

# VU Research Portal

## The Good, the Efficient and the Inductive Biases:

Romero Guzman, David Wilson

2024

**DOI (link to publisher)**  
[10.5463/thesis.738](https://doi.org/10.5463/thesis.738)

### **document version**

Publisher's PDF, also known as Version of record

[Link to publication in VU Research Portal](#)

### **citation for published version (APA)**

Romero Guzman, D. W. (2024). *The Good, the Efficient and the Inductive Biases: Exploring Efficiency in Deep Learning Through the Use of Inductive Biases*. [PhD-Thesis - Research and graduation internal, Vrije Universiteit Amsterdam]. <https://doi.org/10.5463/thesis.738>

### **General rights**

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal

### **Take down policy**

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

**E-mail address:**  
[vuresearchportal.ub@vu.nl](mailto:vuresearchportal.ub@vu.nl)

# The Good, the Efficient and the Inductive Biases

Exploring Efficiency in Deep Learning Through the  
Use of Inductive Biases

## ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad Doctor of Philosophy aan  
de Vrije Universiteit Amsterdam,  
op gezag van de rector magnificus  
prof.dr. J.J.G. Geurts,  
in het openbaar te verdedigen  
ten overstaan van de promotiecommissie  
van de Faculteit der Bètawetenschappen  
op dinsdag 10 september 2024 om 15.45 uur  
in een bijeenkomst van de universiteit,  
De Boelelaan 1105

door  
David Wilson Romero Guzman  
geboren te Tunja, Colombia

promotor: prof.dr. M. Hoogendoorn

copromotoren: dr. E.J. Bekkers  
dr. J.M. Tomczak

promotiecommissie: prof.dr. S. Bhulai  
prof.dr. M. Cuturi  
prof.dr. C. Ré  
prof.dr. M. Welling  
dr. R. Yu

SIKS Dissertation Series No. 2024-29

The research reported in this thesis has been carried out  
under the auspices of SIKS, the Dutch Research School  
for Information and Knowledge Systems.



Funding for this research was provided by the Netherlands Organization for Scientific Research (NWO) grant number P16-25 and a 2021 Qualcomm Innovation Fellowship. Additional funding from Google Research and Mitsubishi Electric Research Laboratories was provided during the internships done at the corresponding companies.

Copyright © 2024 David W. Romero



# Summary

The emergence of Deep Learning has marked a profound shift in the paradigm of machine learning, a change driven by the numerous breakthroughs it has achieved in recent years. However, as the field evolves and Deep Learning becomes increasingly present in everyday tools and applications, there is an increasing need to address unresolved challenges related to its efficiency and sustainability.

This dissertation delves into the role of inductive biases, particularly continuous modeling and symmetry preservation, to address these challenges and enhance the efficiency of Deep Learning. The dissertation is structured in two main parts.

The first part investigates continuous modeling as a tool to improve the efficiency of Deep Learning algorithms. Continuous modeling involves the idea of parameterizing neural operations directly in a continuous space. The research presented in this part highlights the substantial benefits of continuous modeling for the *(i)* computational efficiency –in time and memory–, *(ii)* the parameter efficiency, and *(iii)* the complexity of designing neural architectures for new datasets and tasks, coined “*design efficiency*”.

In the second half, the focus shifts towards the influence of symmetry preservation on the efficiency of Deep Learning algorithms. Symmetry preservation involves designing neural operations that align with the inherent symmetries of data. The research presented in this part highlights significant gains both in data and parameter efficiency through the use of symmetry preservation. However, it also acknowledges a resulting trade-off of increased computational costs.

The dissertation concludes with a thorough critical evaluation of the research findings, openly discussing their limitations and proposing strategies to address them, informed by literature and the author’s insights. It ends by identifying promising future research avenues in the exploration of inductive biases for efficiency, and their wider implications for Deep Learning.



# Acknowledgements

I would like to start by expressing my profound gratitude to my advisors Mark, Erik and Jakub for their continued confidence and support in all of my endeavors during my PhD. Mark for believing in me from the very beginning and always supporting and encouraging me to grow both as a scientist and as a person. Erik for helping me improve my mathematical knowledge, rigor and precision. Without your guidance, this dissertation would not have been what it is today. And Jakub, for always being there to listen and for making my PhD journey as fun as it was! I will never forget how fun our meetings were –and your technical contributions too, of course.

Further, I would like to thank the many great researchers I had the honor to collaborate with during my PhD, many of whom –perhaps unknowingly– have deeply impacted the way I conduct research: Anna Kuzina, Robert-Jan Bruintjes, Jan van Gemert, Jean-Baptiste Cordonnier, Gabriele Cesa, Guillaume Sautière, Auke Wiggers, Taco Cohen, Suhas Lohit, Tycho van der Ouderaa, Albert Gu, Efstratios Gavves, Neil Zeghidour, Stephano Massaroli, Michael Poli, Hermann Kumbong, Putri van der Linden and Sharvaree Vadgama. I am particularly grateful to Jean-Baptiste Cordonnier for teaching me great deal of good software engineering practices; to Neil Zeghidour for his great advice on so many different topics and for encouraging me to continue valuing the principles of signal processing, regardless of how far Deep Learning has come; and to Jan van Gemert for teaching me the value of asking the right questions. I am very thankful for all of our discussions and conversations. I learned a tremendous amount from each of you.

To the students I had the pleasure of supervising: Joep van Genderingen, Cees Kandorp, Meena Alfons, Alonso Urbano and especially David Knigge, who became a close collaborator and friend afterward. I am extremely proud of all your achievements.

To my fellow QDA, CI and VU members: Ali el Hassouni, Alessandro Zonta, Luis Silvestrin, Jan Klein, Floris den Hengst, Anne Fischer, Tariq Dam, Bülent Ündes, Jacob Kooi, Arwin Gansekoele, Louk Smabil, Olivier Moulin, Tariq Dam, Yvonne Blokland, Frank Bennis, Vincent François-Lavet, Shujian Yu, Yura Perugachi-Diaz, Fuda van Diggelen, Jie Luo, Babak Kargar, Mateo de Carlo, Tugay Karagüzel, Karine Miras, Anil Yaman, Guszti Eiben, and those mentioned above. I really enjoyed those Friday-drinks evenings in the old building and playing real-life *Among Us!* at De Jansakkerhoeve.

