}_n^{\top}, \quad (4.13)$$

where $\|$ denotes matrix concatenation. Similarly, the prior \mathbf{AV} is cached as \mathbf{AV}_{mem} and updated based on the outer products of the newly computed entries of \mathbf{A} as well as \mathbf{v}_{new} and \mathbf{v}_{old} :

$$\mathbf{AV}^{(-n+1:-1)} = \mathbf{AV}_{\text{mem}}^{(-n+2:0)} - \exp(\mathbf{Q}_{\text{mem}} \mathbf{k}_{\text{old}}^{\top}) \mathbf{v}_{\text{old}} + \exp(\mathbf{Q}_{\text{mem}} \mathbf{k}_{\text{new}}^{\top}) \mathbf{v}_{\text{new}}. \quad (4.14)$$

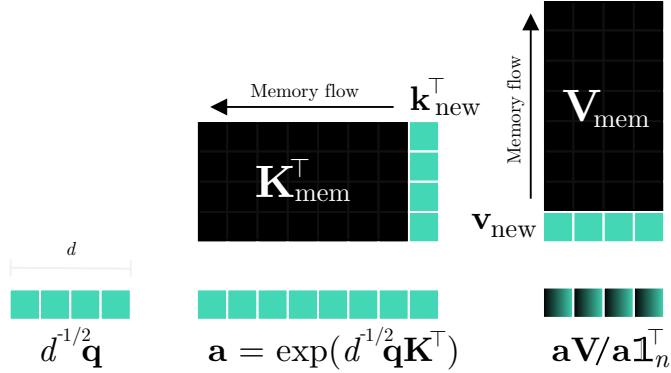


Figure 4.12: Continual Single-Output Dot-Product Attention. The key (\mathbf{K}) and value (\mathbf{V}) matrices are aggregated over time by caching the step vectors \mathbf{k}_{new} and \mathbf{v}_{new} in a FIFO queue. During each step, only the attention output associated with \mathbf{q} is computed. The figure first appeared in paper 3.

The row corresponding to the newest value is computed from scratch:

$$\mathbf{AV}^{(0)} = \exp \left(\frac{\mathbf{q}_{\text{new}}}{\sqrt{d}} (\mathbf{K}_{\text{mem}} \| \mathbf{k}_{\text{new}})^{\top} \right) (\mathbf{V}_{\text{mem}} \| \mathbf{v}_{\text{new}}). \quad (4.15)$$

Finally, the attention output is attained by normalizing \mathbf{AV} with \mathbf{d} :

$$\text{CoReAtt}(\mathbf{q}_{\text{new}}, \mathbf{k}_{\text{new}}, \mathbf{v}_{\text{new}}) = \mathbf{d}^{-1} \odot \mathbf{AV}, \quad (4.16)$$

where \odot denotes element-wise multiplication. The above-described process is graphically illustrated in Fig. 4.11.

4.6.2.2 Single-output SDA

To construct an efficient single-output SDA for continual inference, \mathbf{K} and \mathbf{V} are treated as a first-in-first-out queues again, where the oldest token is discarded when a new token is received. Since only a single output is desired, the attention matrix \mathbf{A} is limited to a single row \mathbf{a} corresponding to the desired step query \mathbf{q} :

$$\mathbf{a} = \exp \left(\frac{\mathbf{q}}{\sqrt{d}} (\mathbf{K}_{\text{mem}} \| \mathbf{k}_{\text{new}})^{\top} \right). \quad (4.17)$$

The attention computation is straight-forward:

$$\text{CoSiAtt}(\mathbf{q}, \mathbf{k}_{\text{new}}, \mathbf{v}_{\text{new}}) = \mathbf{a} (\mathbf{V}_{\text{mem}} \| \mathbf{v}_{\text{new}}) / \mathbf{a} \mathbb{1}_n^{\top}. \quad (4.18)$$

Fig. 4.12 provides a visualization of the process.

4.6.3 Continual Transformer Encoder

The full *Continual Transformer Encoder* architecture is structured in a near-identical manner to the non-continual version, with only minor variations required in the implementation of the *Continual Multi-head Attention* and *Continual Transformer Encoder* block.

4.6.3.1 Continual Multi-head Attention

The *Continual Multi-head Attention* is computed by concatenating the outputs of h continual attention heads for the current query, key, and value vectors projected by $\mathbf{W}_Q^i, \mathbf{W}_K^i \in \mathbb{R}^{d \times d_K/h}$, $\mathbf{W}_V^i \in \mathbb{R}^{d \times d_V/h}$, and performing a per-token linear projection using an output weight, $\mathbf{W}_O \in \mathbb{R}^{d_V \times d_O}$:

$$CoMHA(\mathbf{q}, \mathbf{k}, \mathbf{v}) = \left(\parallel_{i=0}^{h-1} CoAtt(\mathbf{q}\mathbf{W}_Q^i, \mathbf{k}\mathbf{W}_K^i, \mathbf{v}\mathbf{W}_V^i) \right) \mathbf{W}_O. \quad (4.19)$$

Here, $CoAtt(\cdot)$ can either be the retroactive or single-output SDA, $CoReAtt(\cdot)$ or $CoSiAtt(\cdot)$, described in Eq. (4.16) and Eq. (4.18), respectively.

4.6.3.2 Continual Transformer Encoder block

The *Continual Transformer Encoder block* first adds the results of the continual multi-head attention for the current input token \mathbf{x} residually to the input.

$$\mathbf{y} = \text{LayerNorm}(\text{Sel}(\mathbf{x}) + CoMHA(\mathbf{x}, \mathbf{x}, \mathbf{x})), \quad (4.20)$$

Here, we use the notation of a selection function, $\text{Sel}(\cdot)$, to note that only a single (*e.g.*, last) token should be selected when the Continual Single-output approach is used. The final output is then given by token-wise projection with a two-layer feed-forward network:

$$\mathbf{z} = \text{LayerNorm}(\mathbf{y} + \text{FF}(\mathbf{y})) \quad (4.21)$$

$$\text{FF}(\mathbf{x}) = \text{ReLU}(\mathbf{x}\mathbf{W}_1 w_1)\mathbf{W}_2 + w_2. \quad (4.22)$$

4.6.3.3 Recycling Positional Encoding

Transformer Encoders do not have inherent positional bias. Hence, it is common to augment input tokens \mathbf{x}_i with positional encodings \mathbf{p}_i . However, for regular transformers, the index i denotes a position in a sequence rather than a position in time. This poses an issue, since the last token at time t will be next-to-last at time $t+1$, and thus in need of a different positional encoding. Stated another way, each new input token step also changes the required positional encoding of prior steps. In effect, caching of prior values is not an option under the common absolute positional encoding. As a solution, the position should be fixed in time rather than in sequence. After T steps have passed, the positional encodings can then be recycled:

$$\tilde{\mathbf{x}}_t = \mathbf{x}_t + \mathbf{p}_{\tau_t}, \quad \tau_t = (\tau_{t-1} + 1) \bmod T. \quad (4.23)$$

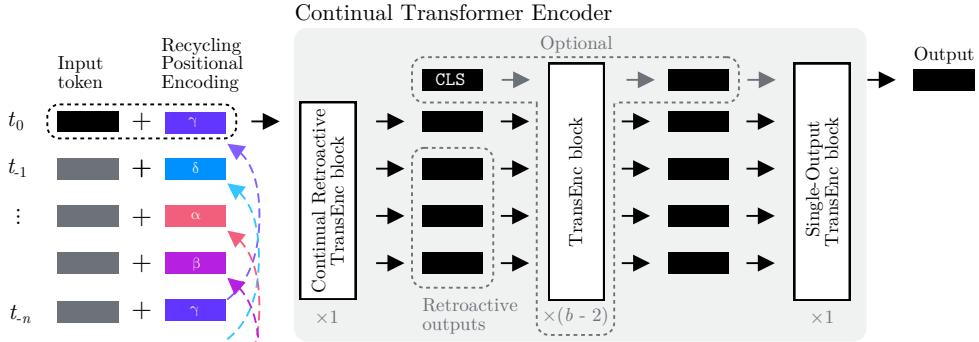


Figure 4.13: Multi-block Continual Transformer Encoder with Recycling Positional Encoding. For $b > 2$ blocks, regular Transformer Encoder blocks can be added between an initial Continual Retroactive block and a final Single-Output block. A class-token may be used after the initial block. The figure first appeared in paper 3.

In the ablation studies conducted in paper 3, the use of this *Recycling Positional Encoding* was found to be crucial to attain good performance during continual inference.

4.6.3.4 Architectural considerations

The full continual Transformer Encoder architecture with Recycling Positional Encoding and Continual Transformer Encoder blocks is illustrated in Fig. 4.13. If a *single* transformer encoder block is desired, the Continual Single-output Transformer Encoder block is ideal for classification tasks and the Continual Retroactive Transformer Encoder block for sequence to sequence tasks. For *two* blocks, a Continual Retroactive Transformer Encoder block should be employed first and a regular Single-output Transformer Encoder last. For *three blocks or more*, the same setup applies for the first and last block, but regular transformer encoders can be stacked in between. Due to the one-to-many nature of Continual Retroactive Transformer Encoders, their efficiency gains quickly diminish as more blocks are added. Consequently, we recommend the use of one- or two block architectures. While this may seem restrictive, concurrent work has shown that shallow Transformer architectures may perform as well as deeper networks if the parameter count is matched by scaling out the number of attention heads [15].

4.6.4 Experiments

The Continual Transformer Encoders are validated in the context of *Online Action Detection* [29], where the task is to predict per-frame human action categories without peeking into the future. Here, recent works primarily utilize pre-extracted features from two CNN streams trained on RGB images and optical flow fields, respectively, and aggregate the temporal information via RNNs [47, 55, 211] or Transformers [203, 212]. Action anticipation is a related task, which may be learned in parallel, as it has been found to improve OAD as well. In the experiments with Continual Transformer

Model	Feat.	THUMOS14 mAP (%)	TVSeries mcAP (%)	FLOPs (M)
RED [55]	A.Net	45.3	79.2	-
TRN [211]		47.2	83.7	1387.5
FATS [100]		51.6	81.7	-
IDN [47]		50.0	84.7	-
TFN [48]		55.7	85.0	-
LSTR [212]		65.3	88.1	-
OadTR [203]		58.3	85.4	2445.6
OadTR [†]		57.0 \pm 0.5	88.6\pm0.1	2445.6
OadTR-b2 [†]		56.6 \pm 0.3	88.3\pm0.2	1008.1
OadTR-b1 [†]		56.3 \pm 0.2	88.1 \pm 0.1	605.5
<i>Co</i> OadTR-b2 (ours)	Kin.	56.8 \pm 0.4	87.7 \pm 0.6	410.9
<i>Co</i> OadTR-b1 (ours)		56.1 \pm 0.7	87.6 \pm 0.7	9.6
TRN [211]		62.1	86.2	1462.0
FATS [100]		59.0	84.6	-
IDN [47]		60.3	86.1	-
PKD [228]		64.5	86.4	-
LSTR [212]		69.5	89.1	-
OadTR [203]		65.2	87.2	2513.5
OadTR [†]		64.2 \pm 0.3	88.6\pm0.1	2513.5
OadTR-b2 [†]		64.5 \pm 0.5	88.3 \pm 0.2	1075.7
OadTR-b1 [†]		63.9 \pm 0.5	88.1 \pm 0.1	673.0
<i>Co</i> OadTR-b2 (ours)		64.4 \pm 0.1	87.6 \pm 0.7	411.9
<i>Co</i> OadTR-b1 (ours)		64.2 \pm 0.4	87.7 \pm 0.4	10.6

Table 4.3: Online Action Detection results. FLOPs per prediction are noted for inference on THUMOS14. The **best** and **next-best** metrics are highlighted. [†] indicates use official source code or modifications there-off. The table first appeared in paper 3.

Encoders, a simplified OadTR [203] architecture was employed, which did not include the action anticipation decoder and class token, and which used fewer Transformer Encoder blocks (denoted by b1 and b2 for one and two blocks). In the ablation studies conducted in paper 3, these modifications were found to have limited effect on predictive performance despite reducing computational complexity significantly.

