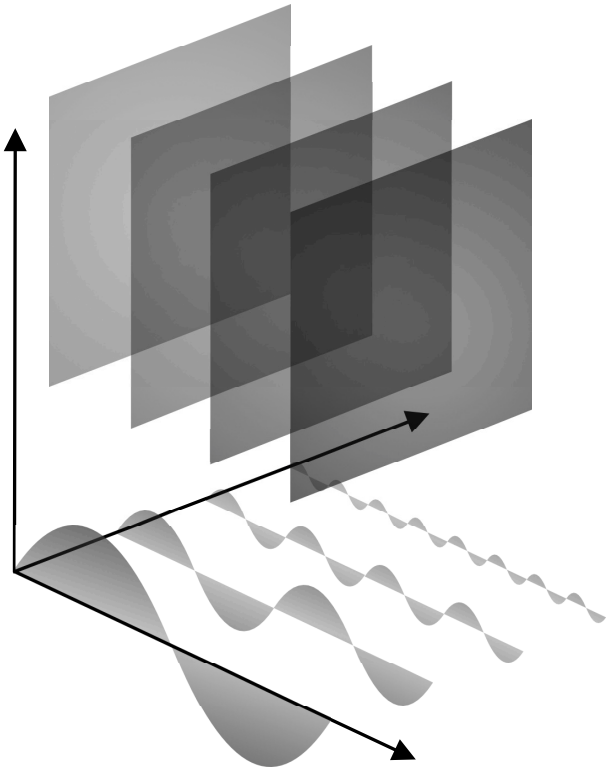
AARHUS
UNIVERSITY

PHD DISSERTATION

Efficient Online Processing with Deep Neural Networks

by Lukas Hedegaard



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 7, 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 Nerual 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 spurt 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 samtaleemne 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 finansiel 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 netværksarkitekturer 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 demonstrerede ved konvertering af adskillige vidt anvendte netværksarkitekturer 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 fell 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 7, 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 LaTeX.

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 iii

Acknowledgments v
List of publications viii

Contents xi

I Overview 1

1 Introduction 3
1.1 The triumph of Deep Learning 3
1.2 The burden of Deep Neural Networks 4
1.3 A pursuit of efficiency 6

2 Quantifying efficiency 9
2.1 Task efficiency 9
2.2 Storage efficiency 9
2.3 Computational efficiency 10
2.4 Efficiency is a trade-off 11

3 Approaches for enhancing inference efficiency 13
3.1 Pruning methods 13
3.2 Knowledge distillation 14
3.3 Quantization 15
3.4 Low-rank factorization of weights 16
3.5 Efficient architectures 16
3.6 Caching and temporal redistribution of features 17
3.7 Efficient run-times 17

4 Redistributing computations through time with Continual Inference Networks 19

4.1 Introduction . . . 19
4.2 Related works . . . 21
4.3 Continual Inference Networks . . . 22
4.4 Continual 3D Convolutional Neural Networks . . . 26
4.5 Continual Spatio-temporal Graph Convolutional Networks . . . 30
4.6 Continual Transformers . . . 34
4.7 A Python library to support implementation . . . 41
4.8 Conclusion . . . 43

5 Compressing and accelerating derived networks with Structured Pruning Adapters 45
5.1 Introduction . . . 45
5.2 Related works . . . 46
5.3 Structured Pruning Adapters . . . 47
5.4 Experiments . . . 48
5.5 Conclusion . . . 49

6 Conclusions and future work 51

Bibliography 53

II Publications 1

1 Continual 3D Convolutional Neural Networks for Real-time Processing of Videos 1

2 Continual Spatio-Temporal Graph Convolutional Networks 27

3 Continual Transformers: Redundancy-Free Attention for Online Inference 41

4 Continual Inference: A Library for Efficient Online Inference with Deep Neural Networks in PyTorch 59

5 Human Activity Recognition 75

6 Graph Neural Networks 107

7 Structured Pruning Adapters 139

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 constant[1], 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:

$$\text{cost} \propto \underbrace{o \cdot n}_{c} \cdot n \cdot e \cdot h, \tag{1.1}$$

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, $n \to n^2$, is required to reduce the error by half. Accordingly, the training cost increases from $n^2 \cdot o \cdot e \cdot h \to n^4 \cdot o \cdot e \cdot h$. 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

[1] *E.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].

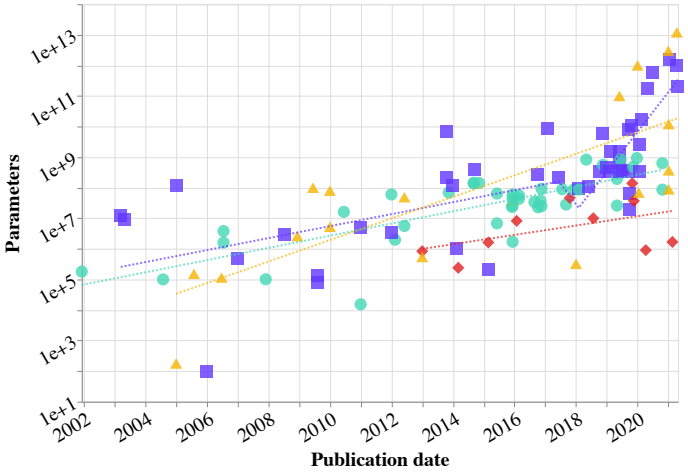


Figure 1.1: Parameter count of prominent deep learning DNNs from 2002 to 2022 for vision (●), language (■), game (◆), and other (▲) models. Data source: [175].

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 CO_2 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* (FLOPs[1]), *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-agnostic[3] 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 meausuring 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

[1] FLOPs 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.

[2] Occastionally referred to as multiply-adds (mult-adds).

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

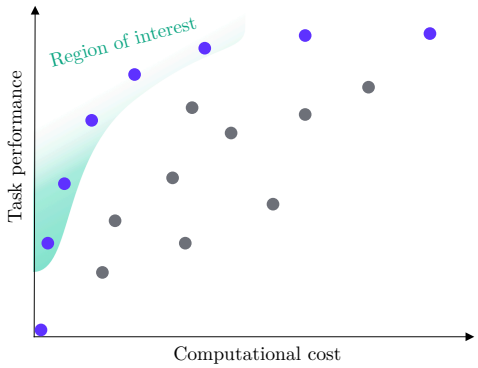


Figure 2.1: Pareto frontier of models •. Models that strike a useful balance between task performance and computational cost, *i.e.*, those withing the green region, are of primary interest.

measurement of computational latency should thus include the data transmission to and from GPU/TPU in addition to the latency of computation[4].

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).

[4]Data 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.

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

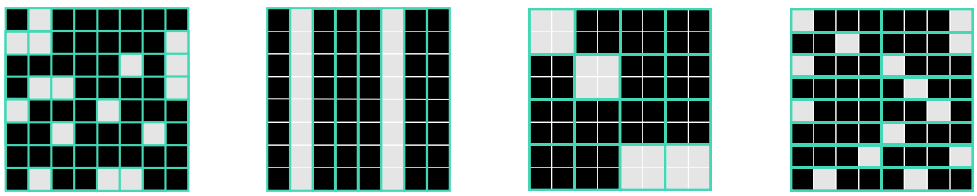
Chapter 3

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



(a) Unstructured (b) Channel (c) Block (d) N:M

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

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

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

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

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 minization 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] of 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 models formats 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, spurted multiple methods that adopted transformer architectures in the video-domain, *e.g.*, ViT-B VTN [136], ViViT [6], TimeSFormer [14], MViT-

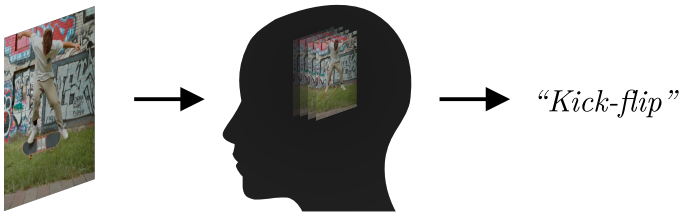


Figure 4.1: Perception in the natural world happens continually, one impression at a time, with temporal context modeled internally.

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?*

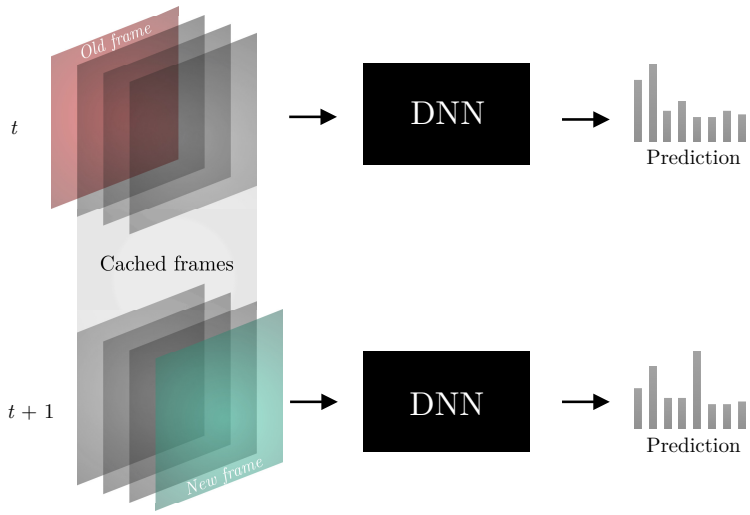
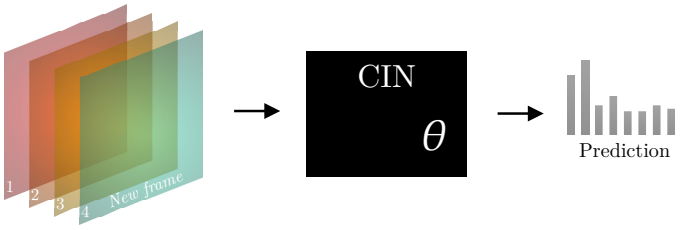


Figure 4.2: Sliding-window processing with DNNs. With each passing time-step, the old frame is discarded while intermediary frames are cached and prepended to the new frame. To process each time-step, many DNNs (*e.g.*, 3D CNN or Transformer-based networks) require the full frame-sequence as input. This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

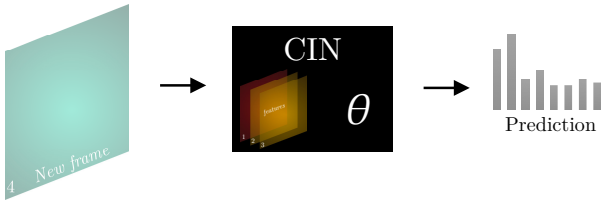
4.2 Related works

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



(a) Offline processing



(b) Online processing

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).

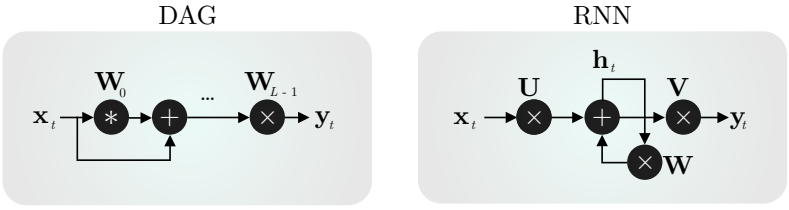
Offline inference: A CIN can process a spatio-temporal input similarly to a regular DNN (Fig. 4.3a).

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

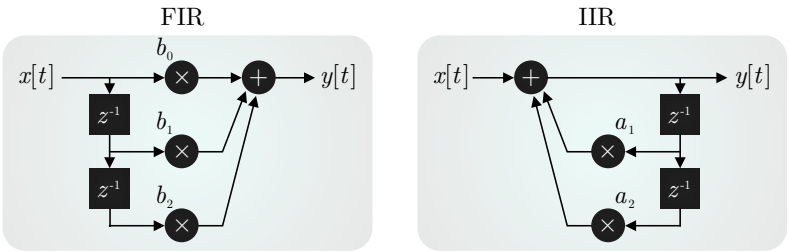
Identical weights: A CIN uses one set of trainable parameters for online and offline inference.

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.



(a) Continual Inference Networks



(b) Discrete-time filters

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

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

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

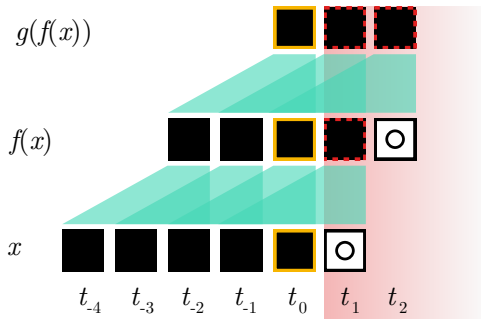


Figure 4.5: Temporal end-padding ⊡ in neural network layers with receptive field of size ▲ results in acausal outputs (red dotted boxes □). The features corresponding to the current time-step t_0 are highlighted yellow □.

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.

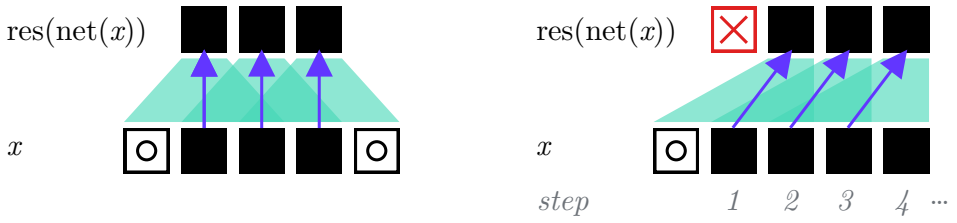
Empirically, it was observed in paper 1 that weights from networks which were trained with temporal end-padding can be reused for efficient stream processing in corresponding CINs without end-padding. This, however, introduced a *model shift*, *i.e.* a deviation of the model operation, with lead to slight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



(a) Offline inference (b) Online inference

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:

$$S_{NN} = \prod_{l=1}^{L} S_l. \tag{4.2}$$

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

$$R_{NN} = 1/S_{NN}. \tag{4.3}$$

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

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

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:

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

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

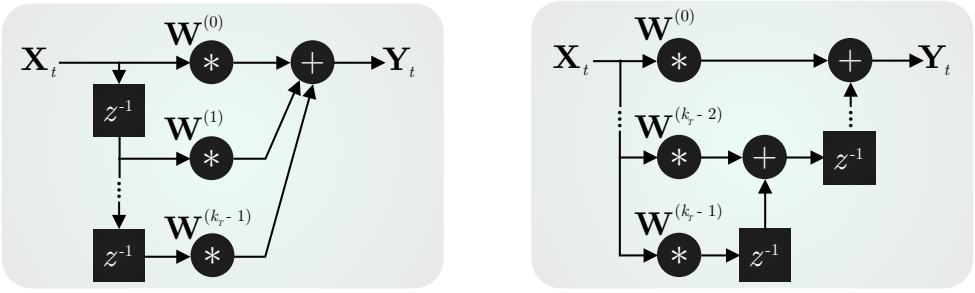
$$\mathbf{Y}^{(t)} = (\mathbf{W} * \mathbf{X})^{(t)} = \sum_{k_t=0}^{K_T-1} \mathbf{W}^{(k_t)} * \mathbf{X}^{(t-k_t)}. \tag{4.6}$$

The above can be extended to any spatial convolution, including dilated, strided, padded, and grouped convolutions, and holds for continual input streams where $T \to \infty$. *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 $\mathbf{W} \in \mathbb{R}^{C_{in} \times C_{out} \times K_T \times \cdots}$ performs an up-projection (*i.e.*, $C_{in} > C_{out}$), the pre-cached structure gives the smallest



(a) Pre-cached (b) Post-cached

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 *Co*I3D, 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× fewer FLOPs for *Co*Slow compared to Slow, which had a clip size of 8, and 15.34× fewer for *Co*X3D-L compared to X3D-L,

which had a clip size of 16), but with lower relative throughput increases than FLOPs reductions due the to 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.

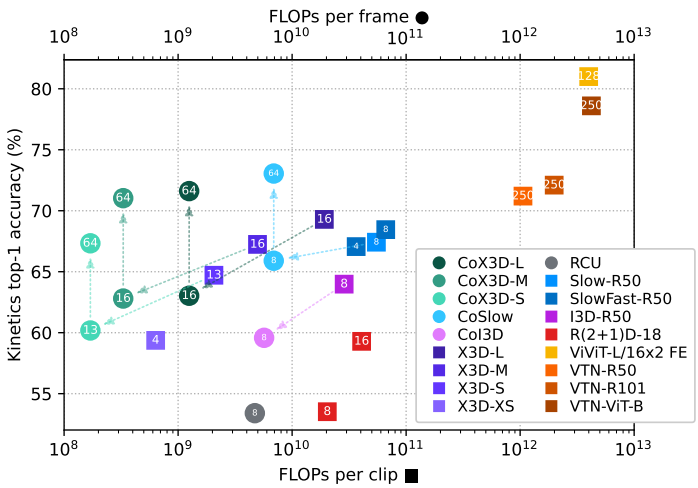


Figure 4.8: Accuracy/complexity trade-off for Continual 3D CNNs and recent state-of-the-art methods on Kinetics-400 using 1-clip/frame testing. ■ FLOPs per *clip* are noted for regular networks, while ● FLOPs per *frame* are shown for the Continual 3D CNNs. Frames per clip / global average pool size 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-8×8-R50	67.42	32.45	266.04	54.87	0.38	1.34	4.31	61.92
	SlowFast-8×8-R50	68.45	66.25	344.84	66.25	0.34	0.87	2.72	30.72
	SlowFast-4×16-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
	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
Frame	RCU_8 [185]	53.40	12.80	-	4.71	-	-	-	-
	Co$I3D_8$	59.58	28.04	235.87	5.68	3.00	2.41	14.88	125.59
	Co$I3D_{64}$	56.86	28.04	236.08	5.68	3.15	2.41	14.89	126.32
	Co$Slow_8$	65.90	32.45	175.98	6.90	2.80	1.60	6.18	113.77
	***Co*Slow$_{64}$**	**73.05**	**32.45**	**176.41**	**6.90**	**2.92**	**1.60**	**6.19**	**102.00**
	*Co*X3D-L_{16}	63.03	6.15	184.29	1.25	2.30	0.99	25.17	206.65
	***Co*X3D-L$_{64}$**	**71.61**	**6.15**	**184.37**	**1.25**	**2.30**	**0.99**	**27.56**	**217.53**
	*Co*X3D-M_{16}	62.80	3.79	68.88	0.33	7.57	7.26	88.79	844.73
	***Co*X3D-M$_{64}$**	**71.03**	**3.79**	**68.96**	**0.33**	**7.51**	**7.04**	**86.42**	**796.32**
	*Co*X3D-S_{13}	60.18	3.79	41.91	0.17	13.16	11.06	219.64	939.72
	***Co*X3D-S$_{64}$**	**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.

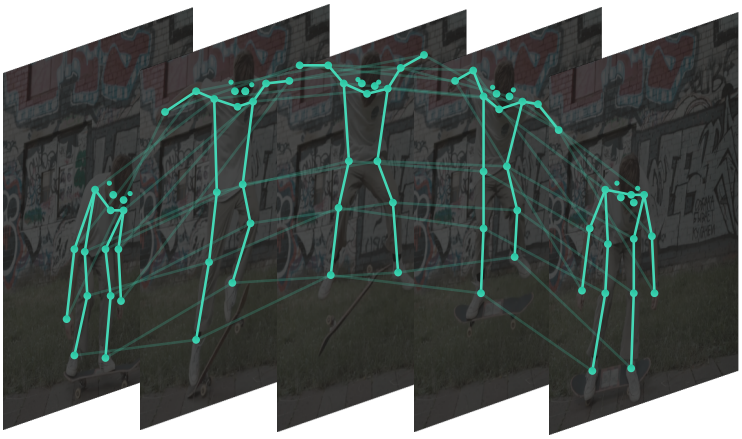


Figure 4.9: Sequence of human body poses modeled as a spatio-temporal graph.

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 methods[1], 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:

$$\mathbf{H}_l = \mathrm{ReLU}\left(\mathrm{Res}(\mathbf{H}_{l-1}) + \mathrm{BN}(\mathrm{TC}(\mathrm{GC}(\mathbf{H}_{l-1})))\right), \tag{4.7}$$

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

[1] See 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 (*Co*TC), and second, the residual connection is delayed according to Eq. (4.1) based on the parameters of *Co*TC. The continual spatio-temporal graph convolution block is given by

$$\mathbf{H}_l^{(t)} = \mathrm{ReLU}\left(\mathrm{Delay}(\mathrm{Res}(\mathbf{H}_{l-1}^{(t)})) + \mathrm{BN}(\mathit{Co}\mathrm{TC}(\mathrm{GC}(\mathbf{H}_{l-1}^{(t)})))\right). \tag{4.8}$$

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 properly[2], 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.*, $\mathcal{O}(\mathit{Co}\mathrm{Net}) \approx \mathcal{O}(\mathrm{Body}) + \mathcal{O}(\mathrm{Head})$, whereas sliding-window processing requires the body to process the whole sequence of T skeletons for each step, *i.e.*, $\mathcal{O}(\mathrm{Net}) \approx T \cdot \mathcal{O}(\mathrm{Body}) + \mathcal{O}(\mathrm{Head})$. For $\mathcal{O}(\mathrm{Body}) \gg \mathcal{O}(\mathrm{Head})$, 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

[2]The *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 (↑ 2.2×)	1.1 (↓ 2.1×)	90.4 (↓ 2.0×)
*Co*ST-GCN	4	85.3 (−0.7)	93.1 (−0.3)	3.14	36.0 (79%)	0.27 (↓ 63.2×)	32.3 (↑ 14.0×)	2375.2 (↑ 13.1×)
*Co*ST-GCN*	1	86.3 (+0.3)	93.8 (+0.4)	3.14	36.1 (80%)	0.16 (↓ 107.7×)	46.1 (↑ 20.0×)	4202.2 (↑ 23.2×)
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 (↑ 2.2×)	1.0 (↓ 2.1×)	71.2 (↓ 2.0×)
*Co*AGCN	4	86.0 (−0.4)	93.9 (−0.4)	3.47	37.3 (77%)	0.30 (↓ 63.2×)	25.0 (↑ 11.9×)	2248.4 (↑ 15.4×)
*Co*AGCN*	1	84.1 (−2.3)	92.6 (−1.7)	3.47	37.4 (77%)	0.17 (↓ 108.8×)	30.4 (↑ 14.5×)	3817.2 (↑ 26.1×)
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 (↑ 2.2×)	0.8 (↓ 2.1×)	85.1 (↓ 1.8×)
*Co*S-TR	4	86.5 (−0.3)	93.3 (−0.5)	3.09	35.9 (48%)	0.22 (↓ 63.2×)	30.3 (↑ 17.8×)	2189.5 (↑ 14.0×)
*Co*S-TR*	1	86.3 (−0.3)	92.4 (−1.4)	3.09	36.1 (49%)	0.15 (↓ 107.6×)	43.8 (↑ 25.8×)	3775.3 (↑ 24.2×)

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 *Co*ST-GCN, *Co*AGCN, and *Co*S-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× 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× 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× for direct conversion and 22.3× for stride-modified models (marked with '∗'). Moreover, maximum allocated memory is reduced by approx. 22% for *Co*ST-GCN and *Co*AGCN and by 51% for *Co*S-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

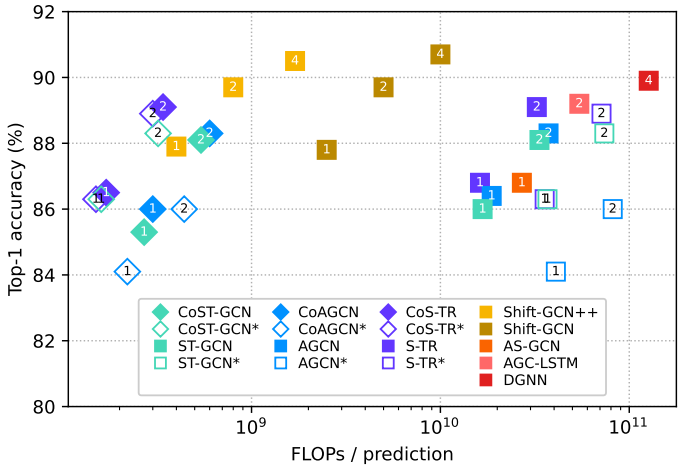


Figure 4.10: Accuracy/complexity trade-off on NTU RGB+D 60 X-Sub for ◆ *Continual* and ■ prior methods during online inference. Numbers denote the number of data streams employed for each evaluation. *Architecture modification with stride one and no padding. The figure first appeared in paper 2.

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:

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

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

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

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

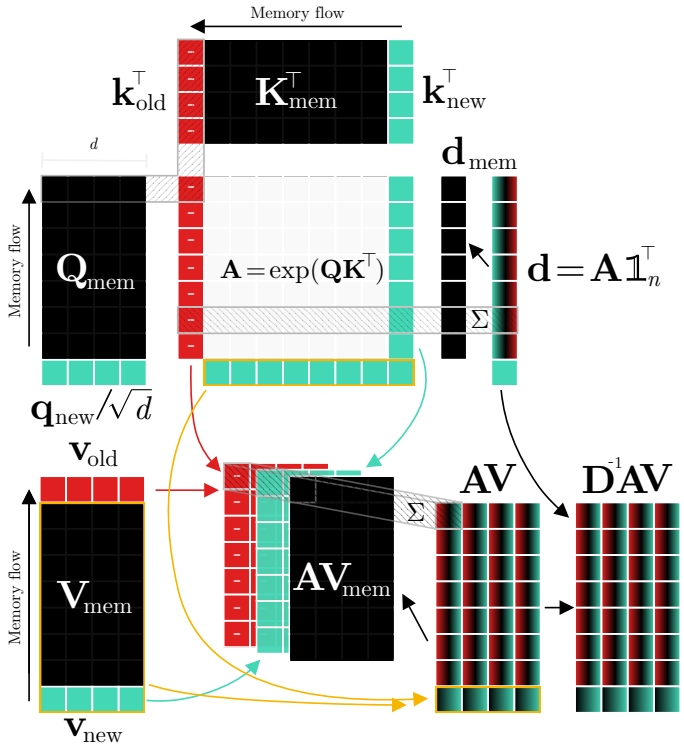


Figure 4.11: Continual Retroactive Dot-Product Attention. The query (**Q**), key (**K**), and value (**V**) matrices are aggregated over time by caching the step vectors $\mathbf{q}_{\text{new}}$, $\mathbf{k}_{\text{new}}$, and $\mathbf{v}_{\text{new}}$ in each their FIFO queue (denoted by $\square_{\text{mem}}$).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

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

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

The newest entry is computed from scratch:

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

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

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

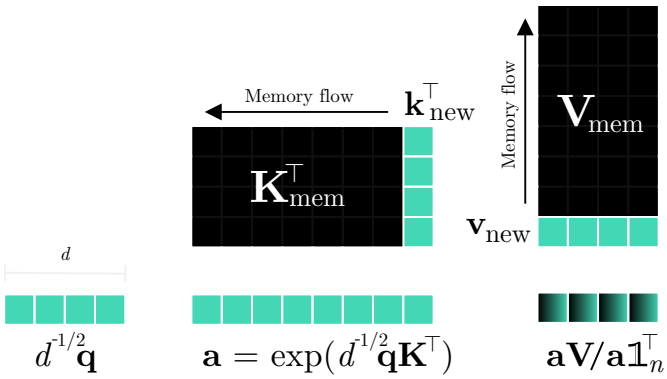


Figure 4.12: Continual Single-Output Dot-Product Attention. The key (**K**) and value (**V**) matrices are aggregated over time by caching the step vectors $\mathbf{k}_{\text{new}}$ and $\mathbf{v}_{\text{new}}$ in a FIFO queue. During each step, only the attention output associated with **q** is computed.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

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

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

Finally, the attention output is attained by normalizing **AV** with **d**:

$$\mathit{CoRe}\text{Att}(\mathbf{q}_{\text{new}}, \mathbf{k}_{\text{new}}, \mathbf{v}_{\text{new}}) = \mathbf{d}^{-1} \odot \mathbf{AV}, \tag{4.16}$$

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, **K** and **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 **A** is limited to a single row **a** corresponding to the desired step query **q**:

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

The attention computation is straight-forward:

$$\mathit{CoSi}\text{Att}(\mathbf{q}, \mathbf{k}_{\text{new}}, \mathbf{v}_{\text{new}}) = \mathbf{a}\left(\mathbf{V}_{\text{mem}} \,\|\, \mathbf{v}_{\text{new}}\right)/\mathbf{a}\mathbb{1}_n^\top. \tag{4.18}$$

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, $\mathbf{W}_O \in \mathbb{R}^{d_V \times d_O}$:

$$Co\text{MHA}(\mathbf{q},\mathbf{k},\mathbf{v}) = \left(\Big\Vert_{i=0}^{h-1} Co\text{Att}(\mathbf{q}\mathbf{W}_Q^i, \mathbf{k}\mathbf{W}_K^i, \mathbf{v}\mathbf{W}_V^i) \right) \mathbf{W}_O. \tag{4.19}$$

Here, $Co\text{Att}(\cdot)$ can either be the retroactive or single-output SDA, $CoRe\text{Att}(\cdot)$ or $CoSi\text{Att}(\cdot)$, 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.

$$\mathbf{y} = \text{LayerNorm}\left(\text{Sel}(\mathbf{x}) + Co\text{MHA}(\mathbf{x},\mathbf{x},\mathbf{x})\right), \tag{4.20}$$

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:

$$\mathbf{z} = \text{LayerNorm}\left(\mathbf{y} + \text{FF}(\mathbf{y})\right) \tag{4.21}$$

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

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:

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

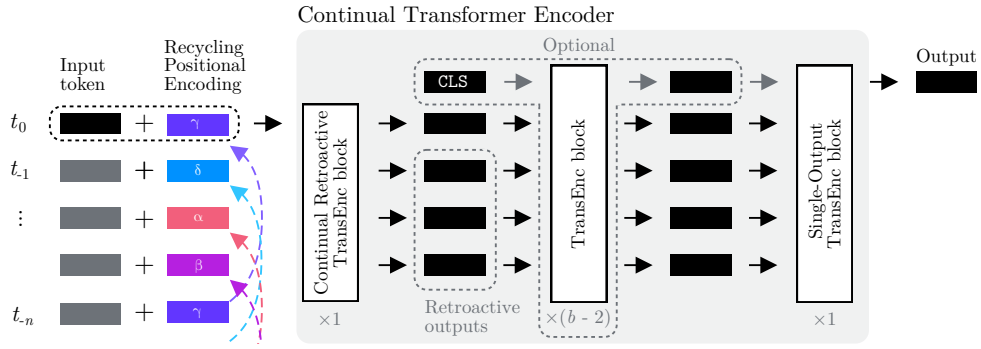


Figure 4.13: Multi-block Continual Transformer Encoder with Recycling Positional Encoding. For $b > 2$ blocks, regular Transformer Encoder blocks can be added between an initial Continual Retroactive block and a final Single-Output block. A class-token may be used after the initial block. The figure 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±0.5	**88.6**±0.1	2445.6
OadTR-b2†		56.6±0.3	***88.3***±0.2	1008.1
OadTR-b1†		56.3±0.2	88.1±0.1	605.5
*Co*OadTR-b2 (ours)		56.8±0.4	87.7±0.6	***410.9***
*Co*OadTR-b1 (ours)		56.1±0.7	87.6±0.7	**9.6**
TRN [211]	Kin.	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±0.3	***88.6***±0.1	2513.5
OadTR-b2†		64.5±0.5	88.3±0.2	1075.7
OadTR-b1†		63.9±0.5	88.1±0.1	673.0
*Co*OadTR-b2 (ours)		64.4±0.1	87.6±0.7	***411.9***
*Co*OadTR-b1 (ours)		64.2±0.4	87.7±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× and 6.1×, 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 *Co*OadTR-b1 reduces FLOPs by approx. 63× and the *Co*OadTR-b2 by 2.5×. 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.

```python
import torch
from torch import nn
import continual as co

# Define net
net = co.Residual(
    co.Sequential(
      co.Conv3d(32, 64, kernel_size=(1, 1, 1)),
      nn.BatchNorm3d(64),
      nn.ReLU6(),
      co.Conv3d(64, 64, kernel_size=(3, 3, 3), padding=(1, 1, 1), groups=64),
      nn.ReLU6(),
      co.Conv3d(64, 32, kernel_size=(1, 1, 1)),
      nn.BatchNorm3d(32),
    )
)
net.eval()

# Example input
x = torch.randn((1, 32, 7, 5, 5))  # (B, C, T, H, W)

# Multiple forward models
y = net.forward(x)  # Offline inference, identical to torch.nn
y_firsts = net.forward_steps(x[:, :, :6])  # Multiple steps, e.g., for init
y_last = net.forward_step(x[:, :, 6])  # Single step, for online inference

# Same results
assert torch.allclose(y[:, :, :-net.delay - 1], y_firsts, atol=1e-7)
assert torch.allclose(y[:, :, -net.delay - 1], y_last, atol=1e-7)

# Temporal properties
assert net.receptive_field == 3
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 re-distributions 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× FLOPs reduction per prediction and 9× and 7× 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× FLOPs reductions and up to 18× and 15× throughput increase on CPU and GPU, respectively, with identical accuracy on clips of 300 frames. *Co*ST-GCN-based networks with minor architectural modifications can be fine-tuned with only few training epochs to achieve similar accuracy, FLOPs reductions of 108×, and throughput increases up to 26× and 24× 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 $\mathcal{O}(n^2 d)$ 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× and 2.5× 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 *Co*OadTR architecture achieved a relative FLOPs reduction of 255× 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 faction 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 ($\odot$):

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

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

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

5.3.1 Structured Pruning Low-rank Adapter

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

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

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

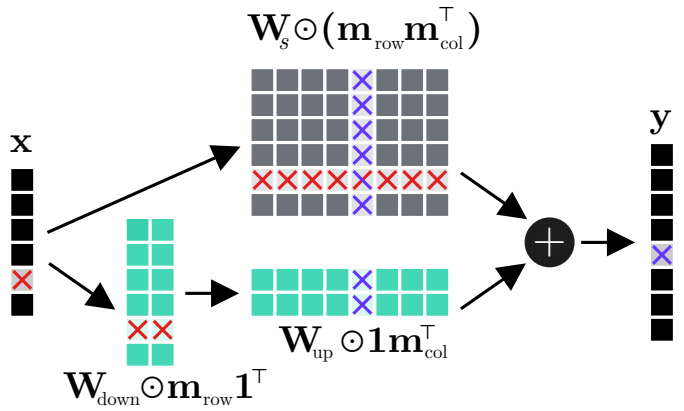


Figure 5.1: Structured Pruning Low-rank Adapter (SPLoRA). The in/out channel pruning mask affects the adapter as well as source weights.

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

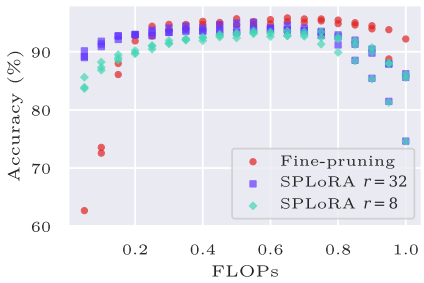
$$\mathbf{W}_t = \mathbf{W}_s \odot \mathbf{m}_{\text{row}} \mathbf{m}_{\text{col}}^\top + (\mathbf{W}_{\text{down}} \odot \mathbf{m}_{\text{row}} \mathbb{1}^\top)(\mathbf{W}_{\text{up}} \odot \mathbb{1} \mathbf{m}_{\text{col}}^\top). \tag{5.4}$$

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

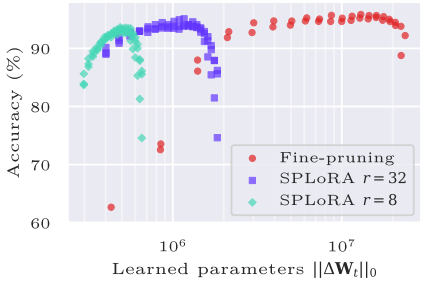
5.4 Experiments

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.

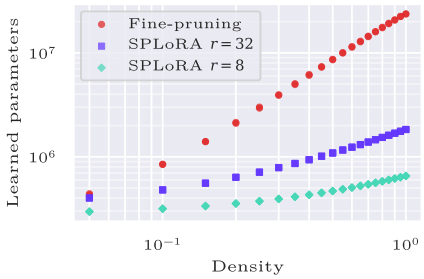
Table 5.1 present experimental results of the average accuracy across datasets shown for each combination of learning and pruning method at 30% and 10% weight density, *i.e.*, at respectively 70% and 90% pruned weights. Here, SPLoRA-$r8$, *i.e.*, SPLoRA with rank 8 bottle-neck, requires 19.38× fewer parameters on average at 30% density with only 1.60% lower accuracy. At 10% pruning, the model requires 4.65× fewer parameters while increasing accuracy by 5.61% on average. Likewise, the SPLoRA-$r32$ configuration requires 6.65× fewer parameters with only 0.54% lower accuracy on average at 30% density. At 10% it uses 2.05× fewer parameters



(a) Accuracy versus FLOPs.



(b) Accuracy versus learned parameters.



(c) Learned parameter count versus model density fraction.

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× 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× fewer learned parameters.

Pruning method	Learning method	Density	ΔParams (K)	FLOPs (M)	Avg. accuracy (%)
Unpruned	Fine-tuning	100%	23,520.8±0.0	1,304.7±0.0	**96.20**±0.05
	LoRA-$r32$	100%	1,644.5±0.0 (↓ 14.3×)	1,304.7±0.0	90.83±2.16 (−5.37)
	LoRA-$r8$	100%	**466.3**±0.0 (↓ 50.4×)	1,304.7±0.0	91.72±2.15 (−4.48)
Weight	Fine-pruning	30%	4,427.1±72.5	785.4±5.4	**96.22**±0.77
		10%	599.1±20.1	352.1±3.9	85.16±1.08
	SPLoRA-$r32$	30%	618.3±2.5 (↓ 7.2×)	778.6±2.9	96.20±0.31 (−0.02)
		10%	294.8±0.4 (↓ 2.0×)	335.4±7.3	**92.88**±0.82 (+7.72)
	SPLoRA-$r8$	30%	**210.0**±1.9 (↓ 21.1×)	773.4±2.1	95.15±0.17 (−1.07)
		10%	**128.9**±0.1 (↓ 4.6×)	338.9±7.0	91.27±0.65 (+6.11)
Gradient	Fine-pruning	30%	3,719.8±59.2	571.9±6.5	**96.13**±0.28
		10%	615.7±4.3	244.7±3.3	83.58±0.69
	SPLoRA-$r32$	30%	601.0±0.4 (↓ 6.2×)	564.6±1.5	95.55±0.20 (−0.58)
		10%	293.1±0.4 (↓ 2.1×)	244.0±1.9	**94.18**±0.33 (+10.60)
	SPLoRA-$r8$	30%	**205.2**±0.2 (↓ 18.1×)	565.2±4.1	94.61±0.22 (−1.52)
		10%	**128.3**±0.0 (↓ 4.8×)	245.4±4.0	91.97±0.40 (+8.39)
Taylor	Fine-pruning	30%	3,392.8±81.1	559.9±0.7	**95.61**±0.25
		10%	576.8±9.9	236.9±3.3	83.01±1.66
	SPLoRA-$r32$	30%	599.7±0.9 (↓ 5.7×)	555.5±6.7	95.37±0.25 (−0.24)
		10%	292.6±0.5 (↓ 2.0×)	242.0±1.5	**93.93**±0.04 (+10.92)
	SPLoRA-$r8$	30%	**205.3**±0.1 (↓ 16.5×)	566.2±10.9	94.46±0.29 (−1.15)
		10%	**128.4**±0.1 (↓ 4.5×)	243.2±9.8	91.60±0.35 (+8.59)
LRP	Fine-pruning	30%	4,428.1±20.6	719.9±0.7	**96.85**±0.10
		10%	608.4±6.3	301.6±2.3	90.54±1.40
	SPLoRA-$r32$	30%	592.6±0.9 (↓ 7.5×)	585.5±6.7	95.52±0.21 (−1.33)
		10%	290.9±0.2 (↓ 2.1×)	270.4±6.4	**93.90**±0.36 (+3.36)
	SPLoRA-$r8$	30%	**203.3**±0.5 (↓ 21.8×)	591.1±12.4	94.20±0.19 (−2.65)
		10%	**128.0**±0.1 (↓ 4.7×)	281.6±1.7	91.17±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 ± 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× and increase throughput up to 9× on CPU and 7× 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× FLOPs reductions as well as a 26× and 24× 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× for a sequence length of 64. A two-block version reduces complexity by 2.5×.

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× 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× 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 the 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. 5

[2] Llm failure archive. https://github.com/giuven95/chatgpt-failures, 2023. Accessed: 2023-02-24. 4

[3] Neuralmagic. https://neuralmagic.com, 2023. Accessed: 2023-02-06. 14

[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. 9, 17, 27

[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. 4, 5, 9

[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. 17, 19

[7] Junjie Bai, Fang Lu, Ke Zhang, et al. Onnx: Open neural network exchange. https://github.com/onnx/onnx, 2019. 17

[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. 16

[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. 16

[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. 5

[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. 10

[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/86e8f7ab32cfd12577bc2619bc635690-Paper.pdf. 6

[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. 9

[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. 19

[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. 39

[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. 9

[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. 19

[18] Cristian Buciluǎ, 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. 14

[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. 16, 17

[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. 30

[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. 6, 9, 17, 19, 26

[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. 17

[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. 6

[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. 16, 32

[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. 32

[26] Krzysztof Marcin Choromanski, Valerii Likhosherstov, 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. 16, 35

[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. 16

[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. 21

[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. 20, 39, 40

[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. 16

[31] BladeDISC developers. Bladedisc. https://github.com/alibaba/BladeDISC, 2022. 17

[32] DeepSparse developers. Deepsparse. https://github.com/neuralmagic/deepsparse, 2021. 17

[33] Nebuly Speedster developers. Speedster. https://github.com/nebuly-ai/nebullvm/tree/main/apps/accelerate/speedster, 2022. 17

[34] NVIDIA TensorRT developers. Nvidia tensorrt. https://github.com/NVIDIA/TensorRT, 2019. 17

[35] ONNX Runtime developers. Onnx runtime. https://www.onnxruntime.ai, 2021. 17

[36] OpenVINO developers. Openvino toolkit repository. https://github.com/openvinotoolkit/openvino, 2019. 17

[37] OpenXLA developers. Openxla. https://github.com/openxla/xla, 2022. 17

[38] TFLite developers. Tflite. https://github.com/tensorflow/tensorflow/tree/master/tensorflow/lite, 2022. 17

[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. 3, 9, 10

[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. 19

[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. 16

[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. 9, 16, 19, 35

[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. 10

[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. 48

[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. 46

[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=rkgO66VKDS. 16

[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. 39, 40

[48] Hyunjun Eun, Jinyoung Moon, Jongyoul Park, Chanho Jung, and Changick Kim. Temporal filtering networks for online action detection. *Pattern Recognition*, 111:107695, 2021. 40

[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. 20

[50] Jun Fang, Ali Shafiee, Hamzah Abdel-Aziz, David Thorsley, Georgios Georgiadis, 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. 16

[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. 3, 10, 11

[52] Christoph Feichtenhofer. X3D: Expanding architectures for efficient video recognition. *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020. 6, 16, 19, 27

[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. 9, 17, 19, 27

[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. 46

[55] Jiyang Gao, Zhenheng Yang, and Ram Nevatia. RED: reinforced encoder-decoder networks for action anticipation. In *British Machine Vision Conference (BMVC)*, 2017. 39, 40

[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. 5

[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. 15

[58] Ross B. Girshick. Fast r-cnn. *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1440–1448, 2015. 9

[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. 3

[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. 15

[61] Scott Gray, Alec Radford, and Diederik P. Kingma. Gpu kernels for block-sparse weights. *OpenAI*, 2017. URL https://cdn.openai.com/blocksparse/blocksparsepaper.pdf. 14

[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. 16

[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. 16

[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. 17

[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. 3, 9, 16

[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. 9

[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. 46

[68] Lukas Hedegaard. Supers, 2020. GitHub. https://github.com/lukashedegaard/supers. ix

[69] Lukas Hedegaard. Co-rider, 2021. GitHub, https://github.com/lukashedegaard/co-rider. ix

[70] Lukas Hedegaard. Ride, 2021. GitHub, https://github.com/lukashedegaard/ride. ix

[71] Lukas Hedegaard. PyTorch-Benchmark, 2022. GitHub, https://github.com/lukashedegaard/pytorch-benchmark. ix

[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. viii, 29, 30

[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. viii

[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. viii, 6

[75] Lukas Hedegaard, Aman Alok, Juby Jose, and Alexandros Iosifidis. Structured pruning adapters, 2022. URL https://arxiv.org/abs/2211.10155. viii

[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/B978-0-32-385787-1.00019-1. URL https://www.sciencedirect.com/science/article/pii/B9780323857871000191. viii

[77] Lukas Hedegaard, Illia Oleksiienko, and Christian Møldrup Legaard. DatasetOps, 2022. GitHub, https://github.com/lukashedegaard/datasetops. ix

[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. viii, 16

[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. viii

[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. 32

[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. 32

[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/B978-0-32-385787-1.00009-9. URL https://www.sciencedirect.com/science/article/pii/B9780323857871000099. viii

[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. 40

[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. 9

[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. 14

[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. 9, 10

[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. 46

[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. 16

[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. 6, 46

[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. 46, 47

[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. 16

[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. 40

[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. 15, 16

[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. 16

[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. 15

[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, Rishub 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. 4

[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. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4345–4354, 2018. 16

[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. 19, 27, 34, 40

[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. 31

[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. 40

[101] Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. In *International Conference on Learning Representations (ICLR)*, 2020. 35

[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. 16

[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. 19, 21

[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. 16

[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. 46

[106] Raghuraman Krishnamoorthi. Quantizing deep convolutional networks for efficient inference: A whitepaper. *CoRR*, abs/1806.08342, 2018. URL http://arxiv.org/abs/1806.08342. 16

[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. 48

[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. 3, 9

[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. 17, 21

[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. 14, 47

[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. 14, 48, 50

[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. 20

[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. 16, 21

[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. 34

[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. 9

[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. 9

[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. 31

[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. 4

[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. 9

[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. 6

[121] R.G. Lyons. *Understanding Digital Signal Processing*. Prentice Hall, 2011. ISBN 9780137027415. 24

[122] Rabeeh Karimi Mahabadi, James Henderson, and Sebastian Ruder. Compacter: Efficient low-rank hypercomplex adapter layers. In *Advances in Neural Information Processing Systems*, 2021. 46

[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. 47

[124] Jacques Matthei. Another way of looking at lee sedol vs alphago, March 2016. URL https://jacquesmattheij.com/another-way-of-looking-at-lee-sedol-vs-alphago/. 5

[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. 3

[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. 15

[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. 6

[128] Szymon Migacz. Nvidia 8-bit inference with tensorrt. In *GPU Technology Conference*, volume 10, pages 1–41, 2017. 15

[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. 14

[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. 15

[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. 14

[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. 9

[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. 14, 48, 50

[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. 16

[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. 4, 5, 9

[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. 19

[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. 48

[138] Graduate School of Technical Sciences. Rules and Regulations, Jan 2023. URL https://phd.tech.au.dk/for-phd-students/rules-regulations. vii

[139] OpenAI. Chatgpt: Optimizing language models for dialogue. https://openai.com/blog/chatgpt/, 2022. 3

[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. 4

[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. 9

[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. 11

[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. 16, 35

[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. ix

[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. 9, 17, 27

[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. 32

[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. 46

[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. 9

[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. 46

[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. 46

[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. 17

[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. 9, 31, 33

[153] Chris Pointon. The carbon footprint of chatgpt, 2023. Medium.com, https://medium.com/@chrispointon/the-carbon-footprint-of-chatgpt-e1bc14e4cc2a. 5

[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. 15

[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. 14

[156] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019. 3

[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. 3

[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. 9

[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. 9

[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. 47

[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. 3

[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. 14, 46

[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. 7, 46

[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. 9

[165] Bernhard Riemann. *Ueber die Darstellbarkeit einer Function durch eine trigonometrische Reihe*. Dieterich, 1867. URL http://eudml.org/doc/203787. 15

[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. 9

[167] F. Rosenblatt. The perceptron - a perceiving and recognizing automaton. Technical Report 85-460-1, Cornell Aeronautical Laboratory, Ithaca, New York, January 1957. 3

[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. 4, 19

[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. 3

[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. 3

[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. 14, 15

[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. 14, 15, 47

[173] Teven Le Scao, Angela Fan, Christopher Akiki, and colleagues. Bloom: A 176b-parameter open-access multilingual language model. *ArXiv*, abs/2211.05100, 2022. 5, 45

[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. 16

[175] James Sevilla. Parameter counts in machine learning, 2022. Towards Datascience, https://towardsdatascience.com/parameter-counts-in-machine-learning-a312dc4753d0. 5

[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. 31, 33, 34

[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. 15

[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. 16

[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. 6, 9, 33

[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. 31

[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. 31

[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. 29

[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. 4

[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. 4

[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. 17, 21, 30

[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. 4, 9

[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. 14, 48, 49, 50

[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. 10, 16, 19

[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. 17

[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. 14

[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. 4

[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/. 4

[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. 15

[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. 9, 16, 19, 27

[195] Dat Thanh Tran, Alexandros Iosifidis, and Moncef Gabbouj. Improving efficiency in convolutional neural network with multilinear filters. *Neural Networks*, 105:328–339, 2018. 16

[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. 17

[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. 9

[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. 41

[199] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, pages 5998–6008, 2017. 3, 6, 9, 16, 17, 19, 22, 34

[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. 9

[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. 40

[202] Sinong Wang, Belinda Z. Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *arXiv:2006.04768*, 2020. 16, 35

[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. 17, 20, 39, 40

[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. 19

[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. 21

[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. 16

[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. 17

[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. 15

[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. 15

[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. 16, 35

[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. 39, 40

[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. 17, 20, 39, 40

[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. 6, 17

[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. 9, 22, 31, 33

[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. 16

[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. 16

[217] Xiahui 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. 6

[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. 14, 47, 48, 50

[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/375c71349b295fbe2dcdca9206f20a06-Paper.pdf. 46

[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. 16

[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. 15

[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. 4, 9

[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. 19

[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. 14

[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. 31

[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. 16

[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. 14

[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. 40

[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. 14

[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. 16

[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. 14

[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. 46

[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. 17

[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. 17

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.

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

Lukas Hedegaard and Alexandros Iosifidis

Department of Electrical and Computer Engineering, Aarhus University, Denmark
{lhm,ai}@ece.au.dk

Abstract. We introduce *Continual* 3D Convolutional Neural Networks (*Co*3D CNNs), a new computational formulation of spatio-temporal 3D CNNs, in which videos are processed frame-by-frame rather than by clip. In online tasks demanding frame-wise predictions, *Co*3D CNNs dispense with the computational redundancies of regular 3D CNNs, namely the repeated convolutions over frames, which appear in overlapping clips. We show that *Continual* 3D CNNs can reuse preexisting 3D-CNN weights to reduce the per-prediction floating point operations (FLOPs) in proportion to the temporal receptive field while retaining similar memory requirements and accuracy. This is validated with multiple models on Kinetics-400 and Charades with remarkable results: *Co*X3D models attain state-of-the-art complexity/accuracy trade-offs on Kinetics-400 with 12.1−15.3× reductions of FLOPs and 2.3−3.8% improvements in accuracy compared to regular X3D models while reducing peak memory consumption by up to 48%. Moreover, we investigate the transient response of *Co*3D CNNs at start-up and perform extensive benchmarks of on-hardware processing characteristics for publicly available 3D CNNs.

Keywords: 3D CNN, Human Activity Recognition, Efficient, Stream Processing, Online Inference, Continual Inference Network.

1 Introduction

Through the availability of large-scale open-source datasets such as ImageNet [37] and Kinetics [25], [4], deep, over-parameterized Convolutional Neural Networks (CNNs) have achieved impressive results in the field of computer vision. In video recognition specifically, 3D CNNs have lead to multiple breakthroughs in the state-of-the-art [3], [43], [11], [10]. Despite their success in competitions and benchmarks where only prediction quality is evaluated, computational cost and processing time remains a challenge to the deployment in many real-life use-cases with energy constraints and/or real-time needs. To combat this general issue, multiple approaches have been explored. These include computationally efficient architectures for image [17], [48], [42] and video recognition [28], [10], [49], pruning of network weights [6], [13], [14], knowledge distillation [16], [47], [36], and network quantisation [19], [2], [12].

The contribution in this paper is complementary to all of the above. It exploits the computational redundancies in the application of regular spatio-temporal 3D CNNs to a continual video stream in a sliding window fashion (Fig. 2). This redundancy was also explored recently [26], [39] using specialised architectures. However, these are not weight-compatible with regular 3D CNNs. We present a weight-compatible reformulation of the 3D CNN and its components as a *Continual* 3D Convolutional Neural Network (*Co*3D CNN). *Co*3D CNNs process input videos frame-by-frame rather than clip-wise and can reuse the weights of regular 3D CNNs, producing identical outputs for networks without temporal zero-padding. Contrary to most deep learning papers, the work presented here needed no training; our goal was to validate the efficacy of converting regular 3D CNNs to Continual CNNs directly, and to explore their characteristics in the online recognition domain. Accordingly, we perform conversions from five 3D CNNs, each at different points on the accuracy/speed pareto-frontier, and evaluate their frame-wise performance. While there is a slight reduction in accuracy after conversion due to zero-padding in the regular 3D CNNs, a simple network modification of extending the temporal receptive field recovers and improves the accuracy significantly *without* any fine-tuning at a negligible increase in computational cost. Furthermore, we measure the transient network response at start-up, and perform extensive benchmarking on common hardware and embedded devices to gauge the expected inference speeds for real-life scenarios. Full source code is available at `https://github.com/lukashedegaard/co3d`.

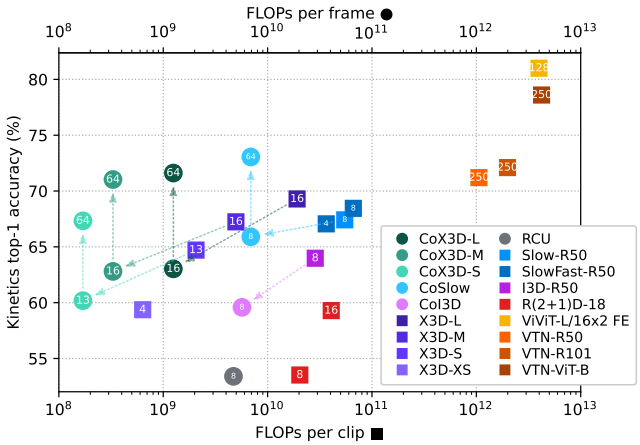


Fig. 1: **Accuracy/complexity trade-off** for *Continual* 3D CNNs and recent state-of-the-art methods on Kinetics-400 using 1-clip/frame testing. ■ FLOPs per *clip* are noted for regular networks, while ● FLOPs per *frame* are shown for the Continual 3D CNNs. Frames per clip / global average pool size 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 receptive field.

2 Related Works

2.1 3D CNNs for video recognition

Convolutional Neural Networks with spatio-temporal 3D kernels may be considered the natural extension of 2D CNNs for image recognition to CNNs for video recognition. Although they did not surpass their 2D CNN + RNN competitors [7], [21] initially [20], [23], [44], arguably due to a high parameter count and insufficient dataset size, 3D CNNs have achieved state-of-the-art results on Human Action Recognition tasks [3], [43], [11] since the Kinetics dataset [25] was introduced. While recent large-scale Transformer-based methods [1], [32] have become leaders in terms of accuracy, 3D CNNs still achieve state-of-the-art accuracy/complexity trade-offs. Nevertheless, competitive accuracy comes with high computational cost, which is prohibitive to many real-life use cases.

In image recognition, efficient architectures such as MobileNet [17], ShuffleNet [48], and EfficientNet [42] attained improved accuracy-complexity trade-offs. These architectures were extended to the 3D-convolutional versions 3D-MobileNet [28], 3D-ShuffleNet [28] and X3D [10] (≈3D-EfficientNet) with similarly improved pareto-frontier in video-recognition tasks. While these efficient 3D CNNs work well for offline processing of videos, they are limited in the context of online processing, where we wish to make updates predictions for each frame; real-time processing rates can only be achieved with the smallest models at severely reduced accuracy. 3D CNNs suffer from the restriction that they must process a whole "clip" (spatio-temporal volume) at a time. When predictions are needed for each frame, this imposes a significant overhead due to repeated computations. In our work, we overcome this challenge by introducing an alternative computational scheme for spatio-temporal convolutions, -pooling, and -residuals, which lets us compute 3D CNN outputs frame-wise (continually) and dispose of the redundancies produced by regular 3D CNNs.

2.2 Architectures for online video recognition

A well-explored approach to video-recognition [7], [21], [22], [40] is to let each frame pass through a 2D CNN trained on ImageNet in one stream alongside a second stream of Optical Flow [9] and integrate these using a recurrent network. Such architectures requires no network modification for deployment in online-processing scenarios, lends themselves to caching [46], and are free of the computational redundancies experienced in 3D CNNs. However, the overhead of Optical Flow and costly feature-extractors pose a substantial disadvantage.

Another approach is to utilise 3D CNNs for feature extraction. In [31], spatio-temporal features from non-overlaping clips are used to train a recurrent network for hand gesture recognition. In [27], a 3D CNN processes a sliding window of the input to perform spatio-temporal action detection. These 3D CNN-based methods have the disadvantage of either not producing predictions for each input frame [31] or suffering from redundant computations from overlapping clips [27].

Massively Parallel Video Networks [5] split a DNN into depth-parallel sub-networks across multiple computational devices to improve online multi-device parallel processing performance. While their approach treats networks layers as atomic operations and doesn't tackle the fundamental redundancy of temporal convolutions, *Continual* 3D CNNs reformulate the network layers, remove redundancy, and accelerate inference on single devices as well.

Exploring modifications of the spatio-temporal 3D convolution, the Recurrent Convolutional Unit (RCU) [39] replaces the 3D convolution by aggregating a spatial 2D convolution over the current input with a 1D convolution over the prior output. Dissected 3D CNNs [26] (D3D) cache the $1 \times n_H \times n_W$ frame-level features in network residual connections and aggregate them with the current frame features via $2 \times 3 \times 3$ convolutions. Like our proposed *Continual* 3D CNNs, both RCU and D3D are causal and operate frame-by-frame. However, they are speciality architectures, which are incompatible with pre-trained 3D CNNs, and must be trained from scratch. We reformulate spatio-temporal convolutions in a one-to-one compatible manner, allowing us to reuse existing model weights.

3 Continual Convolutional Neural Networks

3.1 Regular 3D-convolutions lead to redundancy

Currently, the best performing architectures (e.g., X3D [10] and SlowFast [11]) employ variations on 3D convolutions as their main building block and perform predictions for a spatio-temporal input volume (video-clip). These architectures achieve high accuracy with reasonable computational cost for predictions on clips in the offline setting. They are, however, ill-suited for online video classification, where the input is a continual stream of video frames and a class prediction is needed for each frame. For regular 3D CNNs processing clips of m_T frames to be used in this context, prior $m_T - 1$ input frames need to be stored between temporal time-steps and assembled to form a new video-clip when the next frame is sampled. This is illustrated in Fig. 2.

Recall the computational complexity for a 3D convolution:

$$\Theta([k_H \cdot k_W \cdot k_T + b] \cdot c_I \cdot c_O \cdot n_H \cdot n_W \cdot n_T), \tag{1}$$

where k denotes the kernel size, T, H, and W are time, height, and width dimension subscripts, $b \in \{0, 1\}$ indicates whether bias is used, and c_I and c_O are the number of input and output channels. The size of the output feature map is $n = (m + 2p - d \cdot (k - 1) - 1)/s + 1$ for an input of size m and a convolution with padding p, dilation d, and stride s. During online processing, every frame in the continual video-stream will be processed n_T times (once for each position in the clip), leading to a redundancy proportional with $n_T - 1$. Moreover, the memory-overhead of storing prior input frames is

$$\Theta(c_I \cdot m_H \cdot m_W \cdot [m_T - 1])), \tag{2}$$

and during inference the network has to transiently store feature-maps of size

$$\Theta(c_O \cdot n_H \cdot n_W \cdot n_T). \tag{3}$$

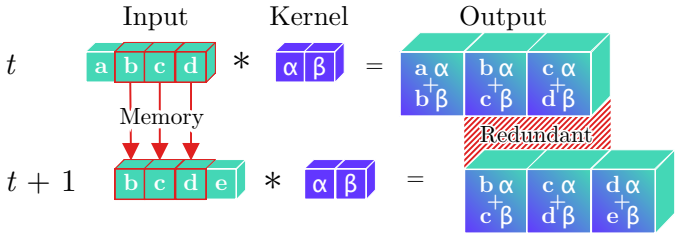


Fig. 2: **Redundant computations** for a temporal convolution during online processing, as illustrated by the repeated convolution of inputs (green $\mathbf{b}, \mathbf{c}, \mathbf{d}$) with a kernel (blue α, β) in the temporal dimension. Moreover, prior inputs ($\mathbf{b}, \mathbf{c}, \mathbf{d}$) must be stored between time-steps for online processing tasks.

3.2 Continual Convolutions

We can remedy the issue described in Sec. 3.1 by employing an alternative sequence of computational steps. In essence, we reformulate the repeated convolution of a (3D) kernel with a (3D) input-clip that continually shifts along the temporal dimension as a *Continual* Convolution (*Co*Conv), where all convolution computations (bar the final sum) for the (3D) kernel with each (2D) input-frame are performed in one time-step. Intermediary results are stored as states to be used in subsequent steps, while previous and current results are summed up to produce the output. The process for a 1D input and kernel, which corresponds to the regular convolution in Fig. 2, is illustrated in Fig. 3. In general, this scheme can be applied for online-processing of any ND input, where one dimension is a temporal continual stream. Continual Convolutions are causal [34] with no information leaking from future to past and can be efficiently implemented by zero-padding the input frame along the temporal dimension with $p = \text{floor}(k/2)$. Python-style pseudo-code of the implementation is shown in Listing 1.1.

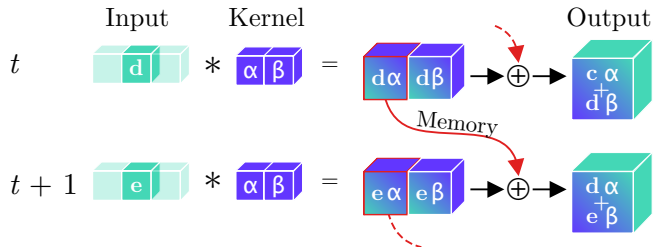


Fig. 3: **Continual Convolution**. An input (green $\mathbf{d}$ or $\mathbf{e}$) is convolved with a kernel (blue α, β). The intermediary feature-maps corresponding to all but the last temporal position are stored, while the last feature map and prior memory are summed to produce the resulting output. For a continual stream of inputs, Continual Convolutions produce identical outputs to regular convolutions.