To the many members of the UvA who made me feel at home both at the UvA and all

around the globe, although I was not formally part of the it: Riccardo Valperga, Samuele Papa, Miltos Kofinas, Alex Gabel, Floor Eijkelboom, Tin Veljković, David Wessels, David Vos, David Zhang, Emiel Hoogeboom, Mohammad Derakhshani, Ivona Najdenkoska and those already mentioned above.

To my TU Berlin, Universidad Nacional and high-school friends who have always believed in me, encouraged me to pursue my dreams, and have always cherished each of my successes, regardless of how small they might seem. I am particularly thankful to Maria Paula Luna, Sebastian Barragan, David Amezquita, Sebastian Medina, Eric Sanchez, Roldany Gutierrez, David Torres, Eric Liberra, Patricia Aponte, Ludwig Winkler, Sami Ede, Lorenz Vaitl and Helge Mohn.

To many other people who have been crucial in my journey to become the person I am today: Ana Cecilia Vargas, Samuel Triana, Yeimi Fajardo, Giselle van Dongen, Marianne van Boxtel, Marja Hortulanus, Gert van Dongen and Ronny Hänsch. I am especially grateful to Lola Chavarro, who always welcomed me into her home with love and affection. I hope you will be with me in spirit at my defense, wherever you may be now.

To my aunts, uncles, cousins and grandparents from both the Romero and Guzman sides. Your unconditional love and support has profoundly marked my life and has been a cornerstone throughout my life. I am particularly grateful to Amparo Gibson, David Gibson and Yesid Romero, who have always treated me as a son of their own and have supported me to the fullest extent of their ability.

To my parents, without whom I would not be writing this here today. My dad, Wilson, who nurtured my curiosity, love and admiration for science since an early age, and to my mom, Olga, who taught me to believe in myself, dream big, and that with hard work I could achieve any goal I set for myself. I admire and love you from the bottom of my heart, and I aspire to be as wonderful a parent as you have been to me.

To my brothers, Yecid and Martin for their love, support and admiration. For keeping me up to date with what the new generation is up to and getting me addicted to new stupid games all the time. You mean the world to me.

To Jacqueline, for her unconditional love and support. Thank you for helping me become a better person and guiding me and supporting me in taking the best decisions, both professionally and personally. I am very excited about our new adventure on the other side of the globe and look forward to sharing many more amazing experiences with you. Finally, to Carla, Hans and Mark for welcoming me into your family and for cherishing each of my successes with such enthusiasm and joy. I am truly happy to be part of your family.

# Publications

## PART I: EFFICIENCY THROUGH CONTINUOUS MODELING

- **David W. Romero**, Anna Kuzina, Erik J Bekkers, Jakub M Tomczak and Mark Hoogendoorn. CKConv: Continuous kernel convolution for sequential data. In *International Conference on Learning Representations*, 2022.
- **David W. Romero\***, David M. Knigge\*, Albert Gu, Erik J. Bekkers, Efstratios Gavves, Jakub M. Tomczak and Mark Hoogendoorn. Towards a General Purpose CNN for Long Range Dependencies in ND. In *ICML Workshop on Continuous Time Methods for Machine Learning*, 2022.
- David M. Knigge\*, **David W. Romero\***, Albert Gu, Efstratios Gavves, Erik J. Bekkers, Jakub M. Tomczak, Mark Hoogendoorn and Jan-Jakob Sonke. Modelling long range dependencies in ND: From task-specific to a general purpose CNN. In *International Conference on Learning Representations*, 2023.
- Putri A. van der Linden\*, **David W. Romero\*** and Erik J. Bekkers. Learned Gridification for Efficient Point Cloud Processing. In *Proceedings of the 2<sup>nd</sup> Annual Workshop on Topology, Algebra, and Geometry in Machine Learning (ICML)*, 2023.
- **David W. Romero\***, Robert-Jan Bruintjes\*, Jakub M. Tomczak, Erik J. Bekkers, Mark Hoogendoorn and Jan C. van Gemert. Flexconv: Continuous kernel convolutions with differentiable kernel sizes. In *International Conference on Learning Representations*, 2022.
- **David W. Romero** and Neil Zeghidour. DNArch: Learning convolutional neural architectures by backpropagation. *Under review*, 2023.

## PART II: EFFICIENCY THROUGH SYMMETRY PRESERVATION

- **David W. Romero** and Mark Hoogendoorn. Co-attentive equivariant neural networks: Focusing equivariance on transformations co-occurring in data. In *International Conference on Learning Representations*, 2020.
- **David W. Romero**, Erik J. Bekkers, Jakub M. Tomczak and Mark Hoogendoorn. Attentive group equivariant convolutional networks. In *International Conference on Machine Learning*, pp. 8188-8199. PMLR, 2020.

---

\*Equal contribution.

- **David W. Romero** and Jean-Baptiste Cordonnier. Group equivariant stand-alone self-attention for vision. In *International Conference on Learning Representations*, 2021.
- **David W. Romero**, Erik J. Bekkers, Jakub M. Tomczak and Mark Hoogendoorn. Wavelet networks: Scale-translation equivariant learning from raw time-series. In *Transactions on Machine Learning Research*, 2023.
- **David W. Romero** and Suhas Lohit. Learning equivariances and partial equivariances from data. In *Advances in Neural Information Processing Systems 35*, pp. 36466-36478, 2022.

#### OTHER PUBLICATIONS

- David M. Knigge, **David W. Romero** and Erik J. Bekkers. Exploiting redundancy: Separable group convolutional networks on lie groups. In *International Conference on Machine Learning* pp. 11359-11386. PMLR, 2022.
- Tycho van der Ouderaa, **David W. Romero** and Mark van der Wilk. Relaxing equivariance constraints with non-stationary continuous filters. In *Advances in Neural Information Processing Systems 35*, pp. 33818-33830, 2022.
- Stefano Massaroli, Michael Poli, Daniel Y. Fu, Hermann Kumbong, **David W. Romero**, Rom N. Parnichkun, Arman Timalsina, Quinn McIntyre, Beidi Chen, Atri Rudra, Ce Zhang, Christopher Ré, Stefano Ermon and Yoshua Bengio. Laughing hyena distillery: Extracting compact recurrences from convolutions. In *Advances in Neural Information Processing Systems*, 2023.
- Erik J. Bekkers, Sharvaree Vadgama, Rob Hesselink, Putri A. van der Linden and **David W. Romero**. Fast, Expressive  $SE(n)$  Equivariant Networks through Weight Sharing in Position-Orientation Space. In *International Conference on Learning Representations*, 2024.
- Alonso Urbano and **David W. Romero**. Self-Supervised Detection of Perfect and Partial Input-Dependent Symmetries. In *ICML Workshop on Geometry-grounded Representation Learning and Generative Modeling*, 2024.