A comparison with prior works on the THUMOS14 [92] and TVSeries [29] datasets is presented in Table 4.3. Please see paper 3 for descriptions of experimental methodology. Two sets of results are shown in Table 4.3 using features from a Temporal Segment Network [201] pre-trained on either ActivityNet [83] (top) or Kinetics-400 [98] (bottom). Compared to OadTR, the proposed *Co*OadTR-b1 and *Co*OadTR-b2 reduce FLOPs by 255 \times and 6.1 \times , respectively. While minor conces-

sions to the predictive performance are observed (at most 1% point), these are expected given the lower parameter count of the b1 and b2 models. Comparing continual and non-continual simplified OadTR-b{1,2} implementations, *i.e.*, near-identical architectures which either use sliding-window processing or continual inference processing, the *CoOadTR-b1* reduces FLOPs by approx. $63\times$ and the *CoOadTR-b2* by $2.5\times$. The discrepancy in efficiency gains is explained by the additional computations needed in the two-block architecture to aggregate retroactively computed tokens.

Additional results for audio classification on GTZAN [198] as well as ablation studies on positional encoding, number of blocks, and removal of decoder and class token are available in paper 3.

4.7 A Python library to support implementation

While the architectural reformulations presented in this chapter may be simple enough conceptually, a considerable engineering effort is required to handle network delays and inputs shapes properly to retain weight-compatibility between offline and online inference versions. To aid this effort, paper 4 presents a Python library, *Continual Inference* (with shorthand *co*), which augments the standard PyTorch *nn* modules with the ability to perform efficient continual inference.

The library is designed to adhere closely to the CIN definition in Section 4.3.1 and to support the necessary architectural considerations outlined in Section 4.3.4. Moreover, the library is implemented with strict PyTorch compatibility as outlined in implementation principles 1 and 2.

Principle 1 (Compatibility with PyTorch) *co* modules with identical names to *nn* modules also have:

1. *identical forward*,
2. *identical model weights*,
3. *identical or extended constructors*,
4. *identical or extended supporting functions*.

Principle 2 (Call modes) *co* modules provide three forward operations:

1. *forward*: takes a (spatio-) temporal input and operates identically to the *forward* of an *nn* module,
2. *forward_step*: takes a single time-step as input without a time-dimension and produces an output corresponding to *forward*, had its input been shifted by one time-step, given identical prior inputs.

3. **`forward_steps`**: takes multiple time-steps as input and produces outputs identical to applying `forward_step` the number of times corresponding to the temporal size of the input.

These principles are best understood with an example, as presented in Fig. 4.14. The *Continual Inference* library features a comprehensive re-implementation of basic `nn` network modules as well as an enhanced set of composition modules such as `co.Residual`, which automatically matches the delay of the wrapped module. For more details on network modules and advanced uses, please see paper 4 and the library code at <https://github.com/lukashedegaard/continual-inference>.

```

•••
1 import torch
2 from torch import nn
3 import continual as co
4
5 # Define net
6 net = co.Residual(
7     co.Sequential(
8         co.Conv3d(32, 64, kernel_size=(1, 1, 1)),
9         nn.BatchNorm3d(64),
10        nn.ReLU6(),
11        co.Conv3d(64, 64, kernel_size=(3, 3, 3), padding=(1, 1, 1), groups=64),
12        nn.ReLU6(),
13        co.Conv3d(64, 32, kernel_size=(1, 1, 1)),
14        nn.BatchNorm3d(32),
15    )
16 )
17 net.eval()
18
19 # Example input
20 x = torch.randn((1, 32, 7, 5, 5)) # (B, C, T, H, W)
21
22 # Multiple forward models
23 y = net.forward(x) # Offline inference, identical to torch.nn
24 y_firsts = net.forward_steps(x[:, :, :6]) # Multiple steps, e.g., for init
25 y_last = net.forward_step(x[:, :, 6]) # Single step, for online inference
26
27 # Same results
28 assert torch.allclose(y[:, :, :-net.delay - 1], y_firsts, atol=1e-7)
29 assert torch.allclose(y[:, :, -net.delay - 1], y_last, atol=1e-7)
30
31 # Temporal properties
32 assert net.receptive_field == 3
33 assert net.delay == 1

```

Figure 4.14: Depth-wise separable 3D convolution block, `net`, implemented with the *Continual Inference* library. Note the mix of `nn` and `co` modules (lines 8-14). The library modules feature multiple forward modes for offline and online inference (lines 23-25), which produce the same results (lines 28-29). Moreover, modules are enhanced with temporal properties (lines 32-33).

4.8 Conclusion

This chapter provided an introduction to Continual Inference Networks (Section 4.3.1), a class of neural networks tailored for redundancy-free online processing of never-ending (continual) data streams. They tackle the issue of computational redundancy observed during sliding-window processing of spatio-temporal networks (*e.g.*, 3D CNNs, Spatio-temporal GCNs, and Transformers). Through the use of temporal redistributions of computational operations and minor architectural modification, many prior DNNs, which were previously required to use sliding-window processing for online execution, can be reformulated to operate in a sequential manner to efficiently process online data streams.

After outlining the principles of CINs, an overview was given of *Continual 3D CNNs* (Section 4.4, paper 1), a general reformulation of the 3D CNN. Through conversions of multiple prior state-of-the-art 3D CNNs for human activity recognition, Continual 3D CNNs achieved up to $15\times$ FLOPs reduction per prediction and $9\times$ and $7\times$ throughput increase on CPU and GPU for networks with a clip-size of sixteen without sacrificing accuracy or retraining the networks.

Continual Spatio-temporal GCNs (Section 4.5, paper 2) are the augmentation of the ST-GCN network family, which brings efficient online processing to spatio-temporal graphs. Here, architectures which are weight-compatible with prior ST-GCN-based methods, achieve $63\times$ FLOPs reductions and up to $18\times$ and $15\times$ throughput increase on CPU and GPU, respectively, with identical accuracy on clips of 300 frames. *CoST-GCN*-based networks with minor architectural modifications can be fine-tuned with only few training epochs to achieve similar accuracy, FLOPs reductions of $108\times$, and throughput increases up to $26\times$ and $24\times$ on CPU and GPU.

Continual Transformer Encoders (Section 4.6, paper 3) provided a bottom-up reformulation of the Transformer and its scaled dot-product attention, which reduced the per-step computational complexity from $\mathcal{O}(n^2d)$ to $\mathcal{O}(nd)$, where n is the sequence receptive field and d is the token dimension. This yielded reductions in FLOPs per prediction of $63\times$ and $2.5\times$ for one- and two-block transformers, respectively, on token sequences of length 64. Through a systematic ablation of a prior state-of-the-art method, OadTR, a one-block *CoOadTR* architecture achieved a relative FLOPs reduction of $255\times$ with only 0.7%-point lower average predictive performance on online action detection benchmarks.

Finally, the *Continual Inference* library (Section 4.7, paper 4) provides a comprehensive re-implementation of basic PyTorch DNN components which were augmented with the ability to process temporal data efficiently in a sequential manner. In addition, the library features an augmented composition interface, which automatically handles delays and aggregates temporal properties to guarantee compliance with the principles of Continual Inference Networks.

Chapter 5

Compressing and accelerating derived networks with Structured Pruning Adapters

5.1 Introduction

Since the inception of deep neural networks (DNNs), there has been an unfailing trend of steadily increasing model size and amount of training data in pursuit of improved predictive performance (see Fig. 1.1). This has reached a point where the largest models require training on the order of one million GPU hours with NVIDIA A100 GPUs [173]. However, the computational resources needed to train such DNNs are beyond the capacity of most public institutions and universities. While efforts such as BLOOM [173] showcase the possibility of collective training, the large private research labs such as OpenAI and DeepMind currently lead the race. Still, even the largest commercial players will eventually be confronted with severely diminishing returns for the money spent on model training.

Faced with economic and environmental realities, a possible scenario for the 2030s is a gradual slow-down in the trend of progressively larger models and longer intervals between the release of new state-of-the-art DNNs trained on large-scale datasets. The increasingly common term of a *foundation model* itself indicates something solid and slow or unchanging. After all, who wants a shaky foundation? With a smaller diversity in state-of-the-art base-models, their adaptation to use-case specific deployments will likely become the primary focus of most deep learning practitioners. In such a scenario, the practical focus is to achieve the best trade-offs between predictive, computational, and storage efficiency. This issue leads to the primary research question of this chapter:

RQ2 *How can we make parameter-efficient adaptations to pre-trained models while improving their computational efficiency on derived tasks?*

5.2 Related works

5.2.1 Feature-extraction and fine-tuning

The adaptation of pre-trained models to novel tasks, *i.e.*, transfer learning, can take on many forms. *Feature-extraction* methods fix the model parameters of the first model layers, which act as feature extractors. Then either end-layers or a linear weighted combination of intermediary representations can be trained [147] on new tasks. In contrast, *fine-tuning* resumes training of the whole network and only re-initializes the required prediction layer. In computer vision, fine-tuning generally performs better than feature-extraction, at least for transfers from ImageNet-1k to other tasks [105]. However, the usability of fine-tuning is reduced when tasks deviate from the source task [67]. A similar trend can be observed in natural language processing [149]. Multiple variants and extensions of fine-tuning have been explored. These include *discriminative fine-tuning* [89], where later layers, which poses the least general knowledge [219], are trained with higher learning rate than early layers; *gradual fine-tuning* [89], where the last layer is trained initially and earlier layers are gradually added to the set of trainable layers; and *chain-thaw*, which alternates the fine-tuning of individual layers while other layers fixed [54].

5.2.2 Adapters

An alternative to training the original model parameters are *adapters* [162]. These can be used for transfer learning by adding and training a small set of new model parameters throughout the source network while keeping prior weights fixed. While both adapters and feature-extraction methods leave original model weights fixed, adapters modify the intermediary network representations and feed them back to the network. Adapters are particularly suitable when multiple networks are derived from a single source model. Since they learn tasks with orders of magnitude fewer parameters than fine-tuning, they are also significantly easier to share if users already possess the original source weights. This has important uses in both federated learning, where the transfer of gradients between devices in a network is the main training bottleneck [45], as well as for adaptation of foundation models and sharing of adapted weights. For instance, a fine-tuned BLOOM model requires 330GB of storage space, which may make the storage of many fine-tuned model variants untenable, whereas multiple adapters can be stored at low cost.

Adapters for CNNs were proposed with point-wise convolutions in series [162] or parallel [163] with the residual connections. Bottle-neck projection layers in parallel with dense source layers [90, 232] or distributed in series with existing layer [87, 122, 150] have been explored in the adaptation of transformers.

While adapters learn only a fraction of parameters compared to the source model size, the computational complexity of adapted networks remains as high or higher than the source model. If derived tasks, which are often simpler than source tasks, can be learned by training only a few parameters, should it not then also be possible to

use fewer active parameters at inference?

5.2.3 Fine-pruning

Chapter 3 outlined common approaches to making model inference more efficient, one of which is pruning (Section 3.1). While most works prune networks with the goal of producing a light-weight network to perform the same task as the unpruned network, pruning can also be used to learn new tasks. In fact, appropriate selection of network weights without further training is enough to adapt to new tasks [123, 160], even if it does not achieve optimal predictive performance. Better target task performance can be achieved with a combination of pruning and fine-tuning, *i.e.*, *fine-pruning* [172], which reduces the parameter count of the trained model while modifying network parameters. However, the resulting model does not necessarily possess better inference characteristics as such depends on the pruning approach used. As described in Section 3.1, accelerating inference on GPUs requires a structured pruning method. Structured fine-pruning has been explored with both channel-pruning [218] and block-pruning [110]. While these methods successfully learn derived networks with both fewer parameters and improved inference efficiency compared with the source network, it is possible to further improve the parameter:accuracy:FLOPs trade-offs through the use of adapters alongside structured pruning.