```
def coconv3d(frame, prev_state = (mem, i)):
    frame = spatial_padding(frame)
    frame = temporal_padding(frame)
    feat = conv3d(frame, weights)
    output, rest_feat = feat[0], feat[1:]
    mem, i = prev_state or init_state(output)
    M = len(mem)
    for m in range(M):
        output += mem[(i + m) % M, M - m - 1]
    output += bias
    mem[i] = rest_feat
    i = (i + 1) % M
    return output, (mem, i)
```

Listing 1.1: **Pseudo-code** for Continual Convolution. Ready-to-use modules are available in the Continual Inference library [15].

In terms of computational cost, we can now perform frame-by-frame computations much more efficiently than a regular 3D convolution. The complexity of processing a frame becomes:

$$\Theta([k_H \cdot k_W \cdot k_T + b] \cdot c_I \cdot c_O \cdot n_H \cdot n_W). \tag{4}$$

This reduction in computational complexity comes at the cost of a memory-overhead in each layer due to the state that is kept between time-steps. The overhead of storing the partially computed feature-maps for a frame is:

$$\Theta(d_T \cdot [k_T - 1] \cdot c_O \cdot n_H \cdot n_W). \tag{5}$$

However, in the context of inference in a deep neural network, the transient memory usage within each time-step is reduced by a factor of n_T to

$$\Theta(c_O \cdot n_H \cdot n_W). \tag{6}$$

The benefits of Continual Convolutions include the independence of clip length on the computational complexity, state overhead, and transient memory consumption. The change from (non-causal) regular convolutions to (causal) Continual Convolutions has the side-effect of introducing a delay to the output. This is because some intermediary results of convolving a frame with the kernel are only added up at a later point in time (see Fig. 3). The delay amounts to

$$\Theta(d_T \cdot [k_T - p_T - 1]). \tag{7}$$

3.3 Continual Residuals

The delay from Continual Convolutions has an adverse side-effect on residual connections. Despite their simplicity in regular CNNs, we cannot simply add the input to a Continual Convolution with its output because the *Co*Conv may delay the output. Residual connections to a *Co*Conv must therefore be delayed by an equivalent amount (see Eq. (7)). This produces a memory overhead of

$$\Theta(d_T \cdot [k_T - 1] \cdot c_O \cdot m_H \cdot m_W). \tag{8}$$

3.4 Continual Pooling

The associative property of pooling operations allows for pooling to be decomposed across dimensions, i.e. $\text{pool}_{T,H,W}(\mathbf{X}) = \text{pool}_T(\text{pool}_{H,W}(\mathbf{X}))$. For continual spatio-temporal pooling, the pooling over spatial dimensions is equivalent to a regular pooling, while the intermediary pooling results must be stored for prior temporal frames. For a pooling operation with temporal kernel size k_T and spatial output size $n_H \cdot n_W$, the memory consumption and delays are

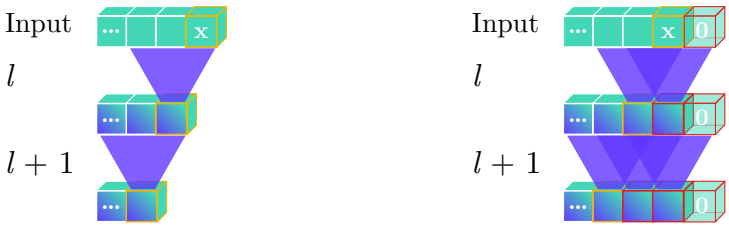
$$\Theta([k_T - 1] \cdot n_H \cdot n_W), \tag{9}$$

$$\Theta(k_T - p_T - 1). \tag{10}$$

Both memory consumption and delay scale linearly with the temporal kernel size. Fortunately, the memory consumed by temporal pooling layers is relatively modest for most CNN architectures (1.5% for *Co*X3D-M, see Appendix A). Hence, the delay rather than memory consumption may be of primary concern for real-life applications. For some network modules it may even make sense to skip the pooling in the conversion to a Continual CNN. One such example is the 3D Squeeze-and-Excitation (SE) block [18] in X3D, where global spatio-temporal average-pooling is used in the computation of channel-wise self-attention. Discarding the temporal pooling component (making it a 2D SE block) shifts the attention slightly (assuming the frame contents change slowly relative to the sampling rate) but avoids a considerable temporal delay.

3.5 The issue with temporal padding

Zero-padding of convolutional layers is a popular strategy for retaining the spatio-temporal dimension of feature-maps in consecutive CNN layers. For Continual CNNs, however, temporal zero-padding poses a problem, as illustrated in Fig. 4. Consider a 2-layer 1D CNN where each layer has a kernel size of 3 and zero padding of 1. For each new frame in a continual stream of inputs, the first layer l should produce two output feature-maps: One by the convolution of the two prior frames and the new frame, and another by convolving with one prior frame, the new frame, and a zero-pad. The next layer $l + 1$ thus receives two inputs and produces three outputs which are dependent on the new input frame of the first layer (one for each input and another from zero-padding). In effect, each zero padding in a convolution forces the next layer to retrospectively update its output for a previous time-step in a non-causal manner. Thus, there is a considerable downside to the use of padding. This questions the necessity of zero padding along the temporal dimension. In regular CNNs, zero padding has two benefits: It helps to avoid spatio-temporal shrinkage of feature-maps when propagated through a deep CNN, and it prevents information at the boarders from "washing away" [24]. The use of zero-padding, however, has the downside that it alters the input-distribution along the boarders significantly [29], [33]. For input data which is a continual stream of frames, a shrinkage of the feature-size in the temporal dimension is not a concern, and an input frame (which may be



(a) No padding (b) Zero padding

Fig. 4: **Issue with temporal padding**: The latest frame $\mathbf{x}$ is propagated through a CNN with (purple) temporal kernels of size 3 (a) without or (b) with zero padding. Highlighted cubes can be produced only in the latest frame, with yellow boarder indicating independence of padded zero and red boarders dependence. In the zero-padded case (b), the number of frame features dependent on $\mathbf{x}$ following a layer l increases with the number of padded zeros.

considered a border frame in a regular 3D CNN) has no risk of "washing away" because it is a middle frame in subsequent time steps. Temporal padding is thus omitted in Continual CNNs. As can be seen in the experimental evaluations presented in the following, this constitutes a "model shift" in the conversion from regular to Continual 3D CNN if the former was trained with temporal padding.

3.6 Initialisation

Before a Continual CNN reaches a steady state of operation, it must have processed $r_T - p_T - 1$ frames where r_T and p_T are the aggregated temporal receptive field and padding of the network. For example, Continual X3D-{S, M, L} models have receptive fields of size $\{69, 72, 130\}$, aggregated padding $\{28, 28, 57\}$, and hence need to process $\{40, 43, 72\}$ frames prior to normal operation. The initial response depends on how internal state variables are initialised. In Sec. 4.2, we explore this further with two initialisation variants: 1) Initialisation with *zeros* and 2) by repeating a *replicate* of the features corresponding to the first input-frame. The latter corresponds to operating in a steady state for a "boring video" [3] which has one frame repeated in the entire clip.

3.7 Design considerations

Disregarding the storage requirement of model weights (which is identical between for regular and continual 3D CNNs), X3D-M has a worst-case total memory-consumption of 7,074,816 floats when prior frames and the transient feature-maps are taken into account. Its continual counterpart, *Co*X3D-M, has a worst case memory only 5,072,688 floats. How can this be? Since Continual 3D CNNs do not store prior input frames and have smaller transient feature maps, memory savings outweigh the cost of caching features in each continual layer. Had the

clip size been four instead of sixteen, X3D-M_4 would have had a worst-case memory consumption of 1,655,808 floats and *Co*X3D-M_4 of 5,067,504 floats. For clip size 64, X3D-M_{64} consumes 28,449,792 floats and *Co*X3D-M_{64} uses 5,093,424 floats. The memory load of regular 3D CNNs is thus highly dependent on clip size, while that of *Co*3D CNNs is not. Continual CNNs utilise longer receptive fields much more efficiently than regular CNNs in online processing scenarios. In networks intended for embedded systems or online processing, we may increase the clip size to achieve higher accuracy with minimal penalty in computational complexity and worst-case memory.

Another consideration, which influences memory consumption is the temporal kernel size and dilation of *Co*Conv layers. Fortunately, the trend to employ small kernel sizes leaves the memory consumption reasonable for recent 3D CNNs [3], [43], [11], [10]. A larger temporal kernel size would not only affect the memory growth through the *Co*Conv filter, but also for co-occuring residual connections. These consume a significant fraction of the total state-memory for real-life networks: in a Continual X3D-M model (*Co*X3D-M) the memory of residuals constitutes 20.5% of the total model state memory (see Appendix A).

3.8 Training

*Co*3D CNNs are trained with back-propagation like other neural networks. However, special care must be taken in the estimation of data statistics in normalisation layers: 1) Momentum should be adjusted to $\text{mom}_{\text{step}} = 2/(1 + \text{timesteps} \cdot (2/\text{mom}_{\text{clip}} - 1))$ to match the exponential moving average dynamics of clip-based training, where T is the clip size; 2) statistics should not be tracked for the transient response. Alternatively, they can be trained offline in their "unrolled" regular 3D-CNN form with no temporal padding. This is similar to utilising pre-trained weights from a regular 3D CNN, as we do in our experiments.

4 Experiments

The experiments in this section aim to show the characteristics and advantages of Continual 3D CNNs as compared with regular 3D CNNs. One of the main benefits of *Co*3D CNNs is their ability to reuse the network weights of regular 3D CNNs. As such, all *Co*3D CNNs in these experiments use publicly available pre-trained network weights of regular 3D CNNs [11], [10], [8] without further fine-tuning. Data pre-processing follows the respective procedures associated with the originating weights unless stated otherwise. The section is laid out as follows: First, we showcase the network performance following weight transfer from regular to Continual 3D on multiple datasets for Human Activity Recognition. This is followed by a study on the transient response of *Co*3D CNNs at startup. Subsequently, we show how the computational advantages of *Co*3D CNNs can be exploited to improve accuracy by extending the temporal receptive field. Finally, we perform an extensive on-hardware benchmark of prior methods and Continual 3D CNNs, measuring the 1-clip/frame accuracy of publicly available models, as well as their inference throughput on various computational devices.

	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-8×8-R50	67.42	32.45	266.04	54.87	0.38	1.34	4.31	61.92
	SlowFast-8×8-R50	68.45	66.25	344.84	66.25	0.34	0.87	2.72	30.72
	SlowFast-4×16-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
	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
Frame	RCU_8 [39]†	53.40	12.80	-	4.71	-	-	-	-
	Co$I3D_8$	59.58	28.04	235.87	5.68	3.00	2.41	14.88	125.59
	Co$I3D_{64}$	56.86	28.04	236.08	5.68	3.15	2.41	14.89	126.32
	Co$Slow_8$	65.90	32.45	175.98	6.90	2.80	1.60	6.18	113.77
	***Co*Slow$_{64}$**	**73.05**	**32.45**	**176.41**	**6.90**	**2.92**	**1.60**	**6.19**	**102.00**
	*Co*X3D-L_{16}	63.03	6.15	184.29	1.25	2.30	0.99	25.17	206.65
	***Co*X3D-L$_{64}$**	**71.61**	**6.15**	**184.37**	**1.25**	**2.30**	**0.99**	**27.56**	**217.53**
	*Co*X3D-M_{16}	62.80	3.79	68.88	0.33	7.57	7.26	88.79	844.73
	***Co*X3D-M$_{64}$**	**71.03**	**3.79**	**68.96**	**0.33**	**7.51**	**7.04**	**86.42**	**796.32**
	*Co*X3D-S_{13}	60.18	3.79	41.91	0.17	13.16	11.06	219.64	939.72
	***Co*X3D-S$_{64}$**	**67.33**	**3.79**	**41.99**	**0.17**	**13.19**	**11.13**	**213.65**	**942.97**

Table 1: **Kinetics-400 benchmark**. 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 speed comparison, predictions per second denote frames per second for the *Co*X3D 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. †Approximate FLOPs derived from paper (see Appendix C).

4.1 Transfer from regular to Continual CNNs

To gauge direct transferability of 3D CNN weights, we implement continual versions of various 3D CNNs and initialise them with their publicly available weights for Kinetics-400 [25] and Charades [38]. While it is common to use an ensemble prediction from multiple clips to boost video-level accuracy on these benchmarks, we abstain from this, as it doesn't apply to online-scenarios. Instead, we report the single-clip/frame model performance.

Kinetics-400. We evaluate the X3D network variants XS, S, M, and L on the test set using one temporally centred clip from each video. The XS network is omitted in the transfer to *Co*X3D, given that it is architecturally equivalent to S, but with fewer frames per clip. In evaluation on Kinetics-400, we faced the challenge that videos were limited to 10 seconds. Due to the longer transient response of Continual CNNs (see Sec. 4.2) and low frame-rate used for training X3D models

	Model	FLOPs (G) × views	mAP (%)
Clip	Slow-8×8 [11]	54.9 × 30	39.0
	Slow-8×8 [11]†	54.9 × 1	21.4
	Slow-8×8 (ours)	54.9 × 1	24.1
Fr.	*Co*Slow$_8$	6.9 × 1	21.5
	*Co*Slow$_{64}$	6.9 × 1	25.2

Table 2: **Charades benchmark.** Noted are the FLOPs × views and video-level mean average precision (mAP) on the validation set using pre-trained model weights. †Results achieved using the publicly available SlowFast code [11].

(5.0, 6.0, 6.0 FPS for S, M, and L), the video-length was insufficient to reach steady-state for some models. As a practical measure to evaluate near steady-state, we repeated the last video-frame for a padded video length of ≈ 80% of the network receptive field as a heuristic choice. The Continual CNNs were thus tested on the last frame of the padded video and initialised with the prior frames. The results of the X3D transfer are shown in Tab. 1 and Fig. 1.

For all networks, the transfer from regular to Continual 3D CNN results in significant computational savings. For the S, M, and L networks the reduction in FLOPs is 12.1×, 15.1×, and 15.3× respectively. The savings do not quite reach the clip sizes since the final pooling and prediction layers are active for each frame. As a side-effect of the transfer from zero-padded regular CNN to Continual CNN without zero-padding, we see a notable reduction in accuracy. This is easily improved by using an extended pooling size for the network (discussed in Sec. 3.7 and in Sec. 4.2). Using a global average pooling with temporal kernel size 64, we improve the accuracy of X3D by 2.6%, 3.8%, and 2.3% in the Continual S, M, and L network variants. As noted, Kinetics dataset did not have sufficient frames to fill the temporal receptive field of all models in these tests. We explore this further in Sections 4.2 and 4.2.

Charades. To showcase the generality of the approach, we repeat the above described procedure with another 3D CNN, the *Co*Slow network [11]. We report the video-level mean average precision (mAP) of the validation split alongside the FLOPs per prediction in Tab. 2. Note the accuracy discrepancy between 30 view (10 temporal positions with 3 spatial positions each) and 1 view (spatially and temporally centred) evaluation. As observed on Kinetics, the *Co*Slow network reduces the FLOPs per prediction proportionally with the original clip size (8 frames), and can recover accuracy by extending the global average pool size.

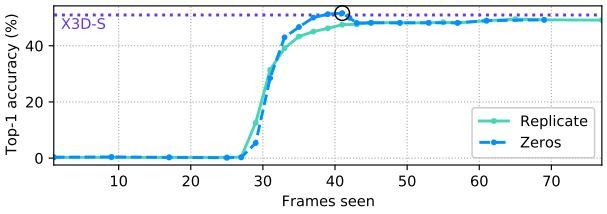
4.2 Ablation Experiments

As described in Sec. 3.6, Continual CNNs exhibit a transient response during their up-start. In order to gauge this response, we perform ablations on the

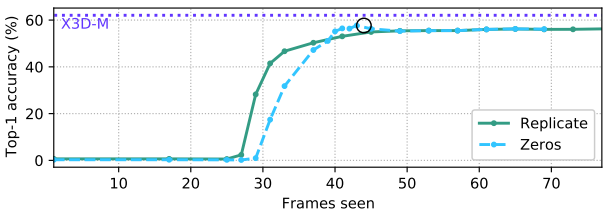
Kinetics-400 validation set, this time sampled at 15 FPS to have a sufficient number of frames available. This corresponds to a data domain shift [45] relative to the pre-trained weights, where time advances slower.

Transient response of Continual CNNs. Our expected upper bound is given by the baseline X3D network 1-clip accuracy at 15 FPS. The transient response is measured by varying the number of prior frames used for initialisation before evaluating a frame using the *Co*X3D model. Note that temporal center-crops of size $T_{\text{init}} + 1$, where T_{init} is the number of initialisation frames, are used in each evaluation to ensure that the frames seen by the network come from the centre. This precaution counters a data-bias, we noticed in Kinetics-400, namely that the start and end of a video are less informative and contribute to worse predictions than the central part. We found results to vary up to 8% for a X3D-S network evaluated at different video positions. The experiment is repeated for two initialisation schemes, "zeros" (used in other experiments) and "replicate", and two model sizes, S and M. The transient responses are shown in Fig. 5.

For all responses, the first ≈25 frames produce near-random predictions, before rapidly increasing at 25−30 frames until a steady-state is reached at 49.2% and 56.2% accuracy for S and M. Relative to the regular X3D, this constitutes a steady-state error of −1.7% and −5.8%. Comparing initialisation schemes, we see that the "replicate" scheme results in a slightly earlier rise. The rise sets in later for the "zeros" scheme, but exhibits a sharper slope, topping with peaks of 51.6% and 57.6% at 41 and 44 frames seen as discussed in Sec. 3.6. This



(a) *Co*X3D-S



(b) *Co*X3D-M

Fig. 5: **Transient response** for Continual X3D-{S,M} on the Kinetics-400 val at 15 FPS. Dotted horizontal lines denote X3D validation accuracy for 1-clip predictions. Black circles highlight the theoretically required initialisation frames.

makes sense considering that the original network weights were trained with this exact amount of zero-padding. Adding more frames effectively replaces the padded zeros and causes a slight drop of accuracy in the steady state, where the accuracy settles at the same values as for the "replication" scheme.

Extended receptive field. Continual CNNs experience a negligible increase in computational cost when larger temporal receptive field are used (see Sec. 3.7). For *Co*X3D networks, this extension can be trivially implemented by increasing the temporal kernel size of the last pooling layer. In this set of experiments, we extend *Co*X3D-{S,M,L} to have temporal pooling sizes 32, 64, and 96, and evaluate them on the Kinetics-400 validation set sampled at 15 FPS. The Continual CNNs are evaluated at frames corresponding to the steady state.

Tab. 3 shows the measured accuracy and floating point operations per frame (*Co*X3D) / clip (X3D) as well as the pool size for the penultimate network layer (global average pooling) and the total receptive field of the network in the temporal dimension. As found in Sec. 4.1, each transfer results in significant computational savings alongside a drop in accuracy. Extending the kernel size of the global average pooling layer increases the accuracy of the Continual CNNs by 11.0−13.3% for 96 frames relative the original 13−16 frames, surpassing that of the regular CNNs. Lying at 0.017−0.009%, the corresponding computational increases can be considered negligible.

Model	**Size**	**Pool**	**Acc.**	**FLOPs (K)**	**Rec. Field**
X3D	S	13	51.0	2,061,366	13
	M	16	62.1	4,970,008	16
	L	16	64.1	19,166,052	16
*Co*X3D	S	13	49.2	166,565	69
		16	50.1	166,567	72
		32	54.7	166,574	88
		64	59.8	166,587	120
		96	*61.8*	166,601	152
	M	16	56.3	325,456	72
		32	60.7	325,463	88
		64	64.9	325,477	120
		96	*67.3*	325,491	152
	L	16	53.0	1,245,549	130
		32	58.5	1,245,556	146
		64	*64.3*	1,245,570	178
		96	*66.3*	1,245,584	210

Table 3: **Effect of extending pool size**. Note that the model weights were trained at different sampling rates than evaluated at (15 FPS), resulting in a lower top-1 val. accuracy. *Italic numbers* denote measurement taken within the transient response due to a lack of frames in the video-clip.

4.3 Inference benchmarks

Despite their high status in activity recognition leader-boards [35], it is unclear how recent 3D CNNs methods perform in the online setting, where speed and accuracy constitute a necessary trade-off. To the best of our knowledge, there has not yet been a systematic evaluation of throughput for these video-recognition models on real-life hardware. In this set of experiments, we benchmark the FLOPs, parameter count, maximum allocated memory and 1-clip/frame accuracy of I3D [3], R(2+1)D [43], SlowFast[41], X3D [10], *Co*I3D, *Co*Slow, and *Co*X3D. To gauge achievable throughputs at different computational budgets, networks were tested on four hardware platforms as described in Appendix B.

As seen in the benchmark results found in Tab. 1, the limitation to one clip markedly lowers accuracy compared with the multi-clip evaluation published in the respective works [3], [43], [11], [10]. Nontheless, the Continual models with extended receptive fields attain the best accuracy/speed trade-off by a large margin. For example, *Co*X3D-L_{64} on the Nvidia Jetson Xavier achieves an accuracy of 71.3% at 27.6 predictions per second compared to 67.2% accuracy at 17.5 predictions per second for X3D-M while reducing maximum allocated memory by 48%! Confirming the observation in [30], we find that the relation between model FLOPs and throughput varies between models, with better ratios attained for simpler models (e.g., I3D) than for complicated ones (e.g., X3D). This relates to different memory access needs and their cost. Tailor-made hardware could plausibly reduce these differences. Supplementary visualisation of the results in Tab. 1 are found in Appendix C.

5 Conclusion

We have introduced Continual 3D Convolutional Neural Networks (*Co*3D CNNs), a new computational model for spatio-temporal 3D CNNs, which performs computations frame-wise rather than clip-wise while being weight-compatible with regular 3D CNNs. In doing so, we are able dispose of the computational redundancies faced by 3D CNNs in continual online processing, giving up to a 15.1× reduction of floating point operations, a 9.2× real-life inference speed-up on CPU, 48% peak memory reduction, and an accuracy improvement of 5.6% on Kinetics-400 through an extension in the global average pooling kernel size.

While this constitutes a substantial leap in the processing efficiency of energy-constrained and real-time video recognition systems, there are still unanswered questions pertaining to the dynamics of *Co*3D CNNs. Specifically, the impact of extended receptive fields on the networks ability to change predictions in response to changing contents in the input video is untested. We leave these as important directions for future work.

Acknowledgement

This work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 871449 (OpenDR).

References

1. Arnab, A., Dehghani, M., Heigold, G., Sun, C., Lučić, M., Schmid, C.: Vivit: A video vision transformer. In: IEEE/CVF International Conference on Computer Vision (ICCV). pp. 6836–6846 (2021)
2. Cai, Z., He, X., Sun, J., Vasconcelos, N.: Deep learning with low precision by half-wave gaussian quantization. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 5406–5414 (2017)
3. Carreira, J., Zisserman, A.: Quo vadis, action recognition? a new model and the kinetics dataset. In: IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 4724–4733 (2017)
4. Carreira, J., Noland, E., Banki-Horvath, A., Hillier, C., Zisserman, A.: A short note about kinetics-600. preprint, arXiv:1808.01340 (2018)
5. Carreira, J., Pătrăucean, V., Mazare, L., Zisserman, A., Osindero, S.: Massively parallel video networks. In: Ferrari, V., Hebert, M., Sminchisescu, C., Weiss, Y. (eds.) Proceedings of the European Conference on Computer Vision (ECCV). pp. 680–697 (2018)
6. Chen, W., Wilson, J.T., Tyree, S., Weinberger, K.Q., Chen, Y.: Compressing neural networks with the hashing trick. In: International Conference on International Conference on Machine Learning (ICML). p. 2285–2294 (2015)
7. Donahue, J., Hendricks, L.A., Guadarrama, S., Rohrbach, M., Venugopalan, S., Darrell, T., Saenko, K.: Long-term recurrent convolutional networks for visual recognition and description. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 2625–2634 (2015)
8. Fan, H., Murrell, T., Wang, H., Alwala, K.V., Li, Y., Li, Y., Xiong, B., Ravi, N., Li, M., Yang, H., Malik, J., Girshick, R., Feiszli, M., Adcock, A., Lo, W.Y., Feichtenhofer, C.: PyTorchVideo: A deep learning library for video understanding. In: ACM International Conference on Multimedia (2021)
9. Farnebäck, G.: Two-frame motion estimation based on polynomial expansion. In: Image Analysis. pp. 363–370. Springer Berlin Heidelberg (2003)
10. Feichtenhofer, C.: X3D: Expanding architectures for efficient video recognition. IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (2020)
11. Feichtenhofer, C., Fan, H., Malik, J., He, K.: Slowfast networks for video recognition. In: IEEE/CVF International Conference on Computer Vision (ICCV) (October 2019)
12. Floropoulos, N., Tefas, A.: Complete vector quantization of feedforward neural networks. Neurocomputing **367**, 55–63 (2019)
13. Han, S., Mao, H., Dally, W.J.: Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In: International Conference on Learning Representations (ICLR) (2016)
14. He, Y., Zhang, X., Sun, J.: Channel pruning for accelerating very deep neural networks. In: 2017 IEEE International Conference on Computer Vision (ICCV). pp. 1398–1406 (2017)
15. Hedegaard, L., Iosifidis, A.: Continual inference: A library for efficient online inference with deep neural networks in pytorch. preprint, arXiv:2204.03418 (2022)
16. Hinton, G., Vinyals, O., Dean, J.: Distilling the knowledge in a neural network. In: NIPS Deep Learning and Representation Learning Workshop (2015)
17. Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H.: Mobilenets: Efficient convolutional neural networks for mobile vision applications. preprint, arXiv:1704.04861 **abs/1704.04861** (2017)

18. Hu, J., Shen, L., Sun, G.: Squeeze-and-excitation networks. In: IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 7132–7141 (2018)
19. Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., Bengio, Y.: Binarized neural networks. In: Lee, D., Sugiyama, M., Luxburg, U., Guyon, I., Garnett, R. (eds.) Advances in Neural Information Processing Systems. vol. 29. Curran Associates, Inc. (2016)
20. Ji, S., Xu, W., Yang, M., Yu, K.: 3d convolutional neural networks for human action recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI) **35**(1), 221–231 (2013)
21. Joe Yue-Hei Ng, Hausknecht, M., Vijayanarasimhan, S., Vinyals, O., Monga, R., Toderici, G.: Beyond short snippets: Deep networks for video classification. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 4694–4702 (2015)
22. Kalogeiton, V., Weinzaepfel, P., Ferrari, V., Schmid, C.: Action tubelet detector for spatio-temporal action localization. In: 2017 IEEE International Conference on Computer Vision (ICCV). pp. 4415–4423 (2017)
23. Karpathy, A., Toderici, G., Shetty, S., Leung, T., Sukthankar, R., Fei-Fei, L.: Large-scale video classification with convolutional neural networks. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 1725–1732 (2014)
24. Karpathy, A.: CS231n convolutional neural networks for visual recognition, `https://cs231n.github.io/convolutional-networks/`. Last visited on 2021/01/26
25. Kay, W., Carreira, J., Simonyan, K., Zhang, B., Hillier, C., Vijayanarasimhan, S., Viola, F., Green, T., Back, T., Natsev, P., Suleyman, M., Zisserman, A.: The kinetics human action video dataset. preprint, arXiv:1705.06950 (2017)
26. Köpüklü, O., Hörmann, S., Herzog, F., Cevikalp, H., Rigoll, G.: Dissected 3d cnns: Temporal skip connections for efficient online video processing. preprint, arXiv:2009.14639 (2020)
27. Köpüklü, O., Wei, X., Rigoll, G.: You only watch once: A unified cnn architecture for real-time spatiotemporal action localization. preprint, arXiv:1911.06644 (2019)
28. Köpüklü, O., Kose, N., Gunduz, A., Rigoll, G.: Resource efficient 3d convolutional neural networks. In: IEEE/CVF International Conference on Computer Vision Workshop (ICCVW). pp. 1910–1919 (2019)
29. Liu, G., Shih, K.J., Wang, T.C., Reda, F.A., Sapra, K., Yu, Z., Tao, A., Catanzaro, B.: Partial convolution based padding. preprint, arXiv:1811.11718 pp. 1–11 (2018)
30. Ma, N., Zhang, X., Zheng, H.T., Sun, J.: Shufflenet v2: Practical guidelines for efficient cnn architecture design. In: Proceedings of the European Conference on Computer Vision (ECCV) (September 2018)
31. Molchanov, P., Yang, X., Gupta, S., Kim, K., Tyree, S., Kautz, J.: Online detection and classification of dynamic hand gestures with recurrent 3d convolutional neural networks. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 4207–4215 (2016)
32. Neimark, D., Bar, O., Zohar, M., Asselmann, D.: Video transformer network. In: 2021 IEEE/CVF International Conference on Computer Vision Workshops (ICCVW). pp. 3156–3165 (2021)
33. Nguyen, A., Choi, S., Kim, W., Ahn, S., Kim, J., Lee, S.: Distribution padding in convolutional neural networks. In: International Conference on Image Processing (ICIP). pp. 4275–4279 (2019)
34. van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., Kavukcuoglu, K.: Wavenet: A generative model for raw audio. preprint, arXiv:1609.03499 (2016)

35. Papers with Code: Kinetics-400 leaderboard, https://paperswithcode.com/sota/action-classification-on-kinetics-400. Last visited on 2021/02/03.
36. Passalis, N., Tefas, A.: Learning deep representations with probabilistic knowledge transfer. In: Proceedings of the European Conference on Computer Vision (ECCV) (2018)
37. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A.C., Fei-Fei, L.: Imagenet large scale visual recognition challenge. International Journal of Computer Vision (ICCV) **115**(3), 211–252 (2015)
38. Sigurdsson, G.A., Varol, G., Wang, X., Farhadi, A., Laptev, I., Gupta, A.: Hollywood in homes: Crowdsourcing data collection for activity understanding. In: Proceedings of the European Conference on Computer Vision (ECCV) (2016)
39. Singh, G., Cuzzolin, F.: Recurrent convolutions for causal 3d cnns. In: 2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW). pp. 1456–1465 (2019)
40. Singh, G., Saha, S., Sapienza, M., Torr, P., Cuzzolin, F.: Online real-time multiple spatiotemporal action localisation and prediction. In: 2017 IEEE International Conference on Computer Vision (ICCV). pp. 3657–3666 (2017)
41. Sovrasov, V.: Ptflops, https://github.com/sovrasov/flops-counter.pytorch. MIT License. Last visited on 2021/03/02
42. Tan, M., Le, Q.: EfficientNet: Rethinking model scaling for convolutional neural networks. In: Proceedings of Machine Learning Research. vol. 97, pp. 6105–6114 (2019)
43. Tran, D., Wang, H., Torresani, L., Ray, J., LeCun, Y., Paluri, M.: A closer look at spatiotemporal convolutions for action recognition. In: IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 6450–6459 (2018)
44. Tran, D., Bourdev, L., Fergus, R., Torresani, L., Paluri, M.: Learning spatiotemporal features with 3d convolutional networks. In: IEEE International Conference on Computer Vision (ICCV). pp. 4489–4497 (2015)
45. Wang, M., Deng, W.: Deep visual domain adaptation: A survey. Neurocomputing **312**, 135–153 (2018)
46. Xu, M., Zhu, M., Liu, Y., Lin, F., Liu, X.: Deepcache: Principled cache for mobile deep vision. International Conference on Mobile Computing and Networking (2018)
47. Yim, J., Joo, D., Bae, J., Kim, J.: A gift from knowledge distillation: Fast optimization, network minimization and transfer learning. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 7130–7138 (2017)
48. Zhang, X., Zhou, X., Lin, M., Sun, J.: Shufflenet: An extremely efficient convolutional neural network for mobile devices. In: IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 6848–6856 (2018)
49. Zhu, L., Sevilla-Lara, L., Yang, Y., Feiszli, M., Wang, H.: Faster recurrent networks for efficient video classification. Proceedings of the AAAI Conference on Artificial Intelligence **34**, 13098–13105 (2020)

Appendix

A Worst-case memory for *Co*X3D-M

In this section, we provide a detailed overview of the memory consumption incurred by the internal state in a Continual X3D-M (*Co*X3D-M) model. For Continual 3D CNNs, there is no need to store input frames between time steps, though this is the case for regular 3D CNNs applied in an online processing scenario. Intermediary computations from prior frames are kept in the continual layers as state if a layer has a temporal receptive field larger than 1. A continual $k_T \times k_H \times k_W = 1 \times 3 \times 3$ convolution is equivalent to a regular convolution, while a $3 \times 1 \times 1$ is not. The same principle holds for pooling layers. As a design decision, the temporal component of the average pooling of Squeeze-and-Excitation (SE) blocks is discarded. Hence, SE blocks do not incur a memory overhead or delay. Keeping the temporal pooling of the SE block would have increased memory consumption by a modest 85.050 (+1.4%). We can compute the total state overhead using Eq. (2), Eq. (8), and Eq. (9) by adding up the state size of each applicable layer shown in Tab. 5. An overview of the resulting computations can be found in Tab. 4. The total memory overhead for the network state is 4,771,632 floating point operations. In addition to the state memory, the worst-case transient memory must be taken into account. The largest intermediary feature-map is produced after the first convolution in conv_1 and has a size of $24 \times 112 \times 112 = 301{,}056$ floats. The total worst-case memory consumption for *Co*X3D-M (excluding models weights) is thus **5,072,688** floats.

If we were to reduce the model clip size from 16 to 4, this would result in a memory reduction of 5,184 floats (only pool_5 is affected) for a total worst-case memory of 5,067,504 floats (−0.1%). Increasing the clip size to 64 would yield an increased state memory of 20,736 floats giving a total worst-case memory of 5,093,424 floats (+0.4%).

Stage	**Layer**		**Mem.** (floats)
conv_1	conv_T	$(5-1)\times 24\times 112\times 112 =$	1,204,224
res_2	residual_1	$(3-1-1)\times 24\times 112\times 112 =$	301,056
	residual_{2-3}	$[(3-1-1)\times 24\times 56\times 56]\times 2 =$	150,528
	conv_{1-3}	$[(3-1-1)\times 54\times 56\times 56]\times 3 =$	508,032
res_3	residual_1	$(3-1-1)\times 24\times 56\times 56 =$	75,264
	residual_{2-5}	$[(3-1-1)\times 48\times 28\times 28]\times 4 =$	150,528
	conv_{1-5}	$[(3-1)\times 108\times 28\times 28]\times 5 =$	846,720
res_4	residual_1	$(3-1-1)\times 48\times 28\times 28 =$	37,632
	residual_{2-11}	$[(3-1-1)\times 96\times 14\times 14]\times 10 =$	188,160
	conv_{1-11}	$[(3-1)\times 216\times 14\times 14]\times 11 =$	931,392
res_5	residual_1	$(3-1-1)\times 96\times 14\times 14 =$	18,816
	residual_{2-3}	$[(3-1-1)\times 192\times 7\times 7]\times 6 =$	56,448
	conv_{1-3}	$[(3-1)\times 432\times 7\times 7]\times 7 =$	296,352
pool_5	-	$(16-1)\times 432 =$	6,480
Total			**4,771,632**

Table 4: ***Co*X3D-M** state memory consumption by layer.

Stage	**Filters**	**Output size** $(T \times H \times W)$
input	-	$16 \times 224 \times 224$
conv_1	$\begin{matrix} 1 \times 3^2, 24 \\ 5^* \times 1^2, 24 \end{matrix}$	$16 \times 112 \times 112$
res_2	res $\begin{bmatrix} 1 \times 1^2, 54 \\ 3 \times 3^2, 54 \\ \text{SE} \\ 1 \times 1^2, 24 \end{bmatrix} \times 3$	$16 \times 56 \times 56$
res_3	res $\begin{bmatrix} 1 \times 1^2, 108 \\ 3 \times 3^2, 108 \\ \text{SE} \\ 1 \times 1^2, 48 \end{bmatrix} \times 5$	$16 \times 28 \times 28$
res_4	res $\begin{bmatrix} 1 \times 1^2, 216 \\ 3 \times 3^2, 216 \\ \text{SE} \\ 1 \times 1^2, 96 \end{bmatrix} \times 11$	$16 \times 14 \times 14$
res_5	res $\begin{bmatrix} 1 \times 1^2, 432 \\ 3 \times 3^2, 432 \\ \text{SE} \\ 1 \times 1^2, 192 \end{bmatrix} \times 7$	$16 \times 7 \times 7$
conv_5	$1 \times 1^2, 432$	$16 \times 7 \times 7$
pool_5	16×7^2	$1 \times 1 \times 1$
fc_1	$1 \times 1^2, 2048$	$1 \times 1 \times 1$
fc_2	1×1^2, #classes	$1 \times 1 \times 1$

Table 5: **X3D-M model architecture**. When converted to a continual CNN, the highlighted components carry an internal state which results in a memory overhead. *Temporal kernel size in conv_1 is set to 5 as found in the official X3D source code [10].

B Benchmarking details

This section should be read in conjunction with Sec. 4.3 of the main paper. To gauge the achievable on-hardware speeds of clip and frame predictions, a benchmark was performed on the following four system: A CPU core of a MacBook Pro (16-inch 2019 2.6 GHz Intel Core i7); Nvidia Jetson TX2; Nvidia Jetson Xavier; and a Nvidia RTX 2080 Ti GPU (on server with Intel XEON Gold processors). A batch size of 1 was used for testing on CPU, while the largest fitting multiple of $2^{\mathbb{N}}$ up to 64 was used for the other hardware platforms which have GPUs and lend themselves better to parallelisation. Thus, the speeds noted for GPU platforms in Tab. 1 of the main paper should not be interpreted as the number of processed clips/frames from a single (high-speed) video stream, but rather as the aggregated number of clips/frames from multiple streams using the available hardware. The exact batch size and input resolutions can be found in Tab. 6. In conducting the measurements, we assume the input data is readily available on the CPU and measure the time it takes for it to transfer from the CPU to GPU (if applicable), process, and transfer back to the CPU. A precision of 16 bits was used for the embedded platforms TX2 and Xavier, while a 32 bit precision was employed for CPU and RTX 2080 Ti. All networks were implemented and tested using PyTorch, and neither Nvidia TensorRT nor ONNX Runtime were used to speed up inference.

Model	**Input shape** $(T \times S^2)$	**Batch size**			
		CPU	**TX2**	**Xavier**	**RTX**
I3D-R50	8×224^2	1	16	16	32
R(2+1)D-18_8	8×112^2	1	16	16	32
R(2+1)D-18_{16}	16×112^2	1	8	16	32
Slow-8×8-R50	8×256^2	1	8	8	8
SlowFast-8×8-R50	8×256^2	1	8	32	32
SlowFast-4×16-R50	16×256^2	1	16	32	32
X3D-L	16×312^2	1	16	32	32
X3D-M	16×224^2	1	32	64	64
X3D-S	13×160^2	1	64	64	64
X3D-XS	4×160^2	1	64	64	64
*Co*I3D	1×224^2	1	8	8	8
*Co*Slow	1×224^2	1	8	8	8
*Co*X3D-L	1×312^2	1	8	16	32
*Co*X3D-M	1×224^2	1	32	64	64
*Co*X3D-S	1×160^2	1	32	64	64

Table 6: **Benchmark model configurations**. For each model, the input shape is noted as $T \times S^2$, where T and S are the temporal and spatial input shape.

Model	FLOPs	Throughput (evaluations/s)			
		CPU	TX2	Xavier	RTX
(*Co*)I3D	5.04×	3.39×	0.95×	1.62×	1.64×
(*Co*)Slow	7.95×	7.68×	1.19×	1.44×	1.65×
(*Co*)X3D-L	15.34×	9.20×	5.21×	5.77×	5.98×
(*Co*)X3D-M	15.06×	9.05×	4.79×	4.95×	6.86×
(*Co*)X3D-S	12.11×	5.91×	4.15×	4.98×	3.41×

Table 7: **Relative improvements** in frame-by-frame inference in Continual 3D CNN relative to regular 3D CNN counterparts. The improvements (× lower for FLOPs and × higher for throughput) correspond to the results in Tab. 1 of the main paper.

C A note on RCU FLOPs

In Tab. 1 of the main paper, we have approximated the FLOPs for RCU [39] as follows: We use a different measure of FLOPs (the one from the `ptflops` [41]) than the RCU authors and therefore employ a translation factor of 28.6/41.0, which is our measured FLOPs for I3D (28.6) divided by theirs (41.0), multiplied with their reported 54.0 for RCU. Considering that their method used 8 frames and can be applied per frame, we also divide by 8. Note that the this approximation lacks the repeat classification layer and may thus be considered on the low side. The resulting computation becomes $28.6/41.0 \cdot 54.0/8 = 4.71$.

D Supplemental visualisations of benchmark

As a supplement to the results presented in the main paper, this appendix supplies additional views of the benchmarking results in Tab. 1. Accordingly, graphical representations of the accuracy versus speed trade-offs from Tab. 1 are shown in Figures 6-9. As in Fig. 1 of the main paper, the noted accuracies on Kinetics-400 were achieved using 1-clip/frame testing on publicly available pretrained models, the *Co*X3D models utilised X3D weights without further fine-tuning, and the numbers noted in each point represent the size of the global average pooling layer. Likewise, Tab. 7 shows the improvements in continual inference relative to the regular models. In general, the FLOPs improvements are higher than on-hardware speed evaluations, with relatively lower improvements on hardware platforms with GPUs. We attribute these differences to a memory operations overhead, which does not enjoy the same computational improvement as multiply-accumulate operations do on massively parallel hardware.

From Figures 6-9 we likewise observe, that the I3D, R(2+1)D and SlowFast models perform relatively better on hardware compared to the X3D and *Co*X3D models, which utilise computation-saving approaches such as 1D-convolutions and grouped 3D-convolutions at the price of increasing memory access cost.

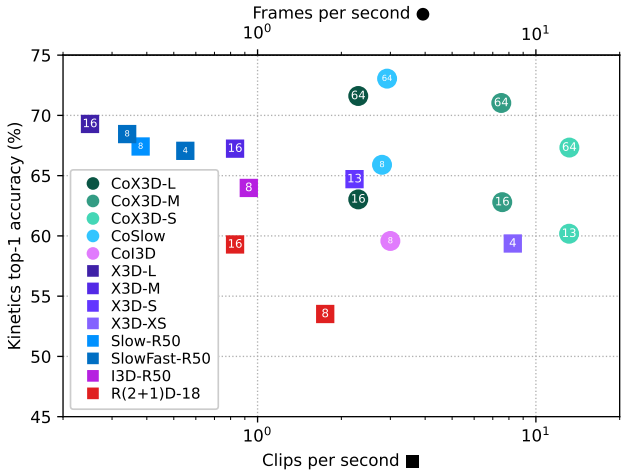


Fig. 6: **CPU** inference throughput versus top-1 accuracy on Kinetics-400.

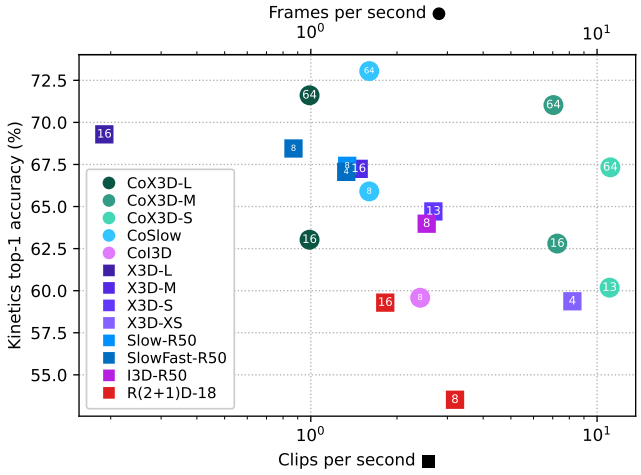


Fig. 7: **TX2** inference throughput versus top-1 accuracy on Kinetics-400.

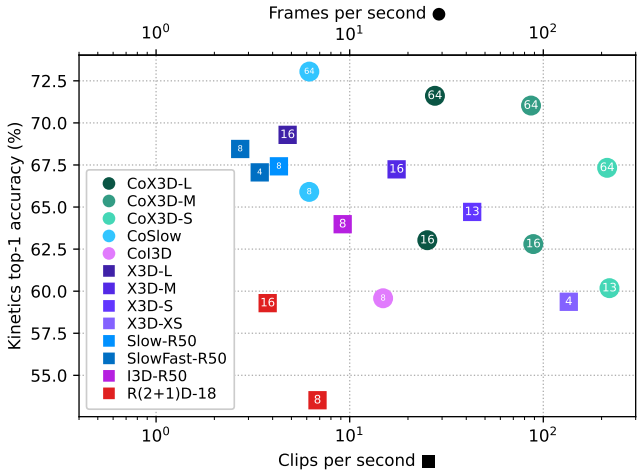


Fig. 8: **Xavier** inference throughput versus top-1 accuracy on Kinetics-400.

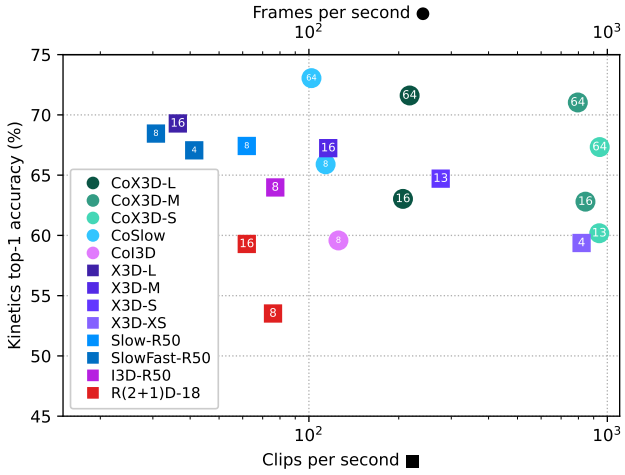


Fig. 9: **RTX2080Ti** inference throughput versus top-1 acc. on Kinetics-400.

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.

Continual Spatio-Temporal Graph Convolutional Networks

Lukas Hedegaard, Negar Heidari, and Alexandros Iosifidis
Department of Electrical and Computer Engineering, Aarhus University, Denmark
{lhm, negar.heidari, ai}@ece.au.dk

***Abstract*—Graph-based reasoning over skeleton data has emerged as a promising approach for human action recognition. However, the application of prior graph-based methods, which predominantly employ whole temporal sequences as their input, to the setting of online inference entails considerable computational redundancy. In this paper, we tackle this issue by reformulating the Spatio-Temporal Graph Convolutional Neural Network as a Continual Inference Network, which can perform step-by-step predictions in time without repeat frame processing. To evaluate our method, we create a continual version of ST-GCN, *Co*ST-GCN, alongside two derived methods with different self-attention mechanisms, *Co*AGCN and *Co*S-TR. We investigate weight transfer strategies and architectural modifications for inference acceleration, and perform experiments on the NTU RGB+D 60, NTU RGB+D 120, and Kinetics Skeleton 400 datasets. Retaining similar predictive accuracy, we observe up to 109× reduction in time complexity, on-hardware accelerations of 26×, and reductions in maximum allocated memory of 52% during online inference.**

***Index Terms*—Graph Convolutional Networks, Continual Inference, Efficient Deep Learning, Skeleton-based Action Recognition**

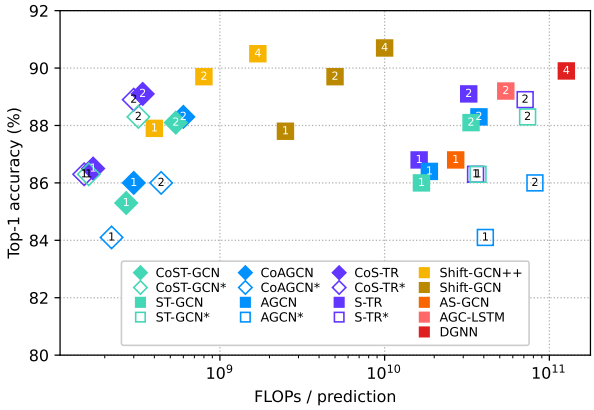


Fig. 1: **Accuracy/complexity trade-off** on NTU RGB+D 60 X-Sub for ◆ *Continual* and ■ prior methods during online inference. Numbers denote streams for each method. One stream contains the joint modality; two streams add the bone modality; and four streams add joint and bone motion. *Architecture modification with stride one and no padding.

I. Introduction

A human action can be described by a temporal sequence of human body poses, each of which is represented by a set of spatial joint coordinates forming a body skeleton. Accordingly, skeleton-based action recognition methods process a sequence of skeletons (instead of an image sequence) to recognize the performed action. Compared with predicting actions from videos, a sequence of skeleton data not only gives the spatial and temporal features of the body poses, but also provides robustness against different background variations and context noise [1]. The estimation of such skeletal data has become a staple in the human action recognition toolkit thanks to publicly available toolboxes such as OpenPose [2].

Early deep learning methods for skeleton-based action recognition either rearrange the body joint coordinates of each skeleton to make a pseudo-image which is used to train a CNN model [3, 4, 5], or concatenate the human body joints as a sequence of feature vectors and train a RNN model [6, 7, 8]. However, these methods cannot take advantage of the non-Euclidean structure of the skeletons. Recently, Graph Convolutional Networks (GCNs) have shown prowess in the modeling of skeleton data [9]. ST-GCN [10] was the first GCN-based method proposed for skeleton-based action recognition. It uses spatial graph convolutions to extract the per time-step features of each skeleton and employs temporal convolutions to capture time-varying dynamics throughout the skeleton sequence. Since its publication, several methods have sprung from ST-GCN, which enhance feature extraction or optimize the structure of the model.

2s-AGCN [11] proposed to learn the graph structure in each GCN layer adaptively based on input graph node similarity and also utilized an attention method which highlights both the existing spatial connections in the graph (bones) and new potential connections between them. MS-AAGCN [12] extended 2s-AGCN by proposing a multi-stream framework which uses four different data streams for training the model. Moreover it enhanced the adaptive graph convolution in 2s-AGCN with a spatio-temporal channel attention module to highlight the most important skeletons, nodes in each skeleton, and features of each node. Following the idea of utilizing different data streams, another multi-stream framework is proposed by [13] which constructs a spatio-temporal view invariant model (STVIM) to capture the spatial and temporal dynamics of joints and bones in skeletons using Geometric Algebra. MS-G3D [14] has proposed multi-scale graph convolutions for long-range feature extraction, and DGNN [15] modeled the spatial connections between the graph nodes with a directed graph and utilized both node features and edge features simultaneously. Similarly, HSR-TSL [16] is a

model with hierarchical spatial reasoning and temporal stack learning network which employs a hierarchical residual graph neural network to capture two-level spatial features, and a temporal stack learning network (TSLN) composed of multiple skip-clip LSTMs to capture temporal dynamics of skeleton sequences. Recently, STF-Net [17] has proposed to capture both robust movement patterns from the skeleton joints and parts topology structures and temporal dependency, using a multi-grain contextual focus module (MCF) and a temporal discrimination focus module (TDF) integrated into a GCN network.

Unfortunately, the high computational complexity of these GCN-based methods makes them infeasible in real-time applications and resource-constrained online inference settings. Multiple approaches have been explored to increase the efficiency of skeleton-based action recognition recently: GCN-NAS [18] and PST-GCN [19] are neural architecture search based methods which try to find an optimized ST-GCN architecture to increase the efficiency of the classification task; Tripool [20] is a novel graph pooling method which optimizes a triplet pooling loss to learn an optimized graph topology, by removing the redundant nodes, and learn hierarchical graph representation.

ShiftGCN [21] replaces graph and temporal convolutions with a zero-FLOPs shift graph operation and point-wise convolutions as an efficient alternative to the feature-propagation rule for GCNs [22]; ShiftGCN++ [23] boost the efficiency of ShiftGCN further via progressive architecture search, knowledge-distillation, explicit spatial positional encodings, and a Dynamic Shift Graph Convolution;

SGN [24] utilizes semantic information such as joint type and frame index as side information to design a compact semantics-guided neural network (SGN) for capturing both spatial and temporal correlations in joint and frame level; TA-GCN [25] tries to make inference more efficient by selecting a subset of key skeletons, which hold the most important features for action recognition, from a sequence to be processed by the spatio-temporal convolutions.

Yet, none of the above-described GCN-based methods are tailored to online inference, were the input is a continual stream of skeletons and step-by-step predictions are required. During online inference, these methods would need to rely on sliding window-based processing, i.e., storing the $T-1$ prior skeletons, appending the newest skeleton to get a sequence of length T, and then performing their prediction on the whole sequence.

In this paper, we reduce such redundant computations by reformulating the ST-GCN and its derived methods as a Continual Inference Network, which processes skeletons one by one and produces updated predictions for each time-step without the need to include past skeletons in every input as is the case for the prior GCN-based methods. This is achieved by using Continual Convolutions in place of regular ones for aggregating temporal information, leading to highly reduced number of floating point operations (see Figure 1). In particular, we propose the *Continual* Spatio-Temporal Graph Convolutional Network (*Co*ST-GCN), *Co*AGCN, and *Co*TR-S and evaluate them on the skeleton-based action recognition datasets NTU RGB+D 60 [26], NTU RGB+D 120 [27], and Kinetics Skeleton 400 [28] with striking results: Our continual models achieve up to 108× FLOPs reduction, 26× speedup, and 52% reduction in max allocated GPU memory compared to the corresponding non-continual models.

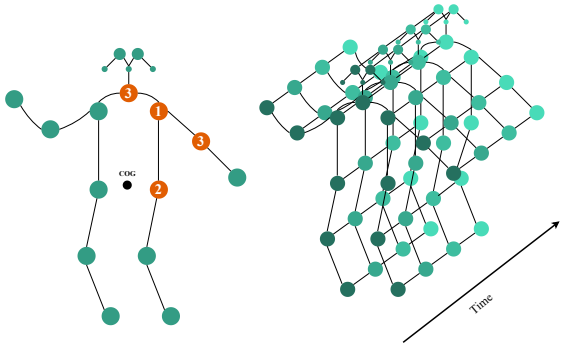


Fig. 2: **Graph illustration** for a spatially partitioned skeleton (left) and spatio-temporal graph (right).

The remainder of the paper is structured as follows: Section II provides an introduction to skeleton-based action recognition and of the related methods, from which we derive a continual counterpart, Section II-D describes Continual Inference Networks, and Section III-D presents our proposed *Continual* Spatio-temporal Graph Convolutional Networks. Experiments on weight transfer strategies, performance benchmarks, and comparisons with prior works are offered in Section IV, and a conclusion is given in Section V.

II. Related Works

A. Spatio-Temporal Graph Convolutional Network

GCN-based models for skeleton-based action recognition [10, 19, 25] operate on sequences of skeleton graphs. The spatio-temporal graph of skeletons $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ has the human body joint coordinates as nodes $\mathcal{V}$ and the spatial and temporal connections between them as edges $\mathcal{E}$. Figure 2 (right) illustrates such a spatio-temporal graph where the spatial graph edges encode the human bones and the temporal edges connect the same joints in subsequent time-steps. We model this graph as a tensor $\mathbf{X} \in \mathbb{R}^{C^{(0)} \times T \times V}$, where $C^{(0)}$ is the number of input-channels of each joint, T denotes the number of skeletons in a sequence, and V is the number of joints in each skeleton. A binary adjacency matrix $\mathbf{A} \in \mathbb{R}^{V \times V}$ encodes the skeleton-structure with ones in positions connecting two vertices in a skeleton and zeros elsewhere.

The ST-GCN [10] and AGCN [11] methods refine the spatial structure of each skeleton by employing a partitioning method which categorizes neighboring nodes of each body joint into three subsets: (1) the root node itself, (2) the root's neighboring nodes which are closer to the skeleton's center of gravity (COG) than the root itself, and (3) the remaining neighboring nodes of the root node. An example of this subset partitioning is shown in Figure 2 (left). Accordingly, the graph-structure of each skeleton is represented by three normalized

binary adjacency matrices $\left\{\mathbf{A}_p \in \mathbb{R}^{V\times V} \mid p = 1, 2, 3\right\}$, each of which is defined as

$$\hat{\mathbf{A}}_p = \mathbf{D}_p^{-\frac{1}{2}}\mathbf{A}_p\mathbf{D}_p^{-\frac{1}{2}}, \tag{1}$$

where $\mathbf{D}_p$ denotes the degree matrix of the neighboring subset p. Inspired by the GCN aggregation rule [22], the spatial graph convolution receives the hidden representation of the previous layer $\mathbf{H}^{(l-1)}$ as input, where $\mathbf{H}^{(0)} = \mathbf{X}$, and performs the following graph convolution (GC) transformation:

$$\text{GC}\left(\mathbf{H}^{(l-1)}\right) = \sigma\left(\text{Res}(\mathbf{H}^{(l-1)}) + \text{BN}\left(\sum_p (\hat{\mathbf{A}}_p \otimes \mathbf{M}_p^{(l)})\mathbf{H}^{(l-1)}\mathbf{W}_p^{(l)}\right)\right) \tag{2}$$

where $\sigma(\cdot)$ denotes a ReLU non-linearity, $\mathbf{W}_p^{(l)} \in \mathbb{R}^{C^{(l)}\times C^{(l-1)}}$ is the weight matrix which transforms the features of the neighboring subset p and $BN(\cdot)$ denotes batch normalization. Moreover, a learnable matrix $\mathbf{M}_p^{(l)} \in \mathbb{R}^{V\times V}$ is multiplied element-wise with its corresponding adjacency matrix $\hat{\mathbf{A}}_p$ as an attention mechanism that highlights the most important connections in each spatial graph. In order to retain the model's stability, the input to a layer is added to the transformed features through a residual connection $\text{Res}(\mathbf{H}^{(l-1)})$ which is defined as:

$$\text{Res}(\mathbf{H}^{(l-1)}) = \begin{cases} \mathbf{H}^{(l-1)}, & C^{(l)} = C^{(l-1)}, \\ \mathbf{H}^{(l-1)}\mathbf{W}_{res}^{(l)}, & \text{otherwise}, \end{cases} \tag{3}$$

where $\mathbf{W}_{res}^{(l)} \in \mathbb{R}^{C^{(l)}\times C^{(l-1)}}$ is a learnable mapping matrix which transforms the layer's input to have the same channel dimension as the layer's output.

The graph convolution block is followed by a temporal convolution, $\text{TC}(\cdot)$, which propagates the features of the graph nodes through different time steps to capture the motions taking place in an action. In the temporal graph, each node only has two fixed neighbors which are its corresponding nodes in the previous and next skeletons. The adjacency matrices and partitioning process are not involved in temporal feature propagation. In practice, the temporal convolution is a standard 2D convolution which receives the output of the graph convolution obtained in Eq. (2) and performs a transformation with a kernel of size $C^{(l)} \times K \times 1$ to keep the node feature dimension unchanged and aggregate the features through K consecutive time steps.