# Contents

|          |   |           |
|----------|---|-----------|
| <b>1</b> | <b>Introduction</b>   | <b>1</b>  |
| 1.1      | Inductive Biases and Deep Learning . . . . .  | 2         |
| 1.2      | Efficiency through Inductive Biases . . . . .   | 3         |
| 1.2.1    | Efficiency Aspects of Deep Learning . . . . .   | 4         |
| 1.2.2    | Continuous modeling and symmetry preservation . . . . .   | 4         |
| 1.3      | Research Objective . . . . .  | 8         |
| 1.4      | Dissertation Structure and Contributions . . . . .  | 9         |
| 1.4.1    | How to read this dissertation . . . . .   | 14        |
| <b>I</b> | <b>EFFICIENCY THROUGH CONTINUOUS MODELING</b>   | <b>15</b> |
| <b>2</b> | <b>Continuous Kernel Convolutions for Sequential Data</b>   | <b>17</b> |
| 2.1      | Introduction . . . . .  | 17        |
| 2.2      | Related Work . . . . .  | 19        |
| 2.3      | The Convolution and Common Kernel Parameterizations . . . . .   | 19        |
| 2.4      | Continuous Kernel Convolution . . . . .   | 21        |
| 2.4.1    | Formulation and Properties . . . . .  | 21        |
| 2.4.2    | The Continuous Convolutional Kernel $\text{MLP}^\psi$ . . . . .   | 23        |
| 2.5      | Experiments . . . . .   | 25        |
| 2.6      | Discussion and Limitations . . . . .  | 28        |
| 2.7      | Conclusion and Future Work . . . . .  | 29        |
| <b>3</b> | <b>Modelling Long Context in ND: From Task-Specific to a General Purpose CNN</b>                          | <b>31</b> |
| 3.1      | Introduction . . . . .  | 31        |
| 3.2      | Related Work . . . . .  | 33        |
| 3.3      | Toward Data-Independent CNN Architectures . . . . .   | 34        |
| 3.3.1    | Pointwise Operations: Linear Layers, Dropout, Pointwise Nonlinearities and Residual Connections . . . . . | 35        |
| 3.3.2    | Global Operations: Normalization Layers and Global Pooling . . . . .                                      | 35        |
| 3.3.3    | Local Operations: Convolutional Layers and Subsampling . . . . .  | 35        |
| 3.4      | A General Purpose CNN Architecture . . . . .  | 38        |
| 3.4.1    | Modifications and Improvements . . . . .  | 39        |
| 3.5      | Experiments . . . . .   | 41        |

|          |   |           |
|----------|---|-----------|
| 3.6      | Limitations and Future Work . . . . .                                   | 44        |
| <b>4</b> | <b>Learnable Gridification for Efficient Point-Cloud Processing</b>     | <b>45</b> |
| 4.1      | Introduction . . . . .  | 45        |
| 4.2      | Method . . . . .  | 47        |
| 4.2.1    | Point cloud and grid representations . . . . .                          | 47        |
| 4.2.2    | Gridification: From a point cloud to a dense grid . . . . .             | 48        |
| 4.2.3    | De-gridification: From a dense grid to a point cloud . . . . .          | 49        |
| 4.2.4    | Requirements and properties of gridification . . . . .                  | 49        |
| 4.2.5    | Materializing the gridification module . . . . .                        | 50        |
| 4.2.6    | Gridified networks for global and dense prediction . . . . .            | 51        |
| 4.3      | Related Work . . . . .  | 52        |
| 4.4      | Experiments . . . . .   | 53        |
| 4.4.1    | Random point cloud reconstruction . . . . .                             | 54        |
| 4.4.2    | ModelNet40 classification . . . . .                                     | 55        |
| 4.4.3    | ShapeNet part segmentation . . . . .                                    | 55        |
| 4.4.4    | Efficiency analysis of gridification . . . . .                          | 56        |
| 4.5      | Limitations and future work . . . . .                                   | 57        |
| 4.6      | Conclusion . . . . .  | 58        |
| <b>5</b> | <b>Continuous Kernel Convolutions with Differentiable Kernel Sizes</b>  | <b>59</b> |
| 5.1      | Introduction . . . . .  | 59        |
| 5.2      | Related Work . . . . .  | 61        |
| 5.3      | Method . . . . .  | 63        |
| 5.3.1    | Flexible Size Continuous Kernel Convolution (FlexConv) . . . . .        | 63        |
| 5.3.2    | Multiplicative Anisotropic Gabor Networks (MAGNets) . . . . .           | 63        |
| 5.3.3    | Analytic Alias-free MAGNets . . . . .                                   | 64        |
| 5.4      | Experiments . . . . .   | 66        |
| 5.4.1    | What kind of functions can MAGNets approximate? . . . . .               | 66        |
| 5.4.2    | Classification Tasks . . . . .  | 67        |
| 5.4.3    | Alias-free FlexNets . . . . .   | 69        |
| 5.5      | Discussion . . . . .  | 70        |
| 5.6      | Limitations . . . . .   | 71        |
| 5.7      | Conclusion . . . . .  | 72        |
| <b>6</b> | <b>Learning Convolutional Neural Architectures by Backpropagation</b>   | <b>73</b> |
| 6.1      | Introduction . . . . .  | 73        |
| 6.2      | Method . . . . .  | 75        |
| 6.2.1    | Differentiable Masking . . . . .  | 75        |
| 6.2.2    | Continuous Kernel Convolutions . . . . .                                | 76        |
| 6.2.3    | The need for learnable architectures . . . . .                          | 76        |
| 6.2.4    | Learning CNN architectures by backpropagation . . . . .                 | 77        |
| 6.2.5    | Learning neural architectures under computational constraints . . . . . | 81        |
| 6.3      | Experiments . . . . .   | 82        |