5.3 Structured Pruning Adapters

Structured Pruning Adapters, which were proposed in paper 7, learn a set of L structurally sparse weight matrices $\{\mathbf{W}_t\}_{t \in 1..L}$ for the target task based on the adaptation $a(\cdot)$ of dense source matrices $\{\mathbf{W}_s\}_{s \in 1..L}$, each using a small set of target weights, $\Delta\mathbf{W}_t$, where $\mathbf{W}_t, \mathbf{W}_s \in \mathbb{R}^{n \times m}$ and $|\Delta\mathbf{W}_t| \ll |\mathbf{W}_s|$. Moreover, a structured binary pruning mask $\mathbf{M}_t \in \{0, 1\}^{n \times m}$ is used. Target weights are derived by fusing source weights with a parallel adapter and applying the binary mask with element-wise multiplication (\odot):

$$\mathbf{W}_t = (\mathbf{W}_s + a(\Delta\mathbf{W}_t)) \odot \mathbf{M}_t. \quad (5.1)$$

In particular, channel-based pruning, where full rows and columns of the matrix \mathbf{M}_t are zero-valued, is expressed as:

$$\mathbf{W}_t = (\mathbf{W}_s + a(\Delta\mathbf{W}_t)) \odot \mathbf{m}_{\text{row}} \mathbf{m}_{\text{col}}^\top. \quad (5.2)$$

5.3.1 Structured Pruning Low-rank Adapter

A simple yet powerful fusible adapter realization is the Low-rank Adapter [90]

$$\mathbf{W}_t = \mathbf{W}_s + \mathbf{W}_{\text{down}} \mathbf{W}_{\text{up}}, \quad (5.3)$$

which uses a bottle-neck projection, $\mathbf{W}_{\text{down}} \mathbf{W}_{\text{up}}$, where $\mathbf{W}_{\text{down}} \in \mathbb{R}^{n \times r}$, $\mathbf{W}_{\text{up}} \in \mathbb{R}^{r \times m}$, and $r \ll n, m$, to adapt source weights. Combining Eq. (5.2) and Eq. (5.3) by setting

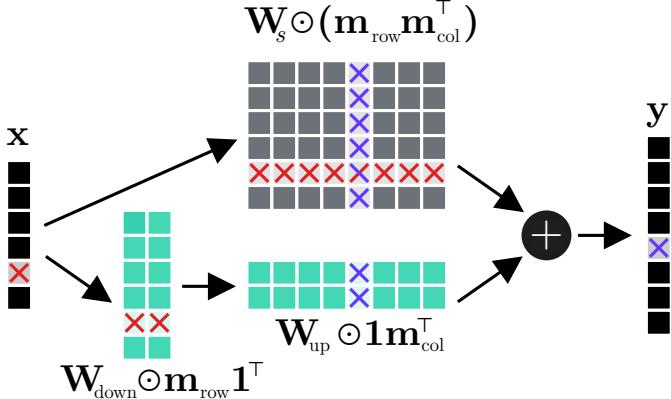


Figure 5.1: Structured Pruning Low-rank Adapter (SPLoRA). The in/out channel pruning mask affects the adapter as well as source weights.

$a(\Delta\mathbf{W}_t)) = \mathbf{W}_{\text{down}}\mathbf{W}_{\text{up}}$, we can define the *Structured Pruning Low-rank Adapter (SPLoRA)*:

$$\mathbf{W}_t = \mathbf{W}_s \odot \mathbf{m}_{\text{row}} \mathbf{m}_{\text{col}}^\top + (\mathbf{W}_{\text{down}} \odot \mathbf{m}_{\text{row}} \mathbf{I}^\top)(\mathbf{W}_{\text{up}} \odot \mathbf{I} \mathbf{m}_{\text{col}}^\top). \quad (5.4)$$

Here, row and column pruning masks, \mathbf{m}_{row} and \mathbf{m}_{col} , explicitly prune both \mathbf{W}_s and the adapter weights, \mathbf{W}_{down} and \mathbf{W}_{up} . This is illustrated in Fig. 5.1. For additional details including a derivation of Eq. (5.4) please see paper 7.

5.4 Experiments

Paper 7 proposed SPLoRA as an alternative to fine-tuning under channel-based pruning. Both learning methods have identical run-time characteristics considering that fused SPLoRA weight contain similar structure and parameter count as fine-pruned weights. Accordingly, the experiments are centered around comparisons of parameter count and predictive accuracy. Specifically, a ResNet-50 network pre-trained on ImageNet-1k is adapted to three image classification tasks, CIFAR-10 [107], Oxford Flowers 102 [137], and Cats and Dogs [44], across four pruning criteria based on weight magnitude [111], gradient [187], a Taylor expansion approximating changes to the cost function [133], and layer-wise relevance propagation (LRP) [218]. Details on experimental protocol and parameter sensitivity are available in paper 7.

Table 5.1 present experimental results of the average accuracy across datasets shown for each combination of learning and pruning method at 30% and 10% weight density, *i.e.*, at respectively 70% and 90% pruned weights. Here, SPLoRA-*r*8, *i.e.*, SPLoRA with rank 8 bottle-neck, requires $19.38\times$ fewer parameters on average at 30% density with only 1.60% lower accuracy. At 10% pruning, the model requires $4.65\times$ fewer parameters while increasing accuracy by 5.61% on average. Likewise, the SPLoRA-*r*32 configuration requires $6.65\times$ fewer parameters with only 0.54% lower accuracy on average at 30% density. At 10% it uses $2.05\times$ fewer parameters

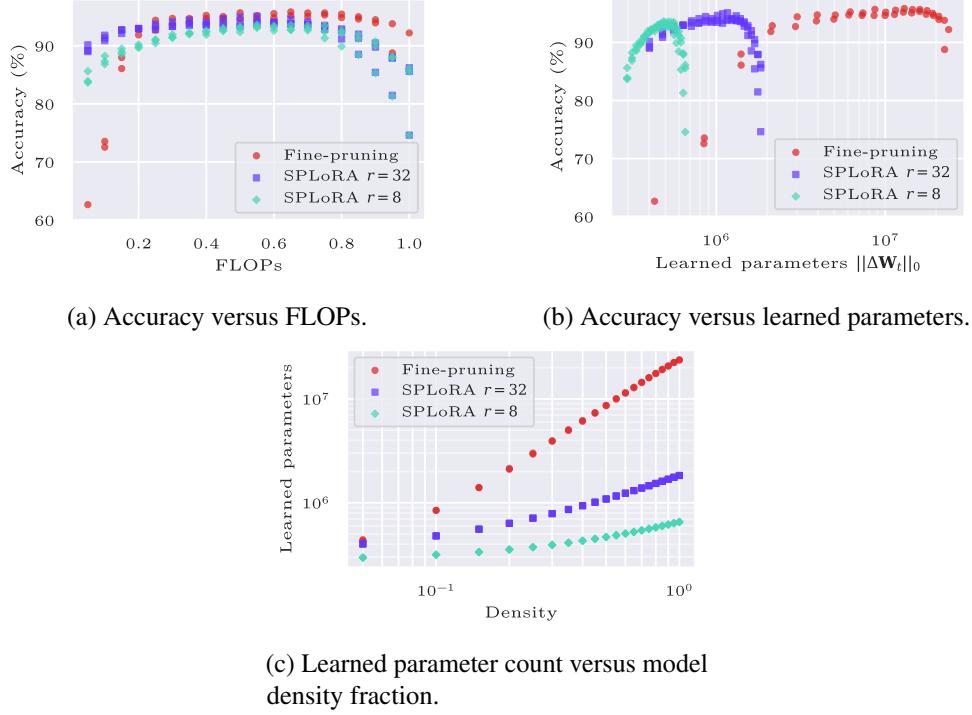


Figure 5.2: Trade-offs between accuracy (%), learned parameter count ($\|\mathbf{W}_t\|_0$) and floating-point operations (FLOPs) on Oxford Flowers 102 using fine-pruning (●) and SPLoRA with ranks 32 (■) and 8 (◆) for gradient-based [187] channel pruning.

while improving accuracy by 8.15%. As observed in Fig. 5.2c as well as Table 5.1, the relative parameter efficiency of SPLoRA adapters becomes lower when more aggressive pruning is used. However, SPLoRA retains accuracy more effectively than fine-tuning as the network is pruned (Fig. 5.2a) and generally exhibits significantly more desirable trade-offs between predictive accuracy and learned parameter count (Fig. 5.2b).

5.5 Conclusion

This chapter presented the work on Structured Pruning Adapters (paper 7). In particular, the proposed channel-based method, SPLoRA, was benchmarked against fine-tuning with channel-pruning with a battery of pruning criteria and on multiple transfer learning tasks in the image domain. SPLoRA systematically exhibited superior parameter efficiency, requiring $19\times$ fewer parameters on average at 30% remaining weight with only 1.60% lower accuracy. In the low-density regimen at 10% remaining weights, SPLoRA retained accuracy better than fine-pruning, achieving 8.15% point higher accuracy on average while requiring $2.05\times$ fewer learned parameters.

Pruning method	Learning method	Density	ΔParams (K)	FLOPs (M)	Avg. accuracy (%)
Unpruned	Fine-tuning	100%	23,520.8 \pm 0.0	1,304.7 \pm 0.0	96.20 \pm 0.05
	LoRA- <i>r</i> 32	100%	1,644.5 \pm 0.0 (\downarrow 14.3 \times)	1,304.7 \pm 0.0	90.83 \pm 2.16 (-5.37)
	LoRA- <i>r</i> 8	100%	466.3 \pm 0.0 (\downarrow 50.4 \times)	1,304.7 \pm 0.0	91.72 \pm 2.15 (-4.48)
Weight	Fine-pruning	30%	4,427.1 \pm 72.5	785.4 \pm 5.4	96.22 \pm 0.77
		10%	599.1 \pm 20.1	352.1 \pm 3.9	85.16 \pm 1.08
	SPLoRA- <i>r</i> 32	30%	618.3 \pm 2.5 (\downarrow 7.2 \times)	778.6 \pm 2.9	96.20 \pm 0.31 (-0.02)
		10%	294.8 \pm 0.4 (\downarrow 2.0 \times)	335.4 \pm 7.3	92.88 \pm 0.82 ($+7.72$)
	SPLoRA- <i>r</i> 8	30%	210.0 \pm 1.9 (\downarrow 21.1 \times)	773.4 \pm 2.1	95.15 \pm 0.17 (-1.07)
		10%	128.9 \pm 0.1 (\downarrow 4.6 \times)	338.9 \pm 7.0	91.27 \pm 0.65 ($+6.11$)
Gradient	Fine-pruning	30%	3,719.8 \pm 59.2	571.9 \pm 6.5	96.13 \pm 0.28
		10%	615.7 \pm 4.3	244.7 \pm 3.3	83.58 \pm 0.69
	SPLoRA- <i>r</i> 32	30%	601.0 \pm 0.4 (\downarrow 6.2 \times)	564.6 \pm 1.5	95.55 \pm 0.20 (-0.58)
		10%	293.1 \pm 0.4 (\downarrow 2.1 \times)	244.0 \pm 1.9	94.18 \pm 0.33 ($+10.60$)
	SPLoRA- <i>r</i> 8	30%	205.2 \pm 0.2 (\downarrow 18.1 \times)	565.2 \pm 4.1	94.61 \pm 0.22 (-1.52)
		10%	128.3 \pm 0.0 (\downarrow 4.8 \times)	245.4 \pm 4.0	91.97 \pm 0.40 ($+8.39$)
Taylor	Fine-pruning	30%	3,392.8 \pm 81.1	559.9 \pm 0.7	95.61 \pm 0.25
		10%	576.8 \pm 9.9	236.9 \pm 3.3	83.01 \pm 1.66
	SPLoRA- <i>r</i> 32	30%	599.7 \pm 0.9 (\downarrow 5.7 \times)	555.5 \pm 6.7	95.37 \pm 0.25 (-0.24)
		10%	292.6 \pm 0.5 (\downarrow 2.0 \times)	242.0 \pm 1.5	93.93 \pm 0.04 ($+10.92$)
	SPLoRA- <i>r</i> 8	30%	205.3 \pm 0.1 (\downarrow 16.5 \times)	566.2 \pm 10.9	94.46 \pm 0.29 (-1.15)
		10%	128.4 \pm 0.1 (\downarrow 4.5 \times)	243.2 \pm 9.8	91.60 \pm 0.35 ($+8.59$)
LRP	Fine-pruning	30%	4,428.1 \pm 20.6	719.9 \pm 0.7	96.85 \pm 0.10
		10%	608.4 \pm 6.3	301.6 \pm 2.3	90.54 \pm 1.40
	SPLoRA- <i>r</i> 32	30%	592.6 \pm 0.9 (\downarrow 7.5 \times)	585.5 \pm 6.7	95.52 \pm 0.21 (-1.33)
		10%	290.9 \pm 0.2 (\downarrow 2.1 \times)	270.4 \pm 6.4	93.90 \pm 0.36 ($+3.36$)
	SPLoRA- <i>r</i> 8	30%	203.3 \pm 0.5 (\downarrow 21.8 \times)	591.1 \pm 12.4	94.20 \pm 0.19 (-2.65)
		10%	128.0 \pm 0.1 (\downarrow 4.7 \times)	281.6 \pm 1.7	91.17 \pm 0.77 (-0.64)

Table 5.1: Channel-based transfer-pruning from ResNet-50 pre-trained on ImageNet to CIFAR-10, Oxford Flowers 102, and Cats & Dogs using pruning methods based on Weight [111], Gradient [187], Taylor [133], and LRP [218]. Please note that SPLoRA and LoRA are identical at 100% density. ΔParams and floating-point operations (FLOPs) are shown for CIFAR-10. Mean \pm standard deviation over three runs is shown for each metric. The average accuracy over datasets is noted as the average of means and standard deviations within a dataset. Changes relative to closest fine-pruning density with same pruning method are specified in parentheses. The best metric for each combination of pruning-method and density is highlighted with bold.