The whole spatio-temporal convolution block has the form

$$\mathbf{H}^{(l)} = \sigma\left(\text{Res}(\mathbf{H}^{(l-1)}) + \text{BN}(\text{TC}(\text{GC}(\mathbf{H}^{(l-1)})))\right). \tag{4}$$

The ST-GCN model is composed of multiple such spatio-temporal convolutional blocks. A global average pool and fully connected layer perform the final classification.

B. Adaptive Graph Convolutional Neural Networks

The fixed graph structure used in Eq. (2) is defined based on natural connections in the human body skeleton which restricts the model's capacity and flexibility in representing different action classes. However, for some action classes such as "touching head" it makes sense to model a connection between hand and head even though such a connection is not naturally present in the skeleton. AGCN [11] allows for such possibilities by adopting an adaptive graph convolution which utilizes a data-dependent graph structure as follows:

$$\text{AGC}\left(\mathbf{H}^{(l-1)}\right) = \sigma\left(\text{Res}(\mathbf{H}^{(l-1)}) + \text{BN}\left(\sum_p (\hat{\mathbf{A}}_p + \mathbf{M}_p^{(l)})\mathbf{H}^{(l-1)}\mathbf{W}_p^{(l)}\right)\right), \tag{5}$$

where $\mathbf{M}_p^{(l)}$ is defined as:

$$\mathbf{M}_p^{(l)} = \mathbf{B}_p^{(l)} + \mathbf{C}_p^{(l)} \tag{6}$$

The attention matrix in this definition is composed of two learnable matrices which are optimized along with other model parameters in an end-to-end manner. $\mathbf{B}_p^{(l)} \in \mathbb{R}^{N\times N}$ is a squared matrix that can be unique for each layer and each sample, and $\mathbf{C}_p^{(l)} \in \mathbb{R}^{N\times N}$ is a similarity matrix whose elements determine the strength of the pair-wise connections between nodes. This matrix is computed by first transforming the feature matrix $\mathbf{H}^{(l-1)} \in \mathbb{R}^{C^{(l-1)}\times T\times V}$ with two embedding matrices $\mathbf{W}_{p\theta}^{(l)}$, $\mathbf{W}_{p\phi}^{(l)}$ of size $C^{de} \times C^{(l-1)}$. The obtained feature maps are then reshaped to $C^{de}T \times V$ and multiplied to obtain the $\mathbf{C}_p^{(l)} \in \mathbb{R}^{N\times N}$ matrix as follows:

$$\mathbf{C}_p^{(l)} = \text{softmax}({\mathbf{H}^{(l-1)}}^\top {\mathbf{W}_{p\theta}^{(l)}}^\top \mathbf{W}_{p\phi}^{(l)}\mathbf{H}^{(l-1)}), \tag{7}$$

where softmax normalizes the matrix values. The additive attention mechanism in Eq. (5), thus, lets the adaptive graph convolution in Eq. (7) model the skeleton structure as a fully connected graph.

C. Skeleton-based Spatial Transformer Networks

S-TR [29] is an attention-based method which models dependencies between body joints at each time step using the self-attention operation found in Transformers [30]. In this method, a Spatial Self-Attention (SSA) module is designed to adaptively learn data-dependent pairwise body joint correlations using multi-head self-attention.

The SSA module at each layer l applies trainable query, key, and value transformations $\mathbf{W}_q^{(l)} \in \mathbb{R}^{C^{(l-1)}\times dq}$, $\mathbf{W}_k^{(l)} \in \mathbb{R}^{C^{(l-1)}\times dk}$, $\mathbf{W}_v^{(l)} \in \mathbb{R}^{C^{(l-1)}\times dv}$ on the feature vector $\mathbf{h}_i^t \in \mathbb{R}^{C^{(l-1)}}$ of node i at time step t to obtain the query, key, and value vectors $\mathbf{q}_i^t \in \mathbb{R}^{dq}$, $\mathbf{k}_i^t \in \mathbb{R}^{dk}$, $\mathbf{v}_i^t \in \mathbb{R}^{dv}$. The correlation weight for each pair of i, j nodes at time t is obtained using a query-key dot product

$$\alpha_{ij}^t = {\mathbf{q}_i^t}^\top \mathbf{k}_j^t. \tag{8}$$

The updated feature vector of node i at time t has size $C^{(l)}$ and is obtained using a weighted feature aggregation of value vectors:

$$\bar{\mathbf{h}}_i^t = \sum_j \text{softmax}_j\left(\frac{\alpha_{ij}^t}{\sqrt{dk}}\right)\mathbf{v}_j^t. \tag{9}$$

For each attention head, the feature transformation is performed with a different set of learnable parameters while the transformation matrices are shared across all the nodes. The output features of the SSA module are finally computed by applying a learnable linear transformation on the concatenated features from S attention heads:

$$\bar{\mathbf{h}}_i^t = (\Big\|_{s=1}^{S} \bar{\mathbf{h}}_{is}^t)\mathbf{W}_o. \tag{10}$$

SSA has similarities to a graph convolution operation on a fully connected graph for which the node connection weights are learned dynamically. The first three layers of the S-TR model extract features with GC and TC blocks as defined in Eq. (4) while in the remaining layers of the model SSA substitutes GC.

D. Continual Inference Networks

First introduced in [31] and subsequently formalized in [32], Continual Inference Networks are Deep Neural Networks that can operate efficiently on both fixed-size (spatio-)temporal batches of data, where the whole temporal sequence is known up front, as well as on continual data, where new input steps are collected continually and inference needs to be performed efficiently in an online manner for each received frame.

Definition (Continual Inference Network). *A Continual Inference Network is a Deep Neural Network, which*

- *is capable of continual step inference without computational redundancy,*
- *is capable of batch inference corresponding to a non-continual Neural Network,*
- *produces identical outputs for batch inference and step inference given identical receptive fields,*
- *uses one set of trainable parameters for both batch and step inference.*

Recurrent Neural Networks (RNNs) are a common family of Deep Neural Networks, which possess the above-described properties. 3D Convolutional Neural Networks (3D CNNs), Transformers, and Spatio-Temporal Graph Convolutional Networks are not Continual Inference Networks since they cannot make predictions time-step by time-step without considerable computational redundancy; they need to cache a sliding window of prior input frames and assemble them into a fixed-size sequence that is subsequently passed through the network to make a new predictions during online inference.

Recently, *Continual* 3D CNNs were made possible through the proposal of *Continual* 3D Convolutions [31]. Likewise, shallow *Continual* Transformers based on *Continual* Dot-product Attentions were introduced in [32]. We continue this line of work by extending Spatio-Temporal Graph Convolutional Networks (ST-GCNs) with a *Continual* formulation as well. To do so, let us first present and expand on the theory on Continual Convolutions.

III. Continual Spatio-Temporal Graph Convolutional Networks

In the section we present and expand the theory on Continual Convolutions with notes on temporal stride. Then, we describe how the Continual Spatio-Temporal Graph Convolutional Networks are constructed.

A. Continual Convolution

The Continual Convolution operation produces the exact same output as the regular convolution does, but performs the computation in a streaming fashion while caching intermediary results.

Consider a single channel 2D convolution over an input $\mathbf{X} \in \mathbb{R}^{T \times V}$ with temporal dimension T and a dimension of V vertices. Given a convolutional kernel with weights $\mathbf{W} \in \mathbb{R}^{K \times V}$, where K is the temporal kernel size, and a bias w_0, a regular convolution would compute the output $\mathbf{y}^{(t)}$ for time-step $t \in K..T$ as

$$\mathbf{y}^{(t)} = w_0 + \sum_{k=1}^{K} \sum_{v=1}^{V} \mathbf{W}_{k,v} \cdot \mathbf{X}_v^{(t-k-1)}. \tag{11}$$

Considering this computation in the context of online processing, where $T \to \infty$ and one input slice $\mathbf{X}^{(t)}$ is revealed in each time step, we find that $K-1$ previous slices, i.e. $(K-1) \cdot V$ values, need to be stored between time-steps.

An alternative computational sequence is used in Continual Convolutions. Here, the input slice $\mathbf{X}^{(t)}$ is convolved with the kernel $\mathbf{W}$ in the same time-step it is received. This is specified in Eq. (12a). The intermediate results are then cached in memory $\mathbf{m}$ ($K-1$ values stored between time-steps) and aggregated according to Eq. (12b).

$$\mathbf{m}^{(t)} = \left[\sum_{v=1}^{V} \mathbf{W}_{k,v} \cdot \mathbf{X}_v^{(t)} : \ k \in 1..K \right] \tag{12a}$$

$$\mathbf{y}^{(t)} = w_0 + \sum_{k=1}^{K} \mathbf{m}_k^{(t-k-1)} \tag{12b}$$

A graphical representation of this is shown in Fig. 3.

B. Delayed Residual

The temporal convolutions of regular Spatio-Temporal Graph Convolution blocks usually employ zero-padding to ensure equal temporal shape for input and output feature maps. This zero-padding is discarded for Continual Convolutions to avoid continual redundancies [31]. To retain weight compatibility between the regular and continual networks, a delay to the residual connection is necessary. This delay amounts to

$$k_T + (k_T - 1)(d_T - 1) - p_T - 1 \tag{13}$$

steps, where k_T, d_T, and p_T are respectively the temporal kernel size, dilation, and zero-padding of the corresponding regular convolution.

C. Temporal Stride

In Section III-A, it is assumed that one output is produced for each input received. However, many spatio-temporal networks including ST-GCN [10], AGCN [11], and S-TR [29], use temporal stride > 1 in their temporal convolutions. For

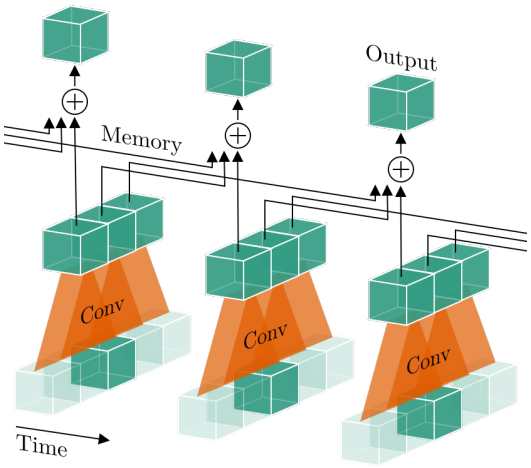


Fig. 3: **Continual Convolutions** are performed in two stages: First, the input is zero-padded and convolved with the convolutional kernel ($K = 3$ in illustration) to produce intermediary results. Subsequently, these are cached and summed up to produce the final output.

offline computation, this has the beneficial effect of reducing the computational and memory complexity, but in the online computational setting, it also reduces the prediction rate. This is illustrated in Fig. 4. For a neural network with L layers, each with a temporal stride s, the effective network stride is given by

$$s_{NN} = \prod_{l=1}^{L} s_l \tag{14}$$

and the corresponding network prediction rate is

$$r_{NN} = 1/s_{NN}. \tag{15}$$

Since a ST-GCN network has two layers with stride two, the corresponding Continual ST-GCN (*Co*ST-GCN) has a prediction rate one fourth the input rate.

D. Continual ST-GCN construction

Many well-performing methods for skeleton-based action recognition, including the ST-GCN [10], AGCN [11], and S-TR [29], share a common block structure, which can be described by Eq. (4). Here, the main difference between methods lies in how the graph information is processed, i.e. in their definition of GC$(\cdot)$.

The regular skeleton-based methods successively extract complete spatio-temporal skeleton features from the whole sequence with each block before classifying an action. Considering one block in isolation, the spatio-temporal feature extraction is given by a spatial (graph) convolution followed by a regular temporal convolution. Here, graph convolutions operate locally within a time-step[1], whereas the temporal convolution does not. Since the next block l takes as input $\mathbf{H}^{(l-1)}$, the output of the prior block and thereby its temporal

[1]AGCN is an exception to this, since the additive attention considers a node's features over all time-steps.

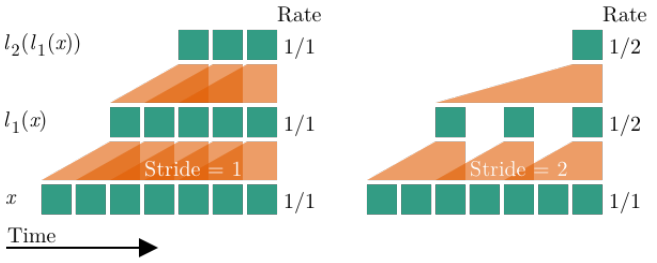


Fig. 4: **Temporal stride** in a Continual Convolution layer l_1 with temporal stride larger than one (right) reduces the prediction rate compared to a layer with stride one (left). The rate reduction is inherited by subsequent layers.

convolution, the output of the next spatial (graph) convolution becomes a function of multiple prior time-steps. With regular temporal convolutions, features produced by multiple blocks cannot be trivially disentangled and cached in time. Accordingly online operation with per-skeleton predictions can be attained by caching $T-1$ prior skeletons, concatenating these with the newest skeleton, and performing regular spatio-temporal inference. However, this comes with significant computational redundancy, where the complexity of online frame-wise inference is the same as for clip-based inference.

To alleviate this issue, we propose to employ Continual Convolutions in the temporal modeling of Spatio-temporal Graph Convolutional Networks. By restricting the GC$(\cdot)$ function to only operate locally within a time-step, we can define a *Continual* Spatio-Temporal block by replacing the original temporal 2D convolution with a continual one. To retain weight-compatibility with regular (non-continual) networks we moreover need to delay the residual to keep temporal alignment. Given $\mathbf{H}_{l-1}^{(t)}$, i.e. the features of layer $l-1$ in a time-step t, the feature in layer l at time t is given by

$$\mathbf{H}_{l}^{(t)} = \sigma\left(\text{Delay}(\text{Res}(\mathbf{H}_{l-1}^{(t)})) + \text{BN}(\mathit{Co}\text{TC}(\text{GC}(\mathbf{H}_{l-1}^{(t)})))\right). \tag{16}$$

Here, Delay(Res($\mathbf{H}_{l-1}^{(t)}$)) outputs the delayed residual in a first-in-first-out manner corresponding to the delay of the *Continual* Temporal Convolutional as computed by Eq. (13). A graphical illustration of such a block is seen in Fig. 5. It should be noted that the restriction of temporal locality does influence the computations of some skeleton-based action recognition methods. For example, the AGCN originally computes one vertex attention weighting based on the whole spatio-temporal feature-map, whereas a *Continual* AGCN (*Co*AGCN) computes separate vertex attentions for each time-step.

The resulting *Continual* Spatio-temporal Graph Convolutional Network is defined by stacking multiple such blocks[2] followed by *Continual* Global Average Pooling [31] and a fully connected layer. The Continual Inference Networks retain the same computational complexity as regular networks during clip-based inference, but can perform online frame-by-frame predictions much more efficiently, as detailed in Section III-E. We should note that all methods, which share the the same

[2]Following the original ST-GCN, AGCN, and S-TR architectures, ten blocks were used for the networks in this paper.

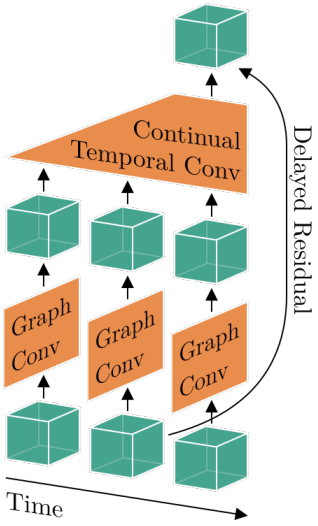


Fig. 5: **Continual Spatio-temporal Graph Convolution Blocks** consist of an in-time Graph Convolution followed by an across-time Continual Convolution (here a kernel size of three is depicted). The residual connection is delayed to ensure temporal alignment with the continual temporal convolution that is weight-compatible with non-continual networks.

structure as ST-GCN, i.e. a decoupled temporal and spatial convolution to perform feature transformation and aggregation over the time domain can be transformed to continual version using the approach outlined above.

E. Computational Complexity

Denote the time complexity of passing a single skeleton frame through the convolutional blocks with stride 1 by $\mathcal{O}(B)$ and time complexity of utilizing the prediction head by $\mathcal{O}(H)$. Given an effective clip-size T, the complexity of producing a prediction with a regular CNN is approximately $\mathcal{O}(\text{CNN}) \approx T \cdot \mathcal{O}(B) + \mathcal{O}(H)$. For a Continual CNN, the corresponding complexity is $\mathcal{O}(Co\text{CNN}) \approx \mathcal{O}(B) + \mathcal{O}(H)$. Computational savings thus scale linearly with the effective clip-size T and are more prominent the larger $\mathcal{O}(B)$ is compared to $\mathcal{O}(H)$.

F. Limitations

While the computational complexity can be greatly reduced for spatio-temporal GCNs during online processing, i.e., where a prediction is made each time a new skeleton is estimated in a live system, no acceleration occurs during offline processing compared to the original model. When the whole skeleton sequence is available beforehand the inference results and computational complexity are identical to prior works. The benefits of the *Continual* ST-GCN augmentation are thus limited to stream processing for networks which employ temporal convolutions. Accordingly, some networks such as AGCN, whose attention was originally based on the whole spatio-temporal sequence, may need modification to avoid peeking into the future.

IV. Experiments

A. Datasets

a) NTU RGB+D 60 [26]*:* A large indoor-captured dataset which is widely used for evaluating skeleton-based action recognition methods. This dataset contains 56,880 action clips and their corresponding 3D skeleton sequences captured by three Microsoft Kinect-v2 cameras from three different views. The clips are performed by 40 different subjects and constitute 60 action classes. The NTU RGB+D 60 dataset comes with two benchmarks, Cross-View (X-View) and Cross-Subject (X-Sub). The X-View benchmark provides 37,920 skeleton sequences coming from the camera views #2 and #3 as training data, and 18,960 skeleton sequences coming from the first camera view as test set. The X-Sub benchmark provides 40,320 skeleton sequences from 20 subjects as training data and 16,560 skeleton sequences from the other 20 subjects as test data. In this dataset, each skeleton has 25 body joints with three different channels each, and each action clip comes with a sequence of 300 skeletons.

b) NTU RGB+D 120 [27]*:* An extension of the NTU RGB+D 60 dataset containing an additional 57,600 skeleton sequences from extra 60 classes. NTU RGB+D 120 is currently the largest dataset providing 3D body joint coordinates for skeletons and in total, it contains 114,480 skeleton sequences from 120 action classes. The action clips in this dataset are performed by 106 subjects and 32 different camera setups are used for capturing the videos. This dataset comes with two benchmarks: Cross-Subject (X-Sub) and Cross-Setup (X-Set). The X-Sub benchmark provides the skeleton sequences of 53 subjects as training data and the remaining skeleton sequences from the other 53 subjects as test data. In the X-Set benchmark, the skeleton sequences with even camera setup IDs are provided as training data and test data contains the remaining skeleton sequences with odd camera setup IDs.

c) Kinetics Skeleton 400 [28]*:* A widely used dataset for action recognition containing 300,000 video action clips of 400 different classes which are collected from YouTube. Skeletons were extracted from each frame of these video clips using the OpenPose toolbox [2]. Each skeleton is represented by 18 body joints and each body joint contains spatial 2D coordinates and the estimation confidence score as its three features. We use the dataset version provided by [10], which contains 240,000 skeleton sequences as training data and 20,000 skeleton sequences as test data, in our experiments.

B. Experimental Settings

All models were implemented within the PyTorch framework [33] using the Ride library [34]. Models were trained using a SGD optimizer with learning rate 0.1 at batch size 64, momentum of 0.9, and a one-cycle learning rate policy [35] using a cosine annealing strategy. For models which could not fit a batch size of 64 on a Nvidia RTX 2080 Ti, the learning rate was adjusted following the linear scaling rule [36]. Our source code is available at www.github.com/lukashedegaard/continual-skeletons.

TABLE I: **Conversion Strategies** from regular (Reg) to Continual (Co) ST-GCN. Noted is the top-1 X-View validation accuracy on NTU RGB+D 60 and the FLOPs per prediction. The superscript p and subscript s indicate network padding and stride respectively. The arrows $\rightarrow$ and $\xrightarrow{FT}$ denote direct conversion and conversion with subsequent fine-tuning. Parentheses show the change relative to the baseline with colours indicating improvement / deterioration.

Conversion Strategy	Acc. (%)	FLOPs (G)
$\text{Reg}_{s=4}^{p=\text{eq}}$ (baseline)	93.4	16.73
$\text{Reg}_{s=4}^{p=\text{eq}} \xrightarrow{FT} \text{Reg}_{s=1}^{p=0}$	93.8 (+0.4)	36.90 ($\uparrow 2.2\times$)
$\text{Reg}_{s=4}^{p=\text{eq}} \rightarrow \text{Co}_{s=4}^{p=0}$	93.1 (−0.3)	0.27 ($\downarrow 63.2\times$)
$\text{Reg}_{s=4}^{p=\text{eq}} \rightarrow \text{Co}_{s=1}^{p=0}$	24.0 (−69.4)	0.16 ($\downarrow 107.7\times$)
$\text{Reg}_{s=4}^{p=\text{eq}} \rightarrow \text{Co}_{s=1}^{p=0} \xrightarrow{FT} \text{Co}*$	93.2 (−0.2)	0.16 ($\downarrow 107.7\times$)
$\text{Reg}_{s=4}^{p=\text{eq}} \xrightarrow{FT} \text{Reg}_{s=1}^{p=0} \rightarrow \text{Co}*$	93.8 (+0.4)	0.16 ($\downarrow 107.7\times$)

C. Conversion and Fine-tuning Strategies

Though regular and Continual CNNs are weight-compatible, the direct transfer of weights is imperfect if the regular CNN was trained with zero-padding [31]. As in most CNNs, it is common practice to utilize padding in skeleton-based spatio-temporal networks to retain the temporal feature size in consecutive layers (though temporal shrinkage is not a concern given the long input clips).

Another common design choice, which has a significant impact in on the performance of Continual Inference Networks, is the utilization of temporal stride larger than one. For regular networks, this has the benefit of reducing the computational complexity per clip prediction. In Continual Inference Networks, however, it reduces the prediction rate, and actually increases the complexity per prediction (see Section III-C). In the continual case, it would thus be computationally beneficial to reduce the stride of all layers to one. However, this results in a stride-inflicted *model-shift*.

Thus far, the *model-shift* inflicted by padding removal and stride reduction, as well as how to best perform the conversion from a regular CNN to a Continual CNN in such cases has not been studied. In this set of experiments, we explore strategies on how to best convert and fine-tune regular networks to achieve good frame-by-frame performance. We use a standard ST-GCN [10] trained on joints only as our starting-point, and explore the accuracy achieved by:

1) Converting to from regular network with equal padding and stride four ($\text{Reg}_{s=4}^{p=\text{eq}}$) to a Continual Inference Network, where zero-padding is omitted ($\text{Co}_{s=4}^{p=0}$).
2) Reducing the network stride to one without fine-tuning ($\text{Co}_{s=1}^{p=0}$).
3) Fine-tuning the $\text{Co}_{s=1}^{p=0}$ network (= Co$*$).
4) Fine-tuning a conversion-optimal regular network which has no zero-padding and a stride of one ($\text{Reg}_{s=1}^{p=0}$).
5) Converting from $\text{Reg}_{s=1}^{p=0}$ to Continual (= Co$*$).

As seen in Table I, the direct transfer of weights was found to have a modest negative impact on the accuracy (by -0.3%) due the removal of zero-padding. This is considerably less than was found in [31]. Our conjecture is that the smaller amount of zeros relative to clip size used in skeleton-based recognition (8 zeros per 300 frames or 2.67%) compared to video-based recognition (e.g., 2 zeros per 16 frames or or 12.5%) makes the removal of zero-padding less detrimental since zeros contribute relatively less to the downstream features. Lowering the stride to one and removing zero-padding reduced accuracy by a substantial amount but allowed the Continual Inference Network to operate at much lower FLOPs. This accuracy drop is alleviated equally effectively by either (a) initializing the $\text{Co}_{s=1}^{p=0}$ with standard weights and fine-tuning in the continual regime or (b) first fine-tuning the conversion-optimal regular network ($\text{Reg}_{s=1}^{p=0}$) and subsequently converting to a Continual Inference Network, though the latter had lower training times in practice. We fine-tuned the networks using the settings described in Section IV-B. As visualised in Fig. 6, we found 20 epochs of fine-tuning using the settings described in Section IV-B recover accuracy on NTU RGB+D 60 with additional training yielding only marginal differences. Following this approach the (padding zero, stride one) optimized Continual ST-GCN (*Co*ST-GCN*) achieves a similar prediction accuracy while reducing the computational complexity by a factor $107.7\times$ relative to original ST-GCN!

D. Conversion of Attention Architectures

As we explored in Section IV-C, the ST-GCN network architecture can easily be modified and fine-tuned to achieve high accuracy for frame-by-frame predictions with exceptionally low computational complexity. A natural follow-up question is whether this conversion is equally successful for more complicated spatio-temporal architectures that employ attention mechanisms. To investigate this, we conduct a similar transfer for two recent ST-GCN variants, the Adaptive GCN (AGCN) [11] and the Spatial Transformer Network (S-TR) [29]. While S-TR is easily converted to a Continual Inference Network (*Co*S-TR) by replacing convolutions, residuals and pooling operators with Continual ones, the AGCN requires additional care. In the original version of AGCN, the vertex attention matrix $\mathbf{C}_p$ (see Eq. (7)) is computed from the global representations in the layer over all time-steps. Since this operation would be acausal in the context of a Continual Inference Network, we restrict it to utilize only the frame-specific subset of features. As a fine-tuning strategy, we first make the conversion from regular network to a conversion-optimal network, and subsequently convert and evaluate the continual version.

Our results are presented in Table II. Here we see that all three architectures can be successfully converted to continual versions. The fine-tuned conversion-optimal models (marked by *) generally exhibit a higher computational complexity than their source models due to their stride decrease. While the ST-GCN* attained increased performance by lowering stride, AGCN* and S-TR* suffer slight accuracy deterioration. This may be due to smaller receptive fields of their attention mechanisms, which likely benefit from observing a larger context. Unlike the transfer from the original models with padding and stride four to continual models, the continual models with weights from ST-GCN*, AGCN*, and S-TR*, i.e.

TABLE II: **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 original model.

Model	Frames per pred	Accuracy (%)		Params (M)	Max mem. (MB)	FLOPs per 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 (↑ 2.2×)	1.1 (↓ 2.1×)	90.4 (↓ 2.0×)
*Co*ST-GCN	4	85.3 (−0.7)	93.1 (−0.3)	3.14	36.0 (79%)	0.27 (↓ 63.2×)	32.3 (↑ 14.0×)	2375.2 (↑ 13.1×)
*Co*ST-GCN*	1	86.3 (+0.3)	93.8 (+0.4)	3.14	36.1 (80%)	0.16 (↓ 107.7×)	46.1 (↑ 20.0×)	4202.2 (↑ 23.2×)
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 (↑ 2.2×)	1.0 (↓ 2.1×)	71.2 (↓ 2.0×)
*Co*AGCN	4	86.0 (−0.4)	93.9 (−0.4)	3.47	37.3 (77%)	0.30 (↓ 63.2×)	25.0 (↑ 11.9×)	2248.4 (↑ 15.4×)
*Co*AGCN*	1	84.1 (−2.3)	92.6 (−1.7)	3.47	37.4 (77%)	0.17 (↓ 108.8×)	30.4 (↑ 14.5×)	3817.2 (↑ 26.1×)
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 (↑ 2.2×)	0.8 (↓ 2.1×)	85.1 (↓ 1.8×)
*Co*S-TR	4	86.5 (−0.3)	93.3 (−0.5)	3.09	35.9 (48%)	0.22 (↓ 63.2×)	30.3 (↑ 17.8×)	2189.5 (↑ 14.0×)
*Co*S-TR*	1	86.3 (−0.3)	92.4 (−1.4)	3.09	36.1 (49%)	0.15 (↓ 107.6×)	43.8 (↑ 25.8×)	3775.3 (↑ 24.2×)

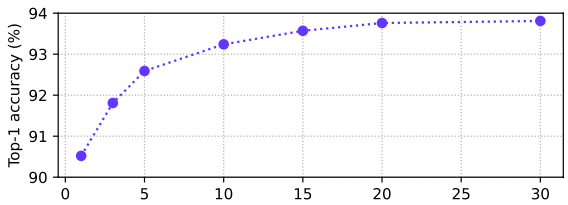


Fig. 6: **Fine-tuning epochs** and associated top-1 accuracy on NTU RGB+D 60 X-View for a transfer from a pre-trained ST-GCN with zero-padding and accumulated stride of four to an equivalent (*Co*)ST-GCN* with no zero-padding and stride one.

*Co*ST-GCN*, *Co*AGCN*, and *Co*S-TR* attain the exact same accuracy as their source models on both the X-Sub and X-View benchmarks, with two orders of magnitude less FLOPs per prediction during online inference.

E. Speed and Memory

Diving deeper into the differences between regular and continual networks, we conduct throughput benchmarks on a MacBook Pro 16" with a 2.6 GHz 6-Core Intel Core i7 CPU and a NVIDIA RTX 2080 Ti GPU. Here, we measure the prediction-time as the time it takes to transfer an input of batch size one from CPU to GPU (if applicable), perform inference, and transfer the results back to CPU again. On CPU, a batch size of one is used, while for GPU, the largest fitting power of two is employed (i.e. {128, 64, 256, 256} for the {Reg, Reg*, *Co*, and *Co**} models). We measure the maximum allocated memory during inference on GPU for batch size one.

As seen in Table II, the change in speed relative to the original models follow a similar trend to those seen for FLOPs. The non-continual stride one variants (denoted by *) exhibit roughly half the speed of the original models, while the continual models enjoy more than a magnitude speed up on both CPU and GPU. As expected, the continual stride one models (*Co**) attain the largest inference throughput. These relative speed-ups are lower than the relative FLOPs reductions due to the read/writes of internal intermediary features in the Continual Convolutions since these are not accounted for by the FLOPs metric while still adding to the runtime. This gap could be reduced on hardware with in- or near-memory computing.

Considering the maximum allocated memory at inference, we find that the continual models reduce memory by 20-52%. While the Continual Convolution and -Pooling layers do add some internal state that adds to the memory consumption, the intermediary features that are passed between network layers are much smaller, i.e. one frame instead of 75 to 300 frames.

F. Comparison with Prior Works

Most current state-of-the-art methods for skeleton-based action recognition are not able to efficiently perform frame-by-frame predictions in the online setting, since they are constrained to operate on whole skeleton-sequences. Some RNN-based methods, e.g. Deep-LSTM [26] and VA-LSTM [7], can be used for redundancy-free frame-wise predictions, but their reported accuracy has been sub-par relative to newer methods that sprung from ST-GCN. The recently proposed AGC-LSTM [38] does report results on-par with CNN-based methods, and might also be able to provide redundancy-free frame-wise results, but we cannot validate this due to the lack of publicly available source code and details in the published paper.

While ShiftGCN and ShiftGCN++ offer impressively low FLOPs, it should be noted that the shift operation is not accounted for by the FLOPs metric. The FLOP count of shift-based methods are therefore low compared to other methods and may not reflect on-hardware performance. Due to the temporal shifts back and forth in time, ShiftGCN and ShiftGCN++ cannot be easily transformed into Continual Inference Networks in their current form, though a *Continual* Shift operation could be devised, where temporal shifts only occur backwards in time. Nevertheless, ShiftGCN++ offers a remarkable accuracy/FLOPs trade-off, which was attained us-

TABLE III: **NTU RGB+D 60** comparison with recent methods, grouped by clip- and frame-based inference. Noted are the number of streams (S.), top-1 validation accuracy, and FLOPs per prediction. †Results for our implementation. Highlights indicate **best**, ***next-best*** and Pareto optimal results.

	Model	S.	Accuracy (%) X-Sub	Accuracy (%) X-View	FLOPs (G)
Clip	SGN [24]	1	89.4	94.5	-
	STVIM [13]	4	85.6	92.0	-
	HSR-TSL [16]	2	87.7	94.4	-
	MS-G3D [14]	1	89.4	95.0	-
		2	**91.5**	96.2	-
	ST-TR [29]	1	89.2	95.8	-
		2	90.3	96.3	-
	MS-AAGCN [12]	4	90.0	96.2	-
	DGNN [15]	4	89.9	96.1	126.80
	AS-GCN [37]	1	86.8	94.2	27.00
	AGC-LSTM [38]	2	89.2	95.0	54.40
	Tripool [20]	1	88.0	95.3	11.76
		2	89.5	96.4	-
		3	90.0	**96.7**	-
	STF-Net [17]	1	88.8	95.0	7.5
		2	90.8	96.2	-
		4	***91.1***	***96.5***	-
	STH-DRL [8]	1	90.8	**96.7**	-
	ShiftGCN [21]	1	87.8	95.1	2.50
		2	89.7	96.0	5.00
		4	90.7	***96.5***	10.00
	ShiftGCN++ [23]	1	87.9	94.8	**0.40**
		2	89.7	95.7	***0.80***
		4	90.5	96.3	1.70
	ST-GCN†	1	86.0	93.4	16.73
		2	88.1	94.9	33.46
	AGCN†	1	86.4	94.3	18.69
		2	88.3	95.3	37.38
	S-TR†	1	86.8	93.8	16.20
		2	89.1	95.3	32.40
Frame	Deep-LSTM [26]	1	60.7	67.3	-
	VA-LSTM [7]	1	79.2	87.7	-
	*Co*ST-GCN (ours)	1	86.0	93.4	0.27
		2	88.1	94.8	0.54
	*Co*ST-GCN* (ours)	1	86.3	93.8	***0.16*** td
		2	88.3	***95.0***	0.32
	*Co*AGCN (ours)	1	86.4	94.2	0.30
		2	88.2	**95.3**	0.60
	*Co*AGCN* (ours)	1	84.1	92.6	0.22
		2	86.0	93.1	0.44
	*Co*S-TR (ours)	1	86.5	93.5	0.17
		2	**88.8**	**95.3**	0.34
	*Co*S-TR* (ours)	1	86.3	92.4	**0.15**
		2	***88.9***	94.8	0.30

TABLE IV: **NTU RGB+D 120** comparison with recent methods, grouped by clip- and frame-based inference. Noted are the number of streams (S.), top-1 validation accuracy, and FLOPs per prediction. †Results for our implementation. Highlights indicate **best**, ***next-best*** and Pareto optimal results.

	Model	S.	Accuracy (%) X-Sub	Accuracy (%) X-Set	FLOPs (G)
Clip	ST-LSTM [6]	1	55.7	57.9	-
	SGN [24]	1	79.2	81.5	-
	MS-G3D [14]	2	**86.9**	**88.4**	-
	Tripool [20]	1	80.1	82.8	-
	STF-Net [17]	4	***86.5***	***88.2***	7.5
	ShiftGCN [21]	1	80.9	83.2	2.50
		2	85.3	86.6	5.00
		4	85.9	87.6	10.00
	ShiftGCN++ [23]	1	80.5	83.0	**0.40**
		2	84.9	86.2	***0.80***
		4	85.6	87.2	1.70
	ST-GCN†	1	79.0	80.7	16.73
		2	83.7	85.1	33.46
	AGCN†	1	79.7	80.7	18.69
		2	84.0	85.4	37.38
	S-TR†	1	80.2	81.8	16.20
		2	84.8	86.2	32.40
Frame	*Co*ST-GCN (ours)	1	78.9	80.7	0.27
		2	83.7	85.1	0.54
	*Co*ST-GCN* (ours)	1	79.4	81.6	***0.16***
		2	84.0	85.5	0.32
	*Co*AGCN (ours)	1	79.6	80.7	0.30
		2	84.0	85.3	0.60
	*Co*AGCN* (ours)	1	77.3	79.1	0.22
		2	80.4	82.0	0.44
	*Co*S-TR (ours)	1	80.1	81.7	0.17
		2	***84.8***	**86.1**	0.34
	*Co*S-TR* (ours)	1	79.7	81.7	**0.15**
		2	**84.8**	***86.1***	0.30

ing multiple complementary techniques, including architecture search and knowledge distillation. Instead of proposing one particular architecture, our proposed approach of translating ST-GCN-based networks into CINs is generic and can be applied alongside many other methods.

Many works have shown that the inclusion of multiple modalities leads to increased accuracy [10, 11, 14, 15, 21]. In our context, these modalities amount to *joints*, which are the original coordinates of the body joints, and *bones*, which are the differences between connected joints. Additional *joint motion* and *bone motion* modalities can be retrieved by computing the differences between adjacent frames in time for the joint and bone streams respectively. Models are trained individually on each stream and combined by adding their softmax outputs prior to prediction.

We evaluate and compare our proposed continual models, which are *Co*ST-GCN, *Co*AGCN, *Co*S-TR, with prior works on the NTU RGB+D 60, NTU RGB+D 120, and Kinetics Skeleton 400 datasets as presented in Tables III, IV, and V.

The *Co*ST-GCN and *Co*S-TR models transfer well across all datasets both with (*) and without padding and stride modifications. For *Co*AGCN, we find that the change to stride one deteriorates accuracy. We surmise that the attention matrix in Eq. (7) may need a larger receptive field (basing the attention on more nodes as in AGCN) to provide beneficial adaptations; a per-step change in attention might provide more noise than clarity in middle and late network layers. As found in prior works, the multi-stream approach with ensemble predictions gives a meaningful boost in accuracy across all experiment.

The Continual Skeleton models provide competitive accu-

TABLE V: **Kinetics Skeleton 400** comparison with recent methods, grouped by clip- and frame-based inference. Noted are the number of streams (S.), top-1 and top-5 validation accuracy, and FLOPs per prediction. †Results for our implementation. Highlights indicate **best**, ***next-best*** and Pareto optimal results.

	Model	S.	Accuracy (%) Top-1	Accuracy (%) Top-5	FLOPs (G)
Clip	Deep LSTM [10, 39]	1	16.4	35.3	-
	TCN [3, 10]	1	20.3	40.0	-
	AS-GCN [37]	1	34.8	56.5	-
	ST-GR [40]	1	33.6	56.1	-
	Tripool [20]	1	34.1	56.2	-
	STF-Net [17]	4	36.1	58.9	-
	DGNN [15]	4	***36.9***	59.6	-
	MS-G3D [14]	2	**38.0**	***60.9***	-
	MS-AAGCN [12]	4	37.8	**61.0**	-
	ST-GCN†	1	33.4	56.1	***12.04***
		2	34.4	57.5	24.09
	AGCN†	1	35.0	57.5	13.45
		2	***36.9***	59.6	26.91
	S-TR†	1	32.0	54.9	**11.62**
		2	34.7	57.9	23.24
Frame	*Co*ST-GCN (ours)	1	31.8	54.6	0.16
		2	33.1	56.1	0.32
	*Co*ST-GCN* (ours)	1	30.2	52.4	**0.11**
		2	32.2	54.5	0.22
	*Co*AGCN (ours)	1	33.0	55.5	0.18
		2	**35.0**	**57.3**	0.36
	*Co*AGCN* (ours)	1	23.3	44.3	***0.12***
		2	27.5	49.1	0.25
	*Co*S-TR (ours)	1	29.7	52.6	0.16
		2	***32.7***	***55.6***	0.31
	*Co*S-TR* (ours)	1	27.4	49.7	**0.11**
		2	29.9	52.7	0.22

racy at multiple orders of magnitude reduction of FLOPs per prediction in the online setting compared to the original non-continual models. While none of our results beat prior state-of-the-art accuracy in absolute terms, this was never the intent with the method. Rather, we have successfully shown that online inference can be greatly accelerated for models in the ST-GCN family with state-of-the-art accuracy/complexity trade-offs to follow. For instance, our one and two-stream *Co*S-TR* achieve Pareto optimal[3] results on all subsets of the NTU RGB+D 60 and NTU RGB+D 120 datasets meaning that no other model improves on either accuracy and FLOPs without reducing the other. Pareto optimal models have been highlighted in Tables III, IV, and V accordingly. Our approach may be used similarly to accelerate other architectures for skeleton-based human action recognition with temporal convolutions.

V. Conclusion

In this paper, we proposed *Continual* Spatio-Temporal Graph Convolutional Networks, an architectural enhancement for networks operating on time-dependent graph structures, which augments prior methods with the ability to perform predictions frame-by-frame during online inference while attaining weight compatibility for batch inference. We re-implement and benchmark three prominent methods for skeleton-based action recognition, the ST-GCN, AGCN, and S-TR, as novel Continual Inference Networks, *Co*ST-GCN, *Co*AGCN, and *Co*S-TR, and propose architectural modifications to maximize their frame-by-frame inference speed. Through experiments on three widely used human skeleton datasets, NTU RGB+D 60, NTU RGB+D 120, and Kinetics Skeleton 400, we show up to 26× on-hardware throughput increases, 109× reduction in FLOPs per prediction, and 52% reduction in maximum memory allocated during online inference with similar accuracy to those of the original networks. During offline inference of full spatio-temporal sequences, the models operate identically to prior works. Our proposed architectural modifications are generic in nature and can be used for a variety of problems involving time-dependent graph structures, like traffic control. Proposal of methods based on *Continual* ST-GCNs which can address challenges posed by such problems can be an interesting future research direction. It is our hope, that this innovation will make online processing of time-varying graphs viable on recourse-constrained devices and systems with real-time requirements.

[3]A solution is pareto optimal if there is no other solution where one metric is improved without causing deterioration in another.

Acknowledgment

This work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 871449 (OpenDR). This publication reflects the authors' views only. The European Commission is not responsible for any use that may be made of the information it contains.

References

[1] F. Han, B. Reily, W. Hoff, and H. Zhang, "Space-time representation of people based on 3D skeletal data: A review," *Computer Vision and Image Understanding*, vol. 158, pp. 85–105, 2017.

[2] 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.

[3] T. S. Kim and A. Reiter, "Interpretable 3D human action analysis with temporal convolutional networks," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2017, pp. 1623–1631.

[4] M. Liu, H. Liu, and C. Chen, "Enhanced skeleton visualization for view invariant human action recognition," *Pattern Recognition*, vol. 68, pp. 346–362, 2017.

[5] M. Naveenkumar and S. Domnic, "Deep ensemble network using distance maps and body part features for skeleton based action recognition," *Pattern Recognition*, vol. 100, p. 107125, 2020.

[6] J. Liu, A. Shahroudy, D. Xu, and G. Wang, "Spatio-temporal LSTM with trust gates for 3D human action recognition," in *European Conference on Computer Vision*. Springer, 2016, pp. 816–833.

[7] P. Zhang, C. Lan, J. Xing, W. Zeng, J. Xue, and N. Zheng, "View adaptive recurrent neural networks for high performance human action recognition from skeleton data," in *IEEE International Conference on Computer Vision*, 2017, pp. 2117–2126.

[8] B. Nikpour and N. Armanfard, "Spatio-temporal hard attention learning for skeleton-based activity recognition," *Pattern Recognition*, p. 109428, 2023.

[9] N. Heidari, L. Hedegaard, and A. Iosifidis, "Graph convolutional networks," in *Deep Learning for Robot Perception and Cognition*. Elsevier, 2022, pp. 71–99.

[10] S. Yan, Y. Xiong, and D. Lin, "Spatial temporal graph convolutional networks for skeleton-based action recognition," in *AAAI Conference on Artificial Intelligence*, 2018, pp. 7444–7452.

[11] L. Shi, Y. Zhang, J. Cheng, and H. Lu, "Two-stream adaptive graph convolutional networks for skeleton-based action recognition," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 12 026–12 035.

[12] ——, "Skeleton-based action recognition with multi-stream adaptive graph convolutional networks," *IEEE Transactions on Image Processing*, vol. 29, pp. 9532–9545, 2020.

[13] Y. Li, R. Xia, and X. Liu, "Learning shape and motion representations for view invariant skeleton-based action recognition," *Pattern Recognition*, vol. 103, p. 107293, 2020.

[14] Z. Liu, H. Zhang, Z. Chen, Z. Wang, and W. Ouyang, "Disentangling and unifying graph convolutions for skeleton-based action recognition," *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 140–149, 2020.

[15] L. Shi, Y. Zhang, J. Cheng, and H. Lu, "Skeleton-based action recognition with directed graph neural networks," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 7912–7921.

[16] C. Si, Y. Jing, W. Wang, L. Wang, and T. Tan, "Skeleton-based action recognition with hierarchical spatial reasoning and temporal stack learning network," *Pattern Recognition*, vol. 107, p. 107511, 2020.

[17] L. Wu, C. Zhang, and Y. Zou, "Spatiotemporal focus for skeleton-based action recognition," *Pattern Recognition*, vol. 136, p. 109231, 2023.

[18] W. Peng, X. Hong, H. Chen, and G. Zhao, "Learning graph convolutional network for skeleton-based human action recognition by neural searching." in *AAAI Conference on Artificial Intelligence*, 2020, pp. 2669–2676.

[19] N. Heidari and A. Iosifidis, "Progressive spatio-temporal graph convolutional network for skeleton-based human action recognition," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2021, pp. 3220–3224.

[20] W. Peng, X. Hong, and G. Zhao, "Tripool: Graph triplet pooling for 3d skeleton-based action recognition," *Pattern Recognition*, vol. 115, p. 107921, 2021.

[21] K. Cheng, Y. Zhang, X. He, W. Chen, J. Cheng, and H. Lu, "Skeleton-based action recognition with shift graph convolutional network," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020.

[22] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *International Conference on Learning Representations*, 2017.

[23] K. Cheng, Y. Zhang, X. He, J. Cheng, and H. Lu, "Extremely lightweight skeleton-based action recognition with shiftgcn++," *IEEE Transactions on Image Processing*, vol. 30, pp. 7333–7348, 2021.

[24] P. Zhang, C. Lan, W. Zeng, J. Xing, J. Xue, and N. Zheng, "Semantics-guided neural networks for efficient skeleton-based human action recognition," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020.

[25] N. Heidari and A. Iosifidis, "Temporal Attention-Augmented Graph Convolutional Network for Efficient Skeleton-Based Human Action Recognition," in *International Conference on Pattern Recognition*, 2020.

[26] A. Shahroudy, J. Liu, T.-T. Ng, and G. Wang, "Ntu rgb+d: A large scale dataset for 3d human activity analysis," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, June 2016.

[27] J. Liu, A. Shahroudy, M. Perez, G. Wang, L.-Y. Duan, and A. 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.

[28] W. Kay, J. Carreira, K. Simonyan, B. Zhang, C. Hillier, S. Vijayanarasimhan, F. Viola, T. Green, T. Back, P. Natsev, M. Suleyman, and A. Zisserman, "The kinetics human action video dataset," *preprint, arXiv:1705.06950*, 2017.

[29] C. Plizzari, M. Cannici, and M. Matteucci, "Skeleton-based action recognition via spatial and temporal transformer networks," *Computer Vision and Image Understanding*, vol. 208, p. 103219, 2021.

[30] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 30, 2017, pp. 5998–6008.

[31] L. Hedegaard and A. Iosifidis, "Continual 3d convolutional neural networks for real-time processing of videos," in *European Conference on Computer Vision*. Springer, 2022, pp. 369–385.

[32] L. Hedegaard, A. Bakhtiarnia, and A. Iosifidis, "Continual Transformers: Redundancy-Free Attention for Online Inference," *preprint, arXiv:2201.06268*, 2022.

[33] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," in *NeurIPS Workshop*, 2017.

[34] L. Hedegaard, "Ride the lightning," *GitHub. Note: https://github.com/LukasHedegaard/ride*, 2021.

[35] L. N. Smith and N. Topin, "Super-convergence: very fast training of neural networks using large learning rates," in *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications*, vol. 11006, International Society for Optics and Photonics. SPIE, 2019, pp. 369 – 386.

[36] P. Goyal, P. Dollár, R. B. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, large minibatch sgd: Training imagenet in 1 hour," *preprint, arXiv:1706.02677*, 2017.

[37] M. Li, S. Chen, X. Chen, Y. Zhang, Y. Wang, and Q. Tian, "Actional-structural graph convolutional networks for skeleton-based action recognition," *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3590–3598, 2019.

[38] C. Si, W. Chen, W. Wang, L. Wang, and T. Tan, "An attention enhanced graph convolutional lstm network for skeleton-based action recognition," *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1227–1236, 2019.

[39] A. Shahroudy, J. Liu, T.-T. Ng, and G. Wang, "NTU RGB+D: A large scale dataset for 3D human activity analysis," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2016, pp. 1010–1019.

[40] B. Li, X. Li, Z. Zhang, and F. Wu, "Spatio-temporal graph routing for skeleton-based action recognition," in *AAAI*, 2019.

Lukas Hedegaard is a PhD candidate at Aarhus University, Denmark. He received his M.Sc. degree in Computer Engineering in 2019 and B.Eng. degree in Electronics in 2017 at Aarhus University, specialising in signal processing and machine learning. With a common theme of efficient deep learning, his research endeavours span from online inference acceleration and human activity recognition to transfer learning and domain adaptation.

Negar Heidari is a Postdoctoral researcher at Aarhus University, Denmark. She completed her PhD in Signal Processing and Machine Learning at the Department of Electrical and Computer Engineering, Aarhus University in 2022. Her current research interests include machine learning, deep learning and computer vision with a focus on computational efficiency.

Alexandros Iosifidis (SM'16) is a Professor at Aarhus University, Denmark. His research interests focus on neural networks and statistical machine learning finding applications in computer vision, financial modelling and graph analysis problems. He serves as Associate Editor in Chief for Neurocomputing (for Neural Networks research area), as an Area Editor for Signal Processing: Image Communication, and as an Associate Editor for IEEE Transactions on Neural Networks and Learning Systems. He was an Area Chair for IEEE ICIP 2018-2022 and EUSIPCO 2019,2021, and Publicity co-Chair of IEEE ICME 2021. He was the recipient of the EURASIP Early Career Award 2021 for contributions to statistical machine learning and artificial neural networks.

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.

CONTINUAL TRANSFORMERS: REDUNDANCY-FREE ATTENTION FOR ONLINE INFERENCE

Lukas Hedegaard, Arian Bakhtiarnia & Alexandros Iosifidis
Department of Electrical and Computer Engineering
Aarhus University
Aarhus, Denmark
{lhm,arianbakh,ai}@ece.au.dk

ABSTRACT

Transformers in their common form are inherently limited to operate on whole token sequences rather than on one token at a time. Consequently, their use during online inference on time-series data entails considerable redundancy due to the overlap in successive token sequences. In this work, we propose novel formulations of the Scaled Dot-Product Attention, which enable Transformers to perform efficient online token-by-token inference on a continual input stream. Importantly, our modifications are purely to the order of computations, while the outputs and learned weights are identical to those of the original Transformer Encoder. We validate our *Continual* Transformer Encoder with experiments on the THUMOS14, TVSeries and GTZAN datasets with remarkable results: Our *Continual* one- and two-block architectures reduce the floating point operations per prediction by up to 63× and 2.6×, respectively, while retaining predictive performance.

1 INTRODUCTION

Many real-life usage scenarios such as the perception in self-driving cars and live monitoring of critical resources process a continual stream of inputs and require near-instantaneous predictions per time-step. This stands in contrast to what many common benchmarks for deep learning evaluate, namely the operation on distinct batches of data with no inter-batch relationships. Consequently, a plethora of methods have been developed (Ji et al., 2013; Carreira & Zisserman, 2017; Varol et al., 2018; Yan et al., 2018; Heidari & Iosifidis, 2021; Vaswani et al., 2017; Arnab et al., 2021; Bakhtiarnia et al., 2021b), which focus on batch-wise processing, but fail to optimise for online operation, where new information (e.g., a video frame / token) arrives at each step from a continual input stream, and future information is not available at the current time-step. We need a class of networks, which operate efficiently on *both batches of data and on continual streams*.

Accordingly, we propose a reformulation of the Transformer Encoder as a Continual Inference Network (CIN, Section 2.1) which accelerates the stream processing on time-series data, while retaining weight-compatibility. Specifically, we derive two variants of Continual Scaled Dot-Product Attention (SDA) for the cases where prior output tokes *should* and *should not* be updated after observing a new input token. Notably, our attention formulations reduce the per-step cost of SDA (Vaswani et al., 2017) from time complexity $\mathcal{O}(n^2d)$ to $\mathcal{O}(nd)$ and memory complexity $\mathcal{O}(n^2)$ to $\mathcal{O}(nd)$ and are readily embedded into Continual Multi-Head Attention (MHA) and Continual Transformer Encoder blocks. Finally, we propose the use of Recycling Positional Encoding to accommodate progressive caching of partial attention results for continual data streams.

Due to the interdependence of SDA outputs, Continual Transformers are most efficient for shallow architectures. Shallow Transformers have many applications such as augmentations of CNNs (Touvron et al., 2021), light-weight Natural Language Processing (Cornia et al., 2020), fusion operations in multi-modal (e.g. audio-visual) settings (Chumachenko et al., 2022) and early exit branches in multi-exit architectures (Bakhtiarnia et al., 2021a;b). In our experiments[1], we validate their exceptional efficiency improvements on common benchmarks in Online Action Detection (Idrees et al., 2017) and Online Audio Classification (Tzanetakis et al., 2001).

[1]Source code: https://github.com/lukashedegaard/continual-transformers.

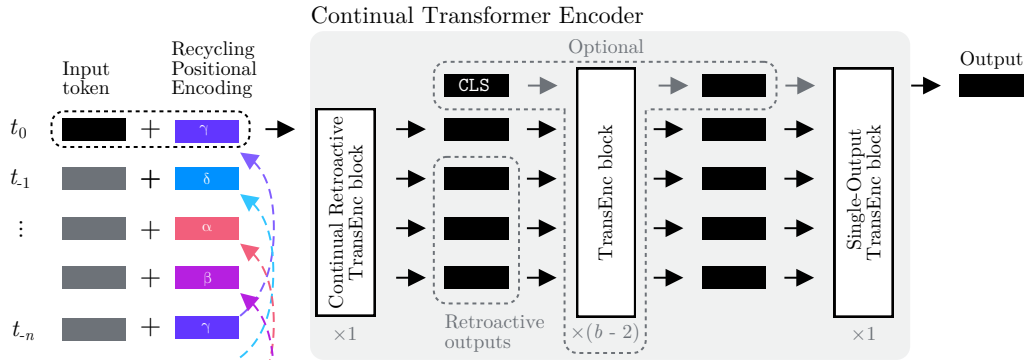


Figure 1: **Multi-block Continual Transformer Encoder with Recycling Positional Encoding.** For $b > 2$ blocks, regular Transformer Encoder blocks can be added between an initial Continual Retroactive block and a final Single-Output block. A class-token may be used after the initial block.

2 Related Work

2.1 Continual Inference Networks

Definition (**Continual Inference Network**). *A Deep Neural Network, which*

- *is capable of continual step inference without computational redundancy,*
- *is capable of batch inference corresponding to a non-continual Neural Network,*
- *produces identical outputs for batch- and step inference given identical receptive fields,*
- *uses one set of trainable parameters for both batch and step inference.*

These requirements ensure that a Neural Network has broad applicability for both (offline) batch-wise inference (i.e., most research benchmarks) and online stream processing. While non-CINs can operate on streams of data by caching prior steps in a first-in first-out (FIFO) queue and aggregating them to a full (spatio-)temporal input, which is processed similarly to an offline batch, this entails computational redundancy in proportion with the sequence length. CINs perform step-wise inference without such caching and repeat computation. Uni-directional Recurrent Neural Networks are an example of Continual Inference Networks. Their default mode of operation is by time-step and they are easily applied to spatio-temporal batches of data by concatenation of the step-wise outputs. Recently, a modification to the spatio-temporal 3D convolution was proposed (Hedegaard & Iosifidis, 2021), which enables existing 3D CNNs to operate efficiently during continual inference. A similar principle was used to enhance Spatio-temporal Graph Convolutions as well (Hedegaard et al., 2022). In Section 3, we derive a CIN formulation for Transformer Encoders.

2.2 Transformer architectures

Initially proposed for sequence-to-sequence modelling in Natural Language Processing, the Transformer (Vaswani et al., 2017) has become a canonical building block in many applications of Deep Learning, including Computer Vision (Dosovitskiy et al., 2021; Arnab et al., 2021; Wang et al., 2021; Carion et al., 2020) and Audio Classification (Gong et al., 2021). Their success can be partly attributed to reduced inductive bias compared with CNNs and RNNs, which allows better adaptations when sufficiently large datasets are available; the Scaled Dot-Product Attention (SDA) maps a set of input tokens to a set of outputs without inherent preconceptions. However, this many-to-many attention exhibits quadratic growth in time and space complexity with the token count n in the set.