|           |   |            |
|-----------|---|------------|
| 6.3.1     | Using DNArch without computational constraints . . . . .  | 83         |
| 6.3.2     | Using DNArch under computational constraints . . . . .  | 84         |
| 6.3.3     | Architectures found by DNArch . . . . .   | 85         |
| 6.4       | Limitations . . . . .   | 86         |
| 6.5       | Outlook and future work . . . . .   | 86         |
| <b>II</b> | <b>EFFICIENCY THROUGH SYMMETRY PRESERVATION</b>   | <b>89</b>  |
| <b>7</b>  | <b>Focusing Equivariance on Transformations Co-Occurring in Data</b>  | <b>91</b>  |
| 7.1       | Introduction . . . . .  | 91         |
| 7.2       | Preliminaries . . . . .   | 95         |
| 7.3       | Co-Attentive Equivariant Neural Networks . . . . .  | 96         |
| 7.3.1     | Co-Attentive Rotation Equivariant Neural Networks . . . . .   | 96         |
| 7.3.2     | Extending $\mathcal{A}^C$ To Multiple Symmetry Groups . . . . .   | 100        |
| 7.4       | Experiments . . . . .   | 101        |
| 7.5       | Discussion and Future Work . . . . .  | 103        |
| 7.6       | Conclusion . . . . .  | 104        |
| <b>8</b>  | <b>Attentive Group Equivariant Convolutional Neural Networks</b>  | <b>105</b> |
| 8.1       | Introduction . . . . .  | 105        |
| 8.2       | Preliminaries . . . . .   | 107        |
| 8.2.1     | Spatial Convolution and Translation Equivariance . . . . .  | 107        |
| 8.2.2     | Group Convolution and Group Equivariance . . . . .  | 108        |
| 8.2.3     | Attention, Self-Attention and Visual Attention . . . . .  | 111        |
| 8.3       | Attentive Group Equivariant Convolution . . . . .   | 112        |
| 8.3.1     | Tying Equivariance and Visual Attention Together . . . . .  | 113        |
| 8.3.2     | Efficient Group Equivariant Attention Maps . . . . .  | 114        |
| 8.3.3     | The Residual Attention Branch . . . . .   | 117        |
| 8.3.4     | Attentive group convolution as a sequence of group convolutions<br>and point-wise non-linearities . . . . . | 118        |
| 8.4       | Experiments . . . . .   | 118        |
| 8.4.1     | rot-MNIST . . . . .   | 119        |
| 8.4.2     | CIFAR-10 . . . . .  | 119        |
| 8.4.3     | PCam . . . . .  | 120        |
| 8.5       | Discussion and future work . . . . .  | 121        |
| 8.6       | Conclusion . . . . .  | 122        |
| <b>9</b>  | <b>Group Equivariant Stand-Alone Self-Attention</b>   | <b>123</b> |
| 9.1       | Introduction . . . . .  | 123        |
| 9.2       | Related Work . . . . .  | 125        |
| 9.3       | Stand-Alone Self-Attention . . . . .  | 125        |
| 9.3.1     | The role of the positional encoding . . . . .   | 126        |
| 9.3.2     | A functional formulation to self-attention . . . . .  | 127        |

|           |   |            |
|-----------|---|------------|
| 9.4       | Equivariance Analysis of Self-Attention . . . . .   | 128        |
| 9.4.1     | Group equivariance and equivariance for functions defined on sets                                     | 128        |
| 9.4.2     | Equivariance properties of self-attention . . . . .   | 129        |
| 9.4.3     | Where exactly is equivariance imposed in self-attention? . . . . .                                    | 130        |
| 9.5       | Group Equivariant Stand-Alone Self-Attention . . . . .  | 131        |
| 9.5.1     | Group self-attention is an steerable operation . . . . .  | 132        |
| 9.5.2     | Lifting and group self-attention . . . . .  | 132        |
| 9.5.3     | Group self-attention is a generalization of the group convolution .                                   | 134        |
| 9.6       | Experiments . . . . .   | 136        |
| 9.7       | Discussion and Future Work . . . . .  | 138        |
| <b>10</b> | <b>Scale-Translation Equivariant Learning From Raw Time-Series</b>                                    | <b>139</b> |
| 10.1      | Introduction . . . . .  | 139        |
| 10.2      | Related Work . . . . .  | 140        |
| 10.3      | Background . . . . .  | 142        |
| 10.3.1    | Group equivariance, group invariance and symmetry preservation  | 142        |
| 10.3.2    | Symmetry-preserving mappings: Group and lifting convolutions .  | 142        |
| 10.4      | The problem of learning 2D convolutional kernels on the time-frequency plane . . . . .                | 143        |
| 10.4.1    | Fundamental differences between visual representations and spectro-temporal representations . . . . . | 144        |
| 10.4.2    | The problem of learning 2D kernels on short-time Fourier spectro-temporal representations . . . . .   | 145        |
| 10.4.3    | The problem of learning 2D kernels on Wavelet spectro-temporal representations . . . . .              | 146        |
| 10.5      | Wavelet networks: Scale-translation equivariant learning from raw waveforms . . . . .                 | 147        |
| 10.5.1    | Scale-translation preserving mappings: group convolutions on the scale-translation group . . . . .    | 147        |
| 10.5.2    | Wavelet Networks: architecture and practical implementation . . .                                     | 150        |
| 10.5.3    | Wavelet networks perform nested non-linear time-frequency transforms . . . . .                        | 154        |
| 10.6      | Experiments . . . . .   | 156        |
| 10.6.1    | Classification of environmental sounds . . . . .  | 156        |
| 10.6.2    | Automatic music tagging . . . . .   | 157        |
| 10.6.3    | Bearing fault detection . . . . .   | 158        |
| 10.6.4    | Discussion . . . . .  | 159        |
| 10.7      | Limitations and future work . . . . .   | 160        |
| 10.8      | Conclusion . . . . .  | 160        |
| <b>11</b> | <b>Learning Equivariances and Partial Equivariances from Data</b>                                     | <b>163</b> |
| 11.1      | Introduction . . . . .  | 163        |
| 11.2      | Background . . . . .  | 165        |
| 11.3      | Partial Group Equivariant Networks . . . . .  | 166        |