Chapter 6

Conclusions and future work

Deep neural networks (DNNs) play an ever more important role in many technological solutions we use everyday. Unlocking phones with facial scans, analyzing photo libraries to suggest compilations, and ensuring our safety while driving by monitoring the road gradually become taken for granted. However, the size of DNNs and energy expenditure associated with their deployment increases at an accelerating pace. While much work in the field has focused primarily on applying DNNs on novel problems or improving predictive performance on prior ones, this dissertation represents an effort to make DNNs more computationally and memory efficient. Hopefully, these improvements will translate to more power and hardware efficient deployment of DNNs.

Specifically, Chapter 4 describes efforts to accelerate the inference of spatio-temporal DNNs on time-series processing tasks during online deployment. These include the development of a new framework of neural network architectures, that operate efficiently on continual streams of data. The theory and experimental validation of *Continual Inference Networks* (CINs), as they were dubbed, was developed over publications 1 to 4, each of which tackled the translation of state-of-the-art architectures for different offline inference tasks to the context of online stream processing.

Continual 3D Convolutional Neural Networks (Section 4.4, paper 1) translate 3D CNNs, which in prior formulations were hamstrung to operate exclusively of full spatio-temporal clips, to a sequential formulation, where each time-step is processed incrementally. Continual 3D CNNs reduce the floating-point operations (FLOPs) by up to $15\times$ and increase throughput up to $9\times$ on CPU and $7\times$ on GPU during online inference compared to ordinary 3D CNNs applied with a sliding-window approach. Importantly, the reformulation is weight-compatible with prior architectures.

Continual Spatio-temporal Graph Convolutional Networks (Section 4.5, paper 2) are the corresponding translation of ST-GCN-based works for classification of spatio-temporal graphs. Since the CIN formulation generally offers improvements in computational complexity in proportion to the corresponding temporal receptive field during sliding-window processing with non-CINs, Continual ST-GCN formulations achieve

up to $108\times$ FLOPs reductions as well as a $26\times$ and $24\times$ throughput increase on CPU and GPU for networks with the commonly used clip size of 300 time-steps.

Continual Transformer Encoders (Section 4.6, paper 3), the CIN reformulation of the Transformer Encoder, likewise offer significant benefits during online processing. By utilizing novel computational schemes for the scaled dot-product attention, a one-block Continual Transformer Encoder can reduce computational complexity by $63\times$ for a sequence length of 64. A two-block version reduces complexity by $2.5\times$.

To support the implementation of CINs, the python library *Continual Inference* was implemented (Section 4.7, paper 4). This augments the standard components of PyTorch with the ability to perform the forward operation efficiently in a progressive manner, time-step by time-step.

Though CINs have left their infancy, there are still many avenues of future work. One such avenue is the exploration of CINs for uses other than classification, *e.g.*, for efficient object detection and tracking or video-based anomaly detection. Moreover, CINs in their current form assume that data in the input stream is uniformly sampled in time. However, data streams with unevenly spaced samples can easily occur on embedded devices, where the limited computational power may occasionally be diverted to other tasks than data acquisition and processing. The development of CINs that operate gracefully in this context would be of great value to such deployments. The current publications and above-described future works on CINs assume hand-crafted translation from ordinary networks to efficient CINs. However, it may be possible to automatically identify similar architectural modifications through automated analysis of the computational graph for consecutive time-steps. Finally, a great deal of work on community outreach is needed to make an impact with CINs, including the creation of tutorials, blogposts, and collaborations with other tools in the machine learning ecosystem.

Chapter 5 and paper 7 presented an orthogonal work on *Structured Pruning Adapters* (SPAs). Motivated by scenarios where a large source model is adapted to many down-stream tasks, SPAs were proposed as an alternative to fine-tuning with pruning. SPAs utilize adapters to learn downstream tasks with an order of magnitude fewer parameters, provided the source model weights are also available. Through the combination of channel-based pruning and fusible parallel adapters, the proposed channel-SPAs required $19\times$ fewer learned parameters at a modest 1.60% lower accuracy at moderate pruning of 30% remaining weights. Under aggressive pruning at 10% remaining weights, the channel-SPAs retained an 8.15%-point higher accuracy than fine-tuning while requiring $2.05\times$ fewer learned parameters.

The work described in Chapter 5 has just scratched the surface of viable methods in the intersection between structured pruning and low-parameter adapter networks. Specifically, SPAs for block-pruning and N:M pruning are interesting directions of future research. Moreover, an explanation of why the channel-SPAs systematically retained their predictive performance better than fine-tuning in the highly pruned regimen is yet to be found. An experimentally validated explanation of this phenomenon could be important in furthering the pruning research field and might enable discovery of even more efficient pruning methods.

Bibliography

- [1] Chatgpt reaches 100 million users two months after launch, 2023. The Guardian, <https://www.theguardian.com/technology/2023/feb/02/chatgpt-100-million-users-open-ai-fastest-growing-app>. 4
- [2] Llm failure archive. <https://github.com/giuven95/chatgpt-failures>, 2023. Accessed: 2023-02-24. 3
- [3] Neuralmagic. <https://neuralmagic.com>, 2023. Accessed: 2023-02-06. 11
- [4] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL <https://www.tensorflow.org/>. Software available from tensorflow.org. 7, 14, 23
- [5] Madhu S. Advani, Andrew M. Saxe, and Haim Sompolinsky. High-dimensional dynamics of generalization error in neural networks. *Neural Networks*, 132:428–446, 2020. ISSN 0893-6080. doi: <https://doi.org/10.1016/j.neunet.2020.08.022>. URL <https://www.sciencedirect.com/science/article/pii/S0893608020303117>. 3, 4, 7
- [6] Anurag Arnab, Mostafa Dehghani, Georg Heigold, Chen Sun, Mario Lučić, and Cordelia Schmid. Vivit: A video vision transformer. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 6836–6846, October 2021. 14, 15
- [7] Junjie Bai, Fang Lu, Ke Zhang, et al. Onnx: Open neural network exchange. <https://github.com/onnx/onnx>, 2019. 14

- [8] Arian Bakhtiarnia, Qi Zhang, and Alexandros Iosifidis. Single-layer vision transformers for more accurate early exits with less overhead. *Neural Networks*, 153:461–473, 2022. ISSN 0893-6080. doi: <https://doi.org/10.1016/j.neunet.2022.06.038>. URL <https://www.sciencedirect.com/science/article/pii/S0893608022002532>. 13
- [9] Ron Banner, Yury Nahshan, and Daniel Soudry. *Post Training 4-Bit Quantization of Convolutional Networks for Rapid-Deployment*. Curran Associates Inc., Red Hook, NY, USA, 2019. 13
- [10] Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854, 2019. doi: 10.1073/pnas.1903070116. URL <https://www.pnas.org/doi/abs/10.1073/pnas.1903070116>. 4
- [11] Irwan Bello, William Fedus, Xianzhi Du, Ekin Dogus Cubuk, Aravind Srinivas, Tsung-Yi Lin, Jonathon Shlens, and Barret Zoph. Revisiting resnets: Improved training and scaling strategies. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. URL <https://openreview.net/forum?id=dsmxf7FKiaY>. 8
- [12] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc., 2011. URL <https://proceedings.neurips.cc/paper/2011/file/86e8f7ab32cf12577bc2619bc635690-Paper.pdf>. 5
- [13] Keni Bernardin and Rainer Stiefelhagen. Evaluating multiple object tracking performance: The clear mot metrics. *EURASIP Journal on Image and Video Processing*, 2008(1):246309, May 2008. ISSN 1687-5281. doi: 10.1155/2008/246309. URL <https://doi.org/10.1155/2008/246309>. 7
- [14] Gedas Bertasius, Heng Wang, and Lorenzo Torresani. Is space-time attention all you need for video understanding? In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 813–824. PMLR, 2021. URL <http://proceedings.mlr.press/v139/bertasius21a.html>. 15
- [15] Jason Ross Brown, Yiren Zhao, Ilia Shumailov, and Robert D Mullins. Wide attention is the way forward for transformers? *ArXiv*, abs/2210.00640, 2022. doi: 10.48550/ARXIV.2210.00640. 35

- [16] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020. URL <https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf>. 7
- [17] A. Buades, B. Coll, and J.-M. Morel. A non-local algorithm for image denoising. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 2, pages 60–65 vol. 2, 2005. doi: 10.1109/CVPR.2005.38. 15
- [18] Cristian Buciluă, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, page 535–541, New York, NY, USA, 2006. Association for Computing Machinery. ISBN 1595933395. doi: 10.1145/1150402.1150464. URL <https://doi.org/10.1145/1150402.1150464>. 11
- [19] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations*, 2020. URL <https://openreview.net/forum?id=HylxE1HKwS>. 13, 14
- [20] Z. Cao, G. Hidalgo Martinez, T. Simon, S. Wei, and Y. A. Sheikh. Openpose: Realtime multi-person 2d pose estimation using part affinity fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2019. 26
- [21] J. Carreira and A. Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4724–4733, 2017. 5, 7, 14, 15, 22
- [22] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Meghan Cowan, Haichen Shen, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. Tvm: An automated end-to-end optimizing compiler for deep learning. OSDI'18, page 579–594, USA, 2018. USENIX Association. ISBN 9781931971478. 14
- [23] Yu-Hsin Chen, Tien-Ju Yang, Joel Emer, and Vivienne Sze. Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices.

- IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 9(2):292–308, 2019. doi: 10.1109/JETCAS.2019.2910232. 5
- [24] Ke Cheng, Yifan Zhang, Xiangyu He, Weihan Chen, Jian Cheng, and Hanqing Lu. Skeleton-based action recognition with shift graph convolutional network. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 180–189, 2020. doi: 10.1109/CVPR42600.2020.00026. 13, 28
 - [25] Ke Cheng, Yifan Zhang, Xiangyu He, Jian Cheng, and Hanqing Lu. Extremely lightweight skeleton-based action recognition with shiftgcn++. *IEEE Transactions on Image Processing*, 30:7333–7348, 2021. doi: 10.1109/TIP.2021.3104182. 28
 - [26] Krzysztof Marcin Choromanski, Valerii Likhoshesterov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Quincy Davis, Afroz Mohiuddin, Lukasz Kaiser, David Benjamin Belanger, Lucy J Colwell, and Adrian Weller. Rethinking attention with performers. In *International Conference on Learning Representations (ICLR)*, 2021. 13, 31
 - [27] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS’15, page 3123–3131, Cambridge, MA, USA, 2015. MIT Press. 13
 - [28] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc Le, and Ruslan Salakhutdinov. Transformer-XL: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2978–2988, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1285. URL <https://aclanthology.org/P19-1285>. 17
 - [29] Roeland De Geest, Efstratios Gavves, Amir Ghodrati, Zhenyang Li, Cees Snoek, and Tinne Tuytelaars. Online action detection. In *European Conference on Computer Vision (ECCV)*, pages 269–284, 2016. 16, 35, 36
 - [30] Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014. URL <https://proceedings.neurips.cc/paper/2014/file/2afe4567e1bf64d32a5527244d104cea-Paper.pdf>. 13
 - [31] BladeDISC developers. Bladedisc. <https://github.com/alibaba/BladeDISC>, 2022. 14

- [32] DeepSparse developers. Deepsparse. <https://github.com/neuralmagic/deepsparse>, 2021. 14
- [33] Nebuly Speedster developers. Speedster. <https://github.com/nebulay-ai/nebullvm/tree/main/apps/accelerate/speedster>, 2022. 14
- [34] NVIDIA TensorRT developers. Nvidia tensorrt. <https://github.com/NVIDIA/TensorRT>, 2019. 14
- [35] ONNX Runtime developers. Onnx runtime. <https://www.onnxruntime.ai>, 2021. 14
- [36] OpenVINO developers. Openvino toolkit repository. <https://github.com/openvinotoolkit/openvino>, 2019. 14
- [37] OpenXLA developers. Openxla. <https://github.com/openxla/xla>, 2022. 14
- [38] TFLite developers. Tflite. <https://github.com/tensorflow/tensorflow/tree/master/tensorflow/lite>, 2022. 14
- [39] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL <https://aclanthology.org/N19-1423>. 2, 7, 8
- [40] Jeff Donahue, Lisa Anne Hendricks, Marcus Rohrbach, Subhashini Venugopalan, Sergio Guadarrama, Kate Saenko, and Trevor Darrell. Long-term recurrent convolutional networks for visual recognition and description. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(4):677–691, 2017. doi: 10.1109/TPAMI.2016.2599174. 15
- [41] Zhen Dong, Zhewei Yao, Amir Gholami, Michael W. Mahoney, and Kurt Keutzer. Hawq: Hessian aware quantization of neural networks with mixed-precision. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019. 13
- [42] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021. URL <https://openreview.net/forum?id=YicbFdNTTy>. 7, 13, 15, 31

- [43] Nan Du, Yanping Huang, Andrew M Dai, Simon Tong, Dmitry Lepikhin, Yuanzhong Xu, Maxim Krikun, Yanqi Zhou, Adams Wei Yu, Orhan Firat, Barret Zoph, Liam Fedus, Maarten P Bosma, Zongwei Zhou, Tao Wang, Emma Wang, Kellie Webster, Marie Pellat, Kevin Robinson, Kathleen Meier-Hellstern, Toju Duke, Lucas Dixon, Kun Zhang, Quoc Le, Yonghui Wu, Zhifeng Chen, and Claire Cui. GLaM: Efficient scaling of language models with mixture-of-experts. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 5547–5569. PMLR, 17–23 Jul 2022. URL <https://proceedings.mlr.press/v162/du22c.html>. 8
- [44] Jeremy Elson, John (JD) Douceur, Jon Howell, and Jared Saul. Asirra: A captcha that exploits interest-aligned manual image categorization. In *Proceedings of 14th ACM Conference on Computer and Communications Security (CCS)*. Association for Computing Machinery, Inc., 2007. 43
- [45] Morten From Elvebakken, Alexandros Iosifidis, and Lukas Esterle. Adaptive parameterization of deep learning models for federated learning, 2023. URL <https://arxiv.org/abs/2302.02949>. 41
- [46] Steven K. Esser, Jeffrey L. McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S. Modha. Learned step size quantization. In *International Conference on Learning Representations*, 2020. URL <https://openreview.net/forum?id=rkg066VKDS>. 13
- [47] H. Eun, Jinyoung Moon, Jongyoul Park, Chanho Jung, and Changick Kim. Learning to discriminate information for online action detection. *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 806–815, 2020. 35, 36
- [48] Hyunjun Eun, Jinyoung Moon, Jongyoul Park, Chanho Jung, and Changick Kim. Temporal filtering networks for online action detection. *Pattern Recognition*, 111:107695, 2021. 36
- [49] Haoqi Fan, Bo Xiong, Karttikeya Mangalam, Yanghao Li, Zhicheng Yan, Jitendra Malik, and Christoph Feichtenhofer. Multiscale vision transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 6824–6835, October 2021. 16
- [50] Jun Fang, Ali Shafiee, Hamzah Abdel-Aziz, David Thorsley, Georgios Giannis, and Joseph H. Hassoun. Post-training piecewise linear quantization for deep neural networks. In *Computer Vision – ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part II*, page 69–86, Berlin, Heidelberg, 2020. Springer-Verlag. ISBN 978-3-030-58535-8. doi: 10.1007/978-3-030-58536-5_5. URL https://doi.org/10.1007/978-3-030-58536-5_5. 13

- [51] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022. URL <http://jmlr.org/papers/v23/21-0998.html>. 2, 8, 9
- [52] Christoph Feichtenhofer. X3D: Expanding architectures for efficient video recognition. *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020. 5, 13, 15, 23
- [53] Christoph Feichtenhofer, Haoqi Fan, Jitendra Malik, and Kaiming He. Slowfast networks for video recognition. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019. 7, 14, 15, 23
- [54] Bjarke Felbo, Alan Mislove, Anders Søgaard, Iyad Rahwan, and Sune Lehmann. Using millions of emoji occurrences to learn any-domain representations for detecting sentiment, emotion and sarcasm. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1615–1625, Copenhagen, Denmark, September 2017. Association for Computational Linguistics. doi: 10.18653/v1/D17-1169. URL <https://aclanthology.org/D17-1169>. 41
- [55] Jiyang Gao, Zhenheng Yang, and Ram Nevatia. RED: reinforced encoder-decoder networks for action anticipation. In *British Machine Vision Conference (BMVC)*, 2017. 35, 36
- [56] Mario Geiger, Stefano Spigler, Stéphane d’Ascoli, Levent Sagun, Marco Baity-Jesi, Giulio Biroli, and Matthieu Wyart. Jamming transition as a paradigm to understand the loss landscape of deep neural networks. *Phys. Rev. E*, 100: 012115, Jul 2019. doi: 10.1103/PhysRevE.100.012115. URL <https://link.aps.org/doi/10.1103/PhysRevE.100.012115>. 4
- [57] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W. Mahoney, and Kurt Keutzer. A survey of quantization methods for efficient neural network inference. In George K. Thiruvathukal, Yung-Hsiang Lu, Jaeyoun Kim, Yiran Chen, and Bo Chen, editors, *Low-Power Computer Vision*, pages 206–232. Chapman and Hall/CRC, 2022. ISBN 9781003162810. doi: <https://doi.org/10.1201/9781003162810>. URL <https://arxiv.org/abs/2103.13630>. 12
- [58] Ross B. Girshick. Fast r-cnn. *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1440–1448, 2015. 7
- [59] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc.,

2014. URL <https://proceedings.neurips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf>. 2
- [60] Jianping Gou, Baosheng Yu, Stephen J. Maybank, and Dacheng Tao. Knowledge distillation: A survey. *International Journal of Computer Vision*, 129(6):1789–1819, Jun 2021. ISSN 1573-1405. doi: 10.1007/s11263-021-01453-z. URL <https://doi.org/10.1007/s11263-021-01453-z>. 12
- [61] Scott Gray, Alec Radford, and Diederik P. Kingma. Gpu kernels for block-sparse weights. *OpenAI*, 2017. URL <https://cdn.openai.com/blocksparse/blocksparsespaper.pdf>. 11
- [62] Kailing Guo, Xiaona Xie, Xiangmin Xu, and Xiaofen Xing. Compressing by learning in a low-rank and sparse decomposition form. *IEEE Access*, 7:150823–150832, 2019. doi: 10.1109/ACCESS.2019.2947846. 13
- [63] Hai Victor Habi, Roy H. Jennings, and Arnon Netzer. Hmq: Hardware friendly mixed precision quantization block for cnns. In *Computer Vision – ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXVI*, page 448–463, Berlin, Heidelberg, 2020. Springer-Verlag. ISBN 978-3-030-58573-0. doi: 10.1007/978-3-030-58574-7_27. URL https://doi.org/10.1007/978-3-030-58574-7_27. 13
- [64] Amirhossein Habibian, Haitam Ben Yahia, Davide Abati, Efstratios Gavves, and Fatih Porikli. Delta distillation for efficient video processing. In Shai Avidan, Gabriel Brostow, Moustapha Cissé, Giovanni Maria Farinella, and Tal Hassner, editors, *Computer Vision – ECCV 2022*, pages 213–229, Cham, 2022. Springer Nature Switzerland. ISBN 978-3-031-19833-5. 14
- [65] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. doi: 10.1109/CVPR.2016.90. 2, 7, 13
- [66] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2980–2988, 2017. doi: 10.1109/ICCV.2017.322. 7
- [67] Kaiming He, Ross Girshick, and Piotr Dollar. Rethinking imagenet pre-training. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019. 41
- [68] Lukas Hedegaard. Supers, 2020. GitHub. <https://github.com/lukashedegaard/supers.vi>
- [69] Lukas Hedegaard. Co-rider, 2021. GitHub, <https://github.com/lukashedegaard/co-rider.vi>

- [70] Lukas Hedegaard. Ride, 2021. GitHub, <https://github.com/lukashedegaard/ride>. vi
- [71] Lukas Hedegaard. PyTorch-Benchmark, 2022. GitHub, <https://github.com/lukashedegaard/pytorch-benchmark>. vi
- [72] Lukas Hedegaard and Alexandros Iosifidis. Continual 3d convolutional neural networks for real-time processing of videos. In Shai Avidan, Gabriel Brostow, Moustapha Cissé, Giovanni Maria Farinella, and Tal Hassner, editors, *Computer Vision – ECCV 2022*, pages 369–385, Cham, 2022. Springer Nature Switzerland. ISBN 978-3-031-19772-7. v, 25, 26
- [73] Lukas Hedegaard and Alexandros Iosifidis. Continual inference: A library for efficient online inference with deep neural networks in pytorch, 2022. URL <https://arxiv.org/abs/2204.03418>. v
- [74] Lukas Hedegaard, Omar Ali Sheikh-Omar, and Alexandros Iosifidis. Supervised domain adaptation: A graph embedding perspective and a rectified experimental protocol. *IEEE Transactions on Image Processing*, 30:8619–8631, 2021. doi: 10.1109/TIP.2021.3118978. v, 5
- [75] Lukas Hedegaard, Aman Alok, Juby Jose, and Alexandros Iosifidis. Structured pruning adapters, 2022. URL <https://arxiv.org/abs/2211.10155>. v
- [76] Lukas Hedegaard, Negar Heidari, and Alexandros Iosifidis. Chapter 14 - human activity recognition. In Alexandros Iosifidis and Anastasios Tefas, editors, *Deep Learning for Robot Perception and Cognition*, pages 341–370. Academic Press, 2022. ISBN 978-0-323-85787-1. doi: <https://doi.org/10.1016/B978-0-323-85787-1.00019-1>. URL <https://www.sciencedirect.com/science/article/pii/B9780323857871000191>. v
- [77] Lukas Hedegaard, Illia Oleksiienko, and Christian Møldrup Legaard. DatasetOps, 2022. GitHub, <https://github.com/lukashedegaard/datasetops>. vi
- [78] Lukas Hedegaard, Arian Bakhtiarnia, and Alexandros Iosifidis. Continual transformers: Redundancy-free attention for online inference. In *The Eleventh International Conference on Learning Representations*, 2023. URL <https://openreview.net/forum?id=PolHquob8M7>. v, 13
- [79] Lukas Hedegaard, Negar Heidari, and Alexandros Iosifidis. Continual spatio-temporal graph convolutional networks. *Pattern Recognition*, 140:109528, 2023. ISSN 0031-3203. doi: <https://doi.org/10.1016/j.patcog.2023.109528>. v
- [80] Negar Heidari and Alexandras Iosifidis. Progressive spatio-temporal graph convolutional network for skeleton-based human action recognition. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3220–3224, 2021. 28