A great deal of research has sought to improve the efficiency of Transformers (Tay et al., 2020). Block-wise or Chunking methods such as Image Transformer (Parmar et al., 2018) and Vision Transformer (Dosovitskiy et al., 2021) group up entities of a local receptive field into a single block, reducing the $\mathcal{O}(n^2)$ complexity to $\mathcal{O}(n_b^2)$, where $n_b < n$ is the number of blocks. Techniques such as sliding windows, dilation and pooling can be used to achieve a similar effect (Beltagy et al., 2020). The Reformer (Kitaev et al., 2020) reduces the complexity to $\mathcal{O}(n \log n)$ by learning group-

ings in a data-driven manner via Locality-Sensitive Hashing (LSH). A different paradigm aims to derive approximations of the self-attention matrix. Methods such as Linformer (Wang et al., 2020), Nyströmformer (Xiong et al., 2021) and Performer (Choromanski et al., 2021) reduce the complexity from $\mathcal{O}(n^2)$ to $\mathcal{O}(n)$. Unlike these efforts, our approach produces the *exact* same computational outputs for temporal sequences as the original Multi-Head Attention.

3 CONTINUAL TRANSFORMERS

In this work, we examine the use of Transformer Encoders for stream-processing, where we receive one token per time-step. Specifically, the query, key and value inputs constitute a continual stream of d-dimensional tokens and we wish to compute the outputs for each step immediately considering $n-1$ prior tokens. We begin our exposition in Section 3.1 by considering the Scaled Dot-Product Attention (SDA) for this task. To alleviate the inefficiencies of SDA, we propose two alternative computational sequences in Section 3.2 and Section 3.3 and compare them to SDA in Section 3.4. Finally, sections 3.5-3.8 build up the full architecture, and discuss architectural considerations.

3.1 REGULAR SCALED DOT-PRODUCT ATTENTION

Denoting query, key, and value sequence matrices by $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V} \in \mathbb{R}^{n \times d}$, the regular Scaled Dot-Product Attention first defined by Vaswani et al. (2017) can be written as:

$$\text{Att}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \boldsymbol{D}^{-1}\boldsymbol{A}\boldsymbol{V} \qquad \boldsymbol{A} = \exp\left(\boldsymbol{Q}\boldsymbol{K}^\top / \sqrt{d}\right) \qquad \boldsymbol{D} = \text{diag}\left(\boldsymbol{A}\mathbb{1}_n^\top\right), \tag{1}$$

where $\boldsymbol{A}, \boldsymbol{D} \in \mathbb{R}^{n \times n}$ and $\mathbb{1}_n$ is a row-vector of n ones. In each time-step, we can update $\boldsymbol{Q}$, $\boldsymbol{K}$, and $\boldsymbol{V}$ by discarding their oldest token and prepending a new one in a FIFO manner. This is a common implementation for step-wise inference, e.g. found in the FAIRSEQ library (Ott et al., 2019).

Each time-step results in $2n^2d + 2nd$ multiplications, $2n^2d - nd - n$ additions, and n^2 exponentiations as accounted for in Appendix A.1, which amounts to a time complexity of $\mathcal{O}(n^2d)$ and a $\mathcal{O}(n^2)$ memory complexity originating from the transient feature-map $\boldsymbol{A}$. Furthermore, a constant-sized cache of size $3(n-1)d$ is needed to store the $n-1$ latest tokens in $\boldsymbol{Q}$, $\boldsymbol{K}$ and $\boldsymbol{V}$. We could avoid considerable redundancy by caching $\boldsymbol{Q}\boldsymbol{K}^\top$ directly. However, this comes with a memory penalty of $(n-1)^2$. Fortunately, another computational scheme can be devised.

3.2 CONTINUAL RETROACTIVE SCALED DOT-PRODUCT ATTENTION

We can compute $\boldsymbol{D}^{-1}\boldsymbol{A}\boldsymbol{V}$ in a step-wise manner using the latest query, key, and value steps, $\boldsymbol{q}_{\text{new}}, \boldsymbol{k}_{\text{new}}, \boldsymbol{v}_{\text{new}} \in \mathbb{R}^{1 \times d}$, alongside appropriately cached partial results. The softmax normalisation with $\boldsymbol{D}^{-1}$ can be efficiently implemented via column-aligned element-wise multiplications (denoted by $\odot$ hereafter) of a column-vector $\boldsymbol{d} = \boldsymbol{A}\mathbb{1}_n^\top$. If we cache the $n-1$ values for the prior step tokens, i.e. $\boldsymbol{d}_{\text{mem}} = \boldsymbol{A}_{\text{prev}}^{(-n+1:-1)}\mathbb{1}_{n-1}^\top$, alongside $\boldsymbol{Q}$ and $\boldsymbol{K}$, we can define the step update as:

$$\boldsymbol{d}^{(-n+1:-1)} = \boldsymbol{d}_{\text{mem}}^{(-n+2:0)} - \exp\left(\boldsymbol{Q}_{\text{mem}}\boldsymbol{k}_{\text{old}}^\top\right) + \exp\left(\boldsymbol{Q}_{\text{mem}}\boldsymbol{k}_{\text{new}}^\top\right) \tag{2}$$

$$\boldsymbol{d}^{(0)} = \exp\left(\frac{\boldsymbol{q}_{\text{new}}}{\sqrt{d}}\left(\boldsymbol{K}_{\text{mem}} \parallel \boldsymbol{k}_{\text{new}}\right)^\top\right)\mathbb{1}_n^\top, \tag{3}$$

where $\boldsymbol{Q}_{\text{mem}}$ ($\boldsymbol{K}_{\text{mem}}$) are the $n-1$ prior query (key) tokens, $\boldsymbol{k}_{\text{old}}$ is the key from n steps ago, and $\parallel$ denotes concatenation of matrices along the first dimension. Negative indices indicate prior time-steps. An update for $\boldsymbol{A}\boldsymbol{V}$ can likewise be defined as a function of the $n-1$ prior values $\boldsymbol{A}\boldsymbol{V}_{\text{mem}}$:

$$\boldsymbol{A}\boldsymbol{V}^{(-n+1:-1)} = \boldsymbol{A}\boldsymbol{V}_{\text{mem}}^{(-n+2:0)} - \exp\left(\boldsymbol{Q}_{\text{mem}}\boldsymbol{k}_{\text{old}}^\top\right)\boldsymbol{v}_{\text{old}} + \exp\left(\boldsymbol{Q}_{\text{mem}}\boldsymbol{k}_{\text{new}}^\top\right)\boldsymbol{v}_{\text{new}} \tag{4}$$

$$\boldsymbol{A}\boldsymbol{V}^{(0)} = \exp\left(\frac{\boldsymbol{q}_{\text{new}}}{\sqrt{d}}\left(\boldsymbol{K}_{\text{mem}} \parallel \boldsymbol{k}_{\text{new}}\right)^\top\right)\left(\boldsymbol{V}_{\text{mem}} \parallel \boldsymbol{v}_{\text{new}}\right). \tag{5}$$

Finally, we compute the Continual Retroactive Attention output in the usual manner:

$$CoRe\text{Att}(\boldsymbol{q}_{\text{new}}, \boldsymbol{k}_{\text{new}}, \boldsymbol{v}_{\text{new}}) = \boldsymbol{d}^{-1} \odot \boldsymbol{A}\boldsymbol{V}. \tag{6}$$

An visual depiction of these update steps is provided in Appendix A.2. A time-step can now be computed with $7nd + 2n - 3d$ multiplications, $6nd + 3n - 6d - 3$ additions, and $3n - 2$ exponentials. This time complexity of $\mathcal{O}(nd)$ per step and a $\mathcal{O}(nd)$ memory complexity is a significant improvement over the prior $\mathcal{O}(n^2d)$ and $\mathcal{O}(n^2)$ complexities in Section 3.1.

3.3 CONTINUAL SINGLE-OUTPUT SCALED DOT-PRODUCT ATTENTION

Both the Regular and Continual Retroactive Dot-Product Attentions produce attention outputs for the current step, as well as $n-1$ retroactively updated steps. In cases where retroactive updates are not needed, we can simplify the computation greatly via a Continual Single-Output Dot-Product Attention ($CoSi$Att). In essence, the regular SDA is reused, but prior values of $\mathbf{k}$ and $\mathbf{v}$ are cached between steps (as in (Ott et al., 2019)), and only the attention corresponding to a single query token $\boldsymbol{q}$ is computed:

$$CoSi\text{Att}(\boldsymbol{q}, \boldsymbol{k}_{\text{new}}, \boldsymbol{v}_{\text{new}}) = \mathbf{a}\left(\boldsymbol{V}_{\text{mem}} \parallel \boldsymbol{v}_{\text{new}}\right) / \mathbf{a}\mathbb{1}_n^\top, \qquad \mathbf{a} = \exp\left(\frac{\boldsymbol{q}}{\sqrt{d}}\left(\boldsymbol{K}_{\text{mem}} \parallel \boldsymbol{k}_{\text{new}}\right)^\top\right). \tag{7}$$

A step output is computed with $2nd + 2d$ multiplications, $2nd - d - 1$ additions, and n exponentials. The time- and memory complexities remain $\mathcal{O}(nd)$ per step. Using the (leading) query $\boldsymbol{q}_{\text{new}}$ as input, the attention is purely causal. Alternatively, prior (lagging) query vectors could be cached and used as query input, though this would introduce a network delay.

3.4 COMPARISON OF SCALED DOT-PRODUCT ATTENTIONS

Assuming $n-1$ prior $\boldsymbol{q}$, $\boldsymbol{k}$ and $\boldsymbol{v}$ steps have been calculated by the Continual SDA modules, and that $\boldsymbol{Q} = (\boldsymbol{Q}_{\text{mem}} \parallel \boldsymbol{q}_{\text{new}})$, $\boldsymbol{K} = (\boldsymbol{K}_{\text{mem}} \parallel \boldsymbol{k}_{\text{new}})$, and $\boldsymbol{V} = (\boldsymbol{V}_{\text{mem}} \parallel \boldsymbol{v}_{\text{new}})$, we have the correspondence:

$$\text{Att}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V})^{(t)} = CoRe\text{Att}(\boldsymbol{q}_{\text{new}}, \boldsymbol{k}_{\text{new}}, \boldsymbol{v}_{\text{new}})^{(t)} = CoSi\text{Att}(\mathbf{q}_t, \boldsymbol{k}_{\text{new}}, \boldsymbol{v}_{\text{new}}) \tag{8}$$

Here, $\boldsymbol{q}_t$ is the t^{th} row of $\boldsymbol{Q}$, i.e. $\boldsymbol{Q}^{(t)}$. During stream processing, the complexity of the Continual Retroactive SDA scales significantly more favourably that the regular SDA. For example, the floating point operations (FLOPs) are reduced by $31\times$ when $n = d = 100$ and $308\times$ when $n = d = 1000$. If retroactive output updates are not needed, the Continual Single-Output SDA reduces FLOPs by respectively $100\times$ and $1000\times$. The scaling properties are detailed in Appendix A.1.

3.5 CONTINUAL MULTI-HEAD ATTENTION

Continual Scaled Dot-Product Attentions can replace regular SDA's directly in a Multi-Head Attention (MHA). Given a new query, key, and value, $\boldsymbol{q}, \boldsymbol{k}, \boldsymbol{v}$, the Continual MHA is defined as

$$Co\text{MHA}(\boldsymbol{q}, \boldsymbol{k}, \boldsymbol{v}) = \left(\mathop{\Big\|}_{i=0}^{h-1} Co\text{Att}(\boldsymbol{q}\boldsymbol{W}_Q^i, \boldsymbol{k}\boldsymbol{W}_K^i, \boldsymbol{v}\boldsymbol{W}_V^i)\right)\boldsymbol{W}_O, \tag{9}$$

where $\parallel$ denotes concatenation of h heads and $\boldsymbol{W}_Q^i, \boldsymbol{W}_K^i \in \mathbb{R}^{d \times d_K/h}$, $\boldsymbol{W}_V^i \in \mathbb{R}^{d \times d_V/h}$, and $\boldsymbol{W}_O \in \mathbb{R}^{d_V \times d_O}$ are projection matrices of head i. CoAtt can be either $CoRe$Att or $CoSi$Att.

3.6 CONTINUAL TRANSFORMER ENCODER

A Continual MHA block can be integrated in a Continual Transformer Encoder block as follows:

$$\boldsymbol{z} = \text{LayerNorm}\left(\boldsymbol{y} + \text{FF}(\boldsymbol{y})\right), \qquad \boldsymbol{y} = \text{LayerNorm}\left(\text{Sel}(\boldsymbol{x}) + Co\text{MHA}(\boldsymbol{x}, \boldsymbol{x}, \boldsymbol{x})\right), \tag{10}$$

where $\boldsymbol{x}$ corresponds to the newest step input and $\text{Sel}(\cdot)$ selects a single (last) token of $\boldsymbol{x}$ if *CoSi*MHA is used, or selects all tokens otherwise. $\text{FF}(\cdot)$ is a two-layer feed-forward network with weights $\boldsymbol{W}_1, \boldsymbol{W}_2$, biases w_1, w_2, and a activation function $\sigma(\cdot)$, i.e. $\text{FF}(\boldsymbol{x}) = \sigma(\boldsymbol{x}\boldsymbol{W}_1 + w_1)\boldsymbol{W}_2 + w_2$. Aside from the residual selection, this is identical to common Transformer Encoder implementations (Vaswani et al., 2017; Dosovitskiy et al., 2021).

3.7 RECYCLING POSITIONAL ENCODING

Since a Transformer Encoder does not provide positional bias, it is common to augment a token $\boldsymbol{x}_i$ with a positional encoding $\boldsymbol{p}$, i.e. $\tilde{\boldsymbol{x}}_i = \boldsymbol{x}_i \circ \boldsymbol{p}_i$, where $\circ$ could be addition or concatenation. In regular Transformers, the index i denotes a position in a sequence rather than a position in time. However, this static positional assignment is problematic in the context of continual inference; the last token at time $t = 0$ will be the next-to-last token at time $t = 1$, and thus in need of a different

positional encoding than in the prior time-step. Instead, CINs require dynamic positions. There have been multiple prior works (Shaw et al., 2018; Huang et al., 2019; Dai et al., 2019) which create relative encodings by augmenting the SDA with positional offsets $\boldsymbol{P}$ between query and keys, i.e. $\boldsymbol{A} = \exp(\boldsymbol{Q}\boldsymbol{K}^\top/\sqrt{d} + \boldsymbol{P})$. While a similar modification to the continual attentions is possible, it is incompatibile with the regular SDA in Eq. (1). Instead, we use a *Recycling Positional Encoding* (RPE), which lets the positional encoding follow each token in time and recycles old encodings:

$$\tilde{\boldsymbol{x}}_t = \boldsymbol{x}_t + \boldsymbol{p}_{\tau_t}, \qquad \tau_t = (\tau_{t-1} + 1) \bmod T, \tag{11}$$

where T is the number of encodings. While RPE does not specify relative encodings explicitly, the absolute positional interpretation of each token changes dynamically when a new token arrives. In practice, the network learns relative, shift-invariant positional information by training with random τ_0 for each batch. Random shifts during training were recently explored in (Kiyono et al., 2021; Likhomanenko et al., 2021; Dehghani et al., 2019) as well. RPE can use either learned or predefined encodings. In the latter case, Cyclic Positional Encoding (Ma et al., 2021), a sinusoidal encoding inspired by Gray code, is a good fit. If we reuse the encoding immediately after an old token has "slided out", i.e. $T = n$, a token will have the same positional encoding relative to another whether it was m steps older or $n - m$ steps newer. The positional ambiguity can be avoided by extending the number of positional tokens to $T = 2n - 1$. We explore both options in Section 4.1.2.

3.8 Architectural considerations

Block count In Section 3.4, we observed an exact correspondence between the results of the continual and regular SDA layers. However, the correspondence does not necessarily hold for stacked layers. Consider the result of stacking two Continual Single-Output Transformer Encoder blocks. While the first block outputs a step t that is identical to that in a corresponding regular block, the second block would have been initialised with prior step-wise inputs, which were the result of prior input windows instead of the current one; the correspondence would not hold. Though it is not convertible to/from a regular Transformer Encoder, the stacked Single-Output Transformer Encoder architecture has the merit of efficiency. This was exploited in Transformer-XL (Dai et al., 2019). Given a single step input, the Continual Retroactive Transformer Encoder block produces output tokens corresponding to the entire observed sequence inside the window. Due to this one-to-many input-output mapping, it is not possible to stack multiple such layers. Nevertheless, it can be used in conjunction with a Continual Single-Output Transformer Encoder with optional regular Transformer Encoder blocks in between as illustrated in Fig. 1. The Regular Transformer Encoder blocks in the resulting architecture have a significantly larger computational complexity than the Continual Retroactive and Single-Output blocks. Consequently, we recommend that Continual Transformer Encoders be used primarily in lightweight architectures with one or two blocks unless compatibility with non-continual Transformers is not required and only Single-output blocks are used.

Class token It is common to add a class token as input to transformers (Devlin et al., 2019; Dosovitskiy et al., 2021), which accumulates information from other tokens prior to classification. However, it cannot be used naïvely with CINs, as this would effectively double the number of input steps. In practice, it can be employed in Continual multi-block Transformer Encoders as input to the second block (see Fig. 1), but this placement limits class token interaction with downstream layers. It can also be used for one-block Transformer Encoders if the value token is omitted as input.

Peak memory reduction trick The FLOPs for $\text{Att}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V})$ are exactly n times those of $CoSi\text{Att}(\mathbf{q}, \boldsymbol{k}_{\text{new}}, \boldsymbol{v}_{\text{new}})$. Comparing their memory complexity, the regular SDA is $\mathcal{O}(n^2)$, while the Single-output SDA is $\mathcal{O}(nd)$. In practical applications where system memory is limited, we may thus reduce the maximum memory requirement of the computational device at inference by up to d/n (assuming $n \gg d$) by computing each row of the attention individually. However, this may reduce throughput due to reduced parallelism.

4 Experiments

We provide case studies within two perception disciplines, namely Online Action Detection (Section 4.1) and Audio Classification (Section 4.2). In each case, we will start with a brief overview of the field, followed by experiments and results.

4.1 ONLINE ACTION DETECTION

Online Action Detection (OAD) (De Geest et al., 2016) entails the per-frame classification of human actions in a video stream as they happen without the ability to change prior predictions nor to use future information. This is fundamentally more restrictive than Temporal Action Localisation, where the whole video clip is processed before start and end frames of an action are determined (Shou et al., 2016; Xu et al., 2017; Shou et al., 2017; Wu et al., 2019).

The dominant design in OAD works at the time of writing is to employ a two-stream Convolutional Neural Network as backbone for frame-wise feature extraction with RGB images as inputs in one stream and Optical Flow fields in the other (Gao et al., 2017; Xu et al., 2019; Eun et al., 2020; Wang et al., 2021; Xu et al., 2021)[2]. On top of these, OAD methods encode temporal information and perform predictions per time-step, e.g. by means of RNNs (Gao et al., 2017; Xu et al., 2019; Eun et al., 2020) or Transformers (Wang et al., 2021; Xu et al., 2021). Alongside the action detection for the current frame, an action anticipation task may be learned in parallel by means of decoder structures, as this has been found to improve the primary OAD task.

Unlike RNNs, an output update for the regular SDA in a Transformer block cannot be naïvely computed for a single step by feeding successive video frames. Instead, prior step features must be cached, re-loaded and re-processed by the Transformer in each step in correspondence with a predefined window-size of prior steps. As laid out in Section 3.8, Continual Transformers are especially efficient when either one or two Continual Transformer Encoder blocks are used. Accordingly, we start our experiments with a set ablation studies to simplify a recent transformer-based architecture, the OadTR (Wang et al., 2021). We further investigate the impact of ablating class token position and the use of Recycling Positional Encoding and compare different RPE schemes for Continual Transformers. Finally, we evaluate our configurations on two widely used OAD datasets, THUMOS14 (Idrees et al., 2017) and TVSeries (De Geest et al., 2016).

4.1.1 EXPERIMENTAL SETUP

The THUMOS14 dataset (Idrees et al., 2017) for OAD has 200 and 213 validation and test videos, respectively, with frame-level class annotations across 20 classes. As in prior OAD works, the model is trained on the validation set and evaluated on the test set. Similar to Wang et al. (2021) we use pre-extracted features from a two-stream Temporal Segment Network (TSN) (Wang et al., 2019) trained on ActivityNet v1.3 (Heilbron et al., 2015) or Kinetics-400 (Carreira & Zisserman, 2017).

For TVSeries (De Geest et al., 2016), the network learns on the train and validations sets (20 videos) and evaluates on the test set (7 videos) as in (Wang et al., 2021). RGB and Optical Flow features were extraced using an MMAction2 (Contributors, 2020) pipeline with ActivityNet v1.3 (Heilbron et al., 2015) and Kinetics-400 (Carreira & Zisserman, 2017) pretrained TSN ResNet-50 (He et al., 2016) backbones. This is similar to the feature extraction process used by LSTR (Xu et al., 2021).

Following Wang et al. (2021), we use a batch size of 128, sequence length 64, initial learning rate 10^{-4} with a factor ten reduction each epoch, alongside weight decay 10^{-4}, and dropout with probability 0.1. We report results using two epochs of training on a Nvidia RTX2080 Ti GPU. We track mean Average Precision (mAP) for THUMOS14 and calibrated mean Average Precision (cmAP) (De Geest et al., 2016) for TVSeries, alongside FLOPs per prediction and parameters of the OAD module (feature extraction excluded). We report the mean $\pm$ standard deviation over five runs.

4.1.2 ABLATION STUDIES

Removing the Decoder As a first step to make an efficient Continual OadTR, we remove the decoder blocks used for action anticipation, which has a large impact on computational efficiency and the ease of transformation to a Continual Inference Network. The first two lines of Table 1a present the results of the removal. Contrary to the observations of Wang et al. (2021), we did not find any drop in accuracy when excluding the decoder. We do, however, gain a large reduction in FLOPs and model size; they were reduced to 58% and 30%, respectively. Given these computational improvements, we exclude the decoder in subsequent experiments.

[2]The feature extraction commonly used in Online Action Detection (OAD) works is in itself quite computationally costly. We consider the optimisation of the backbone as orthogonal future work and will follow the same feature extraction procedure as other OAD works at this time.

Table 1: **Ablation experiments** on THUMOS14 with TSN-Anet features. **Best** metrics are highlighted. '-' indicates that a particular feature was not used.

(a) **Class token** variations with OadTR. `CLS` pos. is the encoder block into which `CLS` is input.

Enc. blocks	Dec.	CLS pos.	mAP (%)	FLOPs (M)	Params (M)
3	✓	1	57.0±0.5	2445.6	74.7
3	-	1	57.0±0.4	1430.6	22.2
3	-	2	**57.3**±0.7	1423.5	22.2
3	-	3	56.7±0.6	1417.2	22.2
3	-	-	56.8±0.3	**1410.9**	22.2
2	-	1	56.5±0.5	1020.7	15.9
2	-	2	**56.7**±0.3	1014.5	15.9
2	-	-	56.6±0.3	**1008.1**	15.9
1	-	1	**57.1**±0.6	611.7	9.6
1	-	-	56.3±0.2	**605.5**	9.6

(b) **Positional encodings** variations for *Co*OadTr.

Enc. blocks	Re-cycling	Learn	Pos. tokens	mAP (%)	FLOPs (M)	Params (K)
2	-	✓	n	45.3±0.9	410.9	15832
2	✓	✓	n	56.4±0.3	410.9	15832
2	✓	✓	$2n-1$	56.0±0.5	410.9	15897
2	✓	-	n	55.8±1.0	410.9	**15767**
2	✓	-	$2n-1$	**56.8**±0.4	410.9	**15767**
1	-	✓	n	44.0±0.8	9.6	9535
1	✓	✓	n	55.6±0.3	9.6	9535
1	✓	✓	$2n-1$	55.6±0.3	9.6	9599
1	✓	-	n	54.4±1.8	9.6	**9469**
1	✓	-	$2n-1$	**56.1**±0.7	9.6	**9469**

(Re)moving the Class token Class tokens should not be input naively to the first Transformer Encoder layer of a CIN (see Section 3.8). Accordingly, we ablate its use and position. In cases where it is removed, we predict on the token corresponding to the last input token. The results of varying `CLS` pos are noted in Table 1a. For the one-block architecture, the removal came with noticeable drop in mAP, while the two-block architecture saw small improvements when removing or introducing the class token later. For the three block model, the use of class tokens in block two achieved the highest mAP. Though it is commonly accepted, that class tokens should be introduced alongside other inputs in the first block, our results indicate that they can accumulate sufficient information with only one or two blocks, and that later stage introduction may work better in some applications. In general, the achieved mAP when varying `CLS` pos. and number of blocks are very similar to one another, while (re)moving the class token and reducing the block size both reduce computational complexity. This encourages the use of shallow Transformer Encoders over deeper ones as well as the removal of class tokens, as we do in the following experiments.

Positional Encodings We can transfer parameters from the simplified one- and two block OadTR to the corresponding Continual architecture, *Co*OadTR. Here, the one block version (*Co*OadTR-b1) uses *CoSi*MHA, and the two block model (*Co*OadTR-b2) uses *CoRe*MHA in the first block and Single-output MHA in the second. However, a regular positional encoding is not suited for continual inference (see Section 3.7). We evaluate the performance of using non-continual encodings for continual inference, as well as of our proposed Recycling Positional Encodings with fixed or learned parameters. In addition, we explore the impact of extending the number of tokens from n to $2n-1$ to avoid positional ambiguity. As seen in Table 1b), non-continual encoding used in the continual setting result in severe mAP drop. Recycling Positional Encodings alleviate this. Comparing learned and fixed encodings, we find the learned encodings to work better when the number of encoding tokens corresponds to the sequence length n and the fixed encoding to work best when positional ambiguity is alleviated by extending the number of tokens to $2n-1$. Fixed encoding with $2n-1$ tokens works best overall and is employed in subsequent experiments unless stated otherwise. There is no difference in FLOPs for either strategy, and the difference in parameter count is negligible.

4.1.3 Comparison with prior works

We evaluate the (*Co*)OadTR architectures on THUMOS14 and TVSeries with two sets of features as described in Section 4.1.1. Since no prior OAD works have reported complexity metrics, we measured the FLOPs for TRN (Xu et al., 2019) based on the publicly available source code to serve as a point of reference. The results of this benchmark are presented in Table 2 and Fig. 2. OadTR and our simplified (continual) one-block (b1) and two-block (b2) versions without decoder and class tokens generally achieve competitive precision in comparison with prior works, surpassing all but OadTR and LSTR. On THUMOS14, our reproduced OadTR results are slightly lower than originally reported (Wang et al., 2021)[3], whereas achieved TVSeries results are higher[4]. The (*Co*)OadTR-b#

[3]The reported 58.3% on THUMOS14 could not be reproduced using their publicly available code.

[4]We attribute our higher mcAP to differences in the feature extraction pipeline.

Table 2: **Online Action Detection** results. FLOPs per prediction are noted for inference on THUMOS14. The **best** and ***next-best*** metrics are highlighted.

Model	Feat.	THUMOS14 mAP (%)	TVSeries mcAP (%)	FLOPs (M)
RED (Gao et al., 2017)	A.Net	45.3	79.2	-
TRN (Xu et al., 2019)		47.2	83.7	1387.5
FATS (Kim et al., 2021)		51.6	81.7	-
IDN (Eun et al., 2020)		50.0	84.7	-
TFN (Eun et al., 2021)		55.7	85.0	-
LSTR (Xu et al., 2021)		**65.3**	88.1	-
OadTR (Wang et al., 2021)		***58.3***	85.4	2445.6
OadTR†		57.0±0.5	**88.6**±0.1	2445.6
OadTR-b2†		56.6±0.3	***88.3***±0.2	1008.1
OadTR-b1†		56.3±0.2	88.1±0.1	605.5
*Co*OadTR-b2 (ours)		56.8±0.4	87.7±0.6	***410.9***
*Co*OadTR-b1 (ours)		56.1±0.7	87.6±0.7	**9.6**
TRN (Xu et al., 2019)	Kin.	62.1	86.2	1462.0
FATS (Kim et al., 2021)		59.0	84.6	-
IDN (Eun et al., 2020)		60.3	86.1	-
PKD (Zhao et al., 2020)		64.5	86.4	-
LSTR (Xu et al., 2021)		**69.5**	**89.1**	-
OadTR (Wang et al., 2021)		***65.2***	87.2	2513.5
OadTR†		64.2±0.3	***88.6***±0.1	2513.5
OadTR-b2†		64.5±0.5	88.3±0.2	1075.7
OadTR-b1†		63.9±0.5	88.1±0.1	673.0
*Co*OadTR-b2 (ours)		64.4±0.1	87.6±0.7	***411.9***
*Co*OadTR-b1 (ours)		64.2±0.4	87.7±0.4	**10.6**

†Using official source code or modifications there-off.

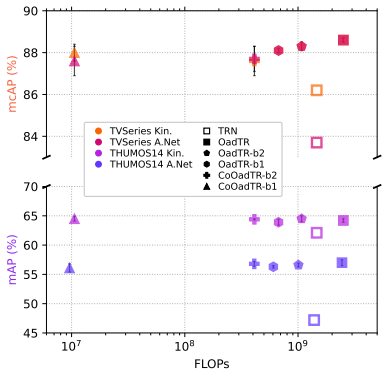


Figure 2: **Visual comparison** of OAD methods on THUMOS14 and TVSeries for backbones trained on ActivityNet 1.3 and Kinetics-400.

architecture largely retain precision and allow significantly reduced FLOPs per prediction. Our proposed continual variants *Co*OadTR-b1 and *Co*OadTR-b2 reduce FLOPs by **255**× and **6.1**×, respectively, compared to OadTR, while either achieving the same performance or conceding no more than one percentage point. On average, continual and non-continual (*Co*)OadTR-b# models achieve similar mAP on THUMOS14, while OadTR-b# have slightly higher mcAP on TVSeries. We attribute these discrepancies to differences in positional encoding. All in all, the *Co*OadTR-b# models provide far-superior computational efficiency to prior works, achieving state-of-the-art performance/efficiency trade-offs by a large margin.

4.1.4 Audio-Visual Online Action Detection

To showcase the validity of our method in audio-visual settings as well, we explore the addition of audio-features to the Online Action Detection task on THUMOS14. As described in Section 4.2, audio-features are extracted using Mel spectrograms and an AudioSet pre-trained VGGish network (Hershey et al., 2017) (output of the penultimate layer) on 1.0 second windows with a step size of 0.2 seconds to match the 5.0 FPS sampling rate of the video features.

The audio-features by themselves do not provide enough signal to reach good Online Action Detection performance (yielding only 6.7% mAP with an OadTR network). When concatenated with RGB and Flow they do provide a modest improvement as seen in Table 3. On average, this amounts to $+0.6\%$ mAP when combined with ActivityNet features and $+0.5\%$ mAP when used with Kinetics-400 features with shallower models enjoying the largest improvements.

4.2 Audio Classification

4.2.1 Background

Audio Classification is the categorisation of audio waveforms. Though waveform sequences can be used directly (Lee et al., 2017), it is common to first convert them to spectrograms. Mel spectrograms are obtained by a nonlinear transformation of a frequency scale (Stevens et al., 1937), which is designed based on empirical knowledge about the human auditory system (Choi et al., 2016). By employing spectrograms, audio classification can be approached in the same way as image classification (Palanisamy et al., 2020).

Table 3: **Audio-Visual** result, THUMOS14.

Model	Feat.	mAP (%)	FLOPs (M)
OadTR		**57.6**$_{\pm 0.6}$	2714.9
OadTR-b2	A.Net	***57.5***$_{\pm 0.5}$	1277.0
OadTR-b1	+	57.4$_{\pm 0.4}$	874.1
*Co*OadTR-b2	AudioSet	56.5$_{\pm 1.1}$	***415.0***
*Co*OadTR-b1		56.8$_{\pm 0.5}$	**13.8**
OadTR		64.4$_{\pm 0.4}$	2781.9
OadTR-b2	Kin.	**65.0**$_{\pm 0.4}$	1344.1
OadTR-b1	+	64.5$_{\pm 0.4}$	941.2
*Co*OadTR-b2	AudioSet	64.7$_{\pm 0.8}$	***416.0***
*Co*OadTR-b1		***64.8***$_{\pm 0.3}$	**14.8**

Table 4: **Audio Classification** results for GTZAN.

Method	Pos. Enc.	Acc. (%)	FLOPs (M)	Par. (K)
Maj. Voting	-	92.0	-	**0**
Trans-b2	learned	**95.0**$_{\pm 0.6}$	47.4	509
Trans-b1	learned	93.8$_{\pm 0.8}$	15.2	***286***
*Co*Trans-b2	fixed	***94.4***$_{\pm 1.0}$	27.0	509
*Co*Trans-b1	learned	93.2$_{\pm 1.1}$	***0.3***	***286***

4.2.2 EXPERIMENTS

We conduct experiments on the Music Genre Classification dataset GTZAN (Tzanetakis & Cook, 2002). It consists of 100 30-second clips for each of ten music genres. Each audio clip is sampled at 22,050 Hz. Since there are no predefined splits for GTZAN, we randomly select 10% of the data for validation and 10% for testing. The input is transformed to a temporal sequence by sliding a one-second window over each 30-second clip with a slide step size of 250ms, leading to 120 one-second clips. These are subsequently converted to Mel spectrograms. We then fine-tune a VGGish network, pre-trained on AudioSet (Hershey et al., 2017) and use the penultimate layer for feature extraction. A batch size of 64 and the Adam optimizer (Kingma & Ba, 2015) are used with an initial learning rate of 10^{-4}. The learning rate is reduced by a factor of 0.6 on plateau with a tolerance of two epochs, and an early stopping mechanism, where a maximum of 100 epochs are allowed. The VGGIsh base-network attains an accuracy of 86.1% on the dataset of one-second clips with 72.1M parameters and 864.7M FLOPs. Subsequently, the audio features are passed to a (Continual) Transformer Encoder which has 16 attention heads, an embedding dimension of 192 and an MLP dimension of 384. The Transformer Encoder is trained on the whole temporal sequence using a batch size of 32 and the AdamW optimizer (Loshchilov & Hutter, 2019) with a learning rate of 10^{-5} and a weight decay of 10^{-4} for 50 epochs. Since the Transformer Encoder is trained on entire 30-second clips, there are less data points available for this training. Accordingly, the size of the validation set is increased to 18%. All audio classification training procedures were carried out on a single Nvidia RTX 2080 Ti GPU. Table 4 presents the accuracy and efficiency of regular and Continual Transformers during online inference. As a baseline, we also include the result of majority voting among the clips to classify the entire sequence. The Continual Transformers obtain similar accuracy as regular a Transformers while consuming $1.76\times$ less FLOPs when using two blocks and $51.5\times$ less FLOPs when using one Transformer Encoder block.

5 CONCLUSION

In this work, we presented Continual Transformers, a redundancy-free reformulation of Transformers tailored for online inference. Central to the Continual Transformer are the Continual Retroactive and Single-Output Attention operations, which produce outputs identical to the original Scaled Dot-Product Attention for continual input sequences, while greatly reducing the time and memory complexity per prediction. The applicability of Continual Transformer architectures was experimentally validated in Online Action Detection and Online Audio Classification settings, observing upwards of multiple orders of magnitude reduction in time complexity for lightweight architectures at modest accuracy concessions. Continual Transformers constitute an algorithmic innovation, which could make possible hitherto unseen precision, speed, and power efficiency in online inference use-cases. With applications spanning enhanced perception and reactivity of robots and autonomous vehicles, weather forecasting, price prediction and surveillance, we hope it will be used for the common good.

ACKNOWLEDGMENTS

This work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 871449 (OpenDR).

REFERENCES

Anurag Arnab, Mostafa Dehghani, Georg Heigold, Chen Sun, Mario Lucic, and Cordelia Schmid. Vivit: A video vision transformer. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

Arian Bakhtiarnia, Qi Zhang, and Alexandros Iosifidis. Single-layer vision transformers for more accurate early exits with less overhead. *preprint, arXiv:2105.09121*, 2021a.

Arian Bakhtiarnia, Qi Zhang, and Alexandros Iosifidis. Multi-exit vision transformer for dynamic inference. *British Machine Vision Conference (BMVC)*, 2021b.

Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv:2004.05150*, 2020.

Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm (eds.), *European Conference on Computer Vision (ECCV)*, pp. 213–229, 2020.

J. Carreira and A. Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4724–4733, 2017.

Keunwoo Choi, George Fazekas, and Mark Sandler. Automatic tagging using deep convolutional neural networks. *International Society of Music Information Retrieval Conference (ISMIR)*, 2016.

Krzysztof Marcin Choromanski, Valerii Likhosherstov, 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.

Kateryna Chumachenko, Alexandros Iosifidis, and Moncef Gabbouj. Self-attention fusion for audiovisual emotion recognition with incomplete data. *arXiv preprint arXiv:2201.11095*, 2022.

MMAction2 Contributors. Openmmlab's next generation video understanding toolbox and benchmark. `https://github.com/open-mmlab/mmaction2`. Apache 2.0 License., 2020.

Marcella Cornia, Lorenzo Baraldi, and Rita Cucchiara. Smart: Training shallow memory-aware transformers for robotic explainability. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1128–1134, 2020. doi: 10.1109/ICRA40945.2020.9196653.

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*, pp. 2978–2988. Association for Computational Linguistics, July 2019. doi: 10.18653/v1/P19-1285.

Roeland De Geest, Efstratios Gavves, Amir Ghodrati, Zhenyang Li, Cees Snoek, and Tinne Tuytelaars. Online action detection. In *European Conference on Computer Vision (ECCV)*, pp. 269–284, 2016.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. Universal transformers. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=HyzdRiR9Y7`.

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)*, pp. 4171–4186. Association for Computational Linguistics, June 2019. doi: 10.18653/v1/N19-1423.

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 (ICLR)*, 2021.

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)*, pp. 806–815, 2020.

Hyunjun Eun, Jinyoung Moon, Jongyoul Park, Chanho Jung, and Changick Kim. Temporal filtering networks for online action detection. *Pattern Recognition*, 111:107695, 2021.

Jiyang Gao, Zhenheng Yang, and Ram Nevatia. RED: reinforced encoder-decoder networks for action anticipation. In *British Machine Vision Conference (BMVC)*, 2017.

Yuan Gong, Yu-An Chung, and James Glass. AST: Audio Spectrogram Transformer. In *Proc. Interspeech 2021*, pp. 571–575, 2021. doi: 10.21437/Interspeech.2021-698.

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)*. IEEE, June 2016. doi: 10.1109/cvpr.2016.90.

Lukas Hedegaard and Alexandros Iosifidis. Continual 3d convolutional neural networks for real-time processing of videos. In *European Conference on Computer Vision (ECCV)*, 2021. Apache 2.0 Licence.

Lukas Hedegaard, Negar Heidari, and Alexandros Iosifidis. Online skeleton-based action recognition with continual spatio-temporal graph convolutional networks. *preprint, arXiv: 2203.11009*, 2022. Apache 2.0 Licence.

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)*, pp. 3220–3224, 2021.

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)*, pp. 961–970, 2015. doi: 10.1109/CVPR.2015.7298698.

Shawn Hershey, Sourish Chaudhuri, Daniel P. W. Ellis, Jort F. Gemmeke, Aren Jansen, R. Channing Moore, Manoj Plakal, Devin Platt, Rif A. Saurous, Bryan Seybold, Malcolm Slaney, Ron J. Weiss, and Kevin Wilson. CNN architectures for large-scale audio classification. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, March 2017. doi: 10.1109/icassp.2017.7952132.

Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Ian Simon, Curtis Hawthorne, Noam M. Shazeer, Andrew M. Dai, Matthew D. Hoffman, Monica Dinculescu, and Douglas Eck. Music transformer: Generating music with long-term structure. In *International Conference on Learning Representations (ICLR)*, 2019.

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.

Shuiwang Ji, Wei Xu, Ming Yang, and Kai Yu. 3d convolutional neural networks for human action recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(1):221–231, 2013. doi: 10.1109/TPAMI.2012.59.

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.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun (eds.), *International Conference on Learning Representations (ICLR)*, 2015.

Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. In *International Conference on Learning Representations (ICLR)*, 2020.

Shun Kiyono, Sosuke Kobayashi, Jun Suzuki, and Kentaro Inui. SHAPE: Shifted absolute position embedding for transformers. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 3309–3321, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.266.

Jongpil Lee, Taejun Kim, Jiyoung Park, and Juhan Nam. Raw waveform-based audio classification using sample-level cnn architectures. *NIPS, Machine Learning for Audio Signal Processing Workshop (ML4Audio)*, 2017.

Tatiana Likhomanenko, Qiantong Xu, Gabriel Synnaeve, Ronan Collobert, and Alex Rogozhnikov. Cape: Encoding relative positions with continuous augmented positional embeddings. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 16079–16092. Curran Associates, Inc., 2021.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*, 2019.

Yining Ma, Jingwen Li, Zhiguang Cao, Wen Song, Le Zhang, Zhenghua Chen, and Jing Tang. Learning to iteratively solve routing problems with dual-aspect collaborative transformer. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, 2021.

Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, pp. 48–53, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-4009.

Kamalesh Palanisamy, Dipika Singhania, and Angela Yao. Rethinking cnn models for audio classification. *arXiv:2007.11154*, 2020.

Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. Image transformer. In Jennifer Dy and Andreas Krause (eds.), *International Conference on Machine Learning (ICML)*, volume 80 of *Proceedings of Machine Learning Research*, pp. 4055–4064. PMLR, 7 2018.

Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations. In *North American Chapter of the Association for Computational Linguistics (NAACL)*, 2018.

Zheng Shou, Dongang Wang, and Shih-Fu Chang. Temporal action localization in untrimmed videos via multi-stage cnns. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1049–1058, 2016. doi: 10.1109/CVPR.2016.119.

Zheng Shou, Jonathan Chan, Alireza Zareian, Kazuyuki Miyazawa, and Shih-Fu Chang. Cdc: Convolutional-de-convolutional networks for precise temporal action localization in untrimmed videos. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1417–1426, 2017. doi: 10.1109/CVPR.2017.155.

S. S. Stevens, J. Volkmann, and E. B. Newman. A scale for the measurement of the psychological magnitude pitch. *The Journal of the Acoustical Society of America*, 8(3):185–190, 1937. doi: 10.1121/1.1915893.

Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. Efficient transformers: A survey. *arXiv:2009.06732*, 2020.

Hugo Touvron, Matthieu Cord, Alaaeldin El-Nouby, Piotr Bojanowski, Armand Joulin, Gabriel Synnaeve, and Hervé Jégou. Augmenting convolutional networks with attention-based aggregation. *preprint, arXiv:2112.13692*, abs/2112.13692, 2021.

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.

George Tzanetakis, Georg Essl, and Perry Cook. Automatic musical genre classification of audio signals, 2001. URL http://ismir2001.ismir.net/pdf/tzanetakis.pdf.

Gül Varol, Ivan Laptev, and Cordelia Schmid. Long-term temporal convolutions for action recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(6):1510–1517, 2018. doi: 10.1109/TPAMI.2017.2712608.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, pp. 5998–6008, 2017.

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.

Sinong Wang, Belinda Z. Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *arXiv:2006.04768*, 2020.

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.

Chao-Yuan Wu, Christoph Feichtenhofer, Haoqi Fan, Kaiming He, Philipp Krähenbühl, and Ross Girshick. Long-term feature banks for detailed video understanding. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 284–293, 2019. doi: 10.1109/CVPR.2019.00037.

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*, 2021.

Huijuan Xu, Abir Das, and Kate Saenko. R-c3d: Region convolutional 3d network for temporal activity detection. In *IEEE International Conference on Computer Vision (ICCV)*, pp. 5794–5803, 2017. doi: 10.1109/ICCV.2017.617.

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)*, pp. 5531–5540, 2019. doi: 10.1109/ICCV.2019.00563. URL https://github.com/xumingze0308/TRN.pytorch. MIT Licence.

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.

Sijie Yan, Yuanjun Xiong, and Dahua Lin. Spatial temporal graph convolutional networks for skeleton-based action recognition. In *AAAI Conference on Artificial Intelligence*, pp. 7444–7452, 2018.

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.

A Appendix

A.1 Scaling properties of Continual and regular Multi-head Attention

A detailed account for the floating point operations involved in computing Regular-, Continual Retroactive-, and Single-output Scaled Product Attentions is given in Tables 5, 6, and 7.

Fig. 3 illustrates the scaling of FLOPs and memory footprint with increasing sequence length n and embedding dimension d. Here, the Continual Retroactive and Single-Output SDAs spend significantly less FLOPs than the Regular SDA, which scales $\mathcal{O}(n^2)$ as opposed to $\mathcal{O}(nd)$ the continual variants. The Continual Single-Output SDA reduces memory footprint for all value combinations, and the Continual Retroactive SDA does so when $n \gtrapprox d$.

Table 5: **Floating Point Operations** for the Scaled Dot-Product Attention in Eq. (1). $\boldsymbol{D}^{-1}(\cdot)$ can be efficiently computed as element-wise multiplication with $\boldsymbol{AV}$.

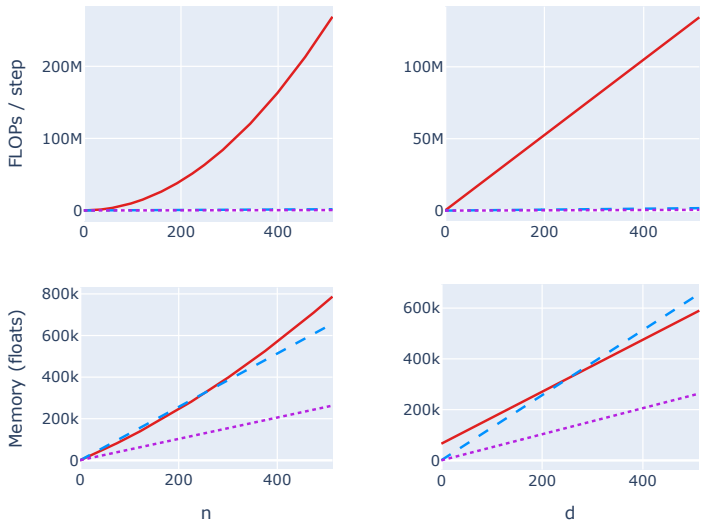
	Mul.	Add	Exp
Eq. (1.1)	$n^2d + nd$	$nd(n-1)$	0
Eq. (1.2)	$n^2d + nd$	$n^2(d-1)$	n^2
Eq. (1.3)	0	$n(n-1)$	0

Table 6: **Floating Point Operations** for the Continual Retroactive Dot-Product Attention in Eqs. (2) to (6). The outputs of the exponentials in Eq. (2) and Eq. (3) can be reused in Eq. (4) and Eq. (5) respectively, and are omitted in the count.

	Mul.	Add	Exp
Eq. (2)	$2(n-1)d$	$2(n-2)d + 2(n-1)$	$2(n-1)$
Eq. (3)	$nd + n + d$	$nd + (n-1) + d$	n
Eq. (4)	$2(n-1)d$	$2(n-1)d$	0
Eq. (5)	nd	$(n-1)d$	0
Eq. (6)	$nd + n$	0	0

Table 7: **Floating Point Operations** for the Continual Single-Output SDA in Eq. (7).

	Mul.	Add	Exp
Eq. (7.1)	$nd + d$	$(n-1)d + n - 1$	0
Eq. (7.2)	$nd + d$	$n(d-1)$	n



(a) $d = 256$ (b) $n = 256$

Figure 3: **FLOPs/step and memory footprint** for Regular, Continual Retroactive, and Continual Single-Output Scaled Dot-Product Attention at varying sequence length n and embedding dimension d. Column (a) has d fixed to 256; Column (b) has n fixed to 256.

A.2 SUPPLEMENTAL VISUALISATIONS

For the visually inclined, we supply a complementary graphical depictions of the Continual Retroactive SDA corresponding to Eqs. (2) to (6) in Fig. 4 and the Single-Output SDA in Eq. (7) in Fig. 5.

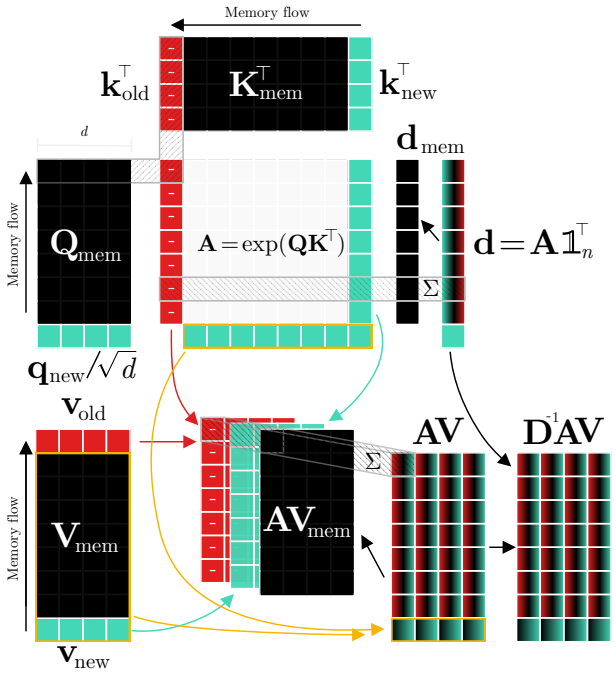


Figure 4: **Continual Retroactive Dot-Product Attention**. The query ($\boldsymbol{Q}$), key ($\boldsymbol{K}$), and value ($\boldsymbol{V}$) matrices are aggregated over time by caching the step vectors $\boldsymbol{q}_{\text{new}}$, $\boldsymbol{k}_{\text{new}}$, and $\boldsymbol{v}_{\text{new}}$ in each their FIFO queue (denoted by $\square_{\text{mem}}$). During each step, only the entries of $\boldsymbol{A}$ associated with $\boldsymbol{q}_{\text{new}}$, $\boldsymbol{k}_{\text{new}}$ and the oldest $\boldsymbol{K}$ step, $\boldsymbol{k}_{\text{old}}$ are computed. The diagonal entries of the row-normalisation matrix $\boldsymbol{D}$ as well as the $\boldsymbol{AV}$ can be updated retroactively by subtracting features corresponding to $\boldsymbol{k}_{\text{old}}$ and adding features related to $\boldsymbol{k}_{\text{new}}$ to the cached outputs of the previous step, $\boldsymbol{D}_{\text{mem}}$ and $\boldsymbol{AV}_{\text{mem}}$, respectively.

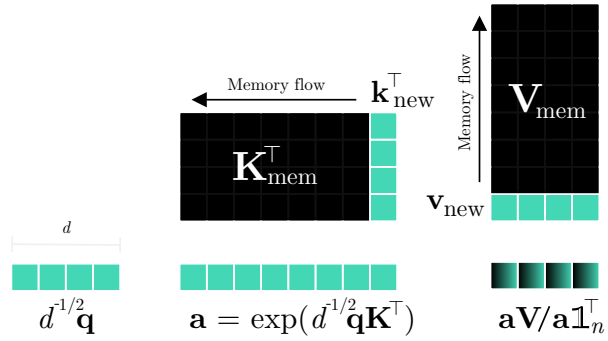


Figure 5: **Continual Single-Output Dot-Product Attention**. The key ($\boldsymbol{K}$) and value ($\boldsymbol{V}$) matrices are aggregated over time by caching the step vectors $\boldsymbol{k}_{\text{new}}$ and $\boldsymbol{v}_{\text{new}}$ in a FIFO queue. During each step, only the attention output associated with $\boldsymbol{q}$ is computed.

A schematic illustration of the Audio Classification experiments architecture is depicted in Fig. 6.

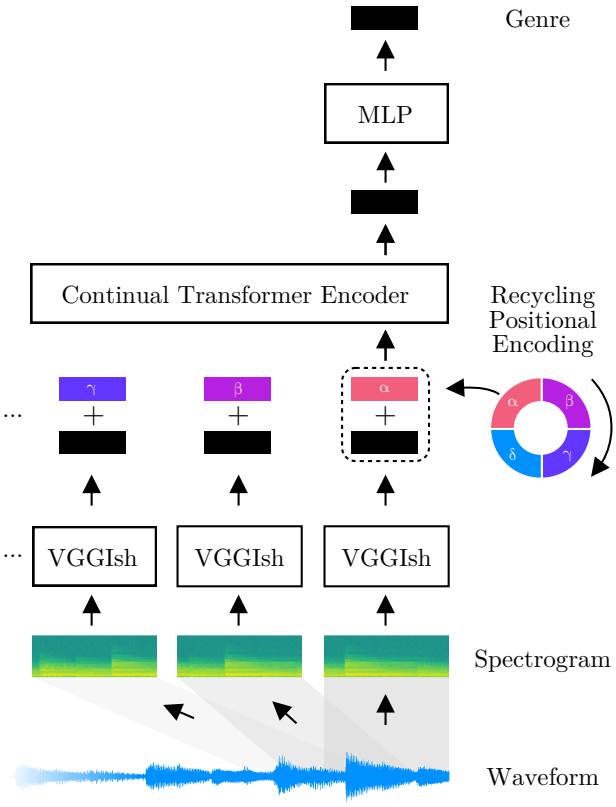


Figure 6: **Audio Classification Architecture**.

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.

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

Lukas Hedegaard and Alexandros Iosifidis

Department of Electrical and Computer Engineering, Aarhus University, Denmark
{lhm,ai}@ece.au.dk

Abstract. We present *Continual Inference*, a Python library for implementing Continual Inference Networks (CINs), a class of Neural Networks designed for redundancy-free online inference. This paper offers a comprehensive introduction and guide to CINs and their implementation, as well as best-practices and code examples for composing basic modules into complex neural network architectures that perform online inference with an order of magnitude less floating-point operations than their non-CIN counterparts. Continual Inference provides drop-in replacements of PyTorch modules and is readily downloadable via the Python Package Index and at www.github.com/lukashedegaard/continual-inference.

Keywords: Online Inference, Continual Inference Network, Deep Neural Network, Python, PyTorch, Library.

1 Introduction

Designing and implementing Deep Neural Networks, which offer good performance in online inference scenarios, is an important but overlooked discipline in Deep Learning and Computer Vision. Research in areas such as Human Activity Recognition focuses heavily on improving accuracy on select benchmark datasets with limited focus on computational complexity and still less on efficient online inference capabilities. Yet, important real-life applications such as human monitoring [13], [18], driver assistance [3], and autonomous vehicles depend on performing predictions on a continual input stream with low latency and low energy consumption.

Continual Inference Networks (CINs) [9], [8], [7], are a recent family of Deep Neural Networks, which can accelerate a wide range of architectures for time-series processing (e.g., CNNs and Transformers) during online inference, even though source networks may have been trained exclusively for offline processing.

This paper provides comprehensive introduction to CINs (Sec. 2), the guiding principles of their design and implementation via the Continual Inference library (Sec. 4), and summarizes and compares achieved reductions in stepwise computational complexity and memory-usage using the library (Sec. 4).

2 Continual Inference Networks

Originally introduced in [9] and subsequently elaborated in [7], [8], Continual Inference Networks denote a variety of Neural Network, which can operate without redundancy during online inference on a continual input stream, as well as offline during batch inference. Specifically, CINs comply with Def. 1 [7]:

Definition 1 (Continual Inference Network). *A Continual Inference Network is a Deep Neural Network, which*

- *is capable of continual step inference without computational redundancy,*
- *is capable of batch inference corresponding to a non-continual Neural Network,*
- *produces identical outputs for batch inference and step inference given identical receptive fields,*
- *uses one set of trainable parameters for both batch and step inference.*

Many prior networks can be viewed as CINs. This includes networks, which perform their task within a single time-step (e.g., object detection and image recognition models), or which inherently process temporal data step-by-step (e.g., Recurrent Neural Networks such as LSTMs [10] and GRUs [2]). Some network types, however, are limited to batch inference exclusively. These include Convolutional Neural Networks (CNNs) with temporal convolutional components (e.g., 3D CNNs), as well as Transformers with tokens spanning the temporal dimension. While they can in principle be used for online inference, it is an inefficient process, where input steps are assembled to full (spatio-)temporal batches and fed to the network in a sliding window fashion, with many redundant intermediary computations as a result.

While some specialty architectures have been devised to let 3D convolutional network variants make predictions step by step [16], [11], and accordingly also qualify as CINs, these were not weight-compatible with regular 3D CNNs. Recently, *Continual* 3D Convolutions [9] changed this. Through a reformulation of the 3D convolution to compute outputs for each time-step individually rather than for the whole spatio-temporal input at once, well-performing 3D CNNs such as X3D [4], Slow [5], and I3D [1] trained for Trimmed Activity Recognition were re-implemented to execute step by step without any re-training. Likewise, Spatio-temporal Graph Convolutional Networks for Skeleton-based Action Recognition [20], [15], [14], which originally operated only on complete sequences of skeleton graphs, were transformed to perform stepwise inference as well though a continual formulation of their Spatio-temporal Graph Convolution blocks [8]. Temporal Transformer networks had also been restricted to operate on batches until a *Continual* Multi-head Attention (*Co*MHA) [7] was introduced, which is weight-compatible with the original MHA [19], while being able to compute updated outputs for each time step.

With these innovations, many existing DNNs can be converted to operate efficiently during online inference. In general, non-continual networks, which are transformed to continual ones attain reductions in per-step computational complexity in proportion to the temporal receptive field of the network. In some

cases, these savings can amount to multiple orders of magnitude [8]. Still, the implementation of Continual Inference Networks with temporal convolutions and Multi-head Attention in frameworks such as PyTorch [12] requires deep knowledge and practical experience with CINs. With the Continual Inference library described in the next section, we hope to change this.

3 Library Design

3.1 Principles

The fundamental feature of CINs, that networks are flexible and perform well on both online inference and batch inference, is a guiding principle in the design of the Continual Inference library as well: Refactoring of existing implementations in pure PyTorch should be straightforward. In the following, we will adopt the Python import abbreviations `import continual as co` and `from torch import nn`. The library follows Principle 1 to ensure that `co` modules can be used as drop-in replacements for `nn` modules without behavior change:

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.*

Before proceeding to the enhanced functionality of `co` modules, let us state our assumption to the input format:

Assumption 1 (Order of input dimensions) *Inputs to* `co` *modules use the order* $(B, C, T, S_1, S_2, ...)$ *for multi-step inputs and* $(B, C, S_1, S_2, ...)$ *for single-step inputs, where* B *is the batch size,* C *is the input channel size,* T *is the temporal size, and* S_n *are additional optional dimensions.*

The core difference between Continual Inference Networks and regular networks is their ability to efficiently compute results for each time-step. Besides the regular `forward` function found in `nn` modules, `co` modules add multiple call modes that allow for continual inference with a simple interface:

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 it's 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.*

Furthermore, the `__call__` *method of* `co` *modules can be changed to use any of the three by either setting the* `call_mode` *attribute of the module or applying the* `co.call_mode()` *context with a string spelling out the wanted forward type.*

Let us exemplify Principle 2 in practice. Example 1.1 shows how the different forward functions introduced in Principle 2.1 can be used. Principle 2.2 is illustrated in Example 1.2.