|                     |   |            |
|---------------------|---|------------|
| 11.3.1              | (Approximate) partial group equivariance . . . . .                                  | 166        |
| 11.3.2              | The partial group convolution . . . . .   | 167        |
| 11.3.3              | From group convolutions to partial group convolutions . . . . .                     | 167        |
| 11.3.4              | Learning group subsets through probability distributions on the group . . . . .     | 169        |
| 11.3.5              | Partial Group Equivariant Networks . . . . .  | 171        |
| 11.4                | Related work . . . . .  | 172        |
| 11.5                | Experiments . . . . .   | 173        |
| 11.5.1              | Experiments with deeper networks . . . . .  | 175        |
| 11.6                | Discussion . . . . .  | 176        |
| 11.7                | Limitations and future work . . . . .   | 178        |
| <b>12</b>           | <b>Conclusion and Future Work</b>   | <b>179</b> |
| 12.1                | Limitations and additional advances . . . . .                                       | 182        |
| 12.1.1              | Computational implications of global convolutions . . . . .                         | 182        |
| 12.1.2              | Input-dependence in long convolutional models . . . . .                             | 183        |
| 12.1.3              | Autoregressive inference with long convolutional models . . . . .                   | 183        |
| 12.1.4              | Computational implications of group convolutions and large groups                   | 184        |
| 12.1.5              | Symmetry pre-specification in group CNNs . . . . .                                  | 185        |
| 12.2                | Future Work . . . . .   | 186        |
| 12.2.1              | Exploring other inductive biases for efficiency and beyond . . . . .                | 186        |
| 12.2.2              | Future work on continuous modeling and symmetry preservation .                      | 187        |
| <b>Bibliography</b> |   | <b>190</b> |
| <b>Appendix A</b>   | <b>Continuous Kernel Convolutions for Sequential Data</b>                           | <b>233</b> |
| A.1                 | Properties of CKConvs . . . . .   | 233        |
| A.1.1               | Very Irregularly Sampled Data . . . . .   | 233        |
| A.1.2               | Data Sampled at Different Sampling Rates . . . . .                                  | 234        |
| A.1.3               | Linear Recurrent Units Are CKConvs . . . . .  | 235        |
| A.2                 | An Spline Interpretation Of ReLU and Sine Networks . . . . .                        | 236        |
| A.2.1               | Kernel Parameterization via ReLU Networks . . . . .                                 | 236        |
| A.2.2               | Kernel Parameterization via Sine Networks . . . . .                                 | 238        |
| A.3                 | Dataset Description . . . . .   | 239        |
| A.4                 | Ablation Studies . . . . .  | 240        |
| A.4.1               | Using Sine Non-Linearities Over Popular Alternatives . . . . .                      | 241        |
| A.4.2               | Going Deeper with CKCNNs . . . . .  | 241        |
| A.5                 | Experimental Details . . . . .  | 243        |
| A.5.1               | General Remarks . . . . .   | 243        |
| A.5.2               | Accounting for Spatial Displacements of the Sampled Convolutional Kernels . . . . . | 244        |
| A.5.3               | Dealing with High-Frequency Components . . . . .                                    | 245        |
| A.5.4               | Hyperparameters and Experimental Details . . . . .                                  | 246        |

|  |            |
|--|------------|
| <b>Appendix B Modelling Long Context in ND: From Task-Specific to a General Purpose CNN</b>    | <b>249</b> |
| B.1 Extended Related Work . . . . .  | 249        |
| B.2 Dataset description . . . . .  | 250        |
| B.3 Experimental details . . . . .   | 252        |
| B.3.1 General remarks . . . . .  | 252        |
| B.3.2 Hyperparameters and training details . . . . .   | 253        |
| B.4 Additional Experiments and Details . . . . .   | 254        |
| B.4.1 Computational Efficiency . . . . .   | 254        |
| B.4.2 Ablation: Performance of FlexCNN, S4 and CCNN Residual Blocks                            | 256        |
| <b>Appendix C Learnable Gridification for Efficient Point-Cloud Processing</b>                 | <b>259</b> |
| C.1 Network structure and convolution blocks . . . . .   | 259        |
| C.2 Hyperparameters . . . . .  | 259        |
| <b>Appendix D Continuous Kernel Convolutions with Differentiable Kernel Sizes</b>              | <b>261</b> |
| D.1 Alias-free FlexConv regularization . . . . .   | 261        |
| D.1.1 Analyzing the frequency spectrum of FlexConv . . . . .                                   | 263        |
| D.1.2 Regularizing the frequency response of FlexConv . . . . .                                | 266        |
| D.2 Dataset Description . . . . .  | 267        |
| D.2.1 Image Fitting Datasets . . . . .   | 267        |
| D.2.2 Sequential Datasets . . . . .  | 268        |
| D.2.3 Image Benchmark Datasets . . . . .   | 269        |
| D.3 Additional Experiments . . . . .   | 269        |
| D.3.1 Image Classification . . . . .   | 269        |
| D.4 Experimental Details . . . . .   | 271        |
| D.4.1 FlexNet . . . . .  | 271        |
| D.4.2 Optimization . . . . .   | 272        |
| D.4.3 Rotated Gaussian masks . . . . .   | 273        |
| <b>Appendix E Learning Convolutional Neural Architectures by Backpropagation</b>               | <b>275</b> |
| E.1 Extended Related Work . . . . .  | 275        |
| E.1.1 Positioning of DNArc on the NAS literature Taxonomy . . . . .                            | 275        |
| E.1.2 DNArc as a differentiable supernet method:<br>DARTS, follow-up works and DNArc . . . . . | 277        |
| E.1.3 Constrained and Multi-Objective NAS . . . . .  | 281        |
| E.2 Architectures Found by DNArc . . . . .   | 282        |
| E.3 Learning downsampling in the spatial Domain . . . . .                                      | 282        |
| E.4 Computational complexity of masked network components . . . . .                            | 286        |
| E.5 Dataset descriptions . . . . .   | 286        |
| E.5.1 The Long Range Arena benchmark . . . . .   | 286        |
| E.5.2 Image classification datasets . . . . .  | 288        |
| E.5.3 NAS-Bench-360 . . . . .  | 288        |
| E.6 Experimental details . . . . .   | 288        |