- [81] Negar Heidari and Alexandros Iosifidis. Temporal Attention-Augmented Graph Convolutional Network for Efficient Skeleton-Based Human Action Recognition. In *International Conference on Pattern Recognition*, 2020. 28
- [82] Negar Heidari, Lukas Hedegaard, and Alexandros Iosifidis. Chapter 4 - graph convolutional networks. In Alexandros Iosifidis and Anastasios Tefas, editors, *Deep Learning for Robot Perception and Cognition*, pages 71–99. Academic Press, 2022. ISBN 978-0-323-85787-1. doi: <https://doi.org/10.1016/B978-0-323-85787-1.00009-9>. URL <https://www.sciencedirect.com/science/article/pii/B9780323857871000099>. v
- [83] Fabian Caba Heilbron, Victor Escorcia, Bernard Ghanem, and Juan Carlos Niebles. Activitynet: A large-scale video benchmark for human activity understanding. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 961–970, 2015. doi: 10.1109/CVPR.2015.7298698. 36
- [84] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL <https://proceedings.neurips.cc/paper/2017/file/8a1d694707eb0fefe65871369074926d-Paper.pdf>. 7
- [85] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531, 2015. URL <http://arxiv.org/abs/1503.02531>. 11
- [86] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 6840–6851. Curran Associates, Inc., 2020. URL <https://proceedings.neurips.cc/paper/2020/file/4c5bcfec8584af0d967f1ab10179ca4b-Paper.pdf>. 7, 8
- [87] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for NLP. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2790–2799, 2019. 41
- [88] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017. URL <http://arxiv.org/abs/1704.04861>. 13

- [89] Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 328–339, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1031. URL <https://aclanthology.org/P18-1031>. 5, 41
- [90] Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. 41, 42
- [91] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. URL <https://proceedings.neurips.cc/paper/2016/file/d8330f857a17c53d217014ee776bfd50-Paper.pdf>. 13
- [92] Haroon Idrees, Amir R. Zamir, Yu-Gang Jiang, Alex Gorban, Ivan Laptev, Rahul Sukthankar, and Mubarak Shah. The thumos challenge on action recognition for videos “in the wild”. *Computer Vision and Image Understanding*, 155:1–23, 2017. ISSN 1077-3142. 36
- [93] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018. 12, 13
- [94] Yunho Jeon and Junmo Kim. Constructing fast network through deconstruction of convolution. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS’18, page 5955–5965, Red Hook, NY, USA, 2018. Curran Associates Inc. 13
- [95] Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. TinyBERT: Distilling BERT for natural language understanding. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4163–4174, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.findings-emnlp.372. URL <https://aclanthology.org/2020.findings-emnlp.372>. 12
- [96] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, Alex Bridgland, Clemens Meyer, Simon A. A. Kohl, Andrew J. Ballard, Andrew Cowie, Bernardino Romera-Paredes, Stanislav

- Nikolov, Rishabh Jain, Jonas Adler, Trevor Back, Stig Petersen, David Reiman, Ellen Clancy, Michal Zielinski, Martin Steinegger, Michalina Pacholska, Tamas Berghammer, Sebastian Bodenstein, David Silver, Oriol Vinyals, Andrew W. Senior, Koray Kavukcuoglu, Pushmeet Kohli, and Demis Hassabis. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, Aug 2021. ISSN 1476-4687. doi: 10.1038/s41586-021-03819-2. URL <https://doi.org/10.1038/s41586-021-03819-2>. 3
- [97] San Hun Jung, Changyong Son, Seohyung Lee, JinWoo Son, Jae-Joon Han, Youngjun Kwak, Sung Ju Hwang, and Changkyu Choi. Learning to quantize deep networks by optimizing quantization intervals with task loss. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4345–4354, 2018. 13
- [98] 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. *preprint, arXiv:1705.06950*, 2017. 15, 23, 30, 36
- [99] QiuHong Ke, Mohammed Bennamoun, Senjian An, Ferdous Sohel, and Farid Boussaid. A new representation of skeleton sequences for 3D action recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 3288–3297, 2017. 27
- [100] Young Hwi Kim, Seonghyeon Nam, and Seon Joo Kim. Temporally smooth online action detection using cycle-consistent future anticipation. *Pattern Recognition*, 116:107954, 2021. ISSN 0031-3203. 36
- [101] Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. In *International Conference on Learning Representations (ICLR)*, 2020. 31
- [102] Tamara G. Kolda and Brett W. Bader. Tensor decompositions and applications. *SIAM Review*, 51(3):455–500, 2009. doi: 10.1137/07070111X. URL <https://doi.org/10.1137/07070111X>. 13
- [103] Dan Kondratyuk, Liangzhe Yuan, Yandong Li, Li Zhang, Mingxing Tan, Matthew Brown, and Boqing Gong. Movinets: Mobile video networks for efficient video recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 16020–16030, June 2021. 15, 17
- [104] Okan Kopuklu, Neslihan Kose, Ahmet Gunduz, and Gerhard Rigoll. Resource efficient 3d convolutional neural networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) Workshops*, Oct 2019. 13

- [105] Simon Kornblith, Jonathon Shlens, and Quoc V. Le. Do better imagenet models transfer better? In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019. [41](#)
- [106] Raghuraman Krishnamoorthi. Quantizing deep convolutional networks for efficient inference: A whitepaper. *CoRR*, abs/1806.08342, 2018. URL <http://arxiv.org/abs/1806.08342>. [13](#)
- [107] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009. URL <https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf>. [43](#)
- [108] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1, NIPS'12*, page 1097–1105, Red Hook, NY, USA, 2012. Curran Associates Inc. [2, 7](#)
- [109] Okan Köpüklü, Stefan Hörmann, Fabian Herzog, Hakan Cevikalp, and Gerhard Rigoll. Dissected 3d cnns: Temporal skip connections for efficient online video processing. *Computer Vision and Image Understanding*, 215: 103318, 2022. ISSN 1077-3142. doi: <https://doi.org/10.1016/j.cviu.2021.103318>. URL <https://www.sciencedirect.com/science/article/pii/S1077314221001594>. [14, 17](#)
- [110] François Lagunas, Ella Charlaix, Victor Sanh, and Alexander Rush. Block pruning for faster transformers. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 10619–10629, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. [11, 42](#)
- [111] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. In *International Conference on Learning Representations (ICLR)*, 2017. URL <https://openreview.net/forum?id=rJqFGTslg>. [11, 43, 45](#)
- [112] Yanghao Li, Chao-Yuan Wu, Haoqi Fan, Karttikeya Mangalam, Bo Xiong, Jitendra Malik, and Christoph Feichtenhofer. Mvitv2: Improved multiscale vision transformers for classification and detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4804–4814, June 2022. [16](#)
- [113] Ji Lin, Chuang Gan, and Song Han. Tsm: Temporal shift module for efficient video understanding. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019. [13, 17](#)

- [114] Jun Liu, Amir Shahroudy, Mauricio Perez, Gang Wang, Ling-Yu Duan, and Alex C. Kot. Ntu rgb+d 120: A large-scale benchmark for 3d human activity understanding. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2019. doi: 10.1109/TPAMI.2019.2916873. 30
- [115] Shuying Liu and Weihong Deng. Very deep convolutional neural network based image classification using small training sample size. In *2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR)*, pages 730–734, 2015. doi: 10.1109/ACPR.2015.7486599. 7
- [116] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. Ssd: Single shot multibox detector. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision – ECCV 2016*, pages 21–37, Cham, 2016. Springer International Publishing. 7
- [117] Ziyu Liu, Hongwen Zhang, Zhenghao Chen, Zhiyong Wang, and Wanli Ouyang. Disentangling and unifying graph convolutions for skeleton-based action recognition. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 140–149, 2020. 27
- [118] Po-Ling Loh. On lower bounds for statistical learning theory. *Entropy*, 19(11), 2017. ISSN 1099-4300. doi: 10.3390/e19110617. URL <https://www.mdpi.com/1099-4300/19/11/617>. 3
- [119] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3431–3440, 2015. doi: 10.1109/CVPR.2015.7298965. 7
- [120] Ilya Loshchilov and Frank Hutter. SGDR: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations*, 2017. URL <https://openreview.net/forum?id=Skq89Scxx>. 5
- [121] R.G. Lyons. *Understanding Digital Signal Processing*. Prentice Hall, 2011. ISBN 9780137027415. 20
- [122] Rabeeh Karimi Mahabadi, James Henderson, and Sebastian Ruder. Compcacter: Efficient low-rank hypercomplex adapter layers. In *Advances in Neural Information Processing Systems*, 2021. 41
- [123] Arun Mallya, Dillon Davis, and Svetlana Lazebnik. Piggyback: Adapting a single network to multiple tasks by learning to mask weights. In *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018. 42
- [124] Jacques Mattheij. Another way of looking at lee sedol vs alphago, March 2016. URL <https://jacquesmattheij.com/another-way-of-looking-at-lee-sedol-vs-alphago/>. 4

- [125] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, Dec 1943. ISSN 1522-9602. doi: 10.1007/BF02478259. URL <https://doi.org/10.1007/BF02478259>. 2
- [126] Jeffrey L. McKinstry, Steven K. Esser, Rathinakumar Appuswamy, Deepika Bablani, John V. Arthur, Izzet B. Yildiz, and Dharmendra S. Modha. Discovering low-precision networks close to full-precision networks for efficient inference. In *2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing - NeurIPS Edition (EMC2-NIPS)*, pages 6–9, 2019. doi: 10.1109/EMC2-NIPS53020.2019.00009. 12
- [127] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. Mixed precision training. In *International Conference on Learning Representations*, 2018. URL <https://openreview.net/forum?id=r1gs9JgRZ>. 5
- [128] Szymon Migacz. Nvidia 8-bit inference with tensorrt. In *GPU Technology Conference*, volume 10, pages 1–41, 2017. 12
- [129] Seyed Iman Mirzadeh, Mehrdad Farajtabar, Ang Li, Nir Levine, Akihiro Matsukawa, and Hassan Ghasemzadeh. Improved knowledge distillation via teacher assistant. In *AAAI Conference on Artificial Intelligence*, 2019. 11
- [130] Asit Mishra and Debbie Marr. Apprentice: Using knowledge distillation techniques to improve low-precision network accuracy. In *International Conference on Learning Representations*, 2018. URL <https://openreview.net/forum?id=B1ae1lZRb>. 12
- [131] Asit Mishra, Jorge Albericio Latorre, Jeff Pool, Darko Stosic, Dusan Stosic, Ganesh Venkatesh, Chong Yu, and Paulius Micikevicius. Accelerating sparse deep neural networks, 2021. URL <https://arxiv.org/abs/2104.08378>. 11
- [132] Rohit Mohan and Abhinav Valada. Efficientps: Efficient panoptic segmentation. *International Journal of Computer Vision*, 129(5):1551–1579, May 2021. ISSN 1573-1405. doi: 10.1007/s11263-021-01445-z. URL <https://doi.org/10.1007/s11263-021-01445-z>. 7
- [133] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. In *International Conference on Learning Representations (ICLR)*, 2017. 11, 43, 45
- [134] Markus Nagel, Mart van Baalen, Tijmen Blankevoort, and Max Welling. Data-free quantization through weight equalization and bias correction. 2019

- IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 1325–1334, 2019. 13
- [135] Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. Deep double descent: Where bigger models and more data hurt. In *International Conference on Learning Representations*, 2020. URL <https://openreview.net/forum?id=B1g5sA4twr>. 3, 4, 7
- [136] Daniel Neimark, Omri Bar, Maya Zohar, and Dotan Asselmann. Video transformer network. *2021 IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)*, pages 3156–3165, 2021. 15
- [137] Maria-Elena Nilsback and Andrew Zisserman. Automated flower classification over a large number of classes. In *2008 Sixth Indian Conference on Computer Vision, Graphics & Image Processing*, pages 722–729, 2008. doi: 10.1109/ICVGIP.2008.47. 43
- [138] Graduate School of Technical Sciences. Rules and Regulations, Jan 2023. URL <https://phd.tech.au.dk/for-phd-students/rules-regulations>. iv
- [139] OpenAI. Chatgpt: Optimizing language models for dialogue. <https://openai.com/blog/chatgpt/>, 2022. 2
- [140] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. In *Proceedings of the 7th International World Wide Web Conference*, pages 161–172, Brisbane, Australia, 1998. 3
- [141] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA, July 2002. Association for Computational Linguistics. doi: 10.3115/1073083.1073135. URL <https://aclanthology.org/P02-1040>. 7
- [142] Vilfredo Pareto. The Maximum of Utility Given by Free Competition. *Giornale degli Economisti*, 67(3):387–403, December 2008. URL https://ideas.repec.org/a/gde/journl/gde_v67_n3_p387-403.html. 9
- [143] Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. Image transformer. In Jennifer Dy and Andreas Krause, editors, *International Conference on Machine Learning (ICML)*, volume 80 of *Proceedings of Machine Learning Research*, pages 4055–4064. PMLR, 7 2018. 13, 31
- [144] Nikolaos Passalis, Stefania Pedrazzi, Robert Babuska, Wolfram Burgard, Daniel Dias, Francesco Ferro, Moncef Gabbouj, Ole Green, Alexandros Iosifidis, Erdal