```python
import torch
import continual as co

con = co.Conv3d(in_channels=4,
                out_channels=8,
                kernel_size=3)
assert con.delay == 2
assert con.receptive_field == 3

reg = torch.nn.Conv3d(in_channels=4,
                      out_channels=8,
                      kernel_size=3)
# Reuse weights
con.load_state_dict(reg.state_dict())

x = torch.randn((2, 3, 5, 6, 7))  # B,C,T,H,W
y = con.forward(x)
assert torch.equal(y, reg.forward(x))

# Multiple steps
firsts = con.forward_steps(x[:, :, :4])
assert torch.allclose(firsts, y[:, :, : con.delay])

# Single step
last = con.forward_step(x[:, :, 4])
assert torch.allclose(last, y[:, :, con.delay])
```

Example 1.1: Definition and usage of `co.Conv3d` and its forward modes.

```python
net(x)            # Invokes ‘forward‘ by default

net.call_mode = "forward_step"
net(x[:, :, 0])   # Invokes ‘forward_step‘

with co.call_mode("forward_steps"):
    net(x)        # Invokes ‘forward_steps‘

net(x[:, :, 0])   # Invokes ‘forward_step‘ again
```

Example 1.2: Changing the `call_mode` for a continual module `net`.

Continual modules, which use information from multiple time-steps, are inherently stateful. Whenever `forward_step` or `forward_steps` is invoked, intermediary results needed for future step results are optimistically computed and stored. Principle 3 states the rules for state-manipulation and updates.

Principle 3 (State) *Module state is updated according to the following rules:*
- `forward_step` *and* `forward_steps` *use and update state by default.*
- *Step results may be computed without updating internal state by passing* `update_state=False` *to either* `forward_step` *or* `forward_steps`*.*
- `forward` *neither uses nor updates state.*
- *Module state can be wiped by invoking the* `clean_state()` *method.*
- *A module produces non-empty outputs after its has conducted a number of stateful forwards steps corresponding to its* `delay`*.*

Regular `nn` modules predominantly operate on input batches in an offline setting and do not have a built-in concept of delay. `co` modules on the other hand are designed to operate on time-series. Since `co` modules often integrate information over multiple time-steps and online operation is causal by nature, some modules produce the output corresponding to a given input only after observing additional steps. For instance, a `co.Conv1d` module with `kernel_size = 3` produces an output from the third input step as illustrated in Fig. 1. The delay of a module is calculated according to Principle 4:

Principle 4 (Delay) `co` *modules produce step outputs that are delayed by*

$$d = f - p - 1 \tag{1}$$

steps relative to the earliest input step used in the computation, where f is the receptive field and p is the temporal padding.

While padding is used in regular networks to retain the size of feature-maps in consecutive layers, this interpretation of temporal padding does not make sense in the context of an infinite, continual input, as handled by CINs. Instead, we may interpret padding as a reduction in delay. For instance, a `co.Conv1d` module with `kernel_size = 3` and `padding = 2` has a delay of zero, because the padded zeros already "saturated" the state before-hand. This is illustrated in Fig. 2. Considering, that `co` modules expect an infinite and continual input stream, end-padding padding is omitted by default. If an end-padding is required, the library supports its use by either passing manually defined zeros as steps or by setting `pad_end = True` for an invocation of the `forward_steps` function.

Similar to padding, the stride of a `co` module impacts the timing of the outputs. Specifically, stride results in empty outputs every $(s-1)/s$ outputs, as well as larger delays for downstream network modules through increased receptive fields. This is stated in Principles 5 and 6.

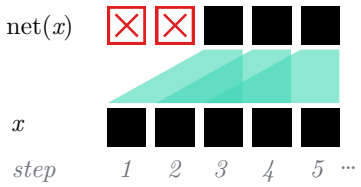


Fig. 1: Sketch of delay and receptive field. Here, the stepwise operation of a `co` module `net` with `receptive_field = 3` is illustrated. ■ are non-zero step-features and ☒ are empty outputs.

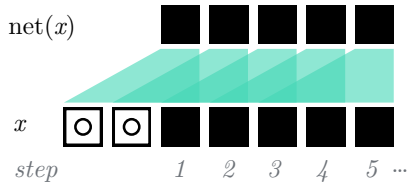


Fig. 2: Sketch of how padding reduces delay. Here, the stepwise operation of a `co` module `net` with `receptive_field = 3`, `padding = 2` is illustrated. ⊡ are padded zeros and ■ are non-zero step-features.

Principle 5 (Stride and prediction rate) *For neural network of N modules with strides $s^{(i)}, i \in \{1..N\}$, the accumulated stride at any given layer is*

$$s_{acc}^{(i)} = s^{(i)} \cdot s_{acc}^{(i-1)} \quad i \in 1..N \tag{2}$$

$$s_{acc}^{(0)} = s^{(0)}. \tag{3}$$

Equivalently, the resulting network stride is

$$s_{NN} = \prod_{i=1}^{N} s^{(i)}, \tag{4}$$

and the network prediction rate is

$$r_{NN} = 1/s_{NN}. \tag{5}$$

Accordingly, the outputs of a `co` *network are empty every $(s_{NN} - 1)/s_{NN}$ steps.*

Principle 6 (Accumulated delay) *The accumulated receptive field of a downstream module i in a network of N modules is given by:*

$$f_{acc}^{(i)} = f^{(i)} + (f_{acc}^{(i-1)} - 1)s^{(i)}, \quad i \in 1..N \tag{6}$$

$$f_{acc}^{(0)} = f^{(0)}. \tag{7}$$

The accumulated delay of layer i in a network is

$$d^{(i)} = f_{acc}^{(i)} - p_{acc}^{(i)} - 1, \tag{8}$$

where the accumulated padding p_{acc} is given by

$$p_{acc}^{(i)} = p^{(i)} \cdot s_{acc}^{(i-1)}, \quad i \in 1..N, \tag{9}$$

$$p_{acc}^{(0)} = p^{(0)}. \tag{10}$$

Fig. 3 illustrates a mixed example, where the first layer of a two-layer network has `padding = 2` and `stride = 2`. Noting layer attributes in consecutive order, and using Equations 2 to 10, the example has the following network attributes:

$$\begin{aligned}
s &= \{2, \quad 1\} \\
p &= \{2, \quad 0\} \\
s_{acc} &= \{2, \quad 2 \cdot 1 = 2\} \\
p_{acc} &= \{2, \quad 2 + 2 \cdot 0 = 2\} \\
f_{acc} &= \{3, \quad 3 + (3-1) \cdot 2 = 7\} \\
d_{acc} &= \{3 - 2 - 1 = 0, \quad 7 - 2 - 1 = 4\} \\
s_{NN} &= s_{acc}^{(1)} = 2 \\
r_{NN} &= 1/s_{NN} = 1/2 \\
d_{NN} &= d_{acc}^{(1)} = 4.
\end{aligned}$$

Before continuing onto the specific modules, we have to discuss a final principle of CINs, namely that of parallel modules.

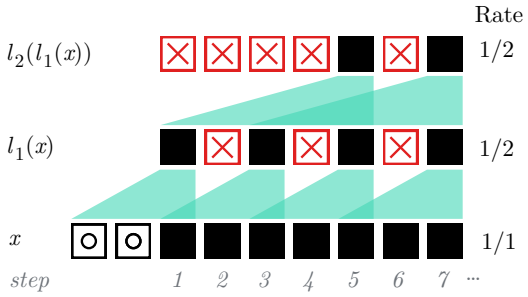


Fig. 3: A mixed example of delay and outputs under padding and stride. Here, we illustrate the stepwise operation of two `co` module layers, l_1 with with `receptive_field = 3`, `padding = 2`, and `stride = 2` and l_2 with `receptive_field = 3`, no padding and `stride = 1`. ⊡ denotes a padded zero, ■ is a non-zero step-feature, and ☒ is an empty output.

Principle 7 (Parallel modules) *Modules can be arranged in parallel to execute on each their separate stream of data under the following rules:*
- *Parallel modules follow the same global clock.*
- *The delay of a collection of parallel modules is the maximum delay of any module in the collection.*
- *If the merger of parallel step values includes an empty value, then the resulting step output of the merger is also empty.*

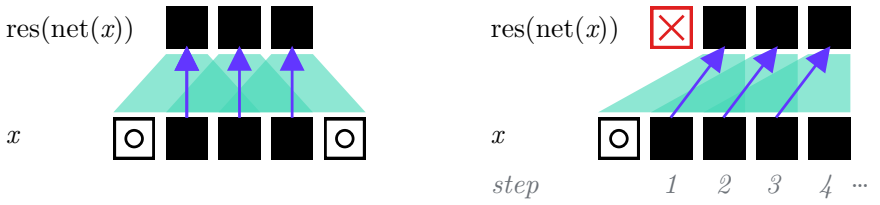
A discussion of residual connections provides a practical example for Principle 7.

Residual connections The residual connection is a simple but crucial tool for avoiding vanishing and exploding gradients; by adding the input of a module to its output, gradients can flow freely through models with hundreds of layers. Without exaggeration, we can state that almost all recent deep architectures at the time of writing use some form of residual connection [6], [19], [20], [5]. Yet, their implementation in Continual Inference Networks may not follow common intuition in all cases. Let us first consider the residual connection during regular `forward` operation as found in a non-continual residual shown in Fig. 4a. Here, the wrapped module will almost always use padding to ensure equal input and output shapes (known as "equal padding"). For a module with receptive field three, we would thus have a padding of one. In this case, the `forward` computation of the residual amounts to adding the input to the output of the convolution. However, the implementation of `forward_step` illustrated in Fig. 4b is different. Since the first output uses information from the second step, the module has a delay of one. Accordingly, the residual connection requires a delay of one as well.

Now consider the same scenario but without padding. This will be quite foreign to many Deep Learning practitioners, and it is not clear how exactly to align residuals. We will use a separate module to shrink the residual by an equivalent amount as the wrapped module. Of the possible alignment choices, a sensible approach is to discard the border values to align the feature maps on *center*. Contrary to other alignment forms, this has the benefit of weight-compatibility between the no-padding case and the case with equal padding described in the former paragraph. The outputs of step 3 in Figures 4b and 5b are equal given the same weights and inputs. However, two issues arise:

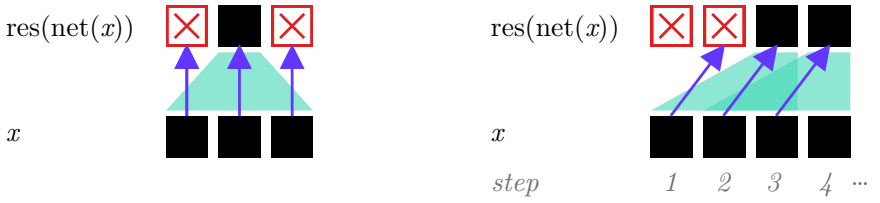
1. Delay mismatch: While the residual connection has a delay of one, the wrapped module has a delay of two.
2. Mix of empty and non-empty results: C.f. the differences in delay, the residual will start producing non-empty outputs before the wrapped module.

Principle 7 helps us navigate this. Despite the internal delay mismatch, the delay of the whole residual module corresponds to the largest delay, in this case two. Consequently, the whole residual module produces outputs from the third step, despite the fact that the delayed input already has non-empty outputs from the second step. Both of these issues can also be avoided if we force residuals to employ the same delay as the wrapped module. This corresponds to a *lagging* alignment. However, using such a strategy breaks weight compatibility between the same residual modules with and without padding.



(a) `forward` (b) `forward_step`

Fig. 4: Residual connections ↑ over a module with receptive field of size ▲ and padding one ("equal padding") ⧈. ☒ are empty outputs.



(a) `forward` (b) `forward_step`

Fig. 5: Centered residual connections ↑ over a module with receptive field of size ▲ and no padding. ☒ are empty outputs.

3.2 Core modules

Designed as an augmentation of PyTorch, the Continual Inference library provides a collection of basic building blocks for composing neural networks. Following Principle 1, we use the same public interfaces as PyTorch , i.e. class constructor, function names and arguments, and attribute names, to ensure that `co` modules can be used as drop-in replacements for `nn` modules. The basic modules can be categorized as follows:

- Convolutions [9]: `co.Conv1d`, `co.Conv2d`, ...
- Pooling: `co.AvgPool1d`, `co.MaxPool1d`, ...
- Linear: `co.Linear`.
- Transformer [7]: `co.TransformerEncoder`, ...
- Shape: `co.Delay`, `co.Reshape`.
- Arithmetic: `co.Lambda`, `co.Add`, ...

Here, the `MultiheadAttention` implementation is a special case, which features two distinct versions of continual operation: 1) "single-output", where only the attention output corresponding to the latest input is produced, and 2) "retrospective", where updates to prior outputs are also produced retrospectively. The details of this are explained in greater detail in [7]. Linear `co` modules follow the `nn` modules closely, but ensure compatibility of dimension c.f. Assumption 1. `co.Delay` adds a specified delay to the input stream. This is handy

Module	Description
`Sequential`	Arrange modules sequentially.
`Broadcast`	Broadcast one stream to multiple parallel streams.
`Parallel`	Apply modules in parallel, each on a separate stream.
`Reduce`	Reduce multiple input streams into one.
`Residual`	Add a residual connection for a wrapped module.
`Conditional`	Conditionally invoke a module (or another) at runtime.

Table 1: Composition modules.

for aligning the delay of multiple streams as required by residual connections (see Sec. 3.1). `co.Lambda` allows a user to pass in functions and functors that are applied stepwise to the inputs. Besides the above list of tailor-made modules, the Continual Inference library has interoperability with most activation functions (`nn.ReLU`, `nn.Softmax`, etc.), normalisation layers (`nn.BatchNorm1d`, `nn.LayerNorm`, etc.), and `nn.DropOut` when used within the composition modules as presented in Sec. 3.3. The full list of compatible modules can be found at `www.github.com/lukashedegaard/continual-inference`.

3.3 Composition modules

In PyTorch, modules are composed by either by using the `nn.Sequential` container or by creating a new class which inherits from `nn.Module` and manually controls data flow within the `forward` function. While the latter is commonly used to handle complex modules in a simple and easily debuggable manner, it is not necessarily the simplest approach for implementing complex Continual Inference Networks. In addition to defining the basic forward flow, a CIN implementation also needs to handle stepwise computations, which require meticulous alignment of delays if Principle 2 is to be kept.

Instead, we expand the container interface of PyTorch to include modules for parallel and conditional processing. While each module is simple in nature, they can be used to compose complex neural network architectures, which retain all the principles in Sec. 3.1 without explicitly needing to consider them. A brief overview and description of each `co` container module is given in Sec. 3.3. To get a practical understanding of these, we will give implementation examples of two common architecture blocks, the residual connection as discussed in Sec. 3.1 and an Inception module [17].

Example 1.3 shows three equivalent implementations of a residual 3D convolution block. `res1` is the verbose version, in which `co.Broadcast` is used to split a single input into two parallel stream, `co.Parallel` specifies that `conv` handles the first stream, while a delay is used on the second. `co.Reduce` merges the streams via an add reduce operation. Due to the commonality of broadcast-apply-reduce operations, the library features a `co.BroadcastReduce` shorthand

```
conv = co.Conv3d(1, 1, kernel_size=3, padding=1)

res1 = co.Sequential(co.Broadcast(2),
                     co.Parallel(conv, co.Delay(1)),
                     co.Reduce("sum"))

res2 = co.BroadcastReduce(conv, co.Delay(1))

res3 = co.Residual(conv)
```

Example 1.3: Equivalent implementations of a residual block.

to specify such composition more succinctly. Even shorter, `co.Residual` can automatically infer the needed delay from the module it wraps. Other reduction functions can be specified in `co.BroadcastReduce` and `co.Residual` using the `reduce` argument, which is `"sum"` by default. The code in Example 1.3 correspond to Fig. 4. The centered residual module in Fig. 5 is easily specified as `co.Residual(conv, residual_shrink=True)` where `conv` has `padding = 0`.

```
def norm_relu(conv):
    return co.Sequential(conv,
                         nn.BatchNorm3d(conv.out_channels),
                         nn.ReLU())