|  |   |     |
|--|---|-----|
| E.6.1  | General remarks . . . . .   | 288 |
| E.6.2  | Hyperparameters and training configurations . . . . .   | 291 |
| <b>Appendix F Focusing Equivariance on Transformations Co-Ocurring in Data</b> | <b>293</b>  |     |
| F.1  | Obtaining Co-Occurrent Attention via Equation 7.5 . . . . .   | 293 |
| <b>Appendix G Attentive Group Equivariant Convolutional Neural Networks</b>    | <b>295</b>  |     |
| G.1  | Generalized Visual Self-Attention . . . . .   | 295 |
| G.1.1  | Self-attention: From Vectors to Feature Maps . . . . .  | 295 |
| G.1.2  | Equivariant Linear Maps are Group Convolutions . . . . .  | 296 |
| G.1.3  | Proof of Theorem 1 . . . . .  | 297 |
| G.1.4  | Equivariance Proof of the Proposed Visual Attention . . . . .   | 298 |
| G.2  | Extended Implementation Details . . . . .   | 299 |
| G.2.1  | General Observations . . . . .  | 300 |
| G.2.2  | rot-MNIST . . . . .   | 300 |
| G.2.3  | CIFAR-10 . . . . .  | 300 |
| G.2.4  | PCam . . . . .  | 300 |
| G.3  | Effects of Stride and Input Size on Equivariance . . . . .  | 300 |
| <b>Appendix H Group Equivariant Stand-Alone Self-Attention</b>                 | <b>305</b>  |     |
| H.1  | Convolution and self-attention: A graphical comparison . . . . .  | 305 |
| H.2  | Concepts From Group Theory . . . . .  | 305 |
| H.3  | Actions and Representations of Groups Acting on Homogeneous Spaces<br>for Functions Defined on Sets . . . . . | 306 |
| H.4  | The Case of Non-Unimodular Groups: Self-Attention on the Scale-Translation<br>Group . . . . .                 | 307 |
| H.4.1  | Current empirical aspects of scale equivariant self-attention . . . . .                                       | 308 |
| H.5  | Experimental Details . . . . .  | 309 |
| H.5.1  | Rotated MNIST . . . . .   | 309 |
| H.5.2  | CIFAR-10 . . . . .  | 310 |
| H.5.3  | PatchCamelyon . . . . .   | 310 |
| H.6  | Proofs . . . . .  | 310 |
| <b>Appendix I Scale-Translation Equivariant Learning From Raw Time-Series</b>  | <b>317</b>  |     |
| I.1  | Group and group action . . . . .  | 317 |
| I.2  | Equivariance properties of time-frequency transforms . . . . .  | 318 |
| I.2.1  | The Fourier transform . . . . .   | 318 |
| I.2.2  | The short-time Fourier transform . . . . .  | 320 |
| I.2.3  | The Wavelet Transform . . . . .   | 322 |
| I.3  | Experimental details . . . . .  | 324 |
| I.3.1  | UrbanSound8K . . . . .  | 324 |
| I.3.2  | MagnaTagATune . . . . .   | 326 |
| <b>Appendix J Learning Equivariances and Partial Equivariances from Data</b>   | <b>329</b>  |     |
| J.1  | Groups, subgroups, group actions and other group theoretical concepts . . . . .                               | 329 |

---

|                              |  |            |
|------------------------------|--|------------|
| J.2                          | (Approximate) equivariance in partial group convolutions . . . . .                                     | 330        |
| J.2.1                        | Partial group convolutions from the group $\mathcal{G}$ to a subset $\mathcal{S}$ . . . . .            | 330        |
| J.2.2                        | Partial group convolutions from a subset $\mathcal{S}^{(1)}$ to a subset $\mathcal{S}^{(2)}$ . . . . . | 331        |
| J.3                          | Equivariance property of Monte-Carlo approximations . . . . .  | 332        |
| J.4                          | Experimental details . . . . .   | 333        |
| J.4.1                        | Dataset description . . . . .  | 333        |
| J.4.2                        | General remarks . . . . .  | 333        |
| J.4.3                        | Hyperparameters and training details . . . . .   | 334        |
| J.5                          | Additional Experiments . . . . .   | 334        |
| <b>SIKS Dissertatiereeks</b> |  | <b>337</b> |

# 1

## Introduction

Before the advent of Deep Learning, the landscape of machine learning was characterized by traditional algorithms that relied heavily on handcrafted features. These algorithms lacked the capacity to automatically extract meaningful patterns from raw information, and often struggled to unravel the intricate complexities of real-world data.

The emergence of Deep Learning marked a profound shift in the paradigm of machine learning. It bestowed machines with the ability to automatically extract patterns, features and abstract concepts directly from raw data. This paradigm shift empowered researchers with an unprecedented level of abstraction for the design of machine learning algorithms, allowing them to conceptualize machine learning models in terms of the functional families these models could and should represent.

An interesting observation is that even though neural networks –the foundation of Deep Learning–, have existed for over five decades [259, 324], it is only in recent years that we have started to witness the true potential of these methods. The turning point came with the ImageNet Large Scale Visual Recognition Challenge in 2012 when AlexNet [196] took center stage. Arguably, the recent success of Deep Learning can be largely attributed to the synergistic advancement of three critical pillars, two of which are (*i*) the access to vast amounts of data, and (*ii*) the exponential growth in compute power.<sup>1</sup> These two pillars underscore an important property of Deep Learning methods, which still holds today: *Deep Learning thrives in rich environments, both in terms of compute and data.*

---

<sup>1</sup>The third pillar is crucial for the motivation and relevance of this dissertation, and will be described in detail in the following subsection.

Nevertheless, these ideal conditions are far from universally met. For example, in the medical domain. Gathering expensive expert-annotated datasets for computer-aided diagnosis poses several challenges due to factors such as the cost associated with data collection and labeling, privacy concerns surrounding sensitive medical information, and variations in protocols across different institution and devices [15, 250]. Furthermore, the deployment and training of neural networks on devices with limited resources introduces another layer of constraints. For instance, for the democratization of these methods to ensure that communities worldwide have equitable access to them. In several parts of the world, computational power remains scarce, making it challenging to use resource-intensive models [175, 399]. Moreover, several promising applications such as autonomous driving and Internet of Things (IoT), demand for neural networks to have the capability to operate on light, portable hardware.