- Kayacan, Jens Kober, Olivier Michel, Nikos Nikolaidis, Paraskevi Nousi, Roel Pieters, Maria Tzelepi, Abhinav Valada, and Anastasios Tefas. Opendr: An open toolkit for enabling high performance, low footprint deep learning for robotics. In *Proceedings of the 2022 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2022. [vi](#)
- [145] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems* 32, pages 8024–8035. Curran Associates, Inc., 2019. [7](#), [14](#), [23](#)
- [146] Wei Peng, Xiaopeng Hong, Haoyu Chen, and Guoying Zhao. Learning graph convolutional network for skeleton-based human action recognition by neural searching. In *AAAI Conference on Artificial Intelligence*, pages 2669–2676, 2020. [28](#)
- [147] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-1202. URL <https://aclanthology.org/N18-1202>. [41](#)
- [148] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-1202. URL <https://aclanthology.org/N18-1202>. [7](#)
- [149] Matthew E. Peters, Sebastian Ruder, and Noah A. Smith. To tune or not to tune? adapting pretrained representations to diverse tasks. In *Proceedings of the 4th Workshop on Representation Learning for NLP (RepL4NLP-2019)*, pages 7–14, Florence, Italy, August 2019. Association for Computational Linguistics. doi: 10.18653/v1/W19-4302. URL <https://aclanthology.org/W19-4302>. [41](#)
- [150] Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. AdapterFusion: Non-destructive task composition for transfer learning. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 487–503, 2021. [41](#)

- [151] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4095–4104. PMLR, 10–15 Jul 2018. URL <https://proceedings.mlr.press/v80/pham18a.html>. 14
- [152] Chiara Plizzari, Marco Cannici, and Matteo Matteucci. Skeleton-based action recognition via spatial and temporal transformer networks. *Computer Vision and Image Understanding*, 208:103219, 2021. 7, 27, 29
- [153] Chris Pointon. The carbon footprint of chatgpt, 2023. Medium.com, <https://medium.com/@chrispointon/the-carbon-footprint-of-chatgpt-e1bc14e4cc2a>. 4
- [154] Antonio Polino, Razvan Pascanu, and Dan Alistarh. Model compression via distillation and quantization. In *International Conference on Learning Representations*, 2018. URL <https://openreview.net/forum?id=S1XolQbRW>. 12
- [155] Jeff Pool and Chong Yu. Channel permutations for n:m sparsity. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 13316–13327. Curran Associates, Inc., 2021. URL <https://proceedings.neurips.cc/paper/2021/file/6e8404c3b93a9527c8db241a1846599a-Paper.pdf>. 11
- [156] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019. 2
- [157] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, 2021. 2
- [158] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020. URL <http://jmlr.org/papers/v21/20-074.html>. 7
- [159] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas, November 2016. Association for Computational

- Linguistics. doi: 10.18653/v1/D16-1264. URL <https://aclanthology.org/D16-1264>. 7
- [160] Vivek Ramanujan, Mitchell Wortsman, Aniruddha Kembhavi, Ali Farhadi, and Mohammad Rastegari. What’s hidden in a randomly weighted neural network? In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020. 42
- [161] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 8821–8831. PMLR, 18–24 Jul 2021. 2
- [162] Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi. Learning multiple visual domains with residual adapters. In *Advances in Neural Information Processing Systems*, volume 30, 2017. 11, 41
- [163] Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi. Efficient parametrization of multi-domain deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018. 6, 41
- [164] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, 2016. doi: 10.1109/CVPR.2016.91. 7
- [165] Bernhard Riemann. *Ueber die Darstellbarkeit einer Function durch eine trigonometrische Reihe*. Dieterich, 1867. URL <http://eudml.org/doc/203787>. 12
- [166] Ergys Ristani, Francesco Solera, Roger Zou, Rita Cucchiara, and Carlo Tomasi. Performance measures and a data set for multi-target, multi-camera tracking. In Gang Hua and Hervé Jégou, editors, *Computer Vision – ECCV 2016 Workshops*, pages 17–35, Cham, 2016. Springer International Publishing. 7
- [167] F. Rosenblatt. The perceptron - a perceiving and recognizing automaton. Technical Report 85-460-1, Cornell Aeronautical Laboratory, Ithaca, New York, January 1957. 2
- [168] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, Dec 2015. ISSN 1573-1405. doi: 10.1007/s11263-015-0816-y. URL <https://doi.org/10.1007/s11263-015-0816-y>. 3, 15

- [169] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (2nd Edition)*. Prentice Hall, December 2002. ISBN 0137903952. URL <http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20&path=ASIN/0137903952.2>
- [170] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily Denton, Seyed Kamyar Seyed Ghasemipour, Raphael Gontijo-Lopes, Burcu Karagol Ayan, Tim Salimans, Jonathan Ho, David J. Fleet, and Mohammad Norouzi. Photorealistic text-to-image diffusion models with deep language understanding. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022. URL <https://openreview.net/forum?id=08Yk-n5l2Al.2>
- [171] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *ArXiv*, abs/1910.01108, 2019. [11](#), [12](#)
- [172] Victor Sanh, Thomas Wolf, and Alexander M. Rush. Movement pruning: Adaptive sparsity by fine-tuning. In *NeurIPS*, 2020. URL <https://proceedings.neurips.cc/paper/2020/hash/eae15aabaa768ae4a5993a8a4f4fa6e4-Abstract.html>. [11](#), [12](#), [42](#)
- [173] Teven Le Scao, Angela Fan, Christopher Akiki, and colleagues. Bloom: A 176b-parameter open-access multilingual language model. *ArXiv*, abs/2211.05100, 2022. [4](#), [40](#)
- [174] Simone Scardapane, Michele Scarpiniti, Enzo Baccarelli, and Aurelio Uncini. Why should we add early exits to neural networks? *Cognitive Computation*, pages 1–13, 2020. [13](#)
- [175] James Sevilla. Parameter counts in machine learning, 2022. Towards DataScience, <https://towardsdatascience.com/parameter-counts-in-machine-learning-a312dc4753d0.4>. [4](#)
- [176] Amir Shahroudy, Jun Liu, Tian-Tsong Ng, and Gang Wang. Ntu rgb+d: A large scale dataset for 3d human activity analysis. In *IEEE Conference on Computer Vision and Pattern Recognition*, June 2016. [27](#), [29](#), [30](#)
- [177] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948. doi: 10.1002/j.1538-7305.1948.tb01338.x. [12](#)
- [178] Sheng Shen, Zhen Dong, Jiayu Ye, Linjian Ma, Zhewei Yao, Amir Gholami, Michael W. Mahoney, and Kurt Keutzer. Q-BERT: hessian based ultra low precision quantization of BERT. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications*

- of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 8815–8821. AAAI Press, 2020. URL <https://ojs.aaai.org/index.php/AAAI/article/view/6409>. 13
- [179] Lei Shi, Yifan Zhang, Jian Cheng, and Hanqing Lu. Two-stream adaptive graph convolutional networks for skeleton-based action recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 12026–12035, 2019. 5, 7, 29
- [180] Lei Shi, Yifan Zhang, Jian Cheng, and Hanqing Lu. Skeleton-based action recognition with directed graph neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 7912–7921, 2019. 27
- [181] Lei Shi, Yifan Zhang, Jian Cheng, and Hanqing Lu. Two-stream adaptive graph convolutional networks for skeleton-based action recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 12026–12035, 2019. 27
- [182] Gunnar A. Sigurdsson, G. Varol, X. Wang, Ali Farhadi, I. Laptev, and A. Gupta. Hollywood in homes: Crowdsourcing data collection for activity understanding. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2016. 25
- [183] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, Jan 2016. ISSN 1476-4687. doi: 10.1038/nature16961. URL <https://doi.org/10.1038/nature16961>. 3
- [184] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018. doi: 10.1126/science.aar6404. URL <https://www.science.org/doi/abs/10.1126/science.aar6404>. 3
- [185] G. Singh and F. Cuzzolin. Recurrent convolutions for causal 3d cnns. In *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, pages 1456–1465, 2019. 14, 17, 26

- [186] Mahdi Soltanolkotabi, Adel Javanmard, and J. Lee. Theoretical insights into the optimization landscape of over-parameterized shallow neural networks. *IEEE Transactions on Information Theory*, 65:742–769, 2017. 3, 7
- [187] Xu Sun, Xuancheng Ren, Shuming Ma, and Houfeng Wang. meProp: Sparsified back propagation for accelerated deep learning with reduced overfitting. In *Proceedings of the 34th International Conference on Machine Learning (ICLR)*, volume 70 of *Proceedings of Machine Learning Research*, pages 3299–3308, International Convention Centre, Sydney, Australia, 2017. 11, 43, 44, 45
- [188] Mingxing Tan and Quoc Le. EfficientNet: Rethinking model scaling for convolutional neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6105–6114. PMLR, 09–15 Jun 2019. URL <https://proceedings.mlr.press/v97/tan19a.html>. 8, 13, 15
- [189] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V. Le. Mnasnet: Platform-aware neural architecture search for mobile. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 2820–2828. Computer Vision Foundation / IEEE, 2019. doi: 10.1109/CVPR.2019.00293. 14
- [190] Raphael Tang, Yao Lu, Linqing Liu, Lili Mou, Olga Vechtomova, and Jimmy Lin. Distilling task-specific knowledge from BERT into simple neural networks. *CoRR*, abs/1903.12136, 2019. URL <http://arxiv.org/abs/1903.12136>. 11
- [191] Neil C. Thompson, Kristjan H. Greenewald, Keeheon Lee, and Gabriel F. Manso. The computational limits of deep learning. *CoRR*, abs/2007.05558, 2020. URL <https://arxiv.org/abs/2007.05558>. 3
- [192] Rob Toews. Alphafold is the most important achievement in ai—ever, 2021. Forbes, <https://www.forbes.com/sites/robtoews/2021/10/03/alphafold-is-the-most-important-achievement-in-ai-ever/>. 3
- [193] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Herve Jegou. Training data-efficient image transformers & distillation through attention. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 10347–10357. PMLR, 18–24 Jul 2021. URL <https://proceedings.mlr.press/v139/touvron21a.html>. 12

- [194] D. Tran, H. Wang, L. Torresani, J. Ray, Y. LeCun, and M. Paluri. A closer look at spatiotemporal convolutions for action recognition. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6450–6459, 2018. [7](#), [13](#), [15](#), [23](#)
- [195] Dat Thanh Tran, Alexandros Iosifidis, and Moncef Gabbouj. Improving efficiency in convolutional neural network with multilinear filters. *Neural Networks*, 105:328–339, 2018. [13](#)
- [196] Dat Thanh Tran, Serkan Kiranyaz, Moncef Gabbouj, and Alexandros Iosifidis. Progressive operational perceptrons with memory. *Neurocomputing*, 379:172–181, 2020. ISSN 0925-2312. doi: <https://doi.org/10.1016/j.neucom.2019.10.079>. URL <https://www.sciencedirect.com/science/article/pii/S0925231219315188>. [14](#)
- [197] Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. Learning spatiotemporal features with 3d convolutional networks. In *IEEE International Conference on Computer Vision (ICCV)*, pages 4489–4497, 2015. [7](#)
- [198] G. Tzanetakis and P. Cook. Musical genre classification of audio signals. *IEEE Transactions on Speech and Audio Processing*, 10(5):293–302, July 2002. doi: 10.1109/tsa.2002.800560. [37](#)
- [199] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, pages 5998–6008, 2017. [2](#), [5](#), [7](#), [13](#), [14](#), [15](#), [18](#), [30](#)
- [200] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355, Brussels, Belgium, November 2018. Association for Computational Linguistics. doi: 10.18653/v1/W18-5446. URL <https://aclanthology.org/W18-5446>. [7](#)
- [201] Limin Wang, Yuanjun Xiong, Zhe Wang, Yu Qiao, Dahua Lin, Xiaoou Tang, and Luc Van Gool. Temporal segment networks for action recognition in videos. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(11):2740–2755, 2019. doi: 10.1109/TPAMI.2018.2868668. [36](#)
- [202] Sinong Wang, Belinda Z. Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *arXiv:2006.04768*, 2020. [13](#), [31](#)