inception_module = co.BroadcastReduce(
    co.Conv3d(192, 64, 1),
    co.Sequential(
        norm_relu(co.Conv3d(192, 96, 1)),
        norm_relu(co.Conv3d(96, 128, 3, padding=1)),
    ),
    co.Sequential(
        norm_relu(co.Conv3d(192, 16, 1)),
        norm_relu(co.Conv3d(16, 32, 5, padding=2))
    ),
    co.Sequential(
        co.MaxPool3d(kernel_size=(1, 3, 3),
                     padding=(0, 1, 1),
                     stride=1),
        norm_relu(co.Conv3d(192, 32, 1)),
    ),
    reduce="concat",
)
```

Example 1.4: *Continual* Inception module using a mix of `co` and `nn` modules.

We can showcase a more advanced application of parallel streams by considering an Inception module [17]. An Inception module broadcasts the input into four streams and applies convolution of varying kernel sizes in parallel before concatenating the channels to produce one output. Without the `co` container modules, it would be complicated to keep track of and align delays of the different branches to create valid `forward`, `forward_step`, and `forward_steps` methods. Using `co.Sequential`, which automatically sums up delays, and `co.BroadcastReduce`, which automatically adds delays to match the branch with highest inherent delay, the implementation becomes simple as shown in Example 1.4.

Model	**Dataset performace** (%)		**Params** (M)	**Max mem.** (MB)	**FLOPs** (G)
	Kinetics-400 (Acc.)				
X3D-L	69.3		6.2	240.7	19.17
*Co*X3D-L$_{64}$	71.6 (+2.3)		6.2	184.4 (75%)	1.25 (↓ 15.34×)
X3D-M	67.2		3.8	126.3	4.97
*Co*X3D-M$_{64}$	71.0 (+3.8)		3.8	69.0 (55%)	0.33 (↓ 15.06×)
X3D-S	64.7		3.8	61.3	2.06
*Co*X3D-S$_{64}$	67.3 (+2.6)		3.8	42.0 (69%)	0.17 (↓ 12.12×)
Slow-8×8	67.4		32.5	266.0	54.87
*Co*Slow$_{64}$	73.1 (+5.7)		32.5	176.4 (66%)	6.90 (↓ 7.95×)
I3D	64.0		28.0	191.6	28.61
*Co*I3D$_8$	59.6 (−4.4)		28.0	235.9 (123%)	5.68 (↓ 5.04×)
	THUMOS14 (mAP)	**TVSeries** (mcAP)			
OadTR-b2	64.2	89.0	15.9	67.6	1.08
*Co*OadTR-b2	64.4 (+0.2)	88.2 (−0.8)	15.9	71.7 (106%)	0.41 (↓ 2.61×)
OadTR-b1	64.4	89.1	9.6	43.3	0.67
*Co*OadTR-b1	64.5 (+0.1)	88.0 (−1.1)	9.6	45.1 (104%)	0.01 (↓ 63.49×)
	NTU RGB+D 60 (Acc.)				
	X-Sub	*X-View*			
ST-GCN	86.0	93.4	3.1	45.3	16.73
*Co*ST-GCN*	86.3 (+0.3)	93.8 (+0.4)	3.1	36.1 (80%)	0.16 (↓ 107.7×)
AGCN	86.4	94.3	3.5	48.4	18.69
*Co*AGCN*	84.1 (−2.3)	92.6 (−1.7)	3.5	37.4 (77%)	0.17 (↓ 108.8×)
S-TR	86.8	93.8	3.1	74.2	16.14
*Co*S-TR*	86.3 (−0.3)	92.4 (−1.4)	3.1	36.1 (49%)	0.15 (↓ 107.6×)

Table 2: Dataset performance, parameter count, maximum allocated memory (Max mem.), and floating-point operations (FLOPs) of continual and non-continual models on video and spatio-temporal graph classification datasets. Subscript$_{xx}$ denotes expanded temporal average pooling, b1 and b2 denote one and two block transformer decoders, and superscript* indicates architectures where network stride was reduced to one. Parentheses show the improvement / deterioration of the continual model relative to the corresponding non-continual model. The noted metrics were originally presented in [9], [7], [8].

4 Performance comparisons

Using the basic `co` modules and composition building blocks, continual versions of advanced neural networks have been implemented in multiple recent works with manyfold speedups and significant reductions in memory consumption during online inference [9], [7], [8]. Specifically, the 3D-CNNs *Co*X3D, *Co*I3D, and *Co*Slow for video-based Human Activity Recognition were proposed in [9]; the Transformer *Co*OadTR for Online Action Detection in [7]; and Spatio-temporal Graph Convolutional Networks *Co*ST-GCN, *Co*AGCN, and *Co*S-TR for Skeleton-based Action Recognition in [8]. While direct conversion from regular to continual versions of the above noted architectures works well in accelerating inference in itself, further improvements can be achieved by exploiting some core characteristics of CINs: in [9], accuracy was improved by increasing model receptive fields through expansions of temporal global average pooling to 64 steps, and in [8], the stride of temporal convolutions was reduced to one to increase prediction rates. Tab. 2 presents a summary of benchmark performance, computational complexity, and maximum allocated memory on GPU for each of these networks alongside with their non-continual counterparts [9], [7], [8].

5 Conclusion

We presented Continual Inference, an easy-to-use library for implementing Continual Inference Networks in Python. Following interfaces closely, the components provided in the library are backwards-compatible drop-in replacements for PyTorch modules, which add the capability of redundancy-free online inference without the need for intimate knowledge of CINs nor their meticulous low-level implementation. Having shown the vast computational advantages of CINs over regular neural networks in multiple settings of video and spatio-temporal graph classification, we hope that this library will contribute to the adoption of CINs and the advancement of use-cases requiring low-latency online inference under recourse constraints in general.

Acknowledgement

This work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 871449 (OpenDR).

References

1. Carreira, J., Zisserman, A.: Quo vadis, action recognition? a new model and the kinetics dataset. In: IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 4724–4733 (2017)
2. Cho, K., van Merriënboer, B., Bahdanau, D., Bengio, Y.: On the properties of neural machine translation: Encoder–decoder approaches. In: Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation. pp. 103–111 (2014)
3. Enkelmann, W.: Video-based driver assistance–from basic functions to applications. International Journal of Computer Vision (IJCV) **45**(3), 201–221 (2001)
4. Feichtenhofer, C.: X3D: Expanding architectures for efficient video recognition. IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (2020)
5. Feichtenhofer, C., Fan, H., Malik, J., He, K.: Slowfast networks for video recognition. In: IEEE/CVF International Conference on Computer Vision (ICCV). pp. 6201–6210 (2019)
6. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 770–778 (2016)
7. Hedegaard, L., Bakhtiarnia, A., Iosifidis, A.: Continual Transformers: Redundancy-Free Attention for Online Inference. In: International Conference on Learning Representations (ICLR) (2023)
8. Hedegaard, L., Heidari, N., Iosifidis, A.: Online skeleton-based action recognition with continual spatio-temporal graph convolutional networks. preprint, arXiv: 2203.11009 (2022)
9. Hedegaard, L., Iosifidis, A.: Continual 3d convolutional neural networks for real-time processing of videos. In: European Conference on Computer Vision (ECCV). pp. 1–12 (2022)
10. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation **9**, 1735–80 (12 1997)
11. Köpüklü, O., Hörmann, S., Herzog, F., Cevikalp, H., Rigoll, G.: Dissected 3D CNNs: Temporal skip connections for efficient online video processing. preprint, arXiv:2009.14639 (2020)
12. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: Pytorch: An imperative style, high-performance deep learning library. In: Advances in Neural Information Processing Systems 32, pp. 8024–8035. Curran Associates, Inc. (2019)
13. Pigou, L., van den Oord, A., Dieleman, S., Van Herreweghe, M., Dambre, J.: Beyond temporal pooling: Recurrence and temporal convolutions for gesture recognition in video. International Journal of Computer Vision (IJCV) **126**(2), 430–439 (2018)
14. Plizzari, C., Cannici, M., Matteucci, M.: Skeleton-based action recognition via spatial and temporal transformer networks. Computer Vision and Image Understanding **208**, 103219 (2021)
15. Shi, L., Zhang, Y., Cheng, J., Lu, H.: Two-stream adaptive graph convolutional networks for skeleton-based action recognition. In: IEEE Conference on Computer Vision and Pattern Recognition. pp. 12026–12035 (2019)

16. Singh, G., Cuzzolin, F.: Recurrent convolutions for causal 3d cnns. In: IEEE/CVF International Conference on Computer Vision Workshop (ICCVW). pp. 1456–1465 (2019)
17. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 1–9 (2015)
18. Tavakolian, M., Hadid, A.: A spatiotemporal convolutional neural network for automatic pain intensity estimation from facial dynamics. International Journal of Computer Vision (IJCV) **127**(10), 1413–1425 (2019)
19. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L.u., Polosukhin, I.: Attention is all you need. In: Advances in Neural Information Processing Systems (NeurIPS). vol. 30, pp. 5998–6008 (2017)
20. Yan, S., Xiong, Y., Lin, D.: Spatial temporal graph convolutional networks for skeleton-based action recognition. In: AAAI Conference on Artificial Intelligence. pp. 7444–7452 (2018)

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.

Chapter 14

Human Activity Recognition

Lukas Hedegaard[1], Negar Heidari[2], and Alexandros Iosifidis[3]

ABSTRACT

With the increasing amount of videos available on the internet and in security settings, Human Activity Recognition has become a highly important topic in Machine Learning. In this chapter, we cover the topic of Deep Learning for Human Activity Recognition and the tasks of Trimmed Action Recognition, Temporal Action Localization, and Spatio-Temporal Action Localization. Throughout our treatise, we discuss different architectural building blocks including 2D Convolutional Neural Networks (CNNs), 3D-CNNs, Recursive Neural Networks (RNNs), and Spatial-Temporal Graph Convolution Networks (ST-GCNs), and how they were used in state-if-the-art models for Human Activity Recognition. Moreover, we discuss how multiple modalities and data resolutions can be fused via a multi-stream network topology, and how video-classification models can be extended for usage in (Spatio-)Temporal Action Localization. Finally, we provide a curated list of datasets for Human Activity Recognition tasks.

KEYWORDS

Human Activity Recognition, Video Classification, Spatio-Temporal Localization, 3D Convolutional Neural Network, Spatial-Temporal Graph Convolution Network

14.1 INTRODUCTION

The explosion of multimedia on the internet and the increased usage of video surveillance monitoring has lead to an increasing need to process and identify explicit and unwanted content online and in public spaces. Since it is unfeasible to assign human workers to watch through and label these massive amounts of content, it is of great interest to automate the process by means of machine learning. However, we cannot simply adopt image classification techniques to handle these problems, since the temporal information is crucial in the distinction of action classes. For instance, an image depicting a person performing the action "Standing Up" would not be easily distinguished by an image depicting the person performing the action "Sitting Down" as they will correspond to very similar human body poses, while image sequences depicting these actions would allow for easy distinction due to the difference in the temporal succession of the human

[1-3]Department of Electrical and Computer Engineering, Aarhus University, Denmark.

poses forming these two actions. To this end, we need methods that can exploit the temporal information in image sequences or videos.

In this chapter, we give an overview of the field of Human Activity Recognition, which (as the name implies) is concerned with recognizing the activity of humans. Here we should note that the terms action, activity and movement are often used interchangeably in the Computer Vision literature, though there have been attempts to distinguish the problems described by these terms [1]. Many different approaches to Human Activity Recognition have been proposed throughout the years, and in this exploration of the subject we focus exclusively on the methods that use Deep Learning and visual input data. First, we will define the tasks in Human Activity Recognition more precisely and give an overview of common input modalities used by the various methods. In Section 14.2 we cover different video classification architectures and methods for Trimmed Action Recognition (Section 14.2.1, Section 14.2.2, Section 14.2.3, and Section 14.2.4), human body pose-based methods using body skeleton data (Section 14.2.5), and multi-stream architectures (Section 14.2.6). Next, we consider the extensions to Temporal Action Localization (Section 14.3) and Spatio-temporal Action Localization (Section 14.4). Finally, we provide an overview of datasets used Human Activity Recognition (Section 14.5).

14.1.1 Tasks in Human Activity Recognition

Often times, "Human Activity Recognition" is understood as the narrow setting of classifying the human action present in a trimmed video. However, the term Human Activity Recognition also encompasses other tasks, where human activity is involved. Broadly speaking, the tasks are concerned with a mix of *what*, *when*, and *where*, expressed as an action class label, action time frames, and the spatial locations of the action in each frame (bounding boxes). These are illustrated in Fig. 14.1. There are many combinations of information to predict. What follows is an overview of the taxonomy[4] used to distinguish the most common tasks:

- **Trimmed Action Recognition** is only concerned with *what*, i.e. what is the class of the human activity depicted in a video, and assumes that the video has been trimmed beforehand to contain only that activity. Often, this task is simply referred to as "Action Recognition".
- **Temporal Action Proposals** determination is the task of estimating *when* any action is depicted in a video, disregarding the class-label of the action itself.
- **Temporal Action Localization**[5] handles both *what* and *when*. The task is to predict both the class label of actions depicted in a video and the corresponding

4. Taxonomy adopted from the ActivityNet challenge, `http://activity-net.org`
5. Another commonly encountered term is "Action Detection". Note however, that Action Detection is often used interchangeably for Temporal Action Localization and Spatio-Temporal Action Localization.

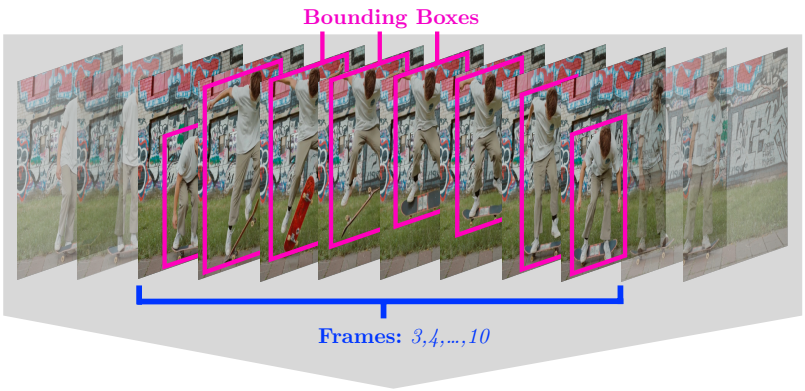


FIGURE 14.1 *Illustration of the types of information involved across Human Activity Recognition tasks.*

time stamps for each of the actions.

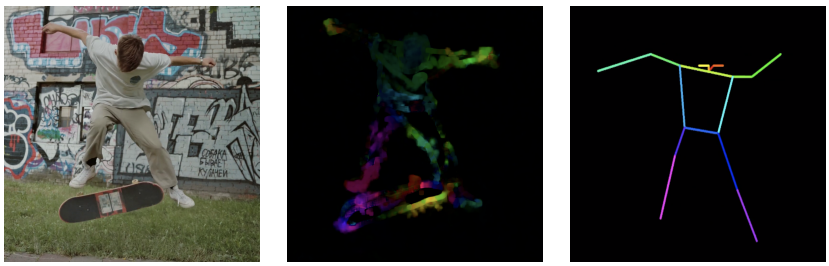
- **Spatio-Temporal Action Localization** is concerned with *what*, *when*, and *where*. The task is to classify the action, note in which frames it is present, and to specify the spatial bounding box for the action in each frame.

An overview of Human Activity Recognition tasks and associated labels is shown in Table 14.1.

TABLE 14.1 Tasks in Human Activity Recognition (rows) and the label information in each of them (columns).

	Class Label	Time Stamps	Bounding Boxes
Trimmed Action Recognition	×		
Temporal Action Proposals		×	
Temporal Action Localization	×	×	
Spatio-Temporal Action Localization	×	×	×

It should be noted that there exist settings where multiple action classes appear in the video simultaneously. This is especially prevalent in datasets for Temporal Action Localization and Spatio-Temporal Action Localization, but can also occur in Trimmed Action Recognition. Other tasks also exist that are human-centric, such as Human Pose Estimation [2], Dense Pose Estimation [3], Human Activity Prediction [4], Activity-based Person Identification [5] and Visual Voice Activity Detection [6], but these are beyond the scope of this chapter.



(a) RGB **(b)** Optical Flow **(c)** Skeleton

FIGURE 14.2 Examples of different input modalities frequently used for Human Activity Recognition. For the image of Optical Flow, the colour indicates the angle of the movement, and the colour intensity denotes movement magnitude. For the image of human body skeleton, different colours indicate connections of different human body joints.

14.1.2 Input Modalities for Human Activity Recognition

Not only does the information we want to obtain vary widely between sub-tasks in Human Activity Recognition; the input modalities are as just as diverse. The most common input is RGB video[6] which consists of a series of RGB images, sampled at a regular interval (usually in the range of 20 to 30 frames per second). Though the videos often vary in length, it is common practice to cut the video into clips containing a fixed number of frames (e.g. 32 frames per clip), sometimes with a lower frame rate (e.g. 8 frames per second) to ensure the clip covers a sufficiently large temporal extend despite the relatively few number of frames. This form of clipping is based on the assumption that human movements are slow-changing and the variations in the contents of a scene between successive video frames are small. It is a common approach for most modalities and is adopted to reduce redundant information and to accommodate network architectures that require a fixed input size, such as the 3D CNNs, which we will be described in Section 14.2.2.

The plain RGB input modality, can be extended with hand-crafted features such as Optical Flow [9] and Improved Dense Trajectories (iDT) [10]. These are calculated from the video inputs and explicitly capture information on the movements of keypoints between frames. Though this type of handcrafted features may be considered a remnant of the pipelines prior to the dominance of deep neural networks, their inclusion still yields a systematic accuracy improvement when used in multi-stream network topologies alongside plain RGB video. We will explore the multi-stream architectures in Section 14.2.6.

In addition to RGB video, some popular devices such as Kinect and Vicon [11] have depth-sensing capabilities that let them produce a depth (D)

6. While under some application scenarios multiple cameras can be used, leading to the Multi-view Human Action Recognition [7] and View-Invariant Action Recognition [8] problems, the most common scenario involves a single camera.

channel. This can either be treated as a fourth channel in the (RGB+D) video input, or more commonly used as a separate stream due to the lower spatial resolution of the depth channel compared to the RGB channels.

Finally, human body skeletons can be extracted for each frame, each corresponding to the body pose of a person appearing in the scene and represented as a set of human body joints interconnected by bones. Such a human body representation can be used to define a graph structure formed by nodes (joints) and edges (bones) that is evolving through time during action execution. The computation of skeleton landmarks is a built-in functionality to both the Kinect and Vicon devices, and alternatively it can be extracted using human body pose methods, such as those in the OpenPose toolkit [12]. The Spatial Temporal Graph Convolutional Network (ST-GCN) has been successfully applied to the skeleton input modality and will be covered in Section 14.2.5.

While there are still other modalities that have been used for Human Activity Recognition, such as Electromyography (EMG), accelerometer and wireless signal measurements, these are beyond the scope of this chapter.

14.2 TRIMMED ACTION RECOGNITION

Often, the terms Human Activity Recognition and Trimmed Action Recognition are used interchangeably, though the former encompasses an array of different tasks, as described in Section 14.1.1. Trimmed Action Recognition is the more precise term, and amounts to video classification, where the video is trimmed to the action of the human subject in the video. The action performed by this subject then constitutes the label. Though the human actions are of special interest here, the architectures described in this section (except for the skeleton-based models) are generic video-classifiers and might just as well be trained to classify the activity of a cat or other scene classification problems, for instance the bathing conditions at a beach, provided a dataset had been collected for this purpose and the corresponding methods have been trained to provide the respective classes as outputs.

In our coverage of the topic, we will start in Section 14.2.1 by describing how image classifiers can be used directly for Trimmed Action Recognition, by using a recursive neural network for knowledge integration over time. In Section 14.2.2, the extension of 2D CNNs for image classification to 3D CNNs for video classification is described. Section 14.2.3 extends our discussion on 3D CNNs by describing how we can "inflate" pretrained 2D CNN image classifiers to improve performance in 3D models. An issue with 3D CNNs as compared to their 2D counterparts is their increased network size. To make 3D CNNs more efficient, it is possible to decompose a 3D kernel into the combination of a 2D and a 1D convolution, as we will detail in Section 14.2.4. Human body pose-based Human Activity Recognition methods which utilize a sequence of skeletons instead of RGB videos as input, will be described in Section 14.2.5. Most of the architectures and models described in this chapter use a multi-stream

topology with multiple different input modalities to achieve their best results, and some methods use different streams to concurrently exploit multiple temporal extends and resolutions. The details on this are given in Section 14.2.6.

14.2.1 2D Convolutional- and Recurrent Neural Network-based Architectures

2D-CNNs have enjoyed almost unrivalled success in the domain of images. Specifically, the ImageNet competition and ever-improving benchmarks on the dataset of the same name, have led to CNN architectures that surpass human level performance on image classification tasks. In the context of Human Activity Recognition, one can thus reuse one of these networks to extract features for each frame of the input clip, and then integrate the information along the temporal dimension using some other method. This is a case of transfer learning, where the architecture and parameters of a model trained on a classification task in the 2D image domain is transferred to the task of frame-wise classification in video clips. There are many possibilities of how to integrate the information across frames, including the bag-of-features approach (or its recurrent version [13]), using convolutions along the temporal dimension, and via recurrent neural networks (RNN). Specifically, the RNN-based approaches with Long Short Term Memory (LSTM) as the recursive modules have seen good success in the years 2015 to 2017 [14, 15]. Their accuracy, however, was surpassed by the I3D [16] and R(2+1)D [17] architectures, which are covered in Section 14.2.3 and Section 14.2.4 respectively. Nevertheless, their appeal still holds today, due to the ease of training and straight-forward application to online systems.

The basic 2D-CNN + LSTM architecture is depicted in Fig. 14.3. When processing a video, each frame is passed through a CNN such as AlexNet [18], that was pre-trained to produce good results on the image classification task, before being fed to the recurrent module. Here, one or more (five in [15]) stacked LSTM layers process the image features and output a preliminary classification for time step t alongside a hidden state $\boldsymbol{h}_t$ that is reused in the computation on the next time-step $t + 1$. The network thus uses the same exact CNN and LSTM for processing each of the input frames forming the video, making it efficient in terms of the number of parameters. Finally, the partial classification results for all time steps are aggregated to produce the action classification result. The aggregation over time steps can take on many variants, i.e. taking the final classification result; averaging over all classification results; max-pooling over time (selecting the classification result with highest certainty among all time steps); and using a weighted sum with a gain g, which increases linearly from 0 in the first output to to 1 in the last [15, 14]. While the linearly weighted gain was found to work slightly better for inference, the difference in performance between aggregation strategies is very small.

Often, when training a recurrent neural network, the loss is computed only for the final step, and the error is then backpropagated through time [19]. Training

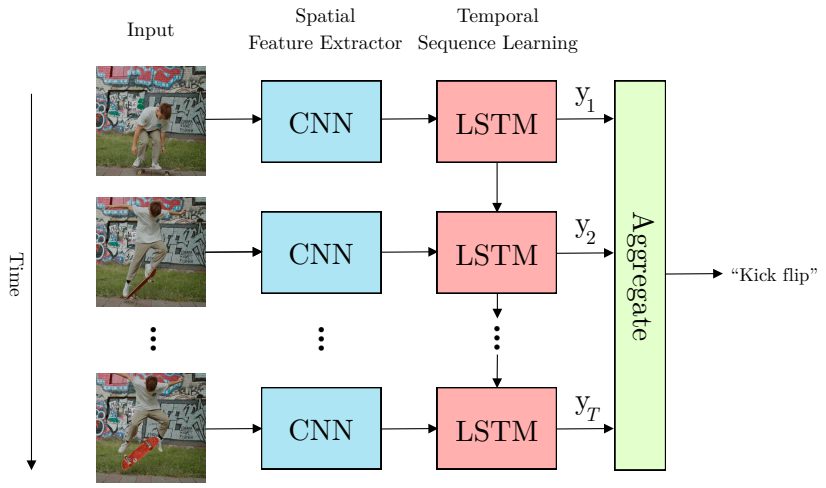


FIGURE 14.3 *2D-CNN + LSTM-based architecture employed in [14] and [15]. Each frame is independently processed through a pre-trained CNN which acts a feature extractor operating on the spatial domain. These features are then introduced to a (stacked) LSTM module, which produces a prediction y_t for time step t and a hidden state $\boldsymbol{h}_t$ that is re-used in the next time-step. Finally, an aggregation function produces a video-level prediction based on the predictions y_t at each time-step. Note that the exact same CNN and LSTM networks are used in each time step.*

using this technique can lead to a slow and memory-inefficient procedure, because all intermediary results through time need to be cached in order to compute the gradients. Moreover, the error information of early time steps diminishes when long sequences are processed. In practice, training can be improved by computing a classification loss for each time step, and weighting each of the derivatives by a gain g (in a similar manner as in the weighted aggregation strategy described above) before backpropagation [15].

14.2.2 3D Convolutional Neural Network Architectures

A straight-forward way of extending CNNs used for image classification tasks to be used for human activity recognition is to replace the 2D convolutions, which were suitable for processing images, with 3D convolutions. In this way, the convolution operation is extended to consider the temporal dimension in addition to the two spatial dimensions found in images. Indeed, many works have used this idea to produce well-performing models for video classification.

Like 2D convolutions, 3D convolutions[7] extract features from a local neigh-

7. In our usage of the terms 2D and 3D convolutions, we disregard the channel dimension of the input, even though a 2D convolution (convolution over spatial dimensions) in reality uses kernels which are 3D-tensors to account for height, width and channel dimensions. Similarly, 3D convolutions (spatial dimensions + temporal dimension) have kernels that constitute a 4D tensor

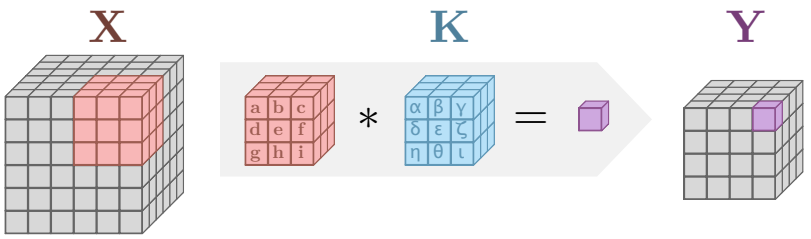


FIGURE 14.4 *Illustration of 3D convolution for a kernel of size* $(3 \times 3 \times 3)$ *without zero padding for a single-channel input video.* $(3 \times 3 \times 3)$ *slices of the input tensor* **X** *are convolved with a kernel* **K** *to produce an output tensor* **Y**. *The convolution amounts to a multiplication of each aligned cube and a summation of all involved cubes:* $a\alpha + b\beta + c\gamma + d\delta + e\epsilon + f\zeta + g\eta + h\theta + i\iota + \ldots$

borhood of an input by multiplying each input position with a corresponding weight tensor, and summing all values alongside a bias term. Only, instead of having a 2D kernel that covers an area of a 2D input, 3D convolutions cover a volume on the 3D input. This is illustrated in Fig. 14.4. Given a 3D input volume **X** and a convolutional kernel **K**, the output feature map **Y** is calculated as follows

$$Y(t,i,j) = \sum_{l} \sum_{m} \sum_{n} X(t-l, i-m, j-n) K(l,m,n) \tag{14.1}$$

In general, properties of 2D convolutions, such as the commutative property, also hold for 3D convolutions. In context of a 3D convolutional neural network, 3D convolutions are usually followed by a non-linear activation function such as the Rectified Linear Unit (ReLU) applied on each element of the output. These are then arranged in layers of multiple convolutional kernels. Using multiple convolutional kernels in a layer gives rise to outputs with an additional channel dimension, and likewise, an input can consist of multiple channels. Denoting by C the number of input channels, T the number of frames in a video clip (i.e. temporal dimension), and H and W the spatial height and width of each input frame, a network operating on video takes an input of size $(C \times T \times H \times W)$. For example, a video formed by 16 frames, each having a resolution of 112×112 pixels and three color channels (corresponding to RGB color space) is a tensor of size $(3 \times 16 \times 112 \times 112)$.

3D convolutions have been exploited by a number of works on Trimmed Action Recognition, which either use 3D convolutions as the primary building block in a deep CNN [20, 21, 22, 23] or as a part of a multistage pipeline [24]. An efficient and conceptually simple model introduced in [22] is the Convolutional 3D (C3D) architecture. It consists entirely of 3D convolutional layers with kernel size of $(3 \times 3 \times 3)$, 3D pooling layers, and fully connected layers. In each 3D convolutional layer, the input is zero-padded in the spatial and temporal

in practice.

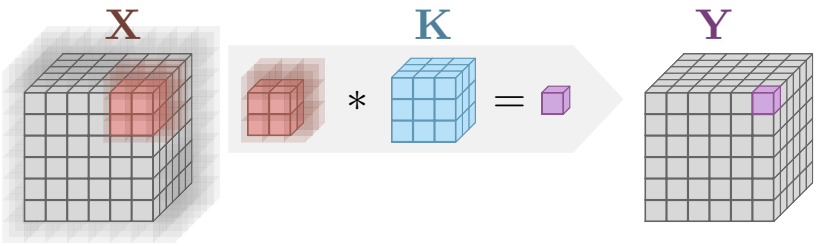


FIGURE 14.5 *Illustration of 3D convolution for a kernel of size* $(3 \times 3 \times 3)$ *with zero padding of size* $(1 \times 1 \times 1)$ *for a single-channel input video. During convolution, slices of the input tensor* **X** *that would have gone 1 step beyond the boundaries are allowed, with values outside the boundaries set to zero. Using zero padding this way the output tensor* **Y** *retains the dimensions of the input* **X**.

dimension in order to keep the dimensionality for the output feature-map. A 3D Convolution with zero padding is illustrated in Fig. 14.5. To reduce the size of the feature-maps throughout the network, max pooling is used. In C3D, this is implemented as a $(2 \times 2 \times 2)$ pooling with stride 2 after all convolutional layers but the first, where a $(1 \times 2 \times 2)$ pooling is used, thus omitting pooling along the temporal dimension initially. This was done to not collapse the temporal signal too early, and to postpone temporal collapse until the last layer (an input of size 16 can be pooled by 2 at most 4 times). Finally, the network has two fully connected layers with ReLU non-linearities before the final prediciton layer with a Softmax activation function. The architecture in its entirety is summarized in Table 14.2. It is worth noting the remarkable similarity between the C3D architecture and a 2D architecture such as VGG16 [25]. They are both built with blocks of padded convolutions of kernel size 3 and ReLU activations, followed by max pooling to reduce the size of the feature map, before being passed to two fully connected layers and a prediction layer. Works proposing these two network architectures were both published in 2015, and represent clear examples of the dominant design for deep neural networks in visual tasks at the time.

While 3D-CNN architectures have shown promising results, they did not achieved state-of-the-art accuracy initially. An issue with these models is that the additional kernel dimension results in significantly more parameters per neuron compared to a 2D CNN. This, coupled with the fact that we cannot reuse parameters from ImageNet pre-trained models, makes them harder to train than their 2D-CNN counterparts. Due to difficulties related to effectively optimizing networks with such a large number of parameters, the employed architectures have been relatively shallow compared to the deep network architectures used for image classification.

TABLE 14.2 Architecture summary for C3D [22]. All convolution and pooling operators operate in 3D across the time T, height H and width W dimensions. C denotes the number of channels or neurons in the layer.

Layer name	Neurons	Kernel size $T \times H \times W$	Padding $T \times H \times W$	Feature size $C \times T \times H \times W$
Input	-	-	-	$3 \times 16 \times 112 \times 112$
Conv1a + ReLU	64	$3 \times 3 \times 3$	$1 \times 1 \times 1$	$64 \times 16 \times 112 \times 112$
Pool1	-	$1 \times 2 \times 2$	-	$64 \times 16 \times 56 \times 56$
Conv2a + ReLU	128	$3 \times 3 \times 3$	$1 \times 1 \times 1$	$128 \times 16 \times 56 \times 56$
Pool2	-	$2 \times 2 \times 2$	-	$128 \times 8 \times 28 \times 28$
Conv3a + ReLU	256	$3 \times 3 \times 3$	$1 \times 1 \times 1$	$256 \times 8 \times 28 \times 28$
Conv3b + ReLU	256	$3 \times 3 \times 3$	$1 \times 1 \times 1$	$256 \times 8 \times 28 \times 28$
Pool3	-	$2 \times 2 \times 2$	-	$256 \times 4 \times 14 \times 14$
Conv4a + ReLU	512	$3 \times 3 \times 3$	$1 \times 1 \times 1$	$512 \times 4 \times 14 \times 14$
Conv4b + ReLU	512	$3 \times 3 \times 3$	$1 \times 1 \times 1$	$512 \times 4 \times 14 \times 14$
Pool4	-	$2 \times 2 \times 2$	-	$512 \times 2 \times 7 \times 7$
Conv5a + ReLU	512	$3 \times 3 \times 3$	$1 \times 1 \times 1$	$512 \times 2 \times 7 \times 7$
Conv5b + ReLU	512	$3 \times 3 \times 3$	$1 \times 1 \times 1$	$512 \times 2 \times 7 \times 7$
Pool5	-	$2 \times 2 \times 2$	$0 \times 1 \times 1$	$512 \times 1 \times 4 \times 4$
Flatten	-	-	-	8192
FC6	4096	-	-	4096
FC7	4096	-	-	4096
Preds + Softmax	Num. classes	-	-	Num. classes

14.2.3 Inflated 3D CNN Architectures

Hitherto, a major shortcoming of 3D CNNs was their inability to exploit the architectures and weights of image classification models trained on ImageNet. To solve the issue, Carriera and Zisserman [16] proposed to reuse the architectures and weights of state-of-the-art 2D CNNs for image recognition, and to *inflate* the 2D convolutional kernels to 3D. In practice, this means taking each $(K \times K)$ 2D convolutional kernel in a source model (e.g. VGG16), repeating it K times along a new 3rd dimension and finally dividing the weights by K. At the heart of the technique is the observation that we can treat an image as a "boring video" by repeating it along the temporal dimension is used. The 3D convolutional filters can then be bootstraped by enforcing a "boring video fixed point", i.e. that the output of a 3D filter applied to a boring video, should yield the same result as the corresponding 2D filter would for the image that made the boring video. For example, consider the convolution of a (3×3) patch of an image and a (3×3)

kernel:

$$\begin{aligned}\mathbf{K} * \mathbf{X} &= \begin{bmatrix}1 & 0 & -1\\ 2 & 0 & -2\\ 1 & 0 & -1\end{bmatrix} * \begin{bmatrix}0.1 & 0.2 & 0.3\\ 0.4 & 0.5 & 0.6\\ 0.7 & 0.8 & 0.9\end{bmatrix}\\ &= 1 \cdot 0.1 + 2 \cdot 0.4 + 1 \cdot 0.7 - 1 \cdot 0.3 - 2 \cdot 0.6 - 1 \cdot 0.9\\ &= -0.8\end{aligned}$$

If we naïvely repeat the kernel three times to produce a tensor kernel $\mathbf{K} \in \mathbb{R}^{3\times3\times3}$ to match a boring video patch $\mathbf{X}$, their convolution would produce three times the desired result. But if the kernel weights are divided by three, the results match:

$$\mathbf{K} * \mathbf{X} = \sum_{1}^{3} \frac{1}{3} \cdot 0.1 + \frac{2}{3} \cdot 0.4 + \frac{1}{3} \cdot 0.7 - \frac{1}{3} \cdot 0.3 - \frac{2}{3} \cdot 0.6 - \frac{1}{3} \cdot 0.9 = -0.8$$

Moreover, because the response of activation functions and max-pooling are the same in 3D as for 2D, the whole network satisfies the boring-video fix-point.

Using an inflated Inception-v1 [26] network in a multi-stream setup with an Optical Flow input in the second stream, the inflated 3D (I3D) model was able to yield state-of-the-art results on multiple benchmarks in trimmed Action recognition.

14.2.4 Factorized (2+1)D CNN Architectures

Consider the associative property for convolutions for a feature map $\mathbf{X}$ and two kernels:

$$(\mathbf{X} * \mathbf{K}_1) * \mathbf{K}_2 = \mathbf{X} * (\mathbf{K}_1 * \mathbf{K}_2) \tag{14.2}$$

On the left-hand-side of Eq. (14.2), a feature map is convolved sequentially by multiple kernels in a CNN (disregarding the activation function). Without the non-linearities, we could have freely convolved all the kernels prior to convolving the feature map, as is done on the right-hand-side of Eq. (14.2). This means that we can construct a new kernel $\mathbf{K}$ by convolving two others, $\mathbf{K}_1 * \mathbf{K}_2$, and that convolving the feature map $\mathbf{X}$ with the new kernel $\mathbf{K}$ yields the same result as first convolving the feature map with one kernel $\mathbf{K}_1$ and then the second kernel $\mathbf{K}_2$. For example, consider the decomposition of a Sobel operator [27] used for the detection of horizontal edges:

$$\mathbf{K}_1 * \mathbf{K}_2 = \mathbf{K}$$

$$\begin{bmatrix}1 & 0 & -1\end{bmatrix} * \begin{bmatrix}1\\ 2\\ 1\end{bmatrix} = \begin{bmatrix}1 & 0 & -1\\ 2 & 0 & -2\\ 1 & 0 & -1\end{bmatrix}$$

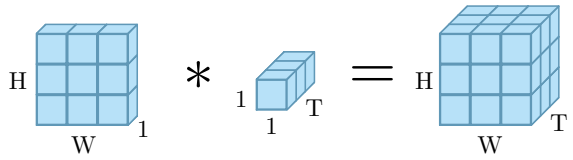


FIGURE 14.6 *The convolution of a* $(1 \times H \times W)$ *and* $(T \times 1 \times 1)$ *kernel with padding yields a resulting convolutional kernel of dimension* $(T \times H \times W)$.

Convolving the feature map $\mathbf{X}$ with the kernels $\mathbf{K}_1$ and $\mathbf{K}_2$ sequentially gives the exact same results as convolving it once with $\mathbf{K}$ (see the prior example):

$$
\begin{aligned}
(\mathbf{X} * \mathbf{K}_1) * \mathbf{K}_2 &= \left(\begin{bmatrix} 0.1 & 0.2 & 0.3 \\ 0.4 & 0.5 & 0.6 \\ 0.7 & 0.8 & 0.9 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \end{bmatrix} \right) * \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \\
&= \begin{bmatrix} (0.1 \cdot 1 + 0.2 \cdot 0 + 0.3 \cdot (-1)) \\ (0.4 \cdot 1 + 0.5 \cdot 0 + 0.6 \cdot (-1)) \\ (0.7 \cdot 1 + 0.8 \cdot 0 + 0.9 \cdot (-1)) \end{bmatrix} * \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \\
&= \begin{bmatrix} -0.2 \\ -0.2 \\ -0.2 \end{bmatrix} * \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \\
&= -0.2 \cdot 1 - 0.2 \cdot 2 + -0.2 \cdot 1 \\
&= -0.8
\end{aligned}
$$

Likewise, we can decompose a $(T \times H \times W)$ 3D kernel as the convolution of a $(1 \times H \times W)$ kernel with a $(T \times 1 \times 1)$ kernel. This is illustrated in Fig. 14.6.

Though not all 3D convolutional kernels can be represented as the convolution of separate 2D and a 1D kernels, doing so can reduce the computational complexity of convolving an input 3D tensor with the resulting kernel from $O(T \cdot H \cdot W)$ to $O(T + H \cdot W)$. For example, using this approach for convolution with a $(3 \times 3 \times 3)$ kernel and without using a bias term, the number of parameters and floating point operations (multiplications and additions) is reduced from $3^3 = 27$ to $3 + 3^2 = 12$. This idea was exploited in the Pseudo 3D blocks (P3D) [28] and the R(2+1)D [17] architectures, by using separate 2D and 1D convolutions to approximate 3D kernels. The "R" in the R(2+1)D alludes to the residual connection (aka. skip connection), that was popularized by ResNet [29]; mechanistically, it adds the input of a block to its output. Unlike our previous example, a ReLU activation was added after each convolution of the (2+1)D block, yielding double the number of nonlinearities as compared to a regular 3D convolution with ReLU. According to the authors, this has the benefit of increasing the complexity of functions that can be represented by the network. The reduction in the number of parameters from using (2+1)D blocks instead of 3D convolutions can either be used to decrease the total network size

and run-time, or it can be used to increase the number of neurons in the network by keeping the overall number of parameters unchanged. For a (2+1)D convolutional block with and input of C_{i-1} channels and output of C_i channels to have the same number of parameters as a regular 3D convolution, the first (spatial) convolution should be formed by M_i neurons:

$$M_i = \frac{T \cdot H \cdot W \cdot C_{i-1} \cdot C_i}{(H \cdot W \cdot C_{i-1}) + (T \cdot C_i)}. \tag{14.3}$$

For example, with $T = H = W = 3$ and $C_{i-1} = C_i$, the intermediary subspace has dimension $M_i = 2.25 \times C_i$. Using this approach, R(2+1)D [17] was able to surpass or match the benchmark results for I3D.

14.2.5 Skeleton-Based Action Recognition

A fundamental way to describe a human action is to determine human body poses which appear during action execution along with their temporal succession. Given a video as a sequence of frames, the human body pose depicted in each frame can be represented by estimating the spatial arrangement of the human body joints and forming the corresponding body skeleton. Efficient and accurate body skeleton detection methods have been exist and can be easily used due to their integration in popular toolboxes such as OpenPose [12]. Compared to other data modalities used for human activity recognition such as RGB videos, optical flow, and depth images, body skeletons provide several advantages: They encode high-level information of human body poses and dynamic body motions in a compact and simple structure, and they are invariant to view-point, illumination variations, body scale, human appearance, motion speed, context noise, and background complexity [30]. While these representational benefits can lead to improved performance for actions involving one person, human body skeletons cannot be easily extended to complex activities and multiple persons, where the scene context is needed to distinguish actions involving similar human body poses. For example, consider a player running during a football game. The exploitation of scene information, e.g. related to the appearance of the field, the ball, and other players, can be the defining factor in distinguishing the action "play football" from the action "run".

Recently, many deep learning based methods have been proposed, which take advantage human skeleton dynamics for skeleton-based human action recognition. These methods are generally categorized into three groups. The first group consists of methods that are based on recurrent neural networks, which mostly utilize LSTM networksto model the temporal dynamics of the action in a sequence of skeletons. Examples in this group of methods include Deep LSTM [31], ST-LSTM [32], VA-LSTM [33] and STA-LSTM [34]. These methods form multiple feature vectors, each representing a body skeleton forming the action using the 2D or 3D coordinates of the body joints, and subsequently apply temporal analysis on the sequence of the feature vectors. The second

group of methods are based on CNNs. Because CNNs require their inputs to follow a regular grid structure, these methods transform the body joint coordinates of each body skeleton to a pseudo-image, and the sequence of such pseudo-images is subsequently processed by the CNN. Noteable works in this group are [35, 36, 37, 38, 39]. Methods belonging to these two groups are not fully capable of exploiting information encoded in the non-Euclidean structure of the body skeletons. Moreover, they are not able to benefit from the information encoded by the spatial connections between the body joints. The third group of methods is based on Graph Convolutional Networks (GCNs) which are able process the structural information encoded in the body skeletons more efficiently. GCNs [40] generalize the notion of convolution from grid-structured data to graph data structures, and have been very successful when applied in skeleton-based human action recognition. Here, each skeleton is represented as a graph which models the corresponding body pose by a set of joints connected with edges following the natural spatial connections indicated by the human body structure. The temporal dynamics in an action are modeled by connecting the graph structures representing the corresponding human body skeletons in successive time steps corresponding to different frames. The Spatial-Temporal Graph Convolution Network (ST-GCN) [41] exploited such a spatio-temporal graph structure for GCN-based action recognition, and several methods have built on top of this idea in recent years. In the remainder of this section, we shall delve further into ST-GCNs.

14.2.5.1 Spatial-Temporal Graph Convolution Network

Given a sequence of skeletons with body joint coordinates, the ST-GCN method first constructs a spatio-temporal graph by connecting each joint to its neighboring joints in the same body skeleton and the corresponding joints in the preceding and successive skeletons in the temporal sequence. Then, spatio-temporal GCN layers are applied to the graph, in order to capture both spatial and temporal patterns of the skeleton's motion, and to finally recognize the human action. Formally, a spatio-temporal graph on a sequence of skeletons is defined as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where

$$\mathcal{V} = \{v_{ti} \mid t,\ i \in \mathbb{Z},\ \ 1 \leq t \leq T,\ \ 1 \leq i \leq V\}$$

denotes the vertice/node set of V body joints in a sequence of skeletons of T time steps, and $\mathcal{E}$ is the set of spatial and temporal edges expressing the connectivity of the body joints through space and time domains. The spatial edges express the natural connections between the body joints in each skeleton and the temporal edges connect each body joint to its corresponding joint in the previous and next time steps. Fig. 14.7 (right) illustrates the spatio-temporal graph constructed on a sequence of 3 skeletons representing a person walking.

Given the spatio-temporal graph and the node features as input, the ST-GCN model updates the features in both spatial and temporal domains by performing

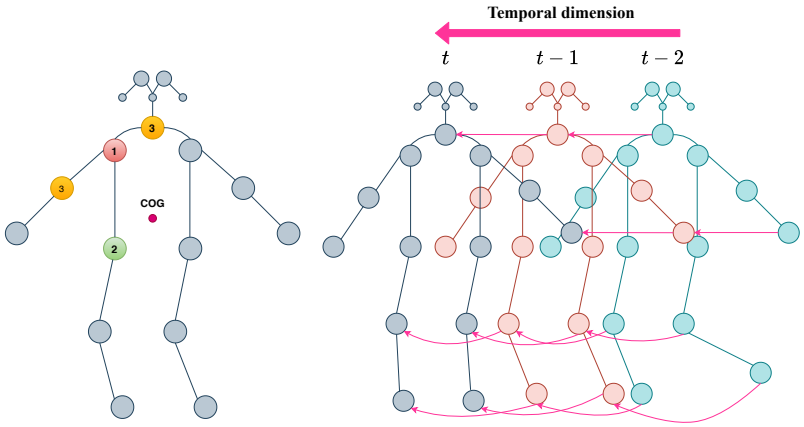


FIGURE 14.7 Left: the 3 neighboring subsets of a joint obtained by the spatial partitioning process in [41] are shown in different colors. The Center of Gravity (COG) is shown as a red dot. The joint under consideration is the one shown in red color, which forms the subset of root node ($p = 1$). Joints closer to the COG than the root node form the second subset ($p = 2$) and are shown in green color. Joints farther to the COG than the root node of the third subset ($p = 3$) are shown in orange color. Right: illustration of spatio-temporal graph constructed on a sequence of three skeletons. In temporal domain, each joint is connected to its corresponding joint in the next time step. The temporal connections for some of the nodes are shown by directed arrows.

graph convolutions. By generalizing the definition of convolution on 2D feature maps, the spatial graph convolution on each skeleton is defined as [41]:

$$f_{out}(v_{ti}) = \sum_{v_{tj} \in \mathcal{N}(v_{ti})} \frac{1}{Z_{ti}(v_{tj})} f_{in}(v_{tj}) \mathbf{W}_{l_{ti}(v_{tj})}, \tag{14.4}$$

where $f_{out}(v_{ti})$ denotes the output features for v_{ti} which is the i^{th} node at time step t, $f_{in}(v_{tj})$ denotes the input features of v_{tj}, and $\mathcal{N}(v_{ti})$ denotes the set of neighboring nodes to v_{ti} which is defined as $\mathcal{N}(v_{ti}) = \{v_{tj} \mid d(v_{ti}, v_{tj}) < D\}$. $d(v_{ti}, v_{tj})$ is the shortest path distance from v_{ti} to v_{tj}. In ST-GCN, the shortest distance is set to $D = 1$ which means that, at each step, the convolution is performed on each node and its immediate neighbors. $\mathbf{W}_{l_{ti}(v_{tj})}$ denotes the transformation used to calculate how input features of node v_{tj} affect the output features of node v_{ti}.

In 2D convolution on grid data, the central pixel and its neighboring pixels which are convolved with a 2D kernel have a fixed spatial order. Therefore, the kernel values are indexed based on the spatial order of pixels and all the pixels are convolved with a shared kernel. In a skeleton graph, the nodes can have different degrees and different number of neighbors, so there is no fixed spatial order around each node of the spatial graph. ST-GCN method addresses this issue by employing a spatial partitioning process to partition the neighboring set of a node v_{ti} into a fixed number of P subsets and assigning each subset a numeric

label $l_{ti} \in \{1, ..., P\}$. The neighboring nodes v_{tj} assigned to a label $l_{ti}(v_{tj}) = p$ form the neighborhood $\mathcal{N}_p(v_{ti})$, where $\bigcup_{p=1}^{P} \mathcal{N}_p(v_{ti}) = \mathcal{N}(v_{ti})$, and are used to update the features of v_{ti}. The spatial partitioning process considers the structure of the skeletons so that the neighboring nodes of each body joint is divided into $P = 3$ subsets which are defined as follows:

1. the root node itself,
2. the neighbors of the root node which are closer to the center of gravity (COG) than the root node,
3. the remaining neighbors of the root node, i.e. those that are farther to the COG than the root node.

The COG is defined as the average of all the body joints' coordinates in a skeleton. Fig. 14.7 (left) shows the partitioned neighbors of a joint in a skeleton which are illustrated in different colors. Accordingly, $l_{ti}(v_{tj})$ in equation (14.4) maps each neighbor v_{tj} of node v_{ti} into one of the above mentioned three subsets. Each neighboring subset is then associated with a transformation matrix $\mathbf{W}_p$, $p \in \{1, 2, 3\}$. Therefore, by employing this spatial partitioning, the model learns a fixed number of $P = 3$ transformation matrices which are shared between all the nodes and their neighboring subsets for feature extraction. The normalization factor $Z_{ti}(v_{tj}) = \left|\left\{v_{tk} \mid l_{ti}(v_{tk}) = l_{ti}(v_{tj})\right\}\right|$ denotes the number of nodes in each partition and balances the contribution of each of the neighboring subsets in the output features $f_{out}(v_{ti})$ of the target node v_{ti}.

Thus, the graph structure in each skeleton is modeled by P Adjacency matrices $\mathbf{A}_p \in \mathbb{R}^{V \times V}$ having values of $\mathbf{A}_p^{(ij)} = 1$ if joints i and j exhibit a connectivity of type $p = \{1, 2, 3\}$, and $\mathbf{A}_{p,ij} = 0$ otherwise. The normalized Adjacency matrix $\hat{\mathbf{A}}_p \in \mathbb{R}^{V \times V}$ for each subset of neighbors p is defined as:

$$\hat{\mathbf{A}}_p = \mathbf{D}_p^{-\frac{1}{2}} \mathbf{A}_p \mathbf{D}_p^{-\frac{1}{2}}, \tag{14.5}$$

where $\mathbf{D}_p^{(ii)} = \sum_j^V \mathbf{A}_p^{(ij)} + \varepsilon$, is the (diagonal) degree matrix, and $\varepsilon = 0.001$ is used to avoid issues related to empty rows in $\mathbf{D}_{(ii)_p}$ when calculating $\hat{\mathbf{A}}_p$. The Adjacency matrix of the root node denotes the nodes' self-connections and it is set to $\mathbf{A}_1 = \mathbf{I}_V$, where $\mathbf{I}_V \in \mathbb{R}^{V \times V}$ is the identity matrix.

In practice, the ST-GCN model receives as input the Adjacency matrices of the spatio-temporal graph $\mathcal{G}$ and a sequence of T_{in} skeletons denoted as $\mathbf{X} \in \mathbb{R}^{C_{in} \times T_{in} \times V}$, where C_{in} is the number of input channels for each body joint and V denotes the number of body joints in each skeleton. The spatial convolution in each layer of ST-GCN model is performed by employing the layer-wise propagation rule of GCNs [40] and it is defined as follows:

$$\mathbf{H}_s^{(l)} = ReLU\left(\sum_p \left(\hat{\mathbf{A}}_p \otimes \mathbf{M}_p^{(l)}\right) \mathbf{H}^{(l-1)} \mathbf{W}_p^{(l)}\right), \tag{14.6}$$

where $\mathbf{H}_s^{(l)}$ is the output of the spatial convolution at layer l and $\mathbf{H}^{(l-1)}$ is the

output of the spatio-temporal GCN at layer $l-1$. The input of the first hidden layer of the model is defined as $\mathbf{H}^{(0)} = \mathbf{X}$. The matrix $\mathbf{W}_p^{(l)} \in \mathbb{R}^{C^{(l)} \times C^{(l-1)}}$ performs a linear transformation of the features associated to the p^{th} partition of the graph, leading to a change of the nodes' representation from a feature space with $C^{(l-1)}$ dimensions to one with $C^{(l)}$ dimensions. $\otimes$ is the element-wise product of two matrices and $\mathbf{M}_p^{(l)} \in \mathbb{R}^{V \times V}$ is a learnable attention mask which highlights the importance of the connections between different body joints (spatial edges) in a skeleton to capture the most important relations between the parts of the human pose at a time step of a human action. The output of the spatial convolution $\mathbf{H}_s^{(l)}$ is introduced into the temporal convolution operation. In practice, the spatial convolution operation performs $C^{(l)}$ standard 2D convolutions with filters of size $C^{(l-1)} \times 1 \times 1$ on $\mathbf{H}^{(l-1)}$ to map the features into a $C^{(l-1)}$ dimensional subspace and multiplies the resulting tensor with the attention masked Adjacency matrix $(\hat{\mathbf{A}}_p \otimes \mathbf{M}_p^{(l)})$ on the last dimension V.

In order to model the temporal dynamics through the sequence of skeletons, the temporal aspect of the spatio-temporal graph which connects the same body joint through consecutive skeletons is employed to extend the spatial convolution to the temporal domain of the data. In this regard, the neighboring set of each node v_{ti} in temporal domain is defined as

$$\mathcal{N}(v_{ti}) = \left\{ v_{qj} \mid d(v_{tj}, v_{ti}) \leq K, |q - t| \leq \lfloor \Gamma/2 \rfloor \right\}, \tag{14.7}$$

where Γ is the temporal kernel size which denotes the number of consecutive skeletons which are involved for feature aggregation in temporal domain. In practice, the temporal convolution performs $C^{(l)}$ standard 2D convolutions with kernels of size $C^{(l)} \times \Gamma \times 1$ on $\mathbf{H}_s^{(l)}$ and the output would be $\mathbf{H}^{(l)}$ which is of size $C^{(l)} \times T^{(l)} \times V$. This means that the features of a node v_{ti} and those corresponding to the same joint in a time window of Γ frames centered at time t are convolved with the filter of the temporal convolution to update the features of each node v_{ti} in the spatio-temporal graph, i.e. a value of $K = 0$ is used. Depending on the temporal kernel size Γ, the stride and zero padding size in temporal convolution operation, the $T^{(l)}$ can be less than or equal to T_{in}. Fig. 14.8 shows the standard 2D convolution operation in both spatial and temporal domains.

The ST-GCN method has some drawbacks related to the graph construction process which are addressed by more recently proposed methods, such as the 2s-AGCN [42], the DGNN [43], and the GCN-NAS [44] methods. It employs a predefined (and fixed) graph structure that represents the naturally existing physical connections between the human body joints. While it uses an attention mechanism in which the learnable attention mask can be used to highlight (or diminish) connections between the joints which are informative for an action, the fact that this attention mask is applied using element-wise multiplication does not allow the method to learn connections between joints that do not appear in the predefined Adjucency matrices $\mathbf{A}_p$. It has been shown that such a graph construction process is not optimal for the human action recognition task, since

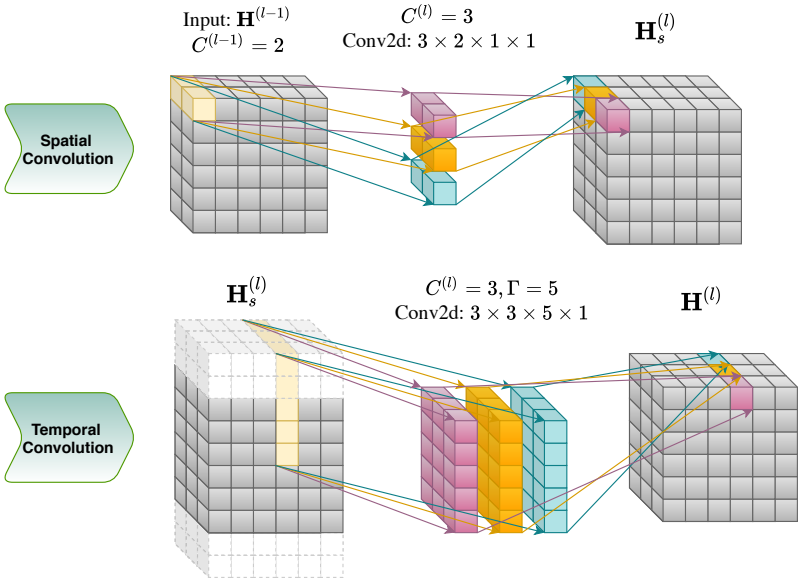


FIGURE 14.8 Illustration of spatial and temporal convolutions at l^{th} layer of the ST-GCN network. The values of $C^{(l-1)} = 2$, $C^{(l)} = 3$ and $\Gamma = 5$ are used. In order to keep the temporal dimension unchanged so that $T^{(l)} = T^{(l-1)}$, zero padding is used in temporal convolution. Conv2d denotes the standard 2D convolutional operation.

in some actions the interaction between some of the body joints which are not naturally connected in the human body is very important to classify the action. As an example, for an action like "hugging" or "clapping" the connection between two hands is very important even though it does not exist naturally in the human body.

The 2s-AGCN [42] method which is built on top of ST-GCN method, successfully addressed this issue and improved the performance of action recognition by proposing an additive attention mechanism in which the learnable attention mask is added to the graph Adjacency matrix to both highlight (or diminish) the existing graph edges for each action and also potentially add some new edges to the graph which connect the disconnected joints to help the model classify the action correctly. Moreover, in addition to the global graph which encodes the human body skeleton in general, the 2s-AGCN method also learns an individual data-dependent graph which represents a unique structure for each input sample and increases the model's flexibility for classifying various actions. In more details, this method utilizes the sum of natural human body structure, the learnable attention mask and a data-dependent graph which encodes the soft connections between human body joints for each action. GCN-NAS [44] method also learns the graph structure adaptively during the training process by exploring a search

space providing various dynamic graph modules. Besides, it considers multiple-hop neighbors of each node in the graph in order to increase the representational capacity of the model. DGNN [43] represents the skeletons as a directed acyclic graph to model the dependencies between joints and bones simultaneously. This method constructs the directed graph using the human body joints as graph nodes and the bones as edges while the edges are grouped into incoming edges and outgoing edges based on their direction.

In most of the ST-GCN-based methods, the model has a fixed architecture with a large number of parameters. Several methods have been proposed to reduce the computational cost and the memory requirements. SGN [45] proposes a compact model by introducing high-level semantics of the human body joints such as joint type and frame index into the model explicitly. Shift-GCN [46] also improves the computational complexity by employing lightweight point-wise convolutions and shift graph convolutions instead of regular graph convolution. DPRL+GCNN [47] and TA-GCN [48] improve the efficiency in terms of floating point operations (FLOPs) by selecting the most informative skeletons in a sequence and PST-GCN [49] method finds a compact and data-dependent network architecture for the ST-GCN in a progressive manner. ST-BLN [50] analyzes the connection of GCN layers with additive spatial attention to bilinear layers and propose the spatio-temporal bilinear network (ST-BLN) which does not require the use of predefined body skeletons and allows for more flexible and potentially more efficient design of the model.

14.2.6 Multi-stream Architectures

So far, we have focused on the neural network design for a single input. However, most of the methods we have covered above use ensembles or topologies with multiple streams in conjunction to achieve their highest performance. The motivation for combining multiple types of inputs is that each separate input stream will contain information that other streams lack (or at least have it more readily available). In this section, we will go through some common network architectures which use multiple streams.

14.2.6.1 Multi-modal

A common combination in the multi-modal setting is to use a stream of hand-crafted features, such as Optical Flow, to explicitly input motion information to the network in addition to the RGB video frames. This combination takes inspiration in the two-stream hypothesis in neuroscience [51], which models the two hypothesized pathways from the human visual cortex: A *ventral* stream performs recognition and detection on still objects, and a *dorsal* stream recognizes motion. The work in [52] proposed a corresponding two-stream neural network which has a stream that operates on RGB-frames, and another one which uses stacked frames of Optical Flow. The architecture of the network streams were 2D CNNs as used in plain image recognition, but in general the two-stream

setup can and has been used for all the architecture types discussed so far, i.e. 2D CNN + RNN [14, 15] (Section 14.2.1), 3D CNNs [23] (Section 14.2.2), Inflated 3D CNNs [16] (Section 14.2.3), and the Factorized (2+1)D CNN [17] (Section 14.2.4). The setup has also been extended beyond two streams by using Improved Dense Trajectories (iDT) in addition to [23] or as a replacement for [22] Optical Flow.

The multi-stream setup has also been used in skeleton-based human action recognition methods to improve the model's performance in terms of classification accuracy. Some methods such as AS-GCN [53], 2s-AGCN [42], GCN-NAS [44] and TA-GCN [48] improved performance by forming a two stream network and training the first and second stream using joint and bone data, respectively. In these methods, the final classification result is obtained by fusing the predicted Softmax scores of the two network streams. DGNN [43] also benefits from the motions taking place in a sequence of skeletons and utilizes the optical flow data to train the model. It forms a four-stream model and trains each network stream using a distinct data, from the set of joint, bone, joint's motion and bone's motion data, and fuses the Softmax predictions of all the network streams to get the final outcome.

There are also multiple ways of fusing input modalities, as explored in [52] and [54]. The most common is to perform *averaging* (or equivalently summation) for each pair of channels in the two streams, when the dimensionality of both streams are equal. This was found to work well by following both the strategies of late fusion, i.e. combining the modalities at the network end [52, 14, 15, 16, 17], and mid-network fusion [54]. Other options include taking the point- and channel-wise *max* across streams, and *concatenation* of channels for the stream, optionally followed by a pointwise 1D *convolution* to reduce the number of channels. The approach using concatenation and pointwise 1D convolutions was found to work best among the options explored in [54] for mid-network fusion. For iDT features as the additional modality, late fusion using channel concatenation in conjunction with a Support Vector Machine (SVM) for performing predictions has also been used successfully [22, 23].

14.2.6.2 Multi-resolution

Instead of using a different input modality, one may also use multiple variations on the same input to improve performance or make the network more efficient. One such setup is to use multiple resolutions of an input video; one covering a large spatial extend (the *context*) of the video at low resolution, and one with a higher resolution cropped to the center of the video (the *fovea*, c.f. the human eye). This takes advantage of the bias present in many videos where the most important motifs are centered. Given an input video formed by frames of size 178×178 pixels, late fusion by concatenation of the input downscaled to 89×89 pixels for context, and a 89×89-pixel center crop of the input as fovea was used in [21], to achieve a $2 - 4\times$ speedup in runtime performance without loss of

accuracy.

14.2.6.3 Multi-temporal

A recent approach, which achieves state of the art performance, is the *SlowFast* network architecture [55]. Here, a *Slow* pathway uses a high temporal stride (low frame rate) input video to a high-capacity network stream in order to capture the semantics of slow changing objects in the scene. Similarly, a *Fast* pathway uses low temporal stride (high frame rate) input to a low capacity network stream to capture more rapidly moving objects in the video. The pathways are fused using mid-network lateral connections from the fast to the slow pathway with concatenation. In order to match the sizes before fusing, a time-strided 3D convolution with a kernel size of $T \times H \times W = 5 \times 1 \times 1$, temporal stride α and $2\beta C$ output channels is used, assuming a slow pathway of feature dimension $C \times T \times H \times W$ and fast pathway dimensions $\beta C \times \alpha T \times H \times W$.

Another approach named *Feature Aggregation for Spatio- TEmporal Redundancy* (FASTER) uses a combination of a computationally expensive network every N frames and a cheap network in the other frames, and accumulates them using a specially made FAST-GRU [56].

14.3 TEMPORAL ACTION LOCALIZATION

In Temporal Action Localization we go beyond the standard single-label paradigm used in classification. Here, the task is to specify the fine-grained time (video frames) in which an action is present in addition to the action-class label. Typically, this also entails recognizing multiple classes with temporal overlap. For instance, the action of performing a "Kick Flip" on a skateboard also includes a simultaneous "Jump" action (see Fig. 14.1). The data is thus labelled differently than in Trimmed Action Recognition; it has not been trimmed to a single action. Accordingly, the input videos are often of longer duration, and multiple (or no) action labels may be present in each time instance.

In order to accommodate the more flexible labelling requirements in Temporal Action Localization, the binary cross entropy over each time t and class c is used as loss:

$$\mathcal{L}(\mathbf{X}) = \sum_{t,c} z_{tc} \log\left(p(c \mid \mathbf{X}_t)\right) + (1 - z_{tc}) \log\left(1 - p(c \mid \mathbf{X}_t)\right) \tag{14.8}$$

where $\mathbf{X}$ is the input video, z_{tc} the ground truth label, which is 1 if the class c is present at time t and 0 otherwise, and $p(c \mid x_t)$ is the model prediction for class c at time t. Contrary to the single-label setting, which uses a Softmax activation function to distribute the probability density over all classes, a Sigmoid activation is used here to let each class probability be computed separately.

All the network topologies we have discussed so far for Trimmed Action Recognition are also applicable for Temporal Action Localization. Networks such as the Two-stream CNN [52] or I3D [16] are often used as base networks

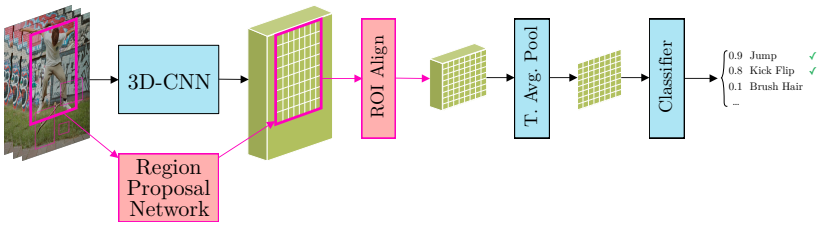


FIGURE 14.9 *Faster R-CNN based pipeline for Spatio-Temporal Action Localization. The blue boxes denote components found in the usual Trimmed Action Recognition pipeline, while the pink boxes are components found in Object Detection. The green 3D boxes illustrate the produced features.*

or as part of an extended architecture. Architectures tailored for Temporal Action Localization predominantly put their focus on exploiting the temporal sequence of actions, learning for instance that "Basketball Dunk" has a high likelihood of coming after a "Basketball Dribble" and to co-occur with a "Jump". LSTM-based methods lend themselves well to this kind of modelling and can be used to capture the dependencies on a high level [57]. The *Temporal Gaussian Mixture Layer* [58] is another method which captures the long-term dependencies by parameterizing a temporal convolution by the location and variance of a collection of Gaussians. In practice, these layers are used on top of the features of a Trimmed Activity Recognition network such as I3D.

In the evaluation of Temporal Action Localization methods, the presence or absence of each actions needs to be considered for each frame. This can be captured by the mean Average Precision metric.

14.4 SPATIO-TEMPORAL ACTION LOCALIZATION

Extending Temporal Action Localization with an estimation of the spatial bounding boxes for the action(s) in each frame leads to the task of Spatio-Temporal Action Localization. As in Temporal Action Localization (Section 14.3), the human subject may perform multiple simultaneous actions. Moreover, we may have multiple human subjects in a clip, the actions of whom must be located and classified separately.

Spatio-Temporal Action Localization can be considered a mix between Trimmed Human Activity Recognition and Object Detection, as techniques from both fields are used for the task. Broadly speaking, Object Detection Techniques are used for localizing the human in each frame, and an Action Recognition model is applied to classify the action that takes place in that location. Thus, the detection and classification pipeline for Spatio-Temporal Action Localization can be adapted from Object Detection with few modifications. The temporal information in the input video is most important to the classification of actions, and less so for detecting the boundaries of the human subject. In [59] and [55],

the pipeline was set up accordingly, using the Faster R-CNN [60] meta-algorithm with minor adaptations. As usual in Human Activity Recognition, a 3D feature extractor is used to produce a set of spatio-temporal features. Any of the previously described networks can be used to produce these, such as the Two-stream CNN [52], the 2D-CNN+LSTM [14, 15], the I3D [16], the R(2+1)D [17]. In parallel, a Region Proposal Network with a deep 2D-CNN as feature extractor (considering only a single frame), evaluates candidate regions using a finite set of anchor boxes. The region(s) detecting a human constitute the bounding boxes, and are used to extract a region from the spatio-temporal features. These are then warped to a predefined size of quadratic shape using either ROI Pooling [61] or the improved ROI Align [62], before being average pooled along the temporal dimension, and finally classified using a multi-class classifier as described in Section 14.3. This pipeline, as depicted in Fig. 14.9, with a SlowFast feature extractor (see Section 14.2.6.3) was used to achieve state of the art results on the challenging AVA benchmark [59, 55].

The above-described pipeline is based on the Region-CNN (R-CNN) family of Object Detectors, which employ an *Anchor-based* approach, evaluating a predefined set of anchor boxes to find a matching location. Alternatively, an *Anchor-free* Object Detection method such as CenterNet [63] could also be adapted as was done in [64].

14.5 DATASETS FOR HUMAN ACTIVITY RECOGNITION

A central part of any task in deep learning is the availability of good datasets. In this section, we give a brief categorization of datasets in the domain, followed by a curated list of open-source datasets. Datasets in Human Activity Recognition come in a plethora of sizes, content types and annotation levels. In terms of annotation, a single video-level action class label is the most basic annotation available. Extending this, a video may have multiple labels associated (e.g. “Jump” and “Kick Flip”). While this is the level of annotation needed for the Trimmed Activity Recognition, more densely labelled videos are required for Temporal Action Localization. Action classes can be assigned at every frame of the video or on select frames only (e.g. every N frames). For Spatio-temporal Action Localization, spatial bounding boxes (BB) must be supplied in addition, noting the action classes in each BB.

The contents of the videos also vary widely. In the simplest case, a video depicts a single person performing some action, in other cases multiple people are present simultaneously. Regarding the camera viewpoint, the part of the subject captured may be restricted to the upper body (*Gesture Recognition*), face (*Facial Expression Recognition*), or captured from a first-person viewpoint (*Egocentric Activity Recognition*). Also, the subjects may be situated in different environments which can be broadly categorized as either a *controlled* setting,

TABLE 14.3 List of datasets in Human Activity Recognition. What follows is a list of abbreviations used. **Task**: Trimmed Action Recognition (TAR), Temporal Action Localization (TAL), Spatio-temporal Action Localization (STL); **Viewpoint**: full body visible (F), upper body visible (U), lower body visible (L), egocentric viewpoint 4et(E); **Environment**: controlled (C), wild (W); **Modalities**: intensity (I), red (R), green (G), blue (B), multi-view (MV), depth (D), skeleton (S), infrared (IR), electromyography (EMG); **Size**: videos (v), hours (h), clips (c), frames (f); **Classes**: * action classes × object classes, † verb classes × noun classes, ‡ weakly annotated data

Name	Task	Viewpoint	Env.	Modalities	Size	Classes	Year
ActivityNet 100 [65]	TAL	F, U, L	W	RGB	9,682v	100	2015
ActivityNet 200 [66]	TAL	F, U, L	W	RGB	19,994v (849h)	203	2016
AVA Actions [59]	STL	F, U	W	RGB	437v (392,000f)	80	2018
AVA Kinetics [67]	STL	F	W	RGB	238,474v (624,430f)	80	2020
Charades [68]	TAL	F, U, L	W	RGB	9,848v	157×46*	2016
EPIC-KITCHENS-55 [69]	TAL	E	W	RGB	55h (90,000c)	125×331†	2019
EPIC-KITCHENS-100 [69]	TAL	E	W	RGB	100h (39,594c)	97×300†	2020
EV-Actions [70]	TAR	F	C	RGB+D+S+EMG	7,000v	20	2019
HMDB-51 [71]	TAR	F, U, L	W	RGB	6,766v	51	2011
Hollywood 1 (HOHA) [72]	TAR	F, U, L	W	RGB	211v	8	2008
Hollywood 2 [73]	TAR	F, U, L	W	RGB	2,517v	12	2009
HUMAN4D [74]	TAR	F	C	RGB+D+S	50,306f	19	2020
Jester [75]	TAR	U	W	RGB	148,092v	27	2019
J-HMDB [76]	STL	F, U, L	W	RGB+S	928v	21	2013
Kinetics400 [77]	TAR	F	W	RGB	306,245v	400	2017
Kinetics600 [78]	TAR	F	W	RGB	495,547v	600	2018
Kinetics700 [79]	TAR	F	W	RGB	650,317v	700	2019
KTH [80]	TAR	F	C	I	2,391v	6	2004
MOBISERV-AIIA [81]	TAL, TAR	U	C	RGB-MV	96v (59.68h)	10 (15)	2015
Moments in Time [82]	TAR	F, U	W	RGB	903,964v	339	2019
MPII Cooking [83]	TAL	F, U	W	RGB	273v	65	2012
MultiTHUMOS [57]	TAL	F, U, L	W	RGB	413v (30h)	65	2015
Multi-Moments in Time [84]	TAR, STL	F, U	W	RGB	1,020,000v	313	2019
NTU RGB+D [85]	TAR	F	C	RGB+D+S+IR	56,880v	60	2016
NTU RGB+D 120 [86]	TAR	F	C	RGB+D+S+IR	114,480v	120	2016
Olympic Sports [87]	TAR	F	W	RGB	800v	16	2010
Something-Something V1 [88]	TAR	U, E	W	RGB	108,499v	174	2017
Something-Something V2 [88]	TAR	U, E	W	RGB	220,874v	174	2018
Sports-1M [21]	TAR	F, U, L	W	RGB	1,000,000v	487‡	2014
THUMOS-14 [89]	TAL	F, U, L	W	RGB	254h	20	2014
THUMOS-15 [90]	TAL	F, U, L	W	RGB	430h	20	2015
UCF-11 (YouTube Action) [91]	TAR	F	W	RGB	1,160v	11	2009
UCF-50 [92]	TAR	F, U, L	W	RGB	6,681v	50	2012
UCF-101 [86]	TAR	F, U, L	W	RGB	13,320v	101	2012

or *in the wild*[8]. Datasets in the controlled setting are predominantly created by having subjects perform the action on command in a well-lit space and capturing them with a still camera. This may be regarded as the easier of the two, because the actions themselves constitute the primary source of variation in videos. However, models trained on data from a controlled setting may not generalize well to real life problems, partly because of the low variation in the datasets, and partly because these datasets are often smaller in size than datasets from the wild. Datasets with activities in the wild are typically collected by gathering and annotating existing video clips from movies, YouTube, and the like. They thus

8. It should be noted that "wild" does not refer to wild nature here, and should rather be understood as a synonym for uncontrolled.

exhibit much larger variation of contents, including background, lightning conditions, camera motion, subjects, subject visibility and clip duration. Moreover, due to the relative ease of gathering data they can grow very large, as exemplified by the Kinetics 700 dataset [79], which contains more 650,000 videos.

Finally, the modalities captured for an action also vary. In the simplest case, an action is represented by an RGB video or derived modalities such as Human Skeleton landmarks. In addition, depth information may be captured by a depth sensor, point clouds by Lidar, muscle activity by Electromyography (EMG) readings, kinetic forces by accelerometers, and thermal radiation by Infrared Sensors. A collection of datasets for Human Activity Recognition is listed in Table 14.3.

14.6 CONCLUSION

This concludes our inquiry into the topic of Human Activity Recognition, and its sub-tasks Trimmed Activity Recognition, Temporal Action Localization and Spatio-temporal Action Localization. A plethora of architectures and methods have been devised to solve these tasks, and we have done our best to cover a wide range of them. Greatly inspired by Image Recognition, most methods use convolutional layers to extract features in the spatial dimension. Different approaches have been adopted in the handling of the temporal dimension, including Recursive Neural Networks, 3D convolutions, and separated 2+1D convolutions. These were covered in Section 14.2.1, Section 14.2.2 and Section 14.2.3, and Section 14.2.4 respectively. Beyond the input modality of RGB, a special class of methods use Graph Convolutional Networks (GCNs) to perform action recognition on a graph composed of the coordinates of human body joints (skeleton) that change over time (Section 14.2.5). In Section 14.2.6, we described how multiple modalities can be fused in a multi-stream network architecture, and how multiple resolutions (both spatially and temporally) can be utilised to make recognition more efficient. In Section 14.3 and Section 14.4, the extended tasks of Temporal Localization and Spatio-Temporal Action Localization, which are heavily inspired by Object Detection techniques, were covered. Finally, in Section 14.5 a curated list of datasets for Human Activity Recognition was supplied. The field of Human Activity Recognition is rapidly evolving, with new and improved methods being published on a regular basis. Luckily, there is great culture of transparency, and almost all new publications are made freely available online, just waiting for you to explore the topic further.

BIBLIOGRAPHY

[1] A. Bobick, Philosophical transactions of the royal society b, Movement, activity and action: the role of knowledge in the perception of motion 352 (1358) (1997) 1257–1265.

[2] T. Pfister, J. Charles, A. Zosserman, Flowing convnets for human pose estimation in videos, in: 2015 International Conference on Computer Vision, 2015, pp. 1913–1921.

[3] R. A. Güler, N. Neverova, I. Kokkinos, Densepose: Dense human pose estimation in the wild, in: IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 7297–7306.

[4] M. Ryoo, Human activity prediction: Early recognition of ongoing activities from streaming videos, in: 2011 International Conference on Computer Vision, 2011, pp. 1036–1043.

[5] A. Iosifidis, A. Tefas, I. Pitas, Activity based person identification using fuzzy representation and discriminant learning, IEEE Transactions on Information Forensics and Security 7 (2) (2012) 530–542.

[6] F. Patrona, A. Iosifidis, A. Tefas, I. Pitas, Visual voice activity detection in the wild, IEEE Transactions on Multimedia 18 (6) (2016) 967–977.

[7] M. Holte, C. Tran, M. Trivedi, T. Moeslund, Human action recognition using multiple views: a comparative perspective on recent developments, 2011 Joint ACM Workshop on Human gesture and behavior understanding (2011) 47–52.

[8] A. Iosifidis, A. Tefas, I. Pitas, View-invariant action recognition based on artificial neural networks, IEEE Transactions on Neural Networks and Learning Systems 23 (3) (2012) 412–434.

[9] G. Farnebäck, Two-frame motion estimation based on polynomial expansion, in: Image Analysis, Springer Berlin Heidelberg, 2003, pp. 363–370.

[10] H. Wang, C. Schmid, Action recognition with improved trajectories, in: IEEE International Conference on Computer Vision (ICCV), 2013.

[11] A. Pfister, A. West, S. Bronner, J. A. Noah, Comparative abilities of microsoft kinect and vicon 3d motion capture for gait analysis, Journal of Medical Engineering & Technology (JMET) 38 (2014) 274 – 280.

[12] Z. Cao, G. Hidalgo Martinez, T. Simon, S. Wei, Y. A. Sheikh, Openpose: Realtime multi-person 2d pose estimation using part affinity fields, IEEE Transactions on Pattern Analysis and Machine Intelligence (2019).

[13] M. Krestenitis, N. Passalis, A. Iosifidis, M. Gabbouj, A. Tefas, Recurrent bag-of-features for visual information analysis, Pattern Recognition 106 (107380) (2020) 1–11.

[14] J. Donahue, L. A. Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, T. Darrell, K. Saenko, Long-term recurrent convolutional networks for visual recognition and description, in: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015, pp. 2625–2634.

[15] Joe Yue-Hei Ng, M. Hausknecht, S. Vijayanarasimhan, O. Vinyals, R. Monga, G. Toderici, Beyond short snippets: Deep networks for video classification, in: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015, pp. 4694–4702.

[16] J. Carreira, A. Zisserman, Quo vadis, action recognition? a new model and the kinetics dataset, in: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, pp. 4724–4733.

[17] D. Tran, H. Wang, L. Torresani, J. Ray, Y. LeCun, M. Paluri, A closer look at spatiotemporal convolutions for action recognition, in: IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2018, pp. 6450–6459.

[18] A. Krizhevsky, I. Sutskever, G. E. Hinton, Imagenet classification with deep convolutional neural networks, in: Advances in Neural Information Processing Systems (NIPS), 2012.

[19] D. E. Rumelhart, G. E. Hinton, R. J. Williams, Learning internal representations by error propagation, in: Parallel distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations, 1986.

[20] S. Ji, W. Xu, M. Yang, K. Yu, 3d convolutional neural networks for human action recognition, IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI) 35 (1) (2013) 221–231.

[21] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, L. Fei-Fei, Large-scale video classification with convolutional neural networks, in: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2014, pp. 1725–1732.

[22] D. Tran, L. Bourdev, R. Fergus, L. Torresani, M. Paluri, Learning spatiotemporal features with 3d convolutional networks, in: IEEE International Conference on Computer Vision (ICCV), 2015, pp. 4489–4497.

[23] G. Varol, I. Laptev, C. Schmid, Long-term temporal convolutions for action recognition, IEEE Transactions on Pattern Analysis and Machine Intelligence 40 (6) (2018) 1510–1517.

[24] G. W. Taylor, R. Fergus, Y. LeCun, C. Bregler, Convolutional learning of spatio-temporal features, in: K. Daniilidis, P. Maragos, N. Paragios (Eds.), European Conference on Computer Vision (ECCV), 2010, pp. 140–153.

[25] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, in: International Conference on Learning Representations (ICLR), 2015.

[26] S. Ioffe, C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, Proceedings of Machine Learning Research, PMLR, 2015.

[27] N. Kanopoulos, N. Vasanthavada, R. L. Baker, Design of an image edge detection filter using the sobel operator, IEEE Journal of solid-state circuits (1988).

[28] Z. Qiu, T. Yao, T. Mei, Learning spatio-temporal representation with pseudo-3d residual networks, in: International Conference on Computer Vision (ICCV), 2017.

[29] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.

[30] F. Han, B. Reily, W. Hoff, H. Zhang, Space-time representation of people based on 3D skeletal data: A review, Computer Vision and Image Understanding 158 (2017) 85–105.

[31] A. Shahroudy, J. Liu, T.-T. Ng, G. Wang, NTU RGB+D: A large scale dataset for 3D human activity analysis, in: IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 1010–1019.

[32] J. Liu, A. Shahroudy, D. Xu, G. Wang, Spatio-temporal LSTM with trust gates for 3D human action recognition, in: European Conference on Computer Vision, Springer, 2016, pp. 816–833.

[33] P. Zhang, C. Lan, J. Xing, W. Zeng, J. Xue, N. Zheng, View adaptive recurrent neural networks for high performance human action recognition from skeleton data, in: IEEE International Conference on Computer Vision, 2017, pp. 2117–2126.

[34] S. Song, C. Lan, J. Xing, W. Zeng, J. Liu, An end-to-end spatio-temporal attention model for human action recognition from skeleton data, in: AAAI Conference on Artificial Intelligence, 2017, pp. 4263–4270.

[35] T. S. Kim, A. Reiter, Interpretable 3D human action analysis with temporal convolutional networks, in: IEEE Conference on Computer Vision and Pattern Recognition Workshops, 2017, pp. 1623–1631.

[36] Q. Ke, M. Bennamoun, S. An, F. Sohel, F. Boussaid, A new representation of skeleton sequences for 3D action recognition, in: IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 3288–3297.

[37] M. Liu, H. Liu, C. Chen, Enhanced skeleton visualization for view invariant human action recognition, Pattern Recognition 68 (2017) 346–362.

[38] B. Li, Y. Dai, X. Cheng, H. Chen, Y. Lin, M. He, Skeleton based action recognition using translation-scale invariant image mapping and multi-scale deep CNN, in: IEEE International Conference on Multimedia & Expo Workshops, 2017, pp. 601–604.

[39] C. Li, Q. Zhong, D. Xie, S. Pu, Skeleton-based action recognition with convolutional neural networks, in: IEEE International Conference on Multimedia & Expo Workshops, 2017, pp.

597–600.

[40] T. N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, International Conference on Learning Representations (2017).

[41] S. Yan, Y. Xiong, D. Lin, Spatial temporal graph convolutional networks for skeleton-based action recognition, in: AAAI Conference on Artificial Intelligence, 2018.

[42] L. Shi, Y. Zhang, J. Cheng, H. Lu, Two-stream adaptive graph convolutional networks for skeleton-based action recognition, in: IEEE Conference on Computer Vision and Pattern Recognition, 2019.

[43] L. Shi, Y. Zhang, J. Cheng, H. Lu, Skeleton-based action recognition with directed graph neural networks, in: IEEE Conference on Computer Vision and Pattern Recognition, 2019.

[44] W. Peng, X. Hong, H. Chen, G. Zhao, Learning graph convolutional network for skeleton-based human action recognition by neural searching., in: AAAI Conference on Artificial Intelligence, 2020.

[45] P. Zhang, C. Lan, W. Zeng, J. Xing, J. Xue, N. Zheng, Semantics-guided neural networks for efficient skeleton-based human action recognition, in: IEEE Conference on Computer Vision and Pattern Recognition, 2020.

[46] K. Cheng, Y. Zhang, X. He, W. Chen, J. Cheng, H. Lu, Skeleton-based action recognition with shift graph convolutional network, in: IEEE Conference on Computer Vision and Pattern Recognition, 2020.

[47] Y. Tang, Y. Tian, J. Lu, P. Li, J. Zhou, Deep progressive reinforcement learning for skeleton-based action recognition, in: IEEE Conference on Computer Vision and Pattern Recognition, 2018.

[48] N. Heidari, A. Iosifidis, Temporal attention-augmented graph convolutional network for efficient skeleton-based human action recognition, in: International Conference on Pattern Recognition, 2020.

[49] N. Heidari, A. Iosifidis, Progressive spatio-temporal graph convolutional network for skeleton-based human action recognition, arXiv preprint arXiv:2011.05668 (2020).

[50] N. Heidari, A. Iosifidis, On the spatial attention in spatio-temporal graph convolutional networks for skeleton-based human action recognition, arXiv preprint arXiv:2011.03833 (2020).

[51] M. A. Goodale, A. Milner, Separate visual pathways for perception and action, Trends in Neurosciences (1992).

[52] K. Simonyan, A. Zisserman, Two-stream convolutional networks for action recognition in videos, in: Advances in Neural Information Processing Systems, 2014.

[53] M. Li, S. Chen, X. Chen, Y. Zhang, Y. Wang, Q. Tian, Actional-structural graph convolutional networks for skeleton-based action recognition, in: IEEE Conference on Computer Vision and Pattern Recognition, 2019.

[54] C. Feichtenhofer, A. Pinz, A. Zisserman, Convolutional two-stream network fusion for video action recognition, in: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.

[55] C. Feichtenhofer, H. Fan, J. Malik, K. He, Slowfast networks for video recognition, in: IEEE/CVF International Conference on Computer Vision (ICCV), 2019.

[56] L. Zhu, D. Tran, L. Sevilla-Lara, Y. Yang, M. Feiszli, H. Wang, FASTER recurrent networks for efficient video classification, in: AAAI Conference on Artificial Intelligence, 2020, pp. 13098–13105.

[57] S. Yeung, O. Russakovsky, N. Jin, M. Andriluka, G. Mori, L. Fei-Fei, Every moment counts: Dense detailed labeling of actions in complex videos, International Journal of Computer Vision (IJCV) (2017).

[58] A. Piergiovanni, M. Ryoo, Temporal Gaussian mixture layer for videos, in: Proceedings of

Machine Learning Research (PMLR), Vol. 97, 2019, pp. 5152–5161.

[59] C. Gu, C. Sun, D. A. Ross, C. Vondrick, C. Pantofaru, Y. Li, S. Vijayanarasimhan, G. Toderici, S. Ricco, R. Sukthankar, C. Schmid, J. Malik, Ava: A video dataset of spatio-temporally localized atomic visual actions, in: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018.

[60] S. Ren, K. He, R. Girshick, J. Sun, Faster r-cnn: Towards real-time object detection with region proposal networks, in: Advances in Neural Information Processing Systems 28, 2015.

[61] R. Girshick, Fast r-cnn, in: IEEE International Conference on Computer Vision (ICCV), 2015.

[62] K. He, G. Gkioxari, P. Dollár, R. Girshick, Mask r-cnn, in: IEEE International Conference on Computer Vision (ICCV), 2017.

[63] X. Zhou, D. Wang, P. Krähenbühl, Objects as points, in: arXiv preprint arXiv:1904.07850, 2019.

[64] Y. Li, Z. Wang, L. Wang, G. Wu, Actions as moving points, in: European Conference on Computer Vision (ECCV), 2020.

[65] F. C. Heilbron, J. C. Niebles, Collecting and annotating human activities in web videos, in: Proceedings of International Conference on Multimedia Retrieval, 2014, p. 377.

[66] F. C. Heilbron, V. Escorcia, B. Ghanem, J. C. Niebles, Activitynet: A large-scale video benchmark for human activity understanding, in: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015, pp. 961–970.

[67] A. Li, M. Thotakuri, D. A. Ross, J. Carreira, A. Vostrikov, A. Zisserman, The ava-kinetics localized human actions video dataset, in: arXiv:2005.00214, 2020.

[68] G. A. Sigurdsson, G. Varol, X. Wang, A. Farhadi, I. Laptev, A. Gupta, Hollywood in homes: Crowdsourcing data collection for activity understanding, in: European Conference on Computer Vision (ECCV), 2016.

[69] D. Damen, H. Doughty, G. M. Farinella, S. Fidler, A. Furnari, E. Kazakos, D. Moltisanti, J. Munro, T. Perrett, W. Price, M. Wray, The epic-kitchens dataset: Collection, challenges and baselines, IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI) (2020).

[70] L. Wang, B. Sun, J. Robinson, T. Jing, Y. Fu, Ev-action: Electromyography-vision multi-modal action dataset, in: IEEE International Conference on Automatic Face and Gesture Recognition, 2020.

[71] H. Kuehne, H. Jhuang, E. Garrote, T. Poggio, T. Serre, Hmdb: A large video database for human motion recognition, in: 2011 International Conference on Computer Vision, 2011, pp. 2556–2563.

[72] I. Laptev, M. Marszalek, C. Schmid, B. Rozenfeld, Learning realistic human actions from movies, in: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2008, pp. 1–8.

[73] M. Marszałek, I. Laptev, C. Schmid, Actions in context, in: IEEE Conference on Computer Vision & Pattern Recognition (CVPR), 2009.

[74] A. Chatzitofis, L. Saroglou, P. Boutis, P. Drakoulis, N. Zioulis, S. Subramanyam, B. Kevelham, C. Charbonnier, P. Cesar, D. Zarpalas, S. Kollias, P. Daras, Human4d: A human-centric multimodal dataset for motions and immersive media, IEEE Access 8 (2020) 176241–176262.

[75] J. Materzynska, G. Berger, I. Bax, R. Memisevic, The jester dataset: A large-scale video dataset of human gestures, in: IEEE/CVF International Conference on Computer Vision Workshop (ICCVW), 2019, pp. 2874–2882.

[76] H. Jhuang, J. Gall, S. Zuffi, C. Schmid, M. J. Black, Towards understanding action recognition, in: International Conference on Computer Vision (ICCV), 2013, pp. 3192–3199.

[77] W. Kay, J. Carreira, K. Simonyan, B. Zhang, C. Hillier, S. Vijayanarasimhan, F. Viola, T. Green, T. Back, P. Natsev, M. Suleyman, A. Zisserman, The kinetics human action video dataset, in:

arXiv:1705.06950, 2017.

[78] J. Carreira, E. Noland, A. Banki-Horvath, C. H. A. Zisserman, A short note about kinetics-600, in: arXiv:1808.01340, 2018.

[79] J. Carreira, E. Noland, C. Hillier, A. Zisserman, A short note on the kinetics-700 human action dataset, in: arXiv:1907.06987, 2019.

[80] I. Laptev, T. Lindeberg, Velocity adaptation of space-time interest points, in: International Conference on Pattern Recognition (ICPR), 2004, pp. 52–56.

[81] A. Iosifidis, E. Marami, A. Tefas, I. Pitas, The mobiserv-aiia eating and drinking multi-view database for vision-based assisted living, Journal of Information Hiding and Multimedia Signal Processing 6 (2) (2015) 254–273.

[82] M. Monfort, A. Andonian, B. Zhou, K. Ramakrishnan, S. A. Bargal, T. Yan, L. Brown, Q. Fan, D. Gutfruend, C. Vondrick, Moments in time dataset: one million videos for event understanding, IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI) (2019) 1–8.

[83] M. Rohrbach, S. Amin, M. Andriluka, B. Schiele, A database for fine grained activity detection of cooking activities, in: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2012, pp. 1194–1201.

[84] M. Monfort, K. Ramakrishnan, A. Andonian, B. A. McNamara, A. Lascelles, B. Pan, Q. Fan, D. Gutfreund, R. Feris, A. Oliva, Multi-moments in time: Learning and interpreting models for multi-action video understanding, in: arXiv:1911.00232, 2019.

[85] A. Shahroudy, J. Liu, T.-T. Ng, G. Wang, Ntu rgb+d: A large scale dataset for 3d human activity analysis, in: IEEE Conference on Computer Vision and Pattern Recognition, 2016.

[86] J. Liu, A. Shahroudy, M. Perez, G. Wang, L.-Y. Duan, A. 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).

[87] J. C. Niebles, C.-W. Chen, L. Fei-Fei, Modeling temporal structure of decomposable motion segments for activity classification, in: European Conference on Computer Vision (ECCV), 2010, pp. 392–405.

[88] R. Goyal, S. E. Kahou, V. Michalski, J. Materzynska, S. Westphal, H. Kim, V. Haenel, I. Fruend, P. Yianilos, M. Mueller-Freitag, F. Hoppe, C. Thurau, I. Bax, R. Memisevic, The “something something” video database for learning and evaluating visual common sense, in: IEEE International Conference on Computer Vision (ICCV), 2017, pp. 5843–5851.

[89] Y.-G. Jiang, J. Liu, A. R. Zamir, G. Toderici, M. Laptev, Ivan anShah, R. Sukthankar, Thumos challenge 2014, 2014.
URL http://crcv.ucf.edu/THUMOS14/

[90] A. Gorban, H. Idrees, Y.-G. Jiang, A. R. Zamir, M. Laptev, Ivaand Shah, R. Sukthankar, Thumos challenge 2015, 2015.
URL http://www.thumos.info

[91] J. Liu, J. Luo, M. Shah, Recognizing realistic actions from videos "in the wild", in: IEEE International Conference on Computer Vision and Pattern Recognition(CVPR), 2009.

[92] K. Reddy, M. Shah, Recognizing 50 human action categories of web videos, Machine Vision and Applications (MVAP) 24 (2012) 971–981.

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.

Chapter 4

Graph Convolutional Networks

Negar Heidari[1], Lukas Hedegaard[2] and Alexandros Iosifidis[3]

ABSTRACT

Deep learning approaches have been very successful in many machine learning tasks including compute vision, natural language processing, audio processing and speech recognition. However, deep neural networks typically work with grid-structured data represented in the Euclidean space and despite their recent successes, they poorly generalize to applications where the data is represented in non-Euclidean space. Recently, due to the increasing amount of graph structured data produced in different areas such as social networks, stock markets and knowledge bases, there is an increasing need to develop learning methods capable of capturing the relational structures in graph data. Recently, graph neural networks have attracted great research attention as they have demonstrated the ability to provide high performance in many of the learning tasks with non-Euclidean data structures. In this chapter, we provide a detailed overview of graph convolutional networks, which extend the convolution operation to graph structured data. We group the existing graph convolutional networks in four different categories, and we provide a discussion on their efficiency and scalability in large graph structures. The application of these networks in different learning tasks is also discussed in this chapter, along with a summary of the existing benchmark datasets and open-source libraries.

KEYWORDS

Graph Neural Network, Graph Convolutional Network, Graph Attention Network, Node Classification, Graph Classification, Graph Sampling

4.1 INTRODUCTION

With the rise of internet and social media, a huge amount of data has been created by the users so that graph structured data can be seen everywhere around us. For instance, the data created by social networks, citation networks and knowledge bases are modeled as graphs of enormous sizes, as the corresponding graphs are formed by billions of nodes and edges. Accordingly, it is important to develop learning methods that are able to capture the patterns appearing in the

1. Department of Electrical and Computer Engineering, Aarhus University, Denmark
2. Department of Electrical and Computer Engineering, Aarhus University, Denmark
3. Department of Electrical and Computer Engineering, Aarhus University, Denmark

graph-structured data and infer the information encoded by them. Deep neural networks have been very successful in processing Euclidean data structures, such as audio, text, images and videos, which are formed by regular 1D, 2D and 3D grids, in different learning tasks such as natural language processing [1, 2], speech recognition [3] and computer vision [4, 5]. However, neural networks used for processing Euclidean data structures are not suited for processing data that do not come in a regular grid structure.

Graph Neural Networks (GNNs) have attracted an increasing research interest and they have been very successful in processing non-Euclidean data structures, such as graphs, in different application areas. Many methods have been developed to extend RNNs, CNNs and MLPs to process graph structured data. These methods are categorized into Recurrent GNNs (RecGNNs), Graph Convolutional Networks (GCNs) and Graph Autoencoders (GAEs), respectively. The first GNN methods were RecGNNs [6, 7, 8, 9], which learn feature representations for the graph nodes by passing information between each node and its neighbors until reaching equilibrium. This idea of message passing was used in GCNs as well. GCNs [10, 11, 12, 13, 14, 15] build a multi-layer neural network, formed by layers that learn feature representations for the graph nodes by applying graph convolution. These representations are employed for classification of individual nodes or the entire graph. The graph convolution layer can be defined based on spectral graph theory or the message passing notion in RecGNNs. GAEs are neural networks trained in an unsupervised manner to reconstruct the structural information of the graph, such as its nodes, its edges, or the graph Adjacency matrix for graph generation [16, 17, 18, 19] or to learn feature representations for graph nodes and edges [20, 21, 22, 23].

In problems where the input data forms a graph structure which is updated over regular time intervals, a spatio-temporal graph structure is created. The spatio-temporal graph can be defined as a sequence of graphs having two types of graph node connections, i.e. connections between the nodes in each graph which form the set of spatial edges, and connections between the nodes at different time steps which form the set of temporal edges. The Spatio-temporal Graph Convolutional Networks (ST-GCN) are able to process spatio-temporal graphs. They are multi-layer neural networks which capture both spatial structure and temporal dynamics of the underlying spatio-temporal graph by applying spatial and temporal graph convolutions at each layer. Recently, ST-GCNs attracted a great research interest in computer vision tasks such as skeleton-based human action recognition where an action is represented by a sequence of human body poses. Each body pose is represented by the arrangement of spatially connected human body joints, and the sequence of skeletons is thus modeled as a spatio-temporal graph.

Using graph structured datasets, three main learning tasks can be formulated, which are briefly described as follows:

- **Node Classification:** The goal of node classification methods is to predict

a label for each graph node in a supervised or semi-supervised setting. In these methods, the GCN model extracts high-level feature representation for each node in a supervised setting or propagates the label information from labeled nodes through the whole graph in a semi-supervised setting. Using the extracted high-level node features, a label will be predicted for each node by a classification layer.

- **Graph Classification:** The methods targeting graph classification tasks use a pooling layer before the classification layer in order to produce a feature vector aggregating information of the whole graph, which is then introduced to the classification layer predicting a label. The pooling layer can be a simple global average pooling which produces a mean vector of all the node features.
- **Edge Prediction and Classification:** The goal of edge prediction and classification tasks is either to generate a graph by predicting the edges between nodes and building a new graph Adjacency matrix, or to learn feature representations for the graph edges.

In this chapter, we focus on GCNs and introduce the most representative methods developed in this area to generalize the notion of convolution from grid structured data to graph structured data. GCN methods are mainly categorized into spectral methods, which are based on spectral graph theory, and spatial methods. In this section, we start with a formal definition a graph. In Section 4.2, spectral-based GCN and the most representative methods in this category are covered. Section 4.3, introduces spatial-based GCN methods and it is followed by a comparison of spatial and spectral-based GCN methods related to their potential advantages and disadvantages. In Section 4.4, the attention based GCN is discussed. In Section 4.5, our focus is on the efficiency and scalability of the GCN methods and we introduce methods which are able to process large graphs. Finally, an overview of benchmark graph datasets used in different GNN tasks and the available software libraries for implementing GCNs are provided in Section 4.6.

4.1.1 Graph Definition

A graph is formally defined as $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{v_i\}_{i=1}^{N}$ denotes the set of N graph nodes and $\mathcal{E} = \{e_{ij} \in \mathcal{V} \times \mathcal{V},\ i, j = 1, \ldots, N\}$ denotes the set of graph edges expressing the connection between each pair of nodes (v_i, v_j). The graph Adjacency matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$ encodes pair-wise connections between the nodes through the edges and it is defined as follows:

$$\mathbf{A}_{ij} = \begin{cases} 1, & \text{if } i \neq j \text{ and } e_{ij} \in \mathcal{E}, \\ 0, & \text{otherwise.} \end{cases} \tag{4.1}$$

When $\mathbf{A}$ is symmetric, i.e. when for every edge $e_{ij} \in \mathcal{E}$ connecting nodes v_i and v_j the corresponding edge e_{ji} also belongs to $\mathcal{E}$, G is an undirected graph. When this is not true, G is a directed graph. When the connections between nodes are

associated with real values encoding their strength, the graph Adjacency matrix is replaced with the corresponding graph Weight matrix[4] formed by real values. In that case, G is a weighted graph. In the following, we consider undirected and unweighted graphs, i.e. graphs defined by symmetric Adjacency matrices.

The degree of each node $deg(v_i)$ indicates the number of its neighbors which are directly connected to it with an edge. The graph Degree matrix $\mathbf{D} \in \mathbb{R}^{N\times N}$ is a diagonal matrix which gives us information about the node degrees, and it is defined as follows:

$$\mathbf{D}_{ij} = \begin{cases} \sum_j \mathbf{A}_{ij}, & \text{if } i = j, \\ 0, & \text{otherwise.} \end{cases} \tag{4.2}$$

Based on this definition, the neighborhood of a node is defined as

$$\mathcal{N}(v_i) = \left\{ v_j \in \mathcal{V} | e_{ij} \in \mathcal{E} \right\}. \tag{4.3}$$

Moreover, each graph node can be represented by a feature vector $\mathbf{x}_i \in \mathbb{R}^D$, leading to the feature matrix of the graph denoted as $\mathbf{X} \in \mathbb{R}^{N\times D}$ storing the feature vectors of all nodes. When the graph feature matrix is not available, information encoded in the graph edge set $\mathcal{E}$ can be exploited to calculate one through graph embedding methods such as DeepWalk [24], node2vec [25] and LINE [26]. The output of graph embedding methods is a set of feature vectors which can be used by GCNs. Here we should note that there exists a variety of graph embedding methods which have been proposed for a wide range of problems, like data visualisation based on neural networks [27, 28, 29], data visualisation based on matrix factorization [30, 31, 32], subspace learning based on spectral analysis [33] and classification based on graph structures [34, 35], which are beyond the scope of this chapter. For more information on these methods we refer the reader to related review articles [36, 37].

4.2 SPECTRAL GRAPH CONVOLUTIONAL NETWORK

Spectral GCNs are based on graph signal processing [38, 39, 40] where the signals are defined on undirected graphs. In other words, a graph function, or graph signal, is defined as a mapping from each node of the graph to a real value. Given the graph Adjacency and the graph Degree matrices, the graph Laplacian matrix is defined as $\mathbf{L} = \mathbf{D} - \mathbf{A}$ so that:

$$\mathbf{L}_{ij} = \begin{cases} deg(v_i), & \text{if } i = j, \\ -1, & \text{if } i \neq j \text{ and } (v_i, v_j) \in \mathcal{E}, \\ 0, & \text{otherwise.} \end{cases} \tag{4.4}$$

$\mathbf{L}$ encodes the graph connectivity and determines the smoothness of the graph function. In a smooth graph function, its value does not vary a lot across

4. The graph Weight matrix should not be confused with the weight matrix of a neural layer used in MLPs.

connected nodes. This can be expressed formally by the Dirichlet energy of a graph function [41, 42]:

$$E(\mathbf{f}) = \frac{1}{2}\sum_{i,j}\mathbf{A}_{ij}\left(\mathbf{f}(v_i) - \mathbf{f}(v_j)\right)^2, \tag{4.5}$$

where $\mathbf{f}$ denotes the graph function which receives as input a node of the graph and provides as output a real value. A smooth graph function $\mathbf{f}$ can be obtained by minimizing the Dirichlet energy $E(\mathbf{f})$. This is equivalent to minimizing the difference of the graph function on pairs of connected nodes. By expanding Eq. (4.5), it can be seen that the graph Laplacian matrix is encountered in this definition:

$$E(\mathbf{f}) = \frac{1}{2}\sum_{i,j}\mathbf{A}_{ij}\left(\mathbf{f}(v_i) - \mathbf{f}(v_j)\right)^2 \tag{4.6}$$

$$= \frac{1}{2}\left(\sum_i\sum_j \mathbf{A}_{ij}\mathbf{f}(v_i)^2 - 2\sum_{i,j}\mathbf{A}_{ij}\mathbf{f}(v_i)\mathbf{f}(v_j) + \sum_j\sum_i \mathbf{A}_{ij}\mathbf{f}(v_j)^2\right) \tag{4.7}$$

$$= \sum_i \mathbf{D}_{ii}\mathbf{f}(v_i)^2 - \sum_{i,j}\mathbf{A}_{ij}\mathbf{f}(v_i)\mathbf{f}(v_j) \tag{4.8}$$

$$= \mathbf{f}^\top\mathbf{D}\mathbf{f} - \mathbf{f}^\top\mathbf{A}\mathbf{f} = \mathbf{f}^\top(\mathbf{D} - \mathbf{A})\mathbf{f} = \mathbf{f}^\top\mathbf{L}\mathbf{f} \tag{4.9}$$

The smoothness of the graph function can also be expressed by the eigenvectors and eigenvalues of the normalized graph Laplacian matrix $\tilde{\mathbf{L}}$, which is defined as:

$$\tilde{\mathbf{L}} = \mathbf{D}^{-1/2}\mathbf{L}\mathbf{D}^{-1/2} = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2} \tag{4.10}$$

and its elements are:

$$\tilde{\mathbf{L}}_{ij} = \begin{cases} 1, & \text{if } i = j \text{ and } deg(v_i) \neq 0, \\ \frac{-1}{\sqrt{deg(v_i)deg(v_j)}}, & \text{if } i \neq j \text{ and } (v_i, v_j) \in \mathcal{E}, \\ 0, & \text{otherwise.} \end{cases} \tag{4.11}$$

Since $\tilde{\mathbf{L}}$ is a square matrix, we can use eigendecomposition $\tilde{\mathbf{L}} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top$, where $\mathbf{\Lambda}$ is a diagonal matrix of eigenvalues, i.e. $\mathbf{\Lambda}_{ii} = \lambda_i$, sorted in an ascending order and $\mathbf{U} = [\mathbf{u}_0, \mathbf{u}_1, ..., \mathbf{u}_{N-1}]$ is the set of eigenvectors ordered according to the eigenvalues. Because $\tilde{\mathbf{L}}$ is symemetric and positive semidefinite, $0 \leq \lambda_0 < \lambda_1 ... \leq \lambda_{N-1}$. The eigenvector $\mathbf{u}_0$ associated with the smallest eigenvalue λ_0, oscillates very slowly across the graph which means that when two nodes are connected by an edge, the values of this eigenvector corresponding to those two nodes are very similar. On the other hand, the eigenvector with the largest eigenvalue, i.e. $\mathbf{u}_{N-1}$, oscillates more rapidly and its values are more likely to be dissimilar in connected nodes.

Using the Fourier transform, the signal can be equivalently represented in both vertex domain and graph spectral domain. Let us assume that $\mathbf{x} \in \mathbb{R}^N$ is

the graph signal in the vertex domain where $\mathbf{x}_i$ denotes the signal value in the i^{th} node v_i. The graph signal $\mathbf{x}$ can be projected into the orthonormal space formed by the normalized graph Laplacian matrix eigenvectors $\mathbf{U}$ using the Fourier transform as $\mathcal{F}(\mathbf{x}) = \mathbf{U}^\top\mathbf{x}$. The graph convolution of the signal $\mathbf{x}$ with a filter $\mathbf{g}$ is defined as $\mathbf{x} * \mathbf{g}$. The spectral convolution in the orthonormal space can be obtained by applying the Fourier transform and its inverse on the graph convolution as follows:

$$\mathbf{x} * \mathbf{g} = \mathcal{F}^{-1}(\mathcal{F}(\mathbf{x}) \odot \mathcal{F}(\mathbf{g})) = \mathbf{U}(\mathbf{U}^\top\mathbf{x} \odot \mathbf{U}^\top\mathbf{g}), \tag{4.12}$$

where $\odot$ denotes the element-wise product. The parametrized filter in the spectral domain is denoted as $\mathbf{g}_\theta = \text{diag}(\mathbf{U}^\top\mathbf{g})$ and the spectral convolution takes the form:

$$\mathbf{x} * \mathbf{g}_\theta = \mathbf{U}\mathbf{g}_\theta\mathbf{U}^\top\mathbf{x}. \tag{4.13}$$

Considering the eigen-decomposition of the normalized graph Laplacian matrix, the spectral convolution in Eq. (4.13) is represented as multiplication of signal $\mathbf{x}$ with a parameterized function of normalized graph Laplacian, $\mathbf{U}\mathbf{g}_\theta\mathbf{U}^\top$, and the filter $\mathbf{g}_\theta$ can be seen as a function of normalized Laplacian eigenvalues $\mathbf{\Lambda}$, i.e. $\mathbf{g}_\theta(\mathbf{\Lambda})$. Computing the eigen-decomposition of $\tilde{\mathbf{L}}$ has a complexity of $O(N^3)$ and the multiplication of the signal with the eigenvector matrix $\mathbf{U}$ has a complexity of $O(N^2)$. Thus, computation of the spectral convolution through Eq. (4.13) is computationally expensive, especially for large graphs. To reduce the computational complexity, it is possible to approximate the filter $\mathbf{g}_\theta(\mathbf{\Lambda})$ with a K^{th} order expansion of Chebyshev polynomials as follows [43]:

$$\mathbf{g}_\theta(\mathbf{\Lambda}) = \sum_{k=0}^{K-1} \theta_k T_k(\hat{\mathbf{\Lambda}}), \tag{4.14}$$

where $\theta \in \mathbb{R}^K$ is a vector of polynomial coefficients and $\hat{\mathbf{\Lambda}} = \frac{2}{\lambda_{\text{max}}}\mathbf{\Lambda} - \mathbf{I}_N$ is a diagonal matrix with the eigenvalues normalized in the range $[-1, 1]$. The Chebyshev polynomials of $\hat{\mathbf{\Lambda}}$ are defined recursively as $T_k(\hat{\mathbf{\Lambda}}) = 2\hat{\mathbf{\Lambda}}T_{k-1}(\hat{\mathbf{\Lambda}}) - T_{k-2}(\hat{\mathbf{\Lambda}})$ with $T_0(\hat{\mathbf{\Lambda}}) = 1$ and $T_1(\hat{\mathbf{\Lambda}}) = \hat{\mathbf{\Lambda}}$.

Considering that $(\mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top)^k = \mathbf{U}\mathbf{\Lambda}^k\mathbf{U}^\top$, the spectral convolution in Eq. (4.13) can be defined by the truncated expansion of Chebyshev polynomials of the normalized graph Laplacian matrix as follows:

$$\mathbf{x} * \mathbf{g} = \sum_{k=0}^{K-1} \theta_k T_k(\hat{\mathbf{L}})\mathbf{x}, \tag{4.15}$$

where $\hat{\mathbf{L}} = \frac{2}{\lambda_{\text{max}}}\tilde{\mathbf{L}} - \mathbf{I}_N$, and $T_k(\hat{\mathbf{L}}) = \mathbf{U}T_k(\hat{\mathbf{\Lambda}})\mathbf{U}^\top$. Using Eq. (4.15) instead of Eq. (4.13) the computation of the spectral convolution operation becomes more efficient, as the polynomial function is calculated recursively. It depends only on the K^{th}-hop neighborhood of each node (i.e. it is K-localized) and the

computation has a complexity of $O(|\mathcal{E}|)$, which is independent to the graph size. ChebNet [11] uses this definition for spectral convolution on graphs.

The GCN method [12] simplifies the spectral graph convolution by a first-order approximation of Chebyshev polynomials, to avoid overfitting on the local neighborhood structure of graphs. By using the values $K = 2$ and $\lambda_{\max} = 2$ in Eq. (4.15), the simplified convolution is defined as follows:

$$\mathbf{x} * \mathbf{g} \approx \theta_0 \mathbf{x} - \theta_1 (\tilde{\mathbf{L}} - \mathbf{I}_N)\mathbf{x} = \theta_0 \mathbf{x} - \theta_1 (\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2})\mathbf{x}. \tag{4.16}$$

To avoid overfitting, it constrains the number of parameters by setting $\theta' = \theta_0 = -\theta_1$ leading to the following formulation for spectral convolution:

$$\mathbf{x} * \mathbf{g} \approx \theta' (\mathbf{I}_N + \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2})\mathbf{x}. \tag{4.17}$$

Since the eigenvalues of $\mathbf{I}_N + \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ are in range $[0, 2]$, employing this operation in a neural network leads to numerical instabilities. Therefore, it is replaced by $\tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}$ where $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}_N$ is the graph Adjacency matrix obtained by introducing self-loops, i.e. edges that connect each graph node to itself, and $\tilde{\mathbf{D}}$ is the graph Degree matrix of $\tilde{\mathbf{A}}$.

The GCN method [12] proposed a two-layer network for semi-supervised node classification by stacking two GCN layers, each of which is built using the convolution operation defined in Eq. (4.17). The first layer receives as input the input feature matrix of the graph nodes $\mathbf{X} \in \mathbb{R}^{N \times D}$ and performs the following transformation:

$$\mathbf{H} = \sigma(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{X}\mathbf{W}) \tag{4.18}$$

where $\mathbf{W}$ is the matrix of filter parameters which should be optimized by training the GCN model, and $\mathbf{H}$ is the matrix storing the transformed features produced by the layer. The graph conovlution is followed by an activation function σ which is applied in an element-wise manner. The second layer receives as input the feature matrix $\mathbf{H}$ and performs another convolution operation, followed by an element-wise activation function. Because the goal of the method is to perform classification, the activation function used for the second layer is the softmax function. The activation function used for the first layer is the Rectified Linear Unit (ReLU) function. The two layer GCN model leads to:

$$\mathbf{Y} = \text{softmax}\Big(\hat{\mathbf{A}}\ \text{ReLU}\left(\hat{\mathbf{A}}\mathbf{X}\mathbf{W}^{(1)}\right)\mathbf{W}^{(2)}\Big), \tag{4.19}$$

where $\hat{\mathbf{A}} = \tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}$ and $\mathbf{W}^{(1)}$, $\mathbf{W}^{(2)}$ are the learnable filter parameters of the first and second layers, respectively. The parameters of the GCN model are trained in an end-to-end manner using the Backpropagation algorithm, or its variants.

Accordingly, the spectral convolution formulation proposed by the GCN method [12] bridged the gap between spectral and spatial convolution by utilizing the graph Adjacency matrix for feature aggregation. In other words, the

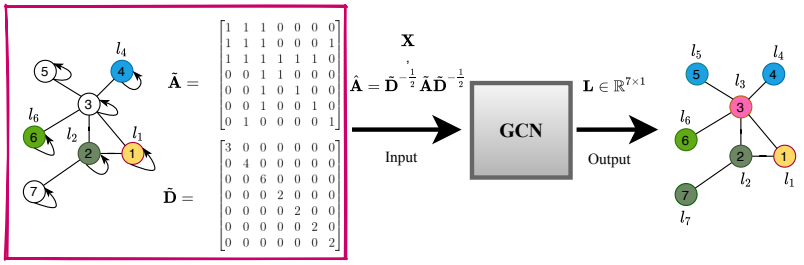


FIGURE 4.1 Illustration of a simple undirected graph with its corresponding graph Adjacency and graph Degree matrices. Only a subset of nodes in the input graphs are labeled with l_i and the remaining nodes are unlabeled. The normalized graph Adjacency matrix and feature matrix are introduced into a GCN model. The model is trained in a semi-supervised manner and propagates the information of the labeled nodes through the whole graph. The output of the model is a vector of labels $L = [l_i \mid i = 1, \ldots, 7]$ containing the predicted label l_i for each graph node.

convolution operation in Eq. (4.18) can be seen as a propagation rule which updates the features of each node and it can be used for semi-supervised node classification by propagating the infromation from labeled nodes to all other nodes in the graph. To make this more clear, we explain it with a simple example. Let us assume that the graph illustrated in Fig. 4.1 is the graph introduced to the GCN model. In this graph, only a subset of nodes are labeled and the goal is to predict labels for all the nodes in the graph. In order to update the features of each node, we need to aggregate the features of the node itself and the features of its neighbors. Therefore, in the illustrated graph each node has a self connection which explains the use of $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}_N$ instead of $\mathbf{A}$. As can be seen in Fig. 4.1, the nodes in the graph may have different degrees. Using the matrix $\tilde{\mathbf{A}}$ for feature propagation would lead to nodes with large degrees having larger values in their feature vectors compared to nodes with smaller degrees. This issue can be solved by normalizing the graph Adjacency matrix, explaining the use of $\hat{\mathbf{A}} = \tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}$. In this way, the connection between each pair of nodes is normalized by the degrees of the two nodes. The normalized graph Adjacency matrix and the feature matrix of the input graph are introduced to a two-layer GCN model which updates the features of each node by performing the propagation rule in Eq. (4.18). As a result, the extracted features are transformed into a C-dimensional space for a C-class classification problem and a label is predicted for each node based on the pseudo-probability values obtained at the output of the second layer. Here we should note that there is no restriction in the number of layers used to build the GCN model, i.e. one can use multiple layers, stacked in a hierarchical manner, to form a multi-layer GCN model. Recently, PGCN method [44] has been proposed to automatically design a problem-specific GCN topology by jointly growing the network's structure both in width and depth in a progressive manner, and optimizing its parameters.

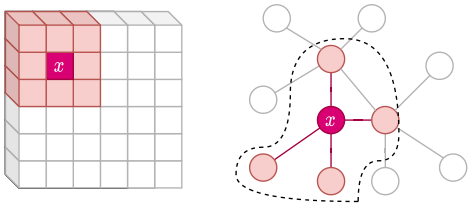


FIGURE 4.2 Left: 2D convolution on a grid where a grid-shaped filter is placed on the 2 dimensional grid data to update the features of the node x by aggregating the features of its ordered neighbors. Right: spatial graph convolution, where the feature representation of the node x is updated by aggregating the features of its neighbors.

One of the main drawbacks of the spectral graph convolutions is that the learned filters in these methods are dependent to the eigenbasis of the underlying graph. Therefore, the learned parameters for a graph cannot be applied to another graph with a different structure. Besides, since these methods need to utilize the whole graph structure (as graph Adjacency matrix) in the convolution operation, they are not scalable to large graphs.

Several methods have been proposed recently such as Adaptive Graph Convolutional Network (AGCN) [45] and Dual Graph Convolutional Network (DGCN) [46] to improve the performance of GCN by making the feature learning independent of input graph structure, and training two graph convolutional layers in parallel, respectively.

4.3 SPATIAL GRAPH CONVOLUTIONAL NETWORK

The spatial convolution operation is directly defined on the graph and it can be easily explained in the context of conventional CNNs in which the spatial structure of the images is considered. As illustrated in Fig. 4.2, the convolution operation in grid-structured data is a process of employing a weighted kernel to update the features of each node in the grid by aggregating information from its local neighbors. Its generalization to graph data is defined as a message passing function which updates the features of each graph node by the means of message passing through the graph edges. Spatial GCN is a combination of such message passing function with a data transformation process. It is trained in an end-to-end manner to optimize an objective function which can be defined on the individual graph nodes or the whole graph.

Diffusion Convolutional Neural Network (DCNN) [13] is a spatial GCN method which defines the convolution operation as a diffusion process using transition matrices. A probability transition matrix $\mathbf{P} \in \mathbb{R}^{N \times N}$ is defined as the degree-normalized graph Adjacency matrix, $\mathbf{P} = \mathbf{D}^{-1}\mathbf{A}$, and $\mathbf{P}_{ij}$ indicates the probability of information transition from node v_i to node v_j. The information is propagated from each node to its neighboring nodes with a transition prob-

ability until the information distribution reaches an equilibrium. This method has several advantages in terms of classification performance, flexibility, and computational efficiency. It is flexible in the sense that the underlying graph in DCNN can be directed or undirected, weighted or unweighted. This method can be used for different tasks including node classification and graph classification in a semi-supervised or supervised manner, respectively. Moreover, DCNN represents graph diffusion as a matrix power series to encode the structural information of the graph. Hereby, we describe in detail the diffusion operation definition in this method.

The DCNN model takes the graph transition matrix and the node feature matrix $\mathbf{X} \in \mathbb{R}^{N \times D}$ as input and outputs a prediction for either nodes, edges or the whole graph. For node classification, the diffusion convolution operation through the k-hop neighbors is defined as follows:

$$\mathbf{H}^{(k)} = \sigma((\mathbf{1}_N.\mathbf{W}^{(k)^\top}) \odot (\mathbf{P}^k\mathbf{X})), \tag{4.20}$$

where $\odot$ denotes the element-wise multiplication and σ is the non-linear activation function. $\mathbf{W} \in \mathbb{R}^{K \times D}$ is the transformation matrix and $\mathbf{W}^{(k)} \in \mathbb{R}^{1 \times D}$ is the transformation matrix to the k-hop neighbors. $\mathbf{1}_N \in \mathbb{R}^N$ is a vector of ones which is multiplied to $\mathbf{W}^{(k)}$ to make N copies of that. $\mathbf{P}^k$ denotes the power k of the transition matrix which is used to aggregate features from hop k neighborhood, and $\mathbf{H}^{(k)} \in \mathbb{R}^{N \times D}$ contains the aggregated features obtained by using the k hop neighborhood. This convolution operation is performed for multiple hop values, i.e. from 1 to K, and the transformed feature matrices $\mathbf{H}^{(1)}$, $\mathbf{H}^{(2)}$, ..., $\mathbf{H}^{(K)}$ are concatenated to produce $\mathbf{H} \in \mathbb{R}^{N \times K \times D}$.

The diffusion can also be expressed using tensor notation. For node classification it can be defined as follows:

$$\mathbf{H} = \sigma(\mathbf{W}^\star \odot (\mathbf{P}^\star \times_3 \mathbf{X})), \tag{4.21}$$

where $\mathbf{P}^\star$ is a tensor of size $N \times K \times N$ which contains the power series $\{\mathbf{P}^1, \mathbf{P}^2, ..., \mathbf{P}^K\}$ of transition matrix $\mathbf{P}$, and $\mathbf{W}^\star \in \mathbb{R}^{N \times K \times D}$ is the transformation tensor to the 1 to K neighbors. This tensor is obtained by making N copies of $\mathbf{W}$ so that it has the same size as $(\mathbf{P}^\star \times_3 \mathbf{X})$, and it contains $K \times D$ learnable parameters. $\mathbf{H} \in \mathbb{R}^{N \times K \times D}$ is the diffusion convolutional representation computed by K-hops graph diffusion over the nodes' features. $\times_3$ in Eq. (4.21) denotes the 3-mode product of the tensor $\mathbf{P}^\star$ to the matrix $\mathbf{X}$. It should be noted that the diffusion convolution doesn't change the features' dimension.

For graph classification, $\mathbf{H}$ is of size $\mathbb{R}^{K \times D}$ and is computed by taking average over the nodes as follows:

$$\mathbf{H} = \sigma(\mathbf{W} \odot ((\mathbf{P}^\star \times_3 \mathbf{X}) \times_1 \mathbf{1}_N^\top)/N), \tag{4.22}$$

The series of tensor operations in this method can be computed in polynomial time and implemented efficiently on a GPU.

Message Passing Neural Network (MPNN) [15], has generated a general framework to formulate all the existing spatial graph convolutional networks. Similar to DCNN, the convolutional operation in MPNN is defined as a K-hop message passing and the information is propagated from one node to its neighboring nodes through the graph edges. Therefore, the graph convolution operation in this framework is defined using message passing and update functions as follows:

$$\mathbf{h}_{\mathcal{N}(v_i)}^{(k)} = \sum_{v_j \in \mathcal{N}(v_i)} M^{(k)} \left(\mathbf{h}_i^{(k-1)}, \mathbf{h}_j^{(k-1)}, \mathbf{x}_{ij}^e \right), \tag{4.23}$$

$$\mathbf{h}_i^{(k)} = U^{(k)} \left(\mathbf{h}_i^{(k-1)}, \mathbf{h}_{\mathcal{N}(v_i)}^{(k)} \right), \tag{4.24}$$

where $\mathbf{h}_i^{(k)}$ is the hidden representation of node v_i obtained by message passing through its k-hop neighbors and $\mathbf{h}_i^{(0)} = \mathbf{x}_i$. $M^{(k)}(\cdot)$ is the message passing function which propagates the features and $U^{(k)}(\cdot)$ is the updating function which updates the features of the node. $\mathbf{h}_{\mathcal{N}(v_i)}^{(k)}$ denotes the aggregated feature vector of the neighbors of v_i which is obtained by the message passing function $M^{(k)}(\cdot)$. $\mathbf{x}_{ij}^e$ denotes the feature vector of the edge connecting the neighboring nodes v_i and v_j. In simple graphs, the edges can be represented by single binary values expressing the graph node connections. The structure of a single layer of the MPNN framework is illustrated in Fig. 4.3.

This convolution operation is performed iteratively for K steps and in each step k the node features are updated using its own features and the features of its direct 1-hop neighboring nodes in step $k-1$. In this way, at the end the information has been propagated through K-hop neighbors and $\mathbf{h}_i^{(K)}$ can be introduced to a classification layer for predicting the labels for each node. For the graph classification task, a readout function is needed to generate a feature vector representing the graph as follows:

$$\mathbf{h}_G = R \left(\mathbf{h}_i^{(K)} | i = 1, ..., N \right), \tag{4.25}$$

where $R(\cdot)$ is the readout function and $\mathbf{h}_G$ is the feature vector of the graph. By specifying $M(\cdot), U(\cdot), R(\cdot)$ which are differentiable functions with learnable parameters in different ways, several spatial graph convolutional network methods such as Convolutional Networks for Learning Molecular Fingerprints [47], Gated Graph Neural Networks (GG-NN) [48], Interaction Networks [49], Molecular Graph Convolutions [50], Deep Tensor Neural Networks [51], and spectral-based GCN methods [10, 52, 12] can be defined in this framework.

Unlike the regular GCN methods that generalize the concept of convolution from CNNs to graph structured data, some methods such as PATCHY-SAN [14] and LGCN [53] transform the graph data into grid structured data in order to apply the convolutional filter directly on the locally connected regions of the graph, similar to image-based convolutional networks that perform convolution on locally connected pixels of an image. PATCHY-SAN method [14] first orders

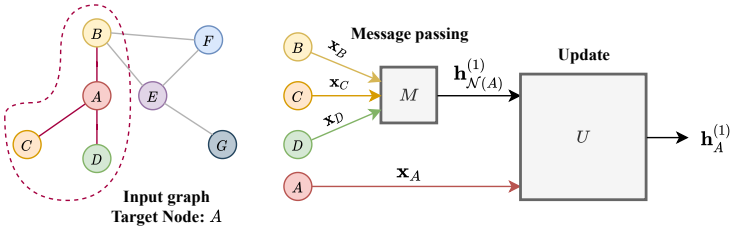


FIGURE 4.3 Illustration of convolution operation in the first layer of MPNN framework. The input graph is shown in the left side. The goal is to learn the hidden representation of the node A, so the first hop neighborhood of this node is considered. In the right side, the layer structure containing the message passing and updating functions is illustrated. The layer receives the feature vectors of node A and its first hop neighbors as input. It learns the representation of the node by performing message passing through its neighbors and applying an updating function to its own input features and the aggregated feature vector of its neighborhood.

the neighbors of each node according to their score which can be defined based on their degree. Then, it determines the number of ordered neighbors of all the nodes by selecting a fixed number of top neighbors for each node. Therefore, similar to grid structured data, each graph node now has a fixed number of ordered neighbors which serves as the convolution receptive field and a standard convolution operation is applied on each node to aggregate its neighborhood features.

LGCN [53] orders the neighbors of each node based on their feature values. For each node, it constructs a feature matrix of size $(k+1) \times D$ containing the D dimensional feature vectors of the node itself and its k neighbors. Then the feature columns of the neighbors are sorted based on their value and a fixed number of top rows are selected as the ordered neighborhood features of that node. In other words, it performs max pooling on the feature matrix of each node's neighbors followed by a standard 1D convolution which is applied on the selected top rows of the feature matrix to generate the transformed features for the node.

4.4 GRAPH ATTENTION NETWORK (GAT)

Consider the Graph Convolutional Network transformation in Eq. (4.18), which defines the feature transformation for all N graph nodes:

$$\mathbf{H} = \sigma\left(\hat{\mathbf{A}}\mathbf{X}\mathbf{W}\right).$$

The matrix multiplication by $\hat{\mathbf{A}}$ produces a weighted summation over the neighbor set for each node. For node i, the computation can be expressed equivalently

as

$$\mathbf{h}_i = \sigma\left(\sum_{j \in \mathcal{N}(v_i)} \alpha_{ij} \mathbf{x}_i \mathbf{W}\right) \tag{4.26}$$

where $\mathbf{x}_i \in \mathbb{R}^D$ is the feature vector of node v_i, and $\alpha_{ij} = \hat{A}_{ij}$ is the importance weighting factor for a node v_i to its neighbor v_j in the set of neighbors $\mathcal{N}(v_i)$.

For a GCN, the importance weights α_{ij} are defined explicitly from the graph Adjacency matrix. This works well for tasks on a fixed graph, i.e. in a *transductive* setting. However, the learned parameters in $\mathbf{W}$ may not generalize to unseen graphs (the *inductive* setting) because they were learned under the specific graph structure with fixed node importance weights.

This limitation can be overcome by the use of a Graph Attention Network (GAT) [54]. Inspired by the attention mechanism in [55], the importance weights α_{ij} are not defined explicitly from the graph structure. Instead, they are computed as a self-attention for neighboring nodes:

$$a_{ij} = \text{attention}(\mathbf{x}_i, \mathbf{x}_j) \tag{4.27}$$

where $a_{ij} \in \mathbb{R}$ is an unnormalized attention coefficient for input feature vectors $\mathbf{x}_i$ and $\mathbf{x}_j$. The attention mechanism could be any function of two input features, but the choice in the original paper [54] was a single-layer feedforward neural network over concatenated transformed features with LeakyReLU and softmax activations:

$$\text{attention}(\mathbf{x}_i, \mathbf{x}_i) = \text{softmax}_j\left(\text{LeakyReLU}\left(\mathbf{v}^\top \left[\mathbf{x}_i \mathbf{W} \,||\, \mathbf{x}_j \mathbf{W}\right]\right)\right). \tag{4.28}$$

Here, $||$ denotes concatenation, and $\text{LeakyReLU}(\mathbf{x}) = \max(\beta \mathbf{x}, \mathbf{x})$ for $1 > \beta > 0$. $\mathbf{W} \in \mathbb{R}^{D' \times D}$ is the weight matrix which transforms the feature vectors, and $\mathbf{v} \in \mathbb{R}^{2D'}$ is a parameter vector for the attention layer. Notice that the attention weight vector $\mathbf{v}$ is learned from node correspondences and does not depend on the global graph structure. This lets GAT generalise better to unseen graphs than a regular GCN. Finally, the unnormalized attention coefficient normalized using a softmax over the neigbourhood of the node to produce the final importance weight:

$$\alpha_{ij} = \text{softmax}_j(a_{ij}) = \frac{e^{a_{ij}}}{\sum_{k \in \mathcal{N}(v_i)} e^{a_{ik}}} \tag{4.29}$$

The softtmax operation yields positive edges for all nodes in the neighborhood of v_i that sum up to $\sum_{\sum_{j \in \mathcal{N}(v_i)} \alpha_{ij} = 1}$. An illustration of this is shown in Figure 4.4. As with other attention-based networks [55], GAT can be extended to use *multi-head attention* where K independent attention mechanisms are aggregated (typically by concatenation) to produce the new feature:

$$\mathbf{h}_i = \Big\|_{k=1}^{K} \sigma\left(\sum_{j \in \mathcal{N}(v_i)} \alpha_{ij}^k \mathbf{x}_i \mathbf{W}^k\right) \tag{4.30}$$

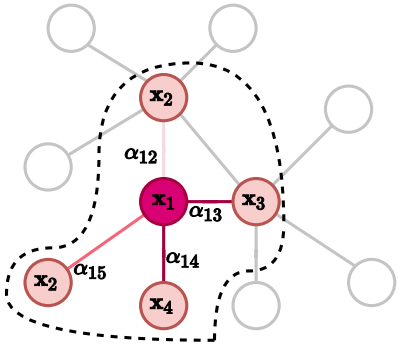


FIGURE 4.4 Illustration of edges weighted by attention coefficients α_{ij} in the neighborhood of node 1. The color intensities on the edges $\mathbf{x}_1$ denote the magnitude of the corresponding attention values α_{1j}.

Here, k denotes the attention index for a layer. Using concatenation to aggregate the outputs, $\mathbf{h}_i$ will consist of KD features.

Now that we have covered a method of generalizing to unseen graphs, we proceed with describing how to leverage and learn from large graphs, that cannot fit in computational memory all at once.

4.5 GRAPH CONVOLUTIONAL NETWORKS FOR LARGE GRAPHS

Many of the existing GCN methods [10, 52, 47, 12, 14] perform the training process in a transductive setting. These methods typically need to load the whole graph data and node features in memory, and introduce them to the model at once, while the training process updates the parameters of the GCN model using full batch iterative optimization. For example, in the GCN method [12], which performs under semi-supervised transductive learning, the features of each node are updated using the features of its neighbors in the previous layer. Similarly, each of these neighbors are also updated by aggregating features from their corresponding neighbors in the previous layer, and this back tracking continues until it reaches k-hops neighbors. Therefore, the whole graph Laplacian and feature matrices are required during the training process. Besides, as we go deeper in the GCN, the total number of neighboring nodes which are used for feature aggregation grows exponentially, which leads to neighbor explosion. Methods with full-batch training only target small-sized graph datasets and their application on large-scale graph data is infeasible due to their high computational complexity and memory consumption. Moreover, in Laplacian based methods like GCN [12], the node representations are dependent on the graph structure. Since they only focus on a fixed graph structure during the training process, they cannot be generalized to unseen nodes or graphs.

Recently, GCN methods which can perform on an inductive setting have been proposed. These methods are scalable to large graph datasets with millions of nodes and they are able to generate feature representations for unseen nodes or graphs efficiently. In order to alleviate the above-mentioned problems and scale GCN methods to large scale graphs, *layer sampling* and *graph sampling* methods have been proposed, which can be trained by following mini-batch optimization. In the following, we will introduce these two groups of methods.

4.5.1 Layer sampling methods

The layer sampling methods [56, 57, 58, 59, 53, 60] form mini-batches by sampling a small number of nodes/edges at each layer of the model and then perform feature propagation among the sampled nodes/edges. These two steps proceed iteratively and the model weights are updated using Stochastic Gradient Descent (SGD) [61].

The GraphSAGE method [56] mitigates the neighbor explosion problem kf GCNs [12] by performing sampling on the neighboring nodes using a distinct sampling function in each layer in order to constrain the batch-size to a pre-defined fixed size. The algorithm receives as input the whole graph $G = (\mathcal{V}, \mathcal{E})$, the nodes feature matrix $\mathbf{X}$, and the mini-batch set $\mathcal{B}$ formed by the target nodes, i.e. the nodes whose features will be updated. The model is composed of K convolutional layers in order to aggregate features from 1 to K-hops neighbors and it samples all the neighboring nodes needed for feature aggregation at first. Sampling K-hop neighbors for each node in $\mathcal{B}$ is like expanding a tree rooted at that node at layer K of the model recursively.

The sampling mechanism starts with the root nodes at layer K, $\mathcal{B}^K = \mathcal{B}$, and samples S_K nodes from their neighbors at layer $K-1$ using a neighborhood sampling function F_K so that $\mathcal{B}^{K-1} = \mathcal{B}^K \cup F_K(v)$ for $\forall v \in \mathcal{B}^K$. This process continues for all the layers $k = \{K, K-1, ..., 1\}$ with fixed sample sizes S_k and sampling functions $F_k : v \rightarrow 2^{\mathcal{V}}$, respectively. The neighborhood samplers F_k used in GraphSAGE are uniform random sampling functions generating independent samples. In cases where the sample size S_k is larger than the node degree, the sampling is performed with replacement. In this way, each set $\mathcal{B}^k$ contains all the needed nodes for feature aggregation in the next layer $k+1$ to compute the feature representation nodes in $\mathcal{B}^{k+1}$. Therefore, $\mathcal{B}^0$ contains all the sampled nodes. As an example, let us assume that we have a 2-layer network and we want to generate feature representations for the nodes in $\mathcal{B}$ using their 2-hop neighbors. For each node, we first sample S_2 nodes from its immediate neighbors and then sample $S_2 \times S_1$ nodes from their 2-hop neighbors.

Instead of learning a discrete feature representation for each node, this method trains a set of aggregation functions, one for each layer, which learn to utilize the local structure of the graph to aggregate the features from each node's neighborhood. Therefore, these trained aggregation functions are able to generate feature representations for completely unseen nodes at inference time. In the forward

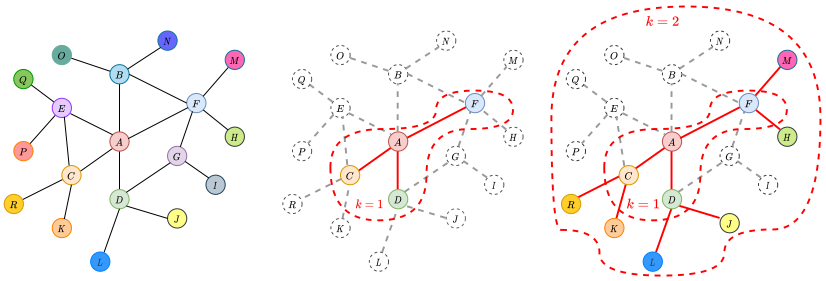


FIGURE 4.5 Visual illustration of the sampling process for a 2-layer network in the GraphSAGE method. Here, the mini-batch contains a single node A, i.e. $\mathcal{B} = A$, and the number of nodes for the first and second layers (or first and second hop neighborhood of node A) are $S_1 = 3$, $S_2 = 2$, respectively. The sampling function first samples S_2 nodes from the immediate neighbors of A, and then it samples S_1 neighbors for each of the S_2 sampled nodes in the previous step. Therefore, it samples $S_2 \times S_1$ nodes in total from its 2-hop neighbors. This sampling process is repeated for each node in the batch set $\mathcal{B}$.

pass, in order to generate the feature representations $\mathbf{h}_v^{(k)}$ for $\forall v \in \mathcal{B}^k$, the k^{th} layer aggregation function $Agg^{(k)}$ aggregates the feature representations of their immediate neighborhood $\left\{\mathbf{h}_{v'}^{(k-1)}, \forall v' \in \mathcal{N}(v)\right\}$ into a single vector $\mathbf{h}_{\mathcal{N}(v)}^{(k-1)}$, while $\mathbf{h}_{v'}^{(k-1)}$ is generated at layer $k-1$. It should be noted that at $k = 0$ the feature representations are the input feature vectors.

In the next step, the aggregated feature vector $\mathbf{h}_{\mathcal{N}(v)}^{(k-1)}$ is concatenated with the current representation vector of the node $\mathbf{h}_v^{(k-1)}$ and the resulting vector is introduced to a fully connected layer which transforms the features by some learnable parameters and is followed by a non-linear activation function. Concatenation of the node features with its aggregated neighborhood feature vectors works like skip connections between different layers of the model. Experimental evaluation of GraphSAGE has shown the effectiveness of the concatenation mechanism, as it leads to significant performance improvement. At the last layer K, the final feature representations $\left\{\mathbf{h}_v^{(K)}, \forall v \in \mathcal{B}\right\}$ for the target nodes are generated, which are then introduced to the classification layer which predicts the labels for the nodes. The sampling process and feature aggregation of the GraphSAGE method are shown in Fig. 4.5, Fig. 4.6, respectively.

GraphSAGE can not only be trained in inductive setting with mini-batches, it can also be trained in both an unsupervised or a supervised manner and it can generalize to unseen data. Another interesting advantage of GraphSAGE is that at different levels of feature extraction, an appropriate aggregation function can be employed to capture the local structure of the graph. The aggregation functions should be differentiable and invariant to node permutations. For instance, mean aggregator, max pooling and mean pooling aggregators are all symmetric and trainable functions which can be employed in GraphSAGE. Mean aggregator

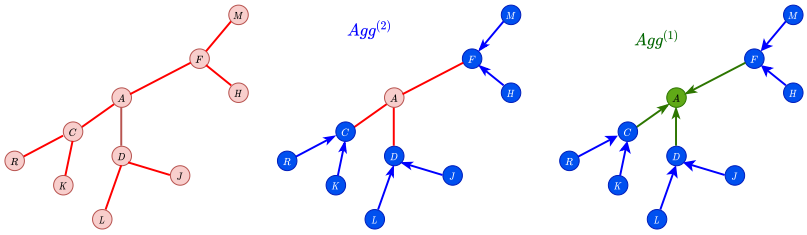


FIGURE 4.6 Illustration of GraphSAGE feature aggregation in a 2-layer network. The node A and its neighbors (which are sampled as in Fig. 4.5) are shown in the left side. In the middle, the first hop neighbors of node A are updated by aggregating features from their neighbors. Arrows indicate the feature aggregation. In the final step, illustrated on the right side, the features of node A are updated by aggregating its first-hop neighbors features. It should be noted that different aggregation functions can be used for each layer of the network (or at each step of feature aggregation process). "Agg" denotes the aggregation function.

computes the element-wise average of vectors in $\left\{\mathbf{h}_{v'}^{(k-1)}, \forall v' \in \mathcal{N}(v)\right\}$. For employing pooling aggregators, the feature vector of each neighboring node is first transformed, and then the element-wise pooling is performed on the transformed features.

An extension of GCN method with an inductive training setting can be obtained by a special form of GraphSAGE with two layers and mean aggregation function in both layers. The convolution operation in the inductive GCN is defined as follows:

$$\mathbf{h}_v^{(k)} \leftarrow \sigma\left(\mathbf{W}^{(k)}\bar{\mathbf{h}}_v^{\prime(k)}\right), \tag{4.31}$$

$$\mathbf{h}_v^{\prime(k)} = \left\{\mathbf{h}_v^{(k-1)}\right\} \cup \left\{\mathbf{h}_{v'}^{(k-1)}, \forall v' \in \mathcal{N}(v)\right\}. \tag{4.32}$$

$\bar{\mathbf{h}}_v^{\prime(k)}$ in Eq. (4.31) is the mean of the node feature vector in the current state and its neighbors aggregated representation vector.

Regardless of the advantages of GraphSAGE method in generalizing GCNs to unseen data by inductive learning and mini-batch training, it still has limitations in terms of scalability to very large graphs with billions of nodes, and it cannot be employed in related real world applications. This limitation is due to the fact that it still needs to store the whole graph in the memory although it performs localized convolution by sampling a fixed number of neighbors around each of the graph nodes in the batch.

The PinSage method [58], addressed this limitation by assigning an importance score to each neighbor using random walk sampling. Moreover, it can employ multi-GPU training methods to improve both the performance and efficiency of the model to scale up to graphs with billions of nodes and edges. The PinSage method has been deployed for different recommendation tasks at Pinterest application which is one of the world's largest graph of images. In this

application, the users interact with visual bookmarks, called pins, to the online images. In the following, we introduce the key properties of this method which led to high scalability of GCNs.

- **Importance pooling:** PinSage developed an importance pooling approach for localized convolution on the nodes by using a random-walk-based sampling method instead of a uniform sampling for defining the neighborhood of each node. Random walks start from a node v and a count is kept to the number of times each of its neighbors has been visited. Each neighbor of the node thus gets an importance score, which is the normalized visit count. The subset of neighbors with the highest importance scores are selected and defined as its neighborhood. Therefore, the localized convolution in this method performs a weighted feature aggregation which leads to less information loss.
- **Mini-batch construction:** Contrary to GraphSAGE, the PinSage method stores the whole graph Adjacency matrix and its nodes' feature vectors on the CPU memory instead of GPU. Since the convolution operations are performed on GPUs, the data needs to be transferred between CPU and GPU which is not computationally efficient. PinSage has solved this issue by creating a small subgraph $G_{sub} = (\mathcal{V}_{sub}, \mathcal{E}_{sub})$ for each mini-batch using the nodes which are involved in the convolution computation of that mini-batch and their sampled neighborhood. The feature matrix corresponding to each subgraph is also constructed using a re-indexing approach. Therefore, at each mini-batch iteration, only a small subgraph with its corresponding feature matrix are transferred to the GPUs. By using such optimized data transfer between GPU and CPU this method has greatly improved the GPU utilization. Moreover, this method proposed a producer-consumer approach for training which reduces the training time. In this approach, while the GPUs are computing convolutions for the nodes in the current mini-batch, the CPU is constructing the subgraph and feature matrix for the next mini-batch in parallel.
- **Multi-GPU training:** In order to maximize the GPU usage during the forward and backward propagation, the graph nodes in each mini-batch are divided into equal-sized portions, each of which is processed by a seperate GPU. It should be noted that all the GPUs use exactly the same neural network model with shared model parameters. For the optimization step, the computed gradients are aggregated across all the GPUs and a synchronous SGD step is performed.

The S-GCN method [59] has improved the time complexity of layer sampling methods by sampling only two arbitrary neighbors for each node in the graph while preserving the performance of the model. It has developed a control variate-based stochastic approximation algorithm for GCN, utilizing the historic activations of nodes in the previous layers, thus minimizing the redundant computations. It was also theoretically proved that S-GCN converges to a local optimum of GCN.

One of the main differences between the standard neural networks and GCNs is that in GCNs, each sample (which is assumed to be a graph node) is convolved

with its neighbors, leading to lack of independence between the samples, while the optimization method SGD is designed based on the additive loss function for independent data samples. FastGCN [57] proposed to view graph convolution from a different perspective and developed the GCN as an integral transform of embedding functions under probability measures. In other words, it interprets each layer of the GCN as an embedding function of the nodes, which are independently sampled from the graph but are from the same probability distribution.

To reduce the computational footprint and the memory consumption caused by recursive neighborhood expansion, the FastGCN method samples the nodes independently for each layer. To compare FastGCN sampling with GraphSAGE, let us assume that at each layer l of the GraphSAGE model, S_l neighbors are sampled for each node in the batch. Then, the total number of expanded neighborhood would be $S_0 \times S_1 \times ... \times S_{K-1}$ for K layers. In FastGCN method, at each layer l, S_l nodes are sampled independently from the graph disregarding their neighbors. Therefore, the total number of sampled nodes for K layers would be $S_0 + S_1 + ... + S_{K-1}$ which makes the convolutions more computationally efficient. The method also applies importance sampling to reduce the variance. However, since the nodes are sampled independently in each layer, the connections between the nodes forming the mini-batches might become too sparse which leads to performance degradation.

The adaptive sampling method proposed in [60] addressed this problem and improved FastGCN by capturing the connections between the network layers. Similar to the FastGCN method, it constructs the network layer by layer by fixed-size sampling but the layers are not independent. In this method, the samples in each layer are conditioned on the samples on their later layers, and the connections between the sampled nodes in the earlier layer and their parents in the later layers can be used. However, training the sampling method proposed in [60] is computationally expensive.

A simple illustration of layer sampling methods is provided in Fig. 4.7. While the layer sampling methods significantly improved the training efficiency of GCNs, methods adopting them still face challenges to achieve high accuracy, scalability, and computation efficiency, simultaneously.

4.5.2 Graph sampling methods

Graph sampling methods [53, 62, 63] perform sampling on the graph scale instead of node scale, which avoids neighbor explosion. The mini-batches are built from subgraphs in these methods. In the following, two of the recently proposed graph sampling methods are introduced. The graph sampling scheme is illustrated in Fig. 4.8.

The work in [62] proposed a representative graph sampling method which assures that the connectivity characteristics of the original training graph are preserved in the sampled subgraphs and that each node in the training graph has

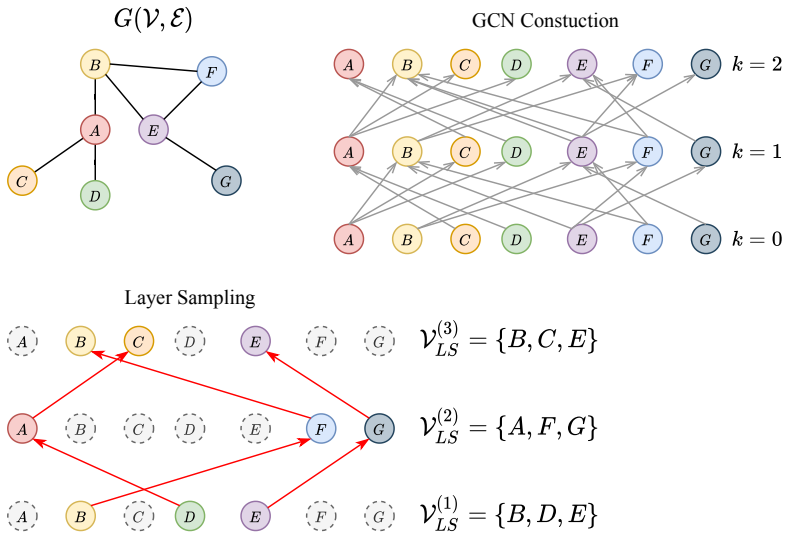


FIGURE 4.7 In layer sampling methods, the GCN model is first constructed on the whole graph. Then a mini-batch is built by a set of sampled nodes in each layer. $\mathcal{V}_{LS}^{th}$ denotes the nodes sampled by the Layer Sampling method in k^{th} layer.

a fair chance of being sampled. Contrary to the traditional GCN methods, where the network is directly constructed on the input graph, this graph sampling method samples a small subgraph $G_{sub} = (\mathcal{V}_{sub}, \mathcal{E}_{sub})$ for each mini-batch training iteration and constructs a complete GCN on the sampled subgraph G_{sub}. The forward propagation in layer l of this network is defined in 3 steps. In the first step, the neighbors' features are aggregated as follows:

$$\mathbf{H}_{neigh}^{(l)} = \mathbf{A}_{sub}^{(l)} \mathbf{H}_{sub}^{(l-1)} \mathbf{W}_{neigh}^{(l)}, \tag{4.33}$$

where $\mathbf{A}_{sub}^{(l)}$ denotes the graph Adjacency matrix of the sampled subgraph, and $\mathbf{H}_{sub}^{(l-1)}$ and $\mathbf{H}_{sub}^{(l)}$ are the feature matrices of nodes in layers $l-1$ and l, respectively. $\mathbf{W}_{neigh}^{(l)}$ denotes the transformation matrix which is learned for neighbors' feature aggregation. In the second step, the features of the node are transformed with another transformation matrix as follows:

$$\mathbf{H}_{self}^{(l)} = \mathbf{H}_{sub}^{(l-1)} \mathbf{W}_{self}^{(l)}, \tag{4.34}$$

and in the final step, the features of the l^{th} layer of the network are built by the concatenation of $\mathbf{H}_{neigh}^{(l)}$ and $\mathbf{H}_{self}^{(l)}$ which is followed by an activation function:

$$\mathbf{H}_{sub}^{(l)} = \sigma(\mathbf{H}_{neigh}^{(l)} \,||\, \mathbf{H}_{self}^{(l)}), \tag{4.35}$$

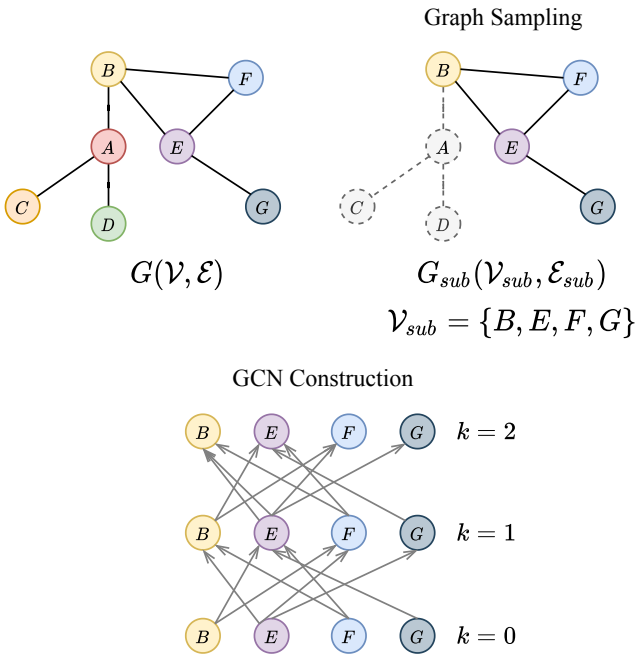


FIGURE 4.8 Illustration of graph sampling and GCN construction on the subgraph. In graph sampling methods, the subgraphs are sampled independently and all the graph nodes have sufficient chance to be sampled. The GCN model is constructed on the full subgraph and at each iteraiton of the training process, the full subgraph is introduced into the GCN model and the features of the nodes are updated by information propagation through their neighbors. In this figure, the model has two layers, $k = 2$, and the features are aggregated from 2-hop neighbors.

It was shown that by sampling a sufficient number representative subgraphs, the GCN model is able to capture both the training graph structure and its nodes' attributes to generate accurate feature representations. The main advantage of this method over the layer sampling methods is that it achieves both high performance and computation efficiency, simultaneously by performing convolution on a small set of nodes of the sampled subgraph. Thus neighbor explosion is avoided, while the node connections in the original graph are preserved which leads to minimum information loss and high accuracy. Besides, the method achieves scalability with respect to graph size and number of GCN layers by developing efficient approaches for parallelizing the training process on shared-memory multiple core processing systems.

ClusterGCN method [63] employs efficient graph clustering algorithms such as Metis [64] and Graclus [65] to generate subgraphs for mini-batch training. These graph clustering methods partition the input graph into a set of dense clusters in such a way that the number of within-cluster edges is much higher

than the number of between-cluster edges. Accordingly, each node and its neighbors are mostly located in the same cluster which leads to minimized information loss. At each iteration of ClusterGCN, a set of generated clusters are selected randomly without replacement and a subgraph is constructed using these clusters. For each mini-batch, the forward pass and the optimization are performed by only using the nodes involved in the subgraph which leads to high computational and memory efficiency.

4.6 DATASETS AND LIBRARIES

GNNs have applications in many different areas and many benchmark graph datasets are available which describe different learning tasks involving graphs. TUDataset [66] contains a collection of more than 120 benchmark datasets, and experimental evaluation tools for graph learning. The statistics of a set of well-known benchmark datasets are summarized in Table 4.1. We categorize these datasets into four groups, citation networks, social networks, bio-chemical networks and a group for the graph datasets used in computer vision applications. Accordingly, some of the applications of GNNs in different domains are briefly described as follows:

- **Natural language processing:** One of the graph-based methods applications in natural language processing domain is document classification [12, 56, 54]. Well-known datasets such as Citeseer [67], Cora [67] and Pubmed [67], include citation networks which represent documents as nodes and the cross references between documents as edges. GCN models extract high-level feature representation for each document using the node labels in a supervised setting or by propagating the label information from labeled nodes through the whole graph in a semi-supervised setting. Using the extracted high-level node features, a label will be predicted for each node by a classification layer. The whole model including the classification layer and GCN layers are trained in an end-to-end manner. Other applications in this domain include neural machine translation [68], text classification [69], and sequence-to-graph learning [70].
- **Social networks and recommender systems:** An usefulness of graph-based methods can be seen in recommendation systems where we need to recommend a new item to each user while the users and items are represented as nodes and the recommendations are represented as graph edges [71, 58, 72]. In social networks, the users are denoted as nodes and the interaction between them or their influence on each other is modeled as graph edges [56, 9, 73]
- **Computer vision:** Graph-based methods have been widely used in computer vision tasks. Example computer vision problems that can be modeled as graph-based tasks are image classification [74, 75], skeleton-based human action recognition [76], semantic segmentation [77], human object interaction [78], and 3D point clouds segmentation and classification [79, 80]. In these problems, the objects in the images are modeled by graphs representing

the semantic relationships between them and the graph-based methods learn to extract features from these graphs for different purposes. The well-known datasets for skeleton-based human action recognition are NTU-RGB+D [81] and Kinetics-skeletons datasets [82]. NTU-RGB+D dataset is one of the largest multi-modality action recognition datasets which contains both video and 3D skeletons data that is captured by the Microsoft Kinect-v2 camera. Kinetics-skeleton dataset is also a very large action recognition dataset that contains YouTube video clips of different human actions. The 2D joints' co-ordinates on every frame are estimated using the OpenPose toolbox [83]. The widely used dataset containing 3D point clouds is FIRSTMM-DB [84, 85] and the MSRC datasets [85] are used for semantic image processing.

- **Chemistry:** In the fields of biology and chemistry, the graphs are built by using molecules or proteins or atoms as nodes and the interaction between them as edges. In such problems the goal is to learn molecular features, study the bio-chemical graph properties and capture molecular or protein-protein interactions. GAEs can be used for solving these tasks, which are formed by an encoder and a decoder sub-networks. In the encoder part, the model employs graph convolution layers to learn graph embedding and the graph structure will be reconstructed in the decoder part by predicting edges between nodes.

TABLE 4.1 Summary of benchmark graph datasets

Category	Name	#Graphs	#Nodes(Avg.)	#Edges(Avg.)	#Features	#Classes	Year
Citation Networks	Cora [67]	1	2708	5429	1433	7	2008
	Citeseer [67]	1	3327	4732	3703	6	2008
	Pubmed [67]	1	19717	44338	500	3	2008
	DBLP (v11) [86]	1	4107340	36624464	-	-	2008
	OAG v1 [86, 87]	2	320,963,344	64,639,608	-	-	2017
Social Networks	BlogCatalog [88]	1	10312	333983	-	39	2009
	Digg [89]	-	279.630	1,548,126	-	-	2012
	Twitter-Higgs [90]	-	456,631	14,855,875	-	-	2013
	Weibo [91]	-	1,776,950	308,489,739	-	-	2013
	COLLAB [92]	1	235,868	235,868	-	-	2015
	IMDB-Binary [92]	1000	19.8	96.5	-	2	2015
	IMDB-Multi [92]	1500	13.0	65.9	-	3	2015
	Reddit [56]	1	232965	11606919	602	41	2017
Computer Vision	MNIST [93]	70000	40-75	-	-	-	1998
	CIFAR10 [94]	60000	85-150	-	-	-	2009
	Fingerprint [95]	2800	5.42	4.42	-	15	2008
	MSRC9 [85]	221	40.58	97.94	-	8	2016
	MSRC21 [85]	563	77.52	198.32	-	20	2016
	MSRC21C [85]	209	40.28	96.60	-	20	2016
	FIRSTMMDB [84, 85]	41	1377.27	3074.10	-	11	2013
	NTU-RGB+D [81]	56,880	25	15000	-	60	2016
	Kinetics-skeleton [82]	300,000	18	10800	-	400	2017
Bio-Chemical Graphs	MUTAG [96]	188	17.93	19.79	7	2	1991
	PTC [97]	344	25.5	-	19	2	2003
	D&D [98]	1178	284.31	715.65	82	2	2003
	ENZYMES [99]	600	32.6	62.1	-	6	2004
	PROTEIN [100]	1113	39.06	72.81	4	2	2005
	NCI1 [101]	4110	29.87	32.30	37	2	2008
	QM9 [102]	133885	-	-	-	-	2014
	PPI [103]	24	56944	818716	50	121	2017
	ZINC [104, 105]	249,456	23.1	24.9	-	12	2018
	Alchemy [106]	119487	-	-	-	-	2019

There exist available open-source libraries for GNNs which facilitate research in this area, allowing for easy model creation and reproduction of GNN results on benchmark datasets:

- **Pytorch Geometric** [107] is an open-source geometric learning library implemented in Pytorch including implementations of many GNN methods.
- **Deep Graph Library (DGL)** [108] is an open-source library which allows for fast implementation of many GNN methods in PyTorch and MXNet platforms.
- **Jraph**[5] is a lighweight open-source library implemented in JAX and provides implementation of GNN models, graph data structures and a set of utilities for working with graphs.

4.7 CONCLUSION

In this chapter, we introduced Graph Convolutional Networks and provided an overview of the most representative methods. We divided the methods in four main categories, namely spectral, spatial, attention and sampling based methods. We started describing the concept of graph convolution in Section 4.2 by introducing the spectral graph convolution which has a solid background in graph signal processing. Learning the feature representations of the nodes and graphs in spectral GCN methods depends heavily on the structure of the input graph which makes the generalization of these methods to unseen data infeasible. Greatly inspired by Convolutional Neural Networks where the spatial structure of the input grid structured data is considered, the spatial GCN methods introduced in Section 4.3 define the graph convolution operation by means of message passing through the graph edges. In Section 4.4, we described how the attention mechanisms can be used in GCN methods to improve their performance. Several methods have been proposed recently to improve the computational efficiency and scalability of the existing GCN methods. In Section 4.5 we introduced the most representative works which are capable to train on large graph datasets efficiently and effectively. Finally in Section 4.6, a list of benchmark graph datasets used in different learning tasks was provided, along with a list open-source libraries which facilitate research in this area by providing the implementations of methods and useful graph utility functions.

BIBLIOGRAPHY

[1] T. Luong, H. Pham, C. D. Manning, Effective approaches to attention-based neural machine translation, in: Conference on Empirical Methods in Natural Language Processing, 2015.

[2] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, et al., Google's neural machine translation system: Bridging the gap between human and machine translation, arXiv preprint arXiv:1609.08144 (2016).

[3] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke,

5. https://github.com/deepmind/jraph

P. Nguyen, T. N. Sainath, et al., Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups, IEEE Signal Processing Magazine 29 (6) (2012) 82–97.

[4] J. Redmon, S. Divvala, R. Girshick, A. Farhadi, You only look once: Unified, real-time object detection, in: Conference on Computer Vision and Pattern Recognition, 2016.

[5] S. Ren, K. He, R. Girshick, J. Sun, Faster r-cnn: Towards real-time object detection with region proposal networks, IEEE Transactions on Pattern Analysis and Machine Intelligence 39 (6) (2016) 1137–1149.

[6] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, G. Monfardini, The graph neural network model, IEEE Transactions on Neural Networks 20 (1) (2008) 61–80.

[7] C. Gallicchio, A. Micheli, Graph echo state networks, in: International Joint Conference on Neural Networks, 2010, pp. 1–8.

[8] Y. Li, D. Tarlow, M. Brockschmidt, R. Zemel, Gated graph sequence neural networks, arXiv preprint arXiv:1511.05493 (2015).

[9] H. Dai, Z. Kozareva, B. Dai, A. Smola, L. Song, Learning steady-states of iterative algorithms over graphs, in: International conference on machine learning, 2018, pp. 1106–1114.

[10] J. Bruna, W. Zaremba, A. Szlam, Y. LeCun, Spectral networks and locally connected networks on graphs, in: International Conference on Learning Representations, 2014.

[11] M. Defferrard, X. Bresson, P. Vandergheynst, Convolutional neural networks on graphs with fast localized spectral filtering, in: Advances in Neural Information Processing Systems, 2016.

[12] T. N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, in: International Conference on Learning Representations, 2017.

[13] J. Atwood, D. Towsley, Diffusion-convolutional neural networks, in: Advances in Neural Information Processing Systems, 2016, pp. 1993–2001.

[14] M. Niepert, M. Ahmed, K. Kutzkov, Learning convolutional neural networks for graphs, in: International Conference on Machine Learning, 2016.

[15] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, G. E. Dahl, Neural message passing for quantum chemistry, in: International Conference on Machine Learning, 2017.

[16] Y. Li, O. Vinyals, C. Dyer, R. Pascanu, P. Battaglia, Learning deep generative models of graphs, arXiv preprint arXiv:1803.03324 (2018).

[17] J. You, R. Ying, X. Ren, W. L. Hamilton, J. Leskovec, Graphrnn: A deep generative model for graphs, arXiv preprint arXiv:1802.08773 (2018).

[18] M. Simonovsky, N. Komodakis, Graphvae: Towards generation of small graphs using variational autoencoders, in: International Conference on Artificial Neural Networks, 2018.

[19] T. Ma, J. Chen, C. Xiao, Constrained generation of semantically valid graphs via regularizing variational autoencoders, in: Advances in Neural Information Processing Systems, 2018.

[20] S. Cao, W. Lu, Q. Xu, Deep neural networks for learning graph representations., in: AAAI Conference on Artificial Intelligence, 2016.

[21] D. Wang, P. Cui, W. Zhu, Structural deep network embedding, in: International Conference on Knowledge Discovery and Data Mining, 2016.

[22] T. N. Kipf, M. Welling, Variational graph auto-encoders, arXiv preprint arXiv:1611.07308 (2016).

[23] S. Pan, R. Hu, G. Long, J. Jiang, L. Yao, C. Zhang, Adversarially regularized graph autoencoder for graph embedding, in: International Joint Conference on Artificial Intelligence, 2018.

[24] B. Perozzi, R. Al-Rfou, S. Skiena, Deepwalk: Online learning of social representations, in: International Conference on Knowledge Discovery and Data mining, 2014.

[25] A. Grover, J. Leskovec, node2vec: Scalable feature learning for networks, in: International Conference on Knowledge Discovery and Data Mining, 2016.

[26] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, Q. Mei, Line: Large-scale information network embedding, in: International Conference on World Wide Web, 2015.

[27] G. Hinton, S. Roweis, Visualizing data using t-SNE, in: Advances in Neural Information Processing Systems, 2002, pp. 833—-840.

[28] L. der Maaten, G.Hinton, Visualizing data using t-SNE, Journal of Machine Learning Research 9 (11) (2008) 2579—-2605.

[29] L. V. D. Maaten, Learning a parametric embedding by preserving local structure, in: Artificial Intelligence and Statistics, 2009, pp. 384—-391.

[30] S. T. Roweis, L. K. Saul, Nonlinear dimensionality reduction by locally linear embedding, science 290 (5500) (2000) 2323–2326.

[31] B. Shaw, T. Jebara, Structure preserving embedding, in: International Conference on Machine Learning, 2009.

[32] M. Ou, P. Cui, J. Pei, Z. Zhang, W. Zhu, Asymmetric transitivity preserving graph embedding, in: International Conference on Knowledge Discovery and Data Mining, 2016.

[33] S. Yan, D. Xu, B. Zhang, H. jiang Zhang, Q. Yang, S. Lin, graph embedding and extensions: A general framework for dimensionality reduction, IEEE Transactions on Pattern Analysis and Machine Intelligence 29 (1) (2007) 40—-51.

[34] V. Mygdalis, A. Iosifidis, A. Tefas, I. Pitas, Graph embedded one-class classifiers for media data classification, Pattern Recognition 60 (2017) 585—-595.

[35] A. Iosifidis, A. Tefas, I. Pitas, graph embedded extreme learning machine, IEEE Transactions on Cybernetics 46 (1) (2016) 311—-324.

[36] P. Goyal, E. Ferrara, Graph embedding techniques, applications, and performance: A survey, Knwoledge-based Systems 151 (7) (2018) 78—-94.

[37] H. Cai, V. W. Zheng, K. C.-C. Chang, A comprehensive survey of graph embedding: Problems, techniques, and applications, IEEE Transactions on Knowledge and Data Engineering 30 (2018) 1616—-1637.

[38] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, P. Vandergheynst, The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains, IEEE Signal Processing Magazine 30 (3) (2013) 83–98.

[39] A. Sandryhaila, J. M. Moura, Discrete signal processing on graphs, IEEE Transactions on Signal Processing 61 (7) (2013) 1644–1656.

[40] S. Chen, R. Varma, A. Sandryhaila, J. Kovačević, Discrete signal processing on graphs: Sampling theory, IEEE Transactions on Signal Processing 63 (24) (2015) 6510–6523.

[41] A. Ancona, R. Lyons, Y. Peres, Crossing estimates and convergence of dirichlet functions along random walk and diffusion paths, Annals of probability (1999) 970–989.

[42] T. Hutchcroft, Harmonic dirichlet functions on planar graphs, Discrete & Computational Geometry 61 (3) (2019) 479–506.

[43] D. K. Hammond, P. Vandergheynst, R. Gribonval, Wavelets on graphs via spectral graph theory, Applied and Computational Harmonic Analysis 30 (2) (2011) 129–150.

[44] N. Heidari, A. Iosifidis, Progressive graph convolutional networks for semi-supervised node classification, IEEE Access (2021).

[45] R. Li, S. Wang, F. Zhu, J. Huang, Adaptive graph convolutional neural networks, in: AAAI Conference on Artificial Intelligence, 2018.

[46] C. Zhuang, Q. Ma, Dual graph convolutional networks for graph-based semi-supervised classification, in: International Conference on World Wide Web, 2018, pp. 499–508.

[47] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik,

R. P. Adams, Convolutional networks on graphs for learning molecular fingerprints, in: Advances in Neural Information Processing Systems, 2015.

[48] Y. Li, D. Tarlow, M. Brockschmidt, R. S. Zemel, Gated graph sequence neural networks, in: Y. Bengio, Y. LeCun (Eds.), International Conference on Learning Representations, 2016.

[49] P. Battaglia, R. Pascanu, M. Lai, D. Jimenez Rezende, et al., Interaction networks for learning about objects, relations and physics, Advances in Neural Information Processing Systems 29 (2016) 4502–4510.

[50] S. Kearnes, K. McCloskey, M. Berndl, V. Pande, P. Riley, Molecular graph convolutions: moving beyond fingerprints, Journal of computer-aided molecular design 30 (8) (2016) 595–608.

[51] K. T. Schütt, F. Arbabzadah, S. Chmiela, K. R. Müller, A. Tkatchenko, Quantum-chemical insights from deep tensor neural networks, Nature communications 8 (1) (2017) 1–8.

[52] M. Defferrard, X. Bresson, P. Vandergheynst, Convolutional neural networks on graphs with fast localized spectral filtering, Advances in Neural Information Processing Systems 29 (2016) 3844–3852.

[53] H. Gao, Z. Wang, S. Ji, Large-scale learnable graph convolutional networks, in: International Conference on Knowledge Discovery & Data Mining, 2018.

[54] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, Y. Bengio, Graph attention networks, in: International Conference on Learning Representations, 2018.

[55] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, I. Polosukhin, Attention is all you need, in: Advances in neural information processing systems, Vol. 30, 2017, pp. 5998–6008.

[56] W. Hamilton, Z. Ying, J. Leskovec, Inductive representation learning on large graphs, in: Advances in Neural Information Processing Systems, 2017.

[57] J. Chen, T. Ma, C. Xiao, Fastgcn: Fast learning with graph convolutional networks via importance sampling, in: International Conference on Learning Representations, 2018.

[58] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, J. Leskovec, Graph convolutional neural networks for web-scale recommender systems, in: International Conference on Knowledge Discovery & Data Mining, 2018.

[59] J. Chen, J. Zhu, L. Song, Stochastic training of graph convolutional networks with variance reduction, in: International Conference on Machine Learning, 2018.

[60] W. Huang, T. Zhang, Y. Rong, J. Huang, Adaptive sampling towards fast graph representation learning, Advances in Neural Information Processing Systems 31 (2018) 4558–4567.

[61] H. Robbins, S. Monro, A stochastic approximation method, The Annals of Mathematical Statistics (1951) 400–407.

[62] H. Zeng, H. Zhou, A. Srivastava, R. Kannan, V. Prasanna, Accurate, efficient and scalable graph embedding, in: International Parallel and Distributed Processing Symposium, 2019, pp. 462–471.

[63] W.-L. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, C.-J. Hsieh, Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks, in: International Conference on Knowledge Discovery & Data Mining, 2019.

[64] G. Karypis, V. Kumar, A fast and high quality multilevel scheme for partitioning irregular graphs, SIAM Journal on Scientific Computing 20 (1) (1998) 359–392.

[65] I. S. Dhillon, Y. Guan, B. Kulis, Weighted graph cuts without eigenvectors a multilevel approach, IEEE Transactions on Pattern Analysis and Machine Intelligence 29 (11) (2007) 1944–1957.

[66] C. Morris, N. M. Kriege, F. Bause, K. Kersting, P. Mutzel, M. Neumann, Tudataset: A collection of benchmark datasets for learning with graphs, arXiv preprint arXiv:2007.08663

(2020).

[67] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, T. Eliassi-Rad, Collective classification in network data, AI magazine 29 (3) (2008) 93–93.

[68] J. Bastings, I. Titov, W. Aziz, D. Marcheggiani, K. Sima'an, Graph convolutional encoders for syntax-aware neural machine translation, in: Conference on Empirical Methods in Natural Language Processing, 2017, pp. 1957–1967.

[69] L. Yao, C. Mao, Y. Luo, Graph convolutional networks for text classification, in: AAAI Conference on Artificial Intelligence, 2019.

[70] D. Beck, G. Haffari, T. Cohn, Graph-to-sequence learning using gated graph neural networks, in: Association for Computational Linguistics, 2018, pp. 273–283.

[71] R. v. d. Berg, T. N. Kipf, M. Welling, Graph convolutional matrix completion, arXiv preprint arXiv:1706.02263 (2017).

[72] F. Monti, M. M. Bronstein, X. Bresson, Geometric matrix completion with recurrent multi-graph neural networks, in: Conference on Neural Information Processing Systems, 2017, pp. 3697–3707.

[73] J. Zhang, X. Shi, J. Xie, H. Ma, I. King, D. Yeung, Gaan: Gated attention networks for learning on large and spatiotemporal graphs, in: Conference on Uncertainty in Artificial Intelligence, 2018, pp. 339–349.

[74] V. G. Satorras, J. B. Estrach, Few-shot learning with graph neural networks, in: International Conference on Learning Representations, 2018.

[75] M. Kampffmeyer, Y. Chen, X. Liang, H. Wang, Y. Zhang, E. P. Xing, Rethinking knowledge graph propagation for zero-shot learning, in: IEEE Conference on Computer Vision and Pattern Recognition, 2019.

[76] S. Yan, Y. Xiong, D. Lin, Spatial temporal graph convolutional networks for skeleton-based action recognition, in: AAAI Conference on Artificial Intelligence, 2018.

[77] X. Qi, R. Liao, J. Jia, S. Fidler, R. Urtasun, 3d graph neural networks for rgbd semantic segmentation, in: IEEE International Conference on Computer Vision, 2017, pp. 5199–5208.

[78] S. Qi, W. Wang, B. Jia, J. Shen, S.-C. Zhu, Learning human-object interactions by graph parsing neural networks, in: Proceedings of the European Conference on Computer Vision (ECCV), 2018, pp. 401–417.

[79] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, J. M. Solomon, Dynamic graph cnn for learning on point clouds, Acm Transactions On Graphics (tog) 38 (5) (2019) 1–12.

[80] L. Landrieu, M. Simonovsky, Large-scale point cloud semantic segmentation with superpoint graphs, in: IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 4558–4567.

[81] A. Shahroudy, J. Liu, T.-T. Ng, G. Wang, Ntu rgb+ d: A large scale dataset for 3d human activity analysis, in: IEEE conference on computer vision and pattern recognition, 2016, pp. 1010–1019.

[82] W. Kay, J. Carreira, K. Simonyan, B. Zhang, C. Hillier, S. Vijayanarasimhan, F. Viola, T. Green, T. Back, P. Natsev, et al., The kinetics human action video dataset, arXiv preprint arXiv:1705.06950 (2017).

[83] Z. Cao, T. Simon, S.-E. Wei, Y. Sheikh, Realtime multi-person 2d pose estimation using part affinity fields, in: IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 7291–7299.

[84] M. Neumann, P. Moreno, L. Antanas, R. Garnett, K. Kersting, Graph kernels for object category prediction in task-dependent robot grasping, in: Online Proceedings of the Eleventh Workshop on Mining and Learning with Graphs, 2013, pp. 0–6.

[85] M. Neumann, R. Garnett, C. Bauckhage, K. Kersting, Propagation kernels: efficient graph

kernels from propagated information, Machine Learning 102 (2) (2016) 209–245.
[86] J. Tang, J. Zhang, L. Yao, J. Li, L. Zhang, Z. Su, Arnetminer: extraction and mining of academic social networks, in: International Conference on Knowledge Discovery and Data Mining, 2008.
[87] A. Sinha, Z. Shen, Y. Song, H. Ma, D. Eide, B.-J. Hsu, K. Wang, An overview of microsoft academic service (mas) and applications, in: International Conference on World Wide Web, 2015, pp. 243–246.
[88] L. Tang, H. Liu, Relational learning via latent social dimensions, in: International Conference on Knowledge Discovery and Data Mining, 2009.
[89] T. Hogg, K. Lerman, Social dynamics of digg, EPJ Data Science 1 (1) (2012) 5.
[90] M. De Domenico, A. Lima, P. Mougel, M. Musolesi, The anatomy of a scientific rumor, Scientific reports 3 (2013) 2980.
[91] J. Zhang, B. Liu, J. Tang, T. Chen, J. Li, Social influence locality for modeling retweeting behaviors., in: International Joint Conference on Artificial Intelligence, 2013.
[92] P. Yanardag, S. Vishwanathan, Deep graph kernels, in: International Conference on Knowledge Discovery and Data Mining, 2015.
[93] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proceedings of the IEEE 86 (11) (1998) 2278–2324.
[94] A. Krizhevsky, G. Hinton, et al., Learning multiple layers of features from tiny images (2009).
[95] K. Riesen, H. Bunke, Iam graph database repository for graph based pattern recognition and machine learning, in: Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR), 2008, pp. 287–297.
[96] A. K. Debnath, R. L. Lopez de Compadre, G. Debnath, A. J. Shusterman, C. Hansch, Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity, Journal of Medicinal Chemistry 34 (2) (1991) 786–797.
[97] H. Toivonen, A. Srinivasan, R. D. King, S. Kramer, C. Helma, Statistical evaluation of the predictive toxicology challenge 2000–2001, Bioinformatics 19 (10) (2003) 1183–1193.
[98] P. D. Dobson, A. J. Doig, Distinguishing enzyme structures from non-enzymes without alignments, Journal of Molecular Biology 330 (4) (2003) 771–783.
[99] I. Schomburg, A. Chang, C. Ebeling, M. Gremse, C. Heldt, G. Huhn, D. Schomburg, Brenda, the enzyme database: updates and major new developments, Nucleic acids research 32 (suppl_1) (2004) D431–D433.
[100] K. M. Borgwardt, C. S. Ong, S. Schönauer, S. Vishwanathan, A. J. Smola, H.-P. Kriegel, Protein function prediction via graph kernels, Bioinformatics 21 (suppl_1) (2005) i47–i56.
[101] N. Wale, I. A. Watson, G. Karypis, Comparison of descriptor spaces for chemical compound retrieval and classification, Knowledge and Information Systems 14 (3) (2008) 347–375.
[102] R. Ramakrishnan, P. O. Dral, M. Rupp, O. A. Von Lilienfeld, Quantum chemistry structures and properties of 134 kilo molecules, Scientific data 1 (1) (2014) 1–7.
[103] M. Zitnik, J. Leskovec, Predicting multicellular function through multi-layer tissue networks, Bioinformatics 33 (14) (2017) i190–i198.
[104] V. P. Dwivedi, C. K. Joshi, T. Laurent, Y. Bengio, X. Bresson, Benchmarking graph neural networks, arXiv preprint arXiv:2003.00982 (2020).
[105] W. Jin, R. Barzilay, T. Jaakkola, Junction tree variational autoencoder for molecular graph generation, arXiv preprint arXiv:1802.04364 (2018).
[106] G. Chen, P. Chen, C.-Y. Hsieh, C.-K. Lee, B. Liao, R. Liao, W. Liu, J. Qiu, Q. Sun, J. Tang, et al., Alchemy: A quantum chemistry dataset for benchmarking ai models, arXiv preprint

arXiv:1906.09427 (2019).

[107] M. Fey, J. Eric Lenssen, F. Weichert, H. Müller, Splinecnn: Fast geometric deep learning with continuous b-spline kernels, in: IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 869–877.

[108] M. Wang, L. Yu, D. Zheng, Q. Gan, Y. Gai, Z. Ye, M. Li, J. Zhou, Q. Huang, C. Ma, et al., Deep graph library: Towards efficient and scalable deep learning on graphs, arXiv preprint arXiv:1909.01315 (2019).

Publication 7

Structured Pruning Adapters

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

Publication: ArXiv preprint arXiv:2211.10155, 2022

Structured Pruning Adapters

Lukas Hedegaard [1] **Aman Alok** [2] **Juby Jose** [2] **Alexandros Iosifidis** [1]

Abstract

Adapters are a parameter-efficient alternative to fine-tuning, which augment a frozen base network to learn new tasks. Yet, the inference of the adapted model is often slower than the corresponding fine-tuned model. To improve on this, we propose Structured Pruning Adapters (SPAs), a family of compressing, task-switching network adapters, that accelerate and specialize networks using tiny parameter sets and structured pruning. Specifically, we propose a channel-based SPA and evaluate it with a suite of pruning methods on multiple computer vision benchmarks. Compared to regular structured pruning with fine-tuning, our channel-SPAs improve accuracy by 6.9% on average while using half the parameters at 90% pruned weights. Alternatively, they can learn adaptations with 17× fewer parameters at 70% pruning with 1.6% lower accuracy. Our experimental code and Python library of adapters are available at `github.com/lukashedegaard/structured-pruning-adapters`.

1. Introduction

Fine-tuning is an established approach to parameter-based transfer learning from a source model pre-trained on a large dataset to a target task with limited training data. However, the resulting model retains the same parameter count and computational characteristics as the source model, even when solving a considerably simpler task. A group of *fine-pruning* methods (Li et al., 2017; Molchanov et al., 2017; Sun et al., 2017; Yeom et al., 2021; Sanh et al., 2020; Lagunas et al., 2021) have combined pruning with fine-tuning to reduce model size while learning parameters for the target task. Generally, this results in compressed models that retain performance down to at least half the relative weight density and which may be better suited for resource-constrained deployments, such as mobile devices, robotics applications, and settings necessitating low-latency predictions.

Meanwhile, Adapters (Rebuffi et al., 2017; 2018) have emerged as a viable alternative to fine-tuning for multi-domain deep neural networks (DNNs), where a single source DNN is specialized and sequentially used for multiple tasks. Instead of continuing training of the source DNN weights directly, Adapters introduce parameter-efficient layer add-ons, which are trained instead. As these add-ons are much more compact than the source DNN weights, they can be transmitted and stored at low cost. This is very useful, *e.g.*, for edge devices and federated learning (McMahan et al., 2016). However, prior work has largely ignored the computational efficiency aspects of Adapters, which either increase the complexity of the network (He et al., 2022; Zhu et al., 2021; Li & Liang, 2021; Houlsby et al., 2019; Pfeiffer et al., 2021; Mahabadi et al., 2021) or leave it unaltered, at best, by utilizing structures that can be fused with the original weights (Rebuffi et al., 2017; 2018; Hu et al., 2022).

In general, a deployed model must strike a balance between predictive performance, storage requirements, inference speed, and flexibility of use. While the combination of pruning and fine-tuning can produce compressed models with good performance at an order of magnitude fewer parameters compared to the source model, *Structured Pruning Adapters (SPAs)* can improve upon this by another order of magnitude for task-switching networks. Specifically, we propose the *Structured Pruning Low-rank Adapter* (SPLoRA) for channel-based pruning and compare its performance and parameter count to pruning with fine-tuning in weight-based transfer learning from ResNet weights pre-trained on ILSVRC 2012 (Russakovsky et al., 2015) to the image classification benchmarks CIFAR-10 (Krizhevsky, 2009), Oxford Flowers 102 (Nilsback & Zisserman, 2008), and Cats and Dogs (Elson et al., 2007). Considering four different pruning methods, we find that SPLoRA not only reduces parameter requirements per task massively, but also retains predictive accuracy better than fine-tuning under aggressive pruning.

In the remainder of this paper, we describe related work on Adapters (Section 2.1) and Pruning (Section 2.3), a framework for SPAs (Section 3), the SPLoRA adapter (Section 3.1), experimental comparisons with fine-pruning (Section 4), and conclusions (Section 5).

[1]Department of Electrical and Computer Engineering, Aarhus University, Aarhus, Denmark [2]Cactus Communications. Correspondence to: Lukas Hedegaard <lhm@ece.au.dk>.

Preprint, 2023.

2. Related Work

2.1. Adapter methods

When multiple specialized versions of a network are deployed on the same device and storage requirements are strict, Adapters (Rebuffi et al., 2017) provide a low-parameter-count alternative to fine-tuning. Instead of deploying multiple sets of full network weights, a single set of full weights can be deployed alongside multiple adapter weights, which augment the main network. For Convolutional Neural Networks (CNNs), point-wise convolutions can be introduced in series (Rebuffi et al., 2017) or parallel (Rebuffi et al., 2018) with a residual connection to adapt fixed source weights to new tasks. For Transformer-based networks, prior work explored bottleneck projections with (Zhu et al., 2021) and without (Hu et al., 2022) low-dimensional non-linearity in parallel with the fixed dense layers of the original network. Prefix-tuning (Li & Liang, 2021), which learns task-specific prompt tokens to adapt the network, may be considered a parallel adapter as well (He et al., 2022). Adapter blocks can also be interspersed in series with existing layers (Houlsby et al., 2019; Pfeiffer et al., 2021; Mahabadi et al., 2021). He at al. (He et al., 2022) proposed a combination of the above. Finally, several works (Stickland & Murray, 2019; Pfeiffer et al., 2021; Rücklé et al., 2021) explored the use of multi-task adapters.

While the above-described methods succeed in learning parameter-efficient network add-ons with very small storage requirements, these adapters often incur an additional computational cost beyond the original network. Considering that the adapted target tasks are often simpler than the source task, it is reasonable to assume that a derived network adaptation can be learned, which reduces computational complexity as well.

2.2. Efficiency approaches

Multiple approaches have been proposed to reduce the compute and memory footprint of neural networks. *Knowledge distillation*[1] (Hinton et al., 2015; Gou et al., 2021) utilize a large network as a teacher for a smaller network, which has more desirable memory and computational characteristics. *Efficient architectures* (Tan & Le, 2019; Feichtenhofer, 2020; Hoffmann et al., 2022) define and optimize expressive yet efficient architectural blocks from random initialization under a multi-metric optimization goal. Low-rank factorizations (Tran et al., 2018; Guo et al., 2019) approximate large tensor weights by factorizing them into multiple lower-rank weights. *Continual Inference Networks* (Hedegaard & Iosifidis, 2022; Hedegaard et al., 2023) reuse the network weights of prior DNNs with a temporal component and accelerate them for online stream processing via optimized computational sequences and appropriate intra-layer caching. *Quantization approaches* (Gray & Neuhoff, 1998; Liang et al., 2021) reduce model size and run-time costs via low-resolution numerical representations of network weights. Finally, *Pruning methods* (LeCun et al., 1989; Han et al., 2015; Frankle & Carbin, 2019) entirely remove unnecessary network weight from a pre-trained model. While all of these are interesting research avenues both in isolation and combination, we focus on pruning-methods hereafter.

[1] Knowledge distillation (Hinton et al., 2015) using the unpruned model as the teacher has been found to help pruning methods (Sanh et al., 2020; Lagunas et al., 2021) retain accuracy better using fine-pruning. We expect this to be true for SPAs as well, but we leave the validation to future work as none of the considered fine-pruning baselines employed knowledge distillation.

2.3. Pruning methods

DNNs can be pruned at multiple levels: *Unstructured* pruning of individual weights results in sparse weight matrices which reduce parameter count but require specialized hardware to improve inference characteristics due to the performance disadvantages of sparse matrix multiplication kernels on graphical processing units (GPUs). On the other hand, *structured* pruning approaches, such as the pruning of entire channels (Yeom et al., 2021) or blocks (Lagunas et al., 2021) of networks weights, generally provide inference speedup across computational devices (Gray et al., 2017).

Many studies have proposed criteria on *what* to prune. Early methods (LeCun et al., 1989; Hassibi & Stork, 1992) proposed the use of second-order Taylor expansion of the loss Hessian for weight selection. As computing the inverse of the Hessian may be computationally intractable, another approach uses a first-order Taylor approximation of the loss change due to the pruning of units (Molchanov et al., 2017). A recent work uses fast Hessian-vector products to retain low complexity (Nonnenmacher et al., 2022). Similarly, the gradient of a weight with respect to the loss can be used for pruning selection (Liu & Wu, 2019; Sun et al., 2017; Sanh et al., 2020). Among the simplest approaches is the use of weight magnitudes in pruning selection. In our experimental comparisons, we employ Magnitude Pruning (Han et al., 2015), which uses a simple average of weight magnitudes, and Weight Pruning (Li et al., 2017), which uses the L_p-norm of weights, for channel selection. Yeom et al. (Yeom et al., 2021) proposed an explainability-inspired approach, computing the relevance of each network component by means of Layer-wise Relevance Pruning (LRP) (Bach et al., 2015; Montavon et al., 2019). We use this method in Section 4.

For all the above methods assigning pruning scores, another consideration is whether to rank and select structural units locally within a layer (keeping pruning evenly spread throughout the network) or globally, with a contest among all network layers. We utilize global selection in our experi-

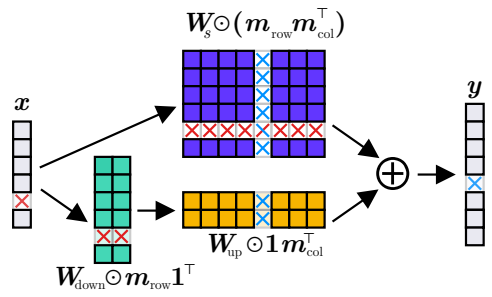


Figure 1. Structured Pruning Low-rank Adapter (SPLoRA). Pruning of in/out channels affects the adapter as well as source weights.

ments (Section 4).

Beyond the pruning criterion, multiple studies proposed *when* to prune. One popular approach (Molchanov et al., 2017; Liu & Wu, 2019; Yeom et al., 2021) is to use an iterative pruning and fine-tuning schedule, masking a predefined fraction of units at a time. Alternatively, Automated Gradual Pruning (Zhu & Gupta, 2018) allows all weights to be updated throughout the pruning schedule, enabling prior masking choices to be altered. We use the former in Section 4.

2.4. Transfer pruning

Pruning is useful not only for compressing a model while retaining predictive performance, but also for transfer learning (Yeom et al., 2021; Sanh et al., 2020; Lagunas et al., 2021). In fact, a task can be "learned" simply by selecting the appropriate subset of weights in a network (Mallya et al., 2018; Ramanujan et al., 2020).

Consider a large (pre-trained) source model f_s and a set of T target tasks for which we desire specialized target models $f_t, t \in \{1..T\}$. Under the framework of transfer learning with pruning ("transfer pruning"), we can concurrently update and mask weights from a source model to benefit a target task t. Consider $g : \boldsymbol{W}_s \times \Delta\boldsymbol{W}_t \times \mathbb{M}_t \rightarrow \boldsymbol{W}_t$, a function that generates target model weights $\boldsymbol{W}_t$, given learned update weights $\Delta\boldsymbol{W}_t$, source weights $\boldsymbol{W}_s$, and a learned masking set $\mathbb{M}_t$ of retained weight indices. Given available source weights $\boldsymbol{W}_s$, every task-specific model f_t can be stored as the parameters $\Phi_t = \{\Delta\boldsymbol{W}_t, \mathbb{M}_t\}$.

Under *fine-pruning* (Sanh et al., 2020), i.e., concurrent pruning and fine-tuning, g constitutes a direct assignment of weights, $\boldsymbol{W}_t := g(\boldsymbol{W}_s, \Delta\boldsymbol{W}_t, \mathbb{M}_t) = \Delta\boldsymbol{W}_t$, where update weights are learned based on a pruned subset, $\{\boldsymbol{W}_s^{(i)}, i \in \mathbb{M}_t\}$. Here, the parameters of the task-specific model are $\Phi_t = \{\boldsymbol{W}_t, \mathbb{M}_t\}$, and the size of the target weights is determined by the weight density $d \in (0, 1]$ and the size of source weights: $\|\boldsymbol{W}_t\|_0 = d\|\boldsymbol{W}_s\|_0$.

3. Structured Pruning Adapters

Although fine-pruning can successfully produce smaller target weights, i.e., $\|\boldsymbol{W}_t\|_0 < \|\boldsymbol{W}_s\|_0$, the set of weights for all tasks $\{\boldsymbol{W}_t\}$ may still consume an intractable amount of storage if many tasks T are involved and/or the average density $\bar{d}$ is large due to high predictive performance requirements. Instead, we seek to utilize adapters alongside pruning to produce an extremely compressed parameter set.

Consider the concurrent pruning and adaptation of a frozen source projection matrix $\boldsymbol{W}_s \in \mathbb{R}^{n\times m}$ with an index mask $\boldsymbol{M} \in \{0, 1\}^{n\times m}$ and an adapter function a. While applicable adapters have been extensively studied (see Section 2.1), we restrict ourselves to fusible parallel adapters to minimize the run-time of the resulting model. Accordingly, pruning adapters take the following basic form:

$$\boldsymbol{W}_t = (\boldsymbol{W}_s + a(\Delta\boldsymbol{W}_t)) \odot \boldsymbol{M}. \tag{1}$$

3.1. Structured Pruning Low-rank Adapter

Unlike unstructed pruning, *structured* methods remove groups of weights and increase computational efficiency. *Channel* pruning, in particular, maps a dense source matrix to a dense pruned matrix with computational improvements proportional to the number of removed parameters. A mask $\boldsymbol{M}$ in this case can be decomposed as row and column masks $\boldsymbol{m}_{\text{row}} \in \{0, 1\}^{n\times 1}$ and $\boldsymbol{m}_{\text{col}} \in \{0, 1\}^{m\times 1}$, respectively. Then, Equation (1) can be expressed as follows:

$$\boldsymbol{W}_t = (\boldsymbol{W}_s + a(\Delta\boldsymbol{W}_t)) \odot \boldsymbol{m}_{\text{row}}\boldsymbol{m}_{\text{col}}^\top. \tag{2}$$

A simple realization of a fusible parallel adapter is the Low-rank Adapter (LoRA) (Hu et al., 2022):

$$\boldsymbol{W}_t = \boldsymbol{W}_s + \boldsymbol{W}_{\text{down}}\boldsymbol{W}_{\text{up}}, \tag{3}$$

where $\boldsymbol{W}_{\text{down}} \in \mathbb{R}^{n\times r}$ and $\boldsymbol{W}_{\text{up}} \in \mathbb{R}^{r\times m}$ are adapter weights and r is the rank hyper-parameter. Following the derivation in Appendix A, we utilize Equations 2 and 3 to define the **S**tructured **P**runing **Lo**w **R**ank **A**dapter (SPLoRA):

$$\boldsymbol{W}_t = \boldsymbol{W}_s \odot \boldsymbol{m}_{\text{row}}\boldsymbol{m}_{\text{col}}^\top + (\boldsymbol{W}_{\text{down}} \odot \boldsymbol{m}_{\text{row}}\boldsymbol{1}^\top)(\boldsymbol{W}_{\text{up}} \odot \boldsymbol{1}\boldsymbol{m}_{\text{col}}^\top). \tag{4}$$

In this form, we see that channel-pruning affects not only the source weights $\boldsymbol{W}_s$, but also the adapter parameters $\boldsymbol{W}_{\text{up}}$ and $\boldsymbol{W}_{\text{down}}$. This effect is illustrated in Figure 1.

Adapters should generally be initialized to perform a near-zero mapping to avoid destabilizing gradients during initial training. Accordingly, we use the initialization $\boldsymbol{W}_{\text{up}}^{(i,j)} \sim \mathcal{U}(-10^{-4}, 10^{-4})$, although other near-zero initialization choices may be equally successful. Experiments validating this choice are presented in Section 4.2.

For a fine-pruned model weight, $\boldsymbol{W}_t$, the updated parameter count is $\|\boldsymbol{m}_{\text{row}}\|_0\|\boldsymbol{m}_{\text{col}}\|_0$. The corresponding SPLoRA

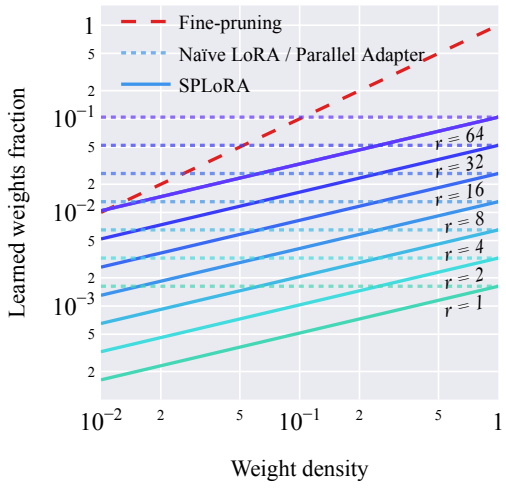


Figure 2. Learned weight fraction ($\|\Delta \boldsymbol{W}_t\|_0 / \|\boldsymbol{W}_s\|_0$) versus weight density ($\|\boldsymbol{W}_t\|_0 / \|\boldsymbol{W}_s\|_0$) for 768×3072 weights under channel-pruning.

has $r(\|\boldsymbol{m}_{\text{row}}\|_0 + \|\boldsymbol{m}_{\text{col}}\|_0)$ parameters. Parameter comparisons depends on the weight retention fraction, weight matrix shape, and SPLoRA bottleneck dimension r. Figure 2 visualizes the parameters achieved by varying the hyper-parameters for a 768×3072 matrix. Here, we depict the learned parameter count of fine-pruning and SPLoRA alongside the naïve application of LoRA or, equivalently, the Parallel Adapter (He et al., 2022; Zhu et al., 2021) for reference. While naïve adapter usage requires far-fewer parameters than fine-tuning at high weight density, the parameter reduction achieved by fine-pruning can produce equally few parameters at low weight densities. SPLoRA combines the benefits of both approaches to produce parameter sets far more compact than those produced by either.

3.2. Limitations

While each set of SPA weights has significantly fewer learned parameters than each set of fine-pruning weights, the superiority of SPAs is dependent on the deployment at hand. When only a single model is deployed, the two approaches are equivalent in terms of model size, considering that the fused parameter count of adapters is identical to that of the fine-pruned network. For T models deployed on one device, the adapter format saves space when $T > 1/\overline{d}$, assuming available source weights $\boldsymbol{W}_s$. Thus, the feasibility depends on the average density $\overline{d}$, with more aggressive pruning requiring more deployed models for SPA formats to consume less storage space.

3.3. Pruning of adapter-specific parameters

The design choices made for SPLoRA (Section 3.1) let us utilize prior channel-based pruning without modification by pruning fused weights, which are identical in form to the source weights, and subsequently pruning the adapter-weights with the same mask. The adapters themselves could also be subject to pruning (Rücklé et al., 2021). For SPLoRA, the bottleneck contains r channels that could be pruned per layer to produce a heterogeneous set of layer adapters with different ranks per layer. However, such pruning adds an additional layer of complexity and hyper-parameters to the already complicated pruning methods, which may reduce the attractiveness of adapters. Moreover, the pruning of intra-adapter channels only reduces the parameter count and does not yield computational benefits. Therefore, we limit the pruning in subsequent experiments to the approach we believe has the best usability in practice; we refrain from pruning over internal adapter structures.

4. Experiments

We seek to compare structured pruning with fine-tuning (fine-pruning) to our proposed channel-based Structured Pruning Adapter method, SPLoRA. As both have identical acceleration benefits during inference, our experimental comparison focuses on prediction performance and the number of learned parameters (ΔParams).

4.1. Experimental setup

In this set of experiments, we reuse and augment a previously reported setup (Yeom et al., 2021) to perform transfer-pruning for ResNet models pretrained on ILSVRC 2012 (Russakovsky et al., 2015) to the image classification benchmarks CIFAR-10 (Krizhevsky, 2009), Oxford Flowers 102 (Nilsback & Zisserman, 2008), and Cats and Dogs (Elson et al., 2007). As no publicly available test split was available for the latter, we defined train-test splits and pre-processed data using DatasetOps (Hedegaard et al., 2022) to match the 8005 training and 2023 test samples reported previously (Yeom et al., 2021). For transfer pruning, we first train the network without pruning for $\{30, 100, 100\}$ epochs and then perform incremental pruning at increments of 5% until 5% of weights remain in total; the incremental pruning is interspersed with $\{20, 50, 50\}$ epochs of training for the {CIFAR-10, Oxford Flowers 102, Cats and Dogs} datasets. Appendix B presents an overview of training times. We employ the Strochastic Gradient Descent optimizer with a momentum of 0.9, weight decay of $5 \cdot 10^{-4}$, and learning rate of 0.01 at a batch size 256 or down-scaled rates following the *linear scaling rule* (Krizhevsky, 2014) when GPU memory limitations must be accomodated. In each training block, we use a step learning rate reduction of $5\times$ after each quarter of epochs. The above setup is used for either

Table 1. Channel-based transfer-pruning from ResNet-50 pre-trained on ImageNet to Cats and Dogs, Oxford Flowers 102, and CIFAR-10 using methods based on Weight (Li et al., 2017), Gradient (Sun et al., 2017), Taylor (Molchanov et al., 2017), and LRP (Yeom et al., 2021). Please note that SPLoRA and LoRA are identical at 100% density. ΔParams and floating point operations (FLOPs) are shown for CIFAR-10. Mean ± standard deviation is shown for each metric. Changes specified relative to closest fine-pruning density with same pruning method. Best metric per pruning-method is highlighted with bold.

Pruning method	Learning method	Density	ΔParams (K)	FLOPs (M)	CIFAR-10 Acc. (%)	Oxford Flowers 102 Acc. (%)	Cats & Dogs Acc. (%)
Unpruned	Fine-tuning	100%	23,520.8±0.0	1,304.7±0.0	**97.10**±0.12	**92.20**±0.00	**99.30**±0.02
	(SP)LoRA-$r32$	100%	1,644.5±0.0 (↓ 14.3×)	1,304.7±0.0	95.32±0.13 (−1.8)	78.57±5.93 (−13.6)	98.60±0.43 (−0.5)
	(SP)LoRA-$r8$	100%	**466.3**±0.0 (↓ 50.4×)	1,304.7±0.0	95.35±0.10 (−1.8)	80.96±5.83 (−11.2)	98.84±0.52 (−0.46)
Weight	Fine-pruning	30%	4,427.1±72.5	785.4±5.4	**96.38**±0.12	93.64±2.15	**98.64**±0.04
		10%	599.1±20.1	352.1±3.9	87.23±2.00	72.83±0.93	95.42±0.31
	SPLoRA-$r32$	30%	618.3±2.5 (↓ 7.2×)	778.6±2.9	95.59±0.22 (−0.8)	**94.57**±0.58 (+0.9)	98.43±0.13 (−0.2)
		10%	294.8±0.4 (↓ 2.0×)	**335.4**±7.3	93.04±0.37 (+5.8)	89.36±1.09 (+16.5)	96.25±0.99 (+0.8)
	SPLoRA-$r8$	30%	210.0±1.9 (↓ 21.1×)	773.4±2.1	94.91±0.24 (−1.5)	92.13±0.15 (−1.5)	98.40±0.12 (−0.2)
		10%	**128.9**±0.1 (↓ 4.6×)	338.9±7.0	91.24±0.51 (+4.0)	86.49±0.74 (+13.7)	96.07±0.69 (+0.6)
Gradient	Fine-pruning	30%	3,719.8±59.2	571.9±6.5	**95.95**±0.09	**94.21**±0.74	**98.22**±0.00
		10%	615.7±4.3	244.7±3.3	91.83±1.17	73.06±0.70	95.84±0.20
	SPLoRA-$r32$	30%	601.0±0.4 (↓ 6.2×)	564.6±1.5	94.91±0.09 (−1.0)	93.58±0.43 (−0.6)	98.17±0.08 (−0.0)
		10%	293.1±0.4 (↓ 2.1×)	**244.0**±1.9	93.65±0.36 (+1.8)	91.35±0.47 (+18.3)	97.54±0.17 (+1.7)
	SPLoRA-$r8$	30%	205.2±0.2 (↓ 18.1×)	565.2±4.1	94.09±0.28 (−1.9)	91.60±0.30 (−2.6)	98.15±0.07 (−0.1)
		10%	**128.3**±0.0 (↓ 4.8×)	245.4±4.0	91.25±0.20 (−0.6)	87.46±0.71 (+14.4)	97.19±0.28 (+1.4)
Taylor	Fine-pruning	30%	3,392.8±81.1	559.9±0.7	**95.71**±0.02	92.91±0.56	**98.22**±0.18
		10%	576.8±9.9	236.9±3.3	88.07±0.66	65.67±4.12	95.30±0.21
	SPLoRA-$r32$	30%	599.7±0.9 (↓ 5.7×)	555.5±6.7	94.88±0.21 (−0.8)	**93.41**±0.06 (+0.5)	97.84±0.48 (−0.4)
		10%	292.6±0.5 (↓ 2.0×)	**242.0**±1.5	93.27±0.12 (+5.2)	91.30±0.10 (+25.6)	97.21±0.10 (+1.9)
	SPLoRA-$r8$	30%	205.3±0.1 (↓ 16.5×)	566.2±10.9	93.98±0.24 (−1.7)	91.51±0.49 (−1.4)	97.90±0.13 (−0.3)
		10%	**128.4**±0.1 (↓ 4.5×)	243.2±9.8	91.22±0.32 (+3.2)	86.76±0.42 (+21.1)	96.83±0.30 (+1.5)
LRP	Fine-pruning	30%	4,428.1±20.6	719.9±0.7	**96.54**±0.14	**95.37**±0.08	**98.65**±0.07
		10%	608.4±6.3	301.6±2.3	93.52±0.05	87.56±2.75	–
	SPLoRA-$r32$	30%	592.6±0.9 (↓ 7.5×)	585.5±6.7	94.85±0.13 (−1.7)	93.62±0.38 (−1.8)	98.09±0.11 (−0.6)
		10%	290.9±0.2 (↓ 2.1×)	**270.4**±6.4	93.47±0.36 (+0.1)	91.22±0.46 (+3.7)	97.01±0.27
	SPLoRA-$r8$	30%	203.3±0.5 (↓ 21.8×)	591.1±12.4	93.53±0.18 (−3.0)	91.26±0.19 (−4.1)	97.80±0.20 (−0.8)
		10%	**128.0**±0.1 (↓ 4.7×)	281.6±1.7	90.94±0.39 (−2.5)	85.69±1.39 (−1.9)	96.88±0.52

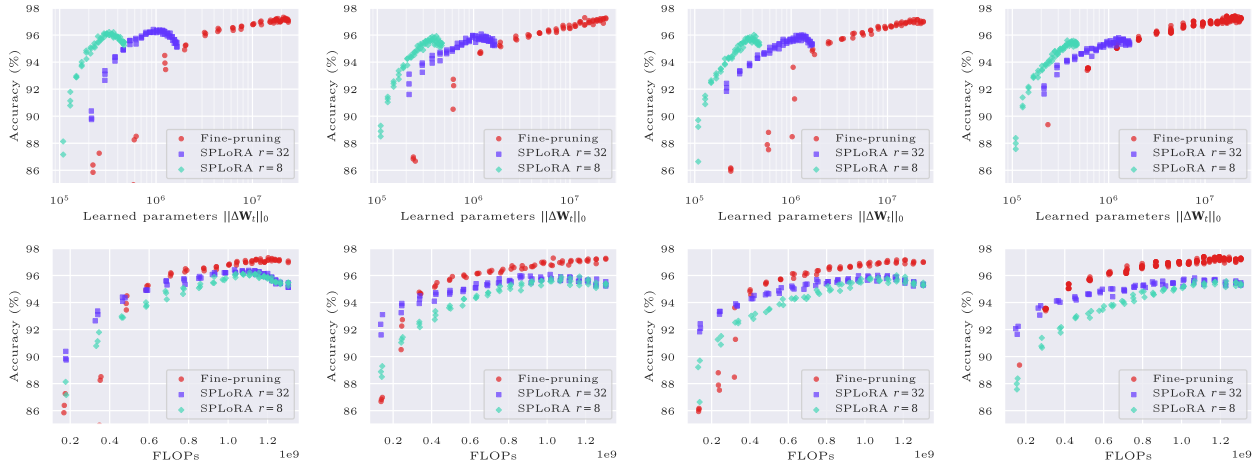
fine-pruning, in which all model weights are updated, or adaptation and pruning, which freezes the original network weights and only trains the adapter weights, normalization, and prediction head.

4.2. SPLoRA initialization and rank choice

To gauge the sensitivity of SPLoRA hyper-parameters and their effect on predictive performance, we perform a set of adaptation and pruning runs using L_2-normalized Taylor pruning (Molchanov et al., 2017) on CIFAR-10 with a ResNet-18 network. Here, we vary the rank $r \in 2^{[0,6]}$ and initialization range in $10^{[-6,-2]}$ and evaluate along densities $\{1.0, 0.5, 0.3, 0.2, 0.1\}$. As illustrated in Figure 4, we observe a clear and expected trend of increasing accuracy as the rank is increased. The increases exhibit diminishing returns with limited benefit beyond $r = 32$ for CIFAR-10. While all tested ranks show similar accuracy at a density of $d = 1.0$, the lowest-rank adapters are more severely affected by a lower d than higher-rank ones. This follows intuition, considering that lower-rank adapters have fewer parameters that might prove redundant during pruning. In some cases ($r \geq 16$) pruning at $d \approx 0.5$ increases accuracy. SPLoRA is generally robust to the chosen initialization range, showing no clear trends in favor of particular ranges. We will use $\boldsymbol{W}_{\text{up}}^{(i,j)} \sim \mathcal{U}(-10^{-4}, 10^{-4})$ in subsequent experiments.

4.3. Comparison with fine-pruning

In our comparison of SPLoRA and fine-pruning, we train a ResNet-50 network for weight-based transfer learning from ILSVRC 2022 to CIFAR-10, Oxford Flower 102, and Cats and Dogs across four structured pruning works, namely, the normalized Weight (Li et al., 2017), Taylor (Molchanov et al., 2017), Gradient (Sun et al., 2017), and LRP (Li et al., 2017) pruning methods. This comparison is conducted for



(a) Weight pruning. (b) Gradient pruning. (c) Taylor pruning. (d) LRP pruning.

Figure 3. CIFAR-10 accuracy versus total learned parameter count $\|\Delta \boldsymbol{W}_t\|_0$ (top row) and FLOPs (bottom row) using fine-pruning and SPLoRA with ranks 32 and 8 for the (a) Weight (Li et al., 2017), (b) Gradient (Sun et al., 2017), (c) Taylor (Molchanov et al., 2017), and (d) LRP (Yeom et al., 2021) channel-pruning methods.

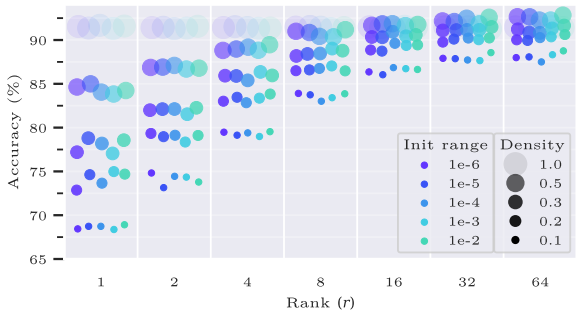


Figure 4. ResNet-18 accuracy on CIFAR-10 using SPLoRA of varying ranks (r) and adapter weight initialization ranges at different model densities (d). Point size $\propto d$ with transparency $\propto -d$.

both fine-pruning and SPLoRA with the ranks $r = 8$ and $r = 32$. To accommodate the stochastic nature of pruning and neural network training, we repeat each experiment three times and report the mean and standard deviation of each metric. The results of our experiments are presented in Table 1 for model densities of 100%, 30%, and 10% retained weights and visualized for CIFAR-10 in Figure 3 as well as for Oxford Flowers 102 and Cats and Dogs[2] in Appendix C

Comparing SPLoRA with fine-pruning, we observed competitive transfer adaptations despite learning a fraction the weights. While fine-pruning generally resulted in higher accuracy at 30% model density (on average 0.6% and 1.6% higher than SPLoRA ranks 32 and 16), SPLoRA had far fewer learned parameters (on average 6.2× and 17.0×). At 10% density, SPLoRA was both more robust to pruning (achieving 6.9% and 4.7% higher average accuracy than fine-pruning for ranks 32 and 16), while reducing the number of learned parameters (by 2.0× and 4.2× on average). The difference in parameter reduction between the 30% and 10% density targets is explained partly by the normalization and linear parameters (73K for ResNet-50), that begin to dominate the learned weight budget at low densities, and partly by the remaining weights of adapter bottleneck channels. Even fewer parameters would be needed if a lower rank hyper-parameter was chosen or if, as discussed in Section 3.2, pruning of bottleneck channels was conducted as well. As FLOPs follow model densities, these are approximately equal for each learning method given equal densities.

5. Conclusion

We proposed Structured Pruning Adapters (SPAs) as an alternative to fine-tuning during structured pruning. Instead of updating all model weights, SPAs consist of prunable lightweight add-on modules, which are learned in place of the original weights but can be fused with them at runtime to obtain the same computational enhancements as regular structured pruning with fine-tuning. Our channel-based SPAs were shown to achieve competitive performance across a battery of pruning methods on computer vision benchmarks while requiring a fraction of the learned parameters per task. Thus, SPAs are ideal for task-switching storage-constrained and/or network-limited usage scenarios,

[2]We found LRP with fine-pruning to be unstable at low model densities. Despite attempts with multiple different seeds, results for Cats and Dogs could not be obtained (denoted by "-" in Table 1).

where the per-model size should be small.

Acknowledgments

Lukas Hedegaard and Alexandros Iosifidis acknowledge funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 871449 (OpenDR).

An acknowledgment also goes to Martin Damgaard Nielsen and Jens Dalgaard Nielsen for the early-stage conversations relating to this project.

References

Bach, S., Binder, A., Montavon, G., Klauschen, F., Müller, K.-R., and Samek, W. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PLoS One*, 10(7):1–46, 2015.

Elson, J., Douceur, J. J., Howell, J., and Saul, J. 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.

Feichtenhofer, C. X3d: Expanding architectures for efficient video recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

Frankle, J. and Carbin, M. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations, (ICLR)*, 2019. URL https://openreview.net/forum?id=rJl-b3RcF7.

Gou, J., Yu, B., Maybank, S. J., and Tao, D. 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.

Gray, R. and Neuhoff, D. Quantization. *IEEE Transactions on Information Theory*, 44(6):2325–2383, 1998. doi: 10.1109/18.720541.

Gray, S., Radford, A., and Kingma, D. P. Gpu kernels for block-sparse weights. *OpenAI*, 2017. URL https://cdn.openai.com/blocksparse/blocksparsepaper.pdf.

Guo, K., Xie, X., Xu, X., and Xing, X. Compressing by learning in a low-rank and sparse decomposition form. *IEEE Access*, 7:150823–150832, 2019. doi: 10.1109/ACCESS.2019.2947846.

Han, S., Pool, J., Tran, J., and Dally, W. J. Learning both weights and connections for efficient neural networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems (NeurIPS)*, pp. 1135–1143. MIT Press, 2015.

Hassibi, B. and Stork, D. G. Second order derivatives for network pruning: Optimal brain surgeon. In *Proceedings of the 5th International Conference on Neural Information Processing Systems*, NIPS'92, pp. 164–171, 1992. ISBN 1558602747.

He, J., Zhou, C., Ma, X., Berg-Kirkpatrick, T., and Neubig, G. Towards a unified view of parameter-efficient transfer learning. In *International Conference on Learning Representations*, 2022.

Hedegaard, L. and Iosifidis, A. Continual 3d convolutional neural networks for real-time processing of videos. In *European Conference on Computer Vision (ECCV)*, pp. 1–18, 2022.

Hedegaard, L., Oleksiienko, I., and Legaard, C. M. DatasetOps, 2022. URL https://github.com/lukashedegaard/datasetops.

Hedegaard, L., Bakhtiarnia, A., and Iosifidis, A. Continual Transformers: Redundancy-free attention for online inference. In *International Conference on Learning Representations (ICLR)*, 2023.

Hinton, G. E., Vinyals, O., and Dean, J. Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531, 2015. URL http://arxiv.org/abs/1503.02531.

Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., Casas, D. d. L., Hendricks, L. A., Welbl, J., Clark, A., Hennigan, T., Noland, E., Millican, K., Driessche, G. v. d., Damoc, B., Guy, A., Osindero, S., Simonyan, K., Elsen, E., Rae, J. W., Vinyals, O., and Sifre, L. Training compute-optimal large language models. *preprint, arXiv:2203.15556*, 2022. doi: 10.48550/ARXIV.2203.15556. URL https://arxiv.org/abs/2203.15556.

Houlsby, N., Giurgiu, A., Jastrzebski, S., Morrone, B., De Laroussilhe, Q., Gesmundo, A., Attariyan, M., and Gelly, S. Parameter-efficient transfer learning for NLP. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 2790–2799, 2019.

Hu, E. J., yelong shen, Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

Krizhevsky, A. 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.

Krizhevsky, A. One weird trick for parallelizing convolutional neural networks. *preprint, arXiv:1404.5997*, 2014. doi: 10.48550/ARXIV.1404.5997.

Lagunas, F., Charlaix, E., Sanh, V., and Rush, A. Block pruning for faster transformers. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 10619–10629, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics.

LeCun, Y., Denker, J., and Solla, S. Optimal brain damage. In Touretzky, D. (ed.), *Advances in Neural Information Processing Systems*, volume 2. Morgan-Kaufmann, 1989. URL https://proceedings.neurips.cc/paper/1989/file/6c9882bbac1c7093bd25041881277658-Paper.pdf.

Li, H., Kadav, A., Durdanovic, I., Samet, H., and Graf, H. P. Pruning filters for efficient convnets. In *International Conference on Learning Representations (ICLR)*, 2017. URL https://openreview.net/forum?id=rJqFGTslg.

Li, X. L. and Liang, P. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 4582–4597, 2021.

Liang, T., Glossner, J., Wang, L., Shi, S., and Zhang, X. Pruning and quantization for deep neural network acceleration: A survey. *Neurocomputing*, 461(C):370–403, oct 2021. ISSN 0925-2312. doi: 10.1016/j.neucom.2021.07.045. URL https://doi.org/10.1016/j.neucom.2021.07.045.

Liu, C. and Wu, H. Channel pruning based on mean gradient for accelerating convolutional neural networks. *Signal Processing*, 156:84–91, 2019. ISSN 0165-1684. doi: https://doi.org/10.1016/j.sigpro.2018.10.019. URL https://www.sciencedirect.com/science/article/pii/S0165168418303517.

Mahabadi, R. K., Henderson, J., and Ruder, S. Compacter: Efficient low-rank hypercomplex adapter layers. In *Advances in Neural Information Processing Systems*, 2021.

Mallya, A., Davis, D., and Lazebnik, S. 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.

McMahan, H. B., Moore, E., Ramage, D., Hampson, S., and y Arcas, B. A. Communication-efficient learning of deep networks from decentralized data. In *International Conference on Artificial Intelligence and Statistics*, 2016.

Molchanov, P., Tyree, S., Karras, T., Aila, T., and Kautz, J. Pruning convolutional neural networks for resource efficient inference. In *International Conference on Learning Representations (ICLR)*, 2017.

Montavon, G., Binder, A., Lapuschkin, S., Samek, W., and Müller, K.-R. *Layer-Wise Relevance Propagation: An Overview*, pp. 193–209. Springer International Publishing, 2019.

Nilsback, M.-E. and Zisserman, A. Automated flower classification over a large number of classes. In *2008 Sixth Indian Conference on Computer Vision, Graphics & Image Processing*, pp. 722–729, 2008. doi: 10.1109/ICVGIP.2008.47.

Nonnenmacher, M., Pfeil, T., Steinwart, I., and Reeb, D. SOSP: Efficiently capturing global correlations by second-order structured pruning. In *International Conference on Learning Representations (ICLR)*, 2022. URL https://openreview.net/forum?id=t5EmXZ3ZLR.

Pfeiffer, J., Kamath, A., Rücklé, A., Cho, K., and Gurevych, I. 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*, pp. 487–503, 2021.

Ramanujan, V., Wortsman, M., Kembhavi, A., Farhadi, A., and Rastegari, M. What's hidden in a randomly weighted neural network? In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

Rebuffi, S.-A., Bilen, H., and Vedaldi, A. Learning multiple visual domains with residual adapters. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Rebuffi, S.-A., Bilen, H., and Vedaldi, A. Efficient parametrization of multi-domain deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

Rücklé, A., Geigle, G., Glockner, M., Beck, T., Pfeiffer, J., Reimers, N., and Gurevych, I. AdapterDrop: On the efficiency of adapters in transformers. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, 2021.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. 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.

Sanh, V., Wolf, T., and Rush, A. M. Movement pruning: Adaptive sparsity by fine-tuning. In *NeurIPS*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/eae15aabaa768ae4a5993a8a4f4fa6e4-Abstract.html.

Stickland, A. C. and Murray, I. BERT and PALs: Projected attention layers for efficient adaptation in multi-task learning. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 5986–5995, 2019.

Sun, X., Ren, X., Ma, S., and Wang, H. 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*, pp. 3299–3308, International Convention Centre, Sydney, Australia, 2017.

Tan, M. and Le, Q. EfficientNet: Rethinking model scaling for convolutional neural networks. In Chaudhuri, K. and Salakhutdinov, R. (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 6105–6114. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/tan19a.html.

Tran, D. T., Iosifidis, A., and Gabbouj, M. Improving efficiency in convolutional neural network with multilinear filters. *Neural Networks*, 105:328–339, 2018.

Yeom, S.-K., Seegerer, P., Lapuschkin, S., Binder, A., Wiedemann, S., Müller, K.-R., and Samek, W. Pruning by explaining: A novel criterion for deep neural network pruning. *Pattern Recognition*, 115, 2021.

Zhu, M. and Gupta, S. 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.

Zhu, Y., Feng, J., Zhao, C., Wang, M., and Li, L. Counter-interference adapter for multilingual machine translation. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pp. 2812–2823, 2021.

A. SPLoRA Derivation

Utilizing Equation (3) in the context of channel-SPAs of the form expressed in Equation (2), we can derive the Structured Pruning Low-rank Adapter defined in Equation (4), where adapter parameters are pruned alongside source weights. This derivation is straightforward, considering that the application of a structured pruning mask $\boldsymbol{M} = \boldsymbol{m}_{\text{row}}\boldsymbol{m}_{\text{col}}^{\top}$ via Hadamard products is equivalent to a projection with diagonalized masking vectors:

$$\boldsymbol{W} \odot \boldsymbol{m}_{\text{row}}\boldsymbol{m}_{\text{col}}^{\top} = \text{diag}(\boldsymbol{m}_{\text{row}})\ \boldsymbol{W}\ \text{diag}(\boldsymbol{m}_{\text{col}}). \tag{5}$$

Similarly, a single diagonalized mask can be expressed via Hadamark products:

$$\text{diag}(\boldsymbol{m}_{\text{row}})\ \boldsymbol{W} = \boldsymbol{W} \odot \boldsymbol{m}_{\text{row}}\mathbf{1}^{\top}. \tag{6}$$

Utilizing Equation (3), Equation (5), and Equation (6), we can rewrite Equation (2) as follows:

$$\begin{aligned}
\boldsymbol{W}_t &= (\boldsymbol{W}_s + a(\Delta\boldsymbol{W}_t)) \odot \boldsymbol{m}_{\text{row}}\boldsymbol{m}_{\text{col}}^{\top} \\
&= (\boldsymbol{W}_s + \boldsymbol{W}_{\text{down}}\boldsymbol{W}_{\text{up}}) \odot \boldsymbol{m}_{\text{row}}\boldsymbol{m}_{\text{col}}^{\top} \\
&= \text{diag}(\boldsymbol{m}_{\text{row}})\ (\boldsymbol{W}_s + \boldsymbol{W}_{\text{down}}\boldsymbol{W}_{\text{up}})\ \text{diag}(\boldsymbol{m}_{\text{col}}) \\
&= \text{diag}(\boldsymbol{m}_{\text{row}})\ \boldsymbol{W}_s\text{diag}(\boldsymbol{m}_{\text{col}}) + (\text{diag}(\boldsymbol{m}_{\text{row}})\boldsymbol{W}_{\text{down}})(\boldsymbol{W}_{\text{up}}\text{diag}(\boldsymbol{m}_{\text{col}})) \\
&= \boldsymbol{W}_s \odot \boldsymbol{m}_{\text{row}}\boldsymbol{m}_{\text{col}}^{\top} + (\boldsymbol{W}_{\text{down}} \odot \boldsymbol{m}_{\text{row}}\mathbf{1}^{\top})(\boldsymbol{W}_{\text{up}} \odot \mathbf{1}\boldsymbol{m}_{\text{col}}^{\top}),
\end{aligned}$$

where the final result is equivalent to Equation (4).

B. Training Durations

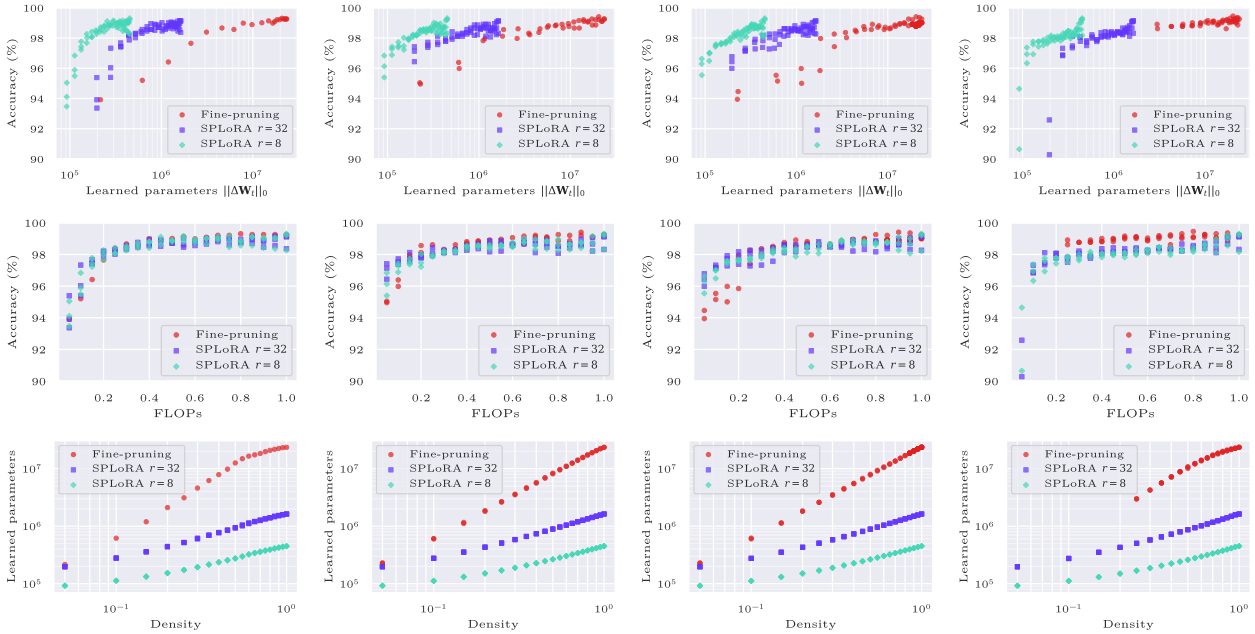
In this section, we provide a brief overview of approximate training durations for the methods tested in the present paper. As training times are comparable among different pruning methods, we report a single metric approximated from multiple pruning methods. These are presented in Table 2 for our experiments using ResNet-50 in image recognition tasks. Here, the pruning methods gradually reduce the network density while producing pruned models at a predefined step reduction in density, cycling the learning rate for each density reduction step. Accordingly, the noted training times for the pruned learning methods in Table 2 includes the training of all models with densities ranging from 100% to 5% at 5% intervals.

Table 2. Training durations for the ResNet-50 model on image recognition transfer tasks using a NVIDIA RTX 2080 Ti GPU. For each dataset, the batch size (BS) and training duration (T) are presented.

Pruning method	**Learning method**	**CIFAR-10**		**Oxf. Fl. 102**		**C. & D.**	
		BS	T	BS	T	BS	T
Unpruned	Fine-tuning	64	0:30h	64	1:05h	64	2:20h
	SPLoRA-$r32$	64	0:30h	32	1:10h	32	2:30h
	SPLoRA-$r8$	64	0:30h	32	1:10h	32	2:30h
Pruned	Fine-pruning	64	10h	4	25h	6	42h
	SPLoRA-$r32$	64	11h	4	33h	6	48h
	SPLoRA-$r8$	64	11h	4	31h	6	45h

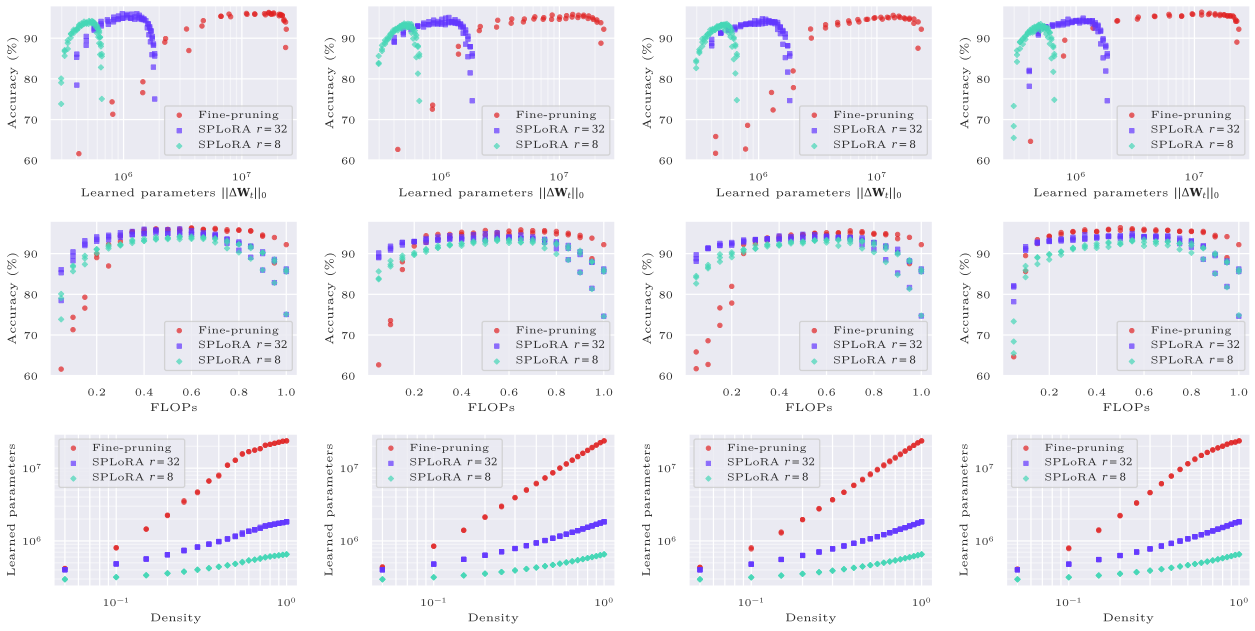
C. Supplemental Visualisations of experimental results with SPLoRA

For completeness, Figure 5 and Figure 6 illustrate trade-offs among accuracy, learned parameters, and FLOPs.



(a) Weight pruning (Li et al., 2017).
(b) Gradient pruning (Sun et al., 2017).
(c) Taylor pruning (Molchanov et al., 2017).
(d) LRP pruning (Yeom et al., 2021).

Figure 5. Cats and Dogs accuracy versus learned parameter count $\|\Delta \boldsymbol{W}_t\|_0$ (top row) and FLOPs (middle row) as well as learned parameter count versus model density (bottom row) using fine-pruning and SPLoRA with ranks 32 and 8 for various channel-pruning methods.



(a) Weight pruning (Li et al., 2017).
(b) Gradient pruning (Sun et al., 2017).
(c) Taylor pruning (Molchanov et al., 2017).
(d) LRP pruning (Yeom et al., 2021).

Figure 6. Oxford Flowers 102 accuracy versus learned parameter count $\|\Delta \boldsymbol{W}_t\|_0$ (top row) and FLOPs (middle row) as well as learned parameter count versus model density (bottom row) using fine-pruning and SPLoRA with ranks 32 and 8 for various channel-pruning methods.