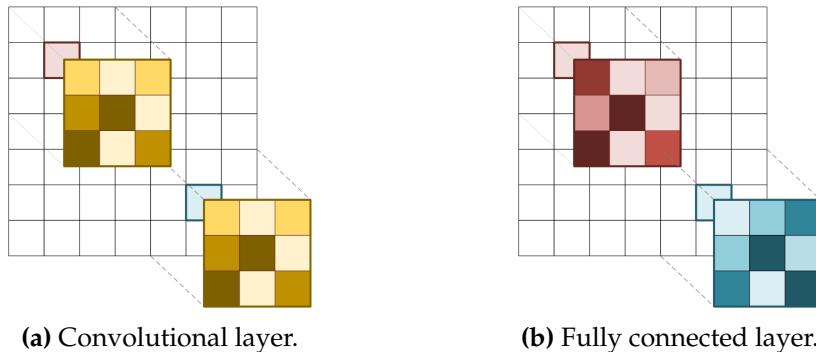
At a global scale, the increasing size and widespread adoption of deep neural networks present compelling reasons to address efficiency from both environmental and financial considerations. From an environmental perspective, research has revealed that the training of a –by now small– model with ~350 million parameters produces carbon emissions comparable to the lifetime emissions of five cars [369]. From a financial standpoint, the costs of training such models currently runs over the millions of dollars [192]. As the interest in deploying large neural networks across diverse modalities and applications increases, these observations raise questions about the environmental and financial sustainability of Deep Learning approaches.

These examples highlight the importance of efficiency in the neural models that we use. The development of more efficient neural architectures holds not only the potential to unlock a myriad of real-world applications where Deep Learning can have a substantial impact, but also to enhance its globally accessibility, reduce its environmental impact, and ensure its economic viability. In doing so, we can fully leverage the power of Deep Learning to drive positive change, both in science and society.

## 1.1 Inductive Biases and Deep Learning

In addition to the two pillars responsible for the success of Deep Learning mentioned earlier, there is a third equally important pillar: *the design of better neural architectures*.

Despite the substantial growth in data and computing power, the neural architectures that sparked the recent Deep Learning revolution differ from the original designs proposed in the early 1950s. Initial models were primarily composed of fully connected layers interleaved with point-wise nonlinearities. The turning point for the recent Deep Learning revolution came with AlexNet [196]: a prominent example of a *Convolutional Neural Network* (CNN) [112, 206]. The main defining characteristic of CNNs is the departure from the use of fully connected layers in favor of *convolutions*: linear operations that respect the translation symmetry of visual data (Fig. 1.1a).



**Figure 1.1:** Convolutional vs fully connected layers. Convolutional layers share the same weights across all positions. Hence, the same mapping is applied (and learned) at all positions –here depicted by weights of the same color in 1.1a. In contrast, fully connected layers apply (and learn) a *unique* set of weights to each position (1.1b). This increases the number of weights needed by  $O(N^2)$  for an  $N \times N$  image, and hampers generalization across positions. Since all features must be independently learned at each position, fully connected layers need  $O(N^2)$  more data samples than convolutions.

Convolutional Neural Networks (CNNs) are a prime example of the use and effectiveness of considering *inductive bias* in the design of neural architectures. Formally, an *inductive bias* represents a set of assumptions, constraints, or prior knowledge considered in the design and training of a machine learning model such that its decision making process favors solutions that align better with the problem at hand. In the case of Convolutional Neural Networks (CNNs), the key inductive bias is *translation preservation*. This inductive bias arises from the use of convolutions, which make the learned feature representations be shared across positions –instead of the independent feature representations per position learned by fully connected layers (Fig. 1.1b). As a result, CNNs learn mappings that inherently respect translation, resulting in vastly improved generalization and data efficiency.

This example underscores the potential that inductive biases may hold for the improvement of Deep Learning algorithms. In this dissertation, we focus on the role that inductive biases can play in improving various facets of Deep Learning efficiency.

## 1.2 Efficiency through Inductive Biases

When thinking about efficiency, probably the first concept that comes to mind is that of *sparsity*. In the context of Deep Learning, sparsity can be seen as an inductive bias that favors small neural architectures over bigger ones, either in terms of the whole architecture [149, 425] or the subset of neurons that are activated for a particular input [116, 342, 380]. The concept of sparsity in Deep Learning has been broadly studied in the literature, and whole dissertations have been written in the topic [109, 122, 212]. How-

ever, sparsity generally touches exclusively upon two aspects of efficiency: *compute* and *parameter efficiency*. In this dissertation we are interested in (complementary) inductive biases that touch upon Deep Learning efficiency in a broader sense.

### 1.2.1 Efficiency Aspects of Deep Learning

Before discussing specific inductive biases, it is beneficial to define the various axes on which Deep Learning efficiency can be measured. We categorize efficiency into the following axes:

- **Compute efficiency:** This refers to the computational resources, e.g., memory, execution time, needed to train and deploy deep neural networks. A compute-efficient model requires fewer computational resources to make predictions.
- **Data efficiency:** Data efficiency pertains to the amount of data required for a network to generalize effectively. Models that achieve high performance with less data are more data-efficient.
- **Parameter efficiency:** Parameter efficiency refers to the size –in terms of the number of parameters used– of the neural network needed to reach a certain accuracy level. Models that require fewer parameters to reach a comparable validation metric are more parameter-efficient.
- **Design efficiency:** Design efficiency refers to the resources in terms of compute, human hours, memory, experimentation, etc., needed to *design* a high-performing architecture for new datasets, tasks or modalities. Architectures that require fewer resources to be adapted to new settings are more design-efficient.
- **Environmental efficiency:** Environmental efficiency relates to the ecological footprint of designing, training and deploying neural networks. Environmentally-efficient architectures incur in lower environmental costs for their design, training and deployment. As Deep Learning models continue to grow in size and complexity, and become increasingly present in everyday tools, the importance of environmental efficiency becomes increasingly critical.
- **Financial efficiency:** Financial efficiency refers to the cost-effectiveness of designing, training, deploying and maintaining neural networks. Factors considered here include hardware investment, operational costs, and maintenance. Architectures that required lower overall inversions are more financially efficient.

As can be seen from this classification, Deep Learning efficiency is a broad multifaceted concept that comprise several different aspects, which go beyond the usual characterization in terms of parameter and computational efficiency.

### 1.2.2 Continuous modeling and symmetry preservation

This dissertation aims to investigate inductive biases that broadly improve the efficiency of Deep Learning algorithms. We focus on two less explored inductive biases which



**Figure 1.2:** Kernels learned by AlexNet [196]. Despite their discrete parameterization, the convolutional kernels learn discrete approximations of continuous functions. Representing the kernels in a continuous domain facilitates the learning of such kernels.

have significant impact on various efficiency facets and the learning, generalization and applicability of Deep Learning algorithms: *continuous modeling* and *symmetry preservation*. This section introduces and showcases the significance of these concepts.

### Continuous modeling

Continuous modeling refers to the idea of modeling neural operations directly in a continuous space. This fundamental inductive bias offers significant advantages:

- **Natural data representation.** Many real-world phenomena are continuous in nature. For example, images, audio signals, sensor readings, and time series are all inherently continuous. Nevertheless, well-established neural architectures such as CNNs approach the modeling of these signals in an inherently discrete manner. A key example lies in the representation of convolutional kernels within CNNs, typically structured as sets of discrete independent weights [143, 161, 196]. For example, a  $3 \times 3$  convolutional kernel is conventionally represented by a collection of 9 independent weights. However, looking at the convolutional kernels that these networks learn, one finds that these kernels in fact represent continuous functions, such as color-gradient or edge detectors (Fig. 1.2). This inherent continuity in the learned kernels suggests an opportunity for a more natural data representation by modeling these kernels as continuous functions.

By harnessing continuous modeling, we can describe learnable mappings *directly* in a continuous space. This allows for the construction of neural architectures that inherently align with the nature of the data, and might be able to capture more nuanced patterns and relationships in the data that may be overlooked or inadequately represented by discrete approximations.

- **Efficient long context.** A crucial challenge in machine learning is modeling long-term input dependencies, especially when dealing with large inputs [392, 393]. This challenge is particularly hard for neural architectures that parameterize their

learnable mappings in a discrete manner, e.g., CNNs, as the parameters needed to consider a certain context window grows linearly with its size. As a result, considering large contexts is impractical for these methods and structural modifications are needed to synthetically increase their context window [454].

A powerful alternative to address this challenge comes from the use of self-attention [403]: a versatile operation able to capture dependencies across long contexts with a fixed number of parameters, and the basis of the popular Transformer architecture. Unfortunately, self-attention induces a quadratic cost both in terms of memory and time relative to the size of the context window, which makes it very expensive for long context applications.

Modeling neural operations directly in a continuous space makes it possible to consider long contexts under a fixed parameter budget and with better scaling properties than self-attention ( $N \log N$  vs.  $N^2$ ) (Ch. 2, 3). Extensions of such parameterizations have been shown to perform on par with Transformers in language modeling [292] with superior scaling properties that enable them to scale to contexts involving millions of tokens [273].

- **Resolution-agnostic architectures.** A recurrent problem in Deep Learning is the need to redesign neural architectures to accommodate inputs of different resolution [142, 209]. Common neural architectures are tied to a particular resolution, e.g., He et al. [143], Krizhevsky et al. [196]. As a consequence, using a neural architecture designed for the processing of  $32 \times 32$  images to process images of other size, e.g.,  $256 \times 256$ , would lead to poor results. This effect is observed even if both images are re-scaled replicas of the exact same image. As discussed in Ch. 3, this problem results from modeling neural operations in a discrete manner. Doing so ties them to a particular resolution, making their output drastically different to inputs of different resolutions, e.g., for a  $3 \times 3$  convolution.

Continuous modeling allows for the scaling of feature maps and receptive fields in a way that maintains consistency across different resolutions. This capability enables the creation of neural networks that generalize across resolutions, even if they have not been seen during training. This eliminates the need for separate models for each resolution [143, 323] or extensive data augmentation techniques to simulate different resolutions [118, 440], thereby simplifying the training process and reducing computational and design costs. Consequently, this approach paves the way for more versatile, efficient, and effective models better suited for the diverse nature of real-world data.

- **Natural handling of irregularly sampled data.** Many real-world scenarios involve data that is not collected uniformly at regular intervals. Prominent examples are (i) healthcare time-series data [174, 307], where metrics such as heart rate or blood pressure can be recorded at varying times, and (ii) sparsely scanned volumetric data [73, 438], which generate point clouds that do not conform to a uniform grid.

Due to their discrete nature, traditional neural architectures, e.g., (discrete) CNNs, are unable to handle irregular data. This stems from the fact that they parameterize kernels in a regular grid, and cannot be sampled at arbitrary positions [355].

Through the use of continuous modeling, it is possible to define neural components that operate in a continuous space [343, 436]. Since the resulting models are defined in a continuous space, they can be easily queried at arbitrary positions. This property gives them the ability to manage irregular data in a native manner, thereby broadening the scope of applications where Deep Learning can be used.

- **Neural Architecture Search on a continuous space.** Traditionally, neural architectures are regarded as discrete entities, characterized by a discrete number of layers, channels, kernel sizes, patches, etc. [187, 206, 403]. This perspective makes Neural Architecture Search (NAS) –the automated process of identifying high-performing neural architectures– very computationally expensive and time-consuming due to the combinatorial nature of the optimization problem involved [289, 482].

Through the use of continuous modeling, it is possible to frame neural architectures as entities in a continuous space (Ch. 5, 6). This turn can be used to dramatically reduce the complexity of designing neural architectures. This continuous perspective enables the use of gradient-optimization methods for Neural Architecture Search [221, 231], which remarkably increases the speed of the search process, and enables the exploration of a much broader set of candidate architectures.

In summary, continuous modeling embodies neural networks with the inherent properties of continuous data, providing several advantages over discrete approaches. These benefits collectively enhance both the efficacy and versatility of deep learning methods across a very diverse set of applications.

### Symmetry preservation

Symmetry preservation refers to the idea of constructing neural operations that consider the inherent symmetries of data. This fundamental inductive bias offers significant advantages for the learning, generalization and inference dynamics of neural networks:

- **Improved learning and generalization.** Leveraging the inherent symmetries in data as prior knowledge for the design of neural architectures that respect them, leads to a substantial reduction in amount of data required to train them [65, 206]. That is, to neural architectures with enhanced data efficiency. Unlike unconstrained models, which must learn these symmetrical patterns from the data itself, architectures design to respect these symmetries inherently incorporate them.

Moreover, models that respect data symmetries inherently generalize better than those without such constraints. This is because symmetry-preserving models can recognize symmetric modifications of learned patterns, even if particular symmetric modifications of these patterns have not been seen during training

[362, 422]. For example, a CNN is able to recognize patterns even if these have been translated to unseen positions through the use of convolutions (Fig. 1.1a). As symmetries are preserved by construction irrespective of