- [203] Xiang Wang, Shiwei Zhang, Zhiwu Qing, Yuanjie Shao, Zhengrong Zuo, Changxin Gao, and Nong Sang. Oadtr: Online action detection with transformers. *International Conference on Computer Vision (ICCV)*, 2021. URL <https://github.com/wangxiang1230/OadTR>. MIT License. 14, 16, 35, 36
- [204] Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. Non-local neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018. 15
- [205] Yiming Wang, Hang Lv, Daniel Povey, Lei Xie, and Sanjeev Khudanpur. Wake word detection with streaming transformers. In *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5864–5868, 2021. doi: 10.1109/ICASSP39728.2021.9414777. 17
- [206] Bichen Wu, Alvin Wan, Xiangyu Yue, Peter H. Jin, Sicheng Zhao, Noah Golmant, Amir Gholaminejad, Joseph Gonzalez, and Kurt Keutzer. Shift: A zero flop, zero parameter alternative to spatial convolutions. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 9127–9135. Computer Vision Foundation / IEEE Computer Society, 2018. doi: 10.1109/CVPR.2018.00951. URL http://openaccess.thecvf.com/content_cvpr_2018/html/Wu_Shift_A_Zero_CVPR_2018_paper.html. 13
- [207] Chao-Yuan Wu, Yanghao Li, Karttikeya Mangalam, Haoqi Fan, Bo Xiong, Jitendra Malik, and Christoph Feichtenhofer. MeMViT: Memory-Augmented Multiscale Vision Transformer for Efficient Long-Term Video Recognition. In *CVPR*, 2022. 14
- [208] Hao Wu, Patrick Judd, Xiaojie Zhang, Mikhail Isaev, and Paulius Micikevicius. Integer quantization for deep learning inference: Principles and empirical evaluation. *ArXiv*, abs/2004.09602, 2020. 12
- [209] Qizhe Xie, Minh-Thang Luong, Eduard H. Hovy, and Quoc V. Le. Self-training with noisy student improves imagenet classification. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*, pages 10684–10695. Computer Vision Foundation / IEEE, 2020. doi: 10.1109/CVPR42600.2020.01070. 12
- [210] Yunyang Xiong, Zhanpeng Zeng, Rudrasis Chakraborty, Mingxing Tan, Glenn Fung, Yin Li, and Vikas Singh. Nyströmformer: A nyström-based algorithm for approximating self-attention. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(16):14138–14148, May 2021. doi: 10.1609/aaai.v35i16.17664. URL <https://ojs.aaai.org/index.php/AAAI/article/view/17664>. 13, 31

- [211] Mingze Xu, Mingfei Gao, Yi-Ting Chen, Larry Davis, and David Crandall. Temporal recurrent networks for online action detection. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 5531–5540, 2019. doi: 10.1109/ICCV.2019.00563. URL <https://github.com/xumingze0308/TRN.pytorch>. MIT Licence. 35, 36
- [212] Mingze Xu, Yuanjun Xiong, Hao Chen, Xinyu Li, Wei Xia, Zhuowen Tu, and Stefano Soatto. Long short-term transformer for online action detection. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2021. 14, 16, 35, 36
- [213] Sijie Yan, Yuanjun Xiong, and Dahua Lin. Spatial temporal graph convolutional networks for skeleton-based action recognition. In *AAAI Conference on Artificial Intelligence*, pages 7444–7452, 2018. 5, 14
- [214] Sijie Yan, Yuanjun Xiong, and Dahua Lin. Spatial temporal graph convolutional networks for skeleton-based action recognition. In *AAAI Conference on Artificial Intelligence*, 2018. 7, 18, 27, 29
- [215] Zhaoyi Yan, Xiaoming Li, Mu Li, Wangmeng Zuo, and Shiguang Shan. Shift-net: Image inpainting via deep feature rearrangement. In Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss, editors, *Computer Vision – ECCV 2018*, pages 3–19, Cham, 2018. Springer International Publishing. ISBN 978-3-030-01264-9. 13
- [216] Yifan Yang, Qijing Huang, Bichen Wu, Tianjun Zhang, Liang Ma, Giulio Gambardella, Michaela Blott, Luciano Lavagno, Kees Vissers, John Wawrzynek, and Kurt Keutzer. Synetgy: Algorithm-hardware co-design for convnet accelerators on embedded fpgas. In *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA ’19, page 23–32, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450361378. doi: 10.1145/3289602.3293902. URL <https://doi.org/10.1145/3289602.3293902>. 13
- [217] Xiaohui Yao, Konstantin Klyukin, Wenjie Lu, Murat Onen, Seungchan Ryu, Dongha Kim, Nicolas Emond, Iradwikanari Waluyo, Adrian Hunt, Jesús A. del Alamo, Ju Li, and Bilge Yildiz. Protonic solid-state electrochemical synapse for physical neural networks. *Nature Communications*, 11(1):3134, Jun 2020. ISSN 2041-1723. doi: 10.1038/s41467-020-16866-6. URL <https://doi.org/10.1038/s41467-020-16866-6>. 5
- [218] Seul-Ki Yeom, Philipp Seegerer, Sebastian Lapuschkin, Alexander Binder, Simon Wiedemann, Klaus-Robert Müller, and Wojciech Samek. Pruning by explaining: A novel criterion for deep neural network pruning. *Pattern Recognition*, 115, 2021. 11, 42, 43, 45

- [219] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014. URL <https://proceedings.neurips.cc/paper/2014/file/375c71349b295fbe2dcda9206f20a06-Paper.pdf>. 41
- [220] Xiyu Yu, Tongliang Liu, Xinchao Wang, and Dacheng Tao. On compressing deep models by low rank and sparse decomposition. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 67–76, 2017. doi: 10.1109/CVPR.2017.15. 13
- [221] Li Yuan, Francis EH Tay, Guilin Li, Tao Wang, and Jiashi Feng. Revisiting knowledge distillation via label smoothing regularization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3903–3911, 2020. 12
- [222] Li Yuan, Qibin Hou, Zihang Jiang, Jiashi Feng, and Shuicheng Yan. Volo: Vision outlooker for visual recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–13, 2022. doi: 10.1109/TPAMI.2022.3206108. 3, 7
- [223] Joe Yue-Hei Ng, Matthew Hausknecht, Sudheendra Vijayanarasimhan, Oriol Vinyals, Rajat Monga, and George Toderici. Beyond short snippets: Deep networks for video classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015. 15
- [224] Mengyao Zhai, Lei Chen, Fred Tung, Jiawei He, Megha Nawhal, and Greg Mori. Lifelong gan: Continual learning for conditional image generation. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 2759–2768, 2019. 11
- [225] Pengfei Zhang, Cuiling Lan, Junliang Xing, Wenjun Zeng, Jianru Xue, and Nanning Zheng. View adaptive recurrent neural networks for high performance human action recognition from skeleton data. In *IEEE International Conference on Computer Vision*, pages 2117–2126, 2017. 27
- [226] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018. 13
- [227] Ying Zhang, Tao Xiang, Timothy M. Hospedales, and Huchuan Lu. Deep mutual learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018. 11

- [228] Peisen Zhao, Jiajie Wang, Lingxi Xie, Ya Zhang, Yanfeng Wang, and Qi Tian. Privileged knowledge distillation for online action detection. *preprint, arXiv:2011.09158*, abs/2011.09158, 2020. 36
- [229] Aojun Zhou, Yukun Ma, Junnan Zhu, Jianbo Liu, Zhijie Zhang, Kun Yuan, Wenxiu Sun, and Hongsheng Li. Learning n:m fine-grained structured sparse neural networks from scratch. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=K9bw7vqp_s. 11
- [230] Tianyi Zhou and Dacheng Tao. Greedy bilateral sketch, completion & smoothing. In Carlos M. Carvalho and Pradeep Ravikumar, editors, *Proceedings of the Sixteenth International Conference on Artificial Intelligence and Statistics*, volume 31 of *Proceedings of Machine Learning Research*, pages 650–658, Scottsdale, Arizona, USA, 29 Apr–01 May 2013. PMLR. URL <https://proceedings.mlr.press/v31/zhou13b.html>. 13
- [231] Michael Zhu and Suyog Gupta. To prune, or not to prune: Exploring the efficacy of pruning for model compression. In *International Conference on Learning Representations (ICLR) Workshop Track Proceedings*, 2018. URL <https://openreview.net/forum?id=Sy1iIDkPM>. 11
- [232] Yaoming Zhu, Jiangtao Feng, Chengqi Zhao, Mingxuan Wang, and Lei Li. Counter-interference adapter for multilingual machine translation. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 2812–2823, 2021. 41
- [233] Barret Zoph and Quoc Le. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*, 2017. URL <https://openreview.net/forum?id=r1Ue8Hcxg>. 14
- [234] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018. 14

Part II

Publications

Publication 1

Continual 3D Convolutional Neural Networks for Real-time Processing of Videos

Authors: Lukas Hedegaard and Alexandros Iosifidis.

Publication: Computer Vision – ECCV 2022. ECCV 2022. Lecture Notes in Computer Science, vol 13664. Springer, Cham. https://doi.org/10.1007/978-3-031-19772-7_22.

Link: https://link.springer.com/chapter/10.1007/978-3-031-19772-7_22

Publication 2

Continual Spatio-Temporal Graph Convolutional Networks

Authors: Lukas Hedegaard, Negar Heidari, and Alexandros Iosifidis.

Publication: Pattern Recognition, Volume 140, 2023, 109528, ISSN 0031-3203.

Link: <https://www.sciencedirect.com/science/article/pii/S0031320323002285>

Publication 3

Continual Transformers: Redundancy-Free Attention for Online Inference

Authors: Lukas Hedegaard, Arian Bakhtiarnia, and Alexandros Iosifidis.

Publication: International Conference on Learning Representations, 2023.

Link: <https://openreview.net/pdf?id=PolHquob8M7>

Publication 4

Continual Inference: A Library for Efficient Online Inference with Deep Neural Networks in PyTorch

Authors: Lukas Hedegaard, and Alexandros Iosifidis.

Publication: Computer Vision – ECCV 2022 Workshops. Lecture Notes in Computer Science, vol 13803. Springer. ISBN: 978-3-031-25066-8.

Link: https://link.springer.com/chapter/10.1007/978-3-031-25082-8_2

Publication 5

Human Activity Recognition

Authors: Lukas Hedegaard, Negar Heidari, and Alexandros Iosifidis.

Publication: “Deep Learning for Robot Perception and Cognition”, A. Iosifidis and A. Tefas, Eds., 1st ed., Academic Press, Jan. 2022, ch. 14, isbn: 9780323857871.

Link: <https://www.sciencedirect.com/science/article/pii/B9780323857871000191>

Publication 6

Graph Neural Networks

Authors: Negar Heidari, Lukas Hedegaard, and Alexandros Iosifidis.

Publication: “Deep Learning for Robot Perception and Cognition”, A. Iosifidis and A. Tefas, Eds., 1st ed., Academic Press, Jan. 2022, ch. 4, isbn: 9780323857871.

Contribution: Primary author of section 4.4 “Graph attention network (GAT)” and reviewer of the remaining chapter contents.

Link: <https://www.sciencedirect.com/science/article/pii/B9780323857871000099>

Publication 7

Structured Pruning Adapters

Authors: Lukas Hedegaard, Omar Ali Sheikh-Omar and Alexandros Iosifidis.

Publication: ArXiv preprint arXiv:2211.10155, 2022.

Link: <https://arxiv.org/abs/2211.10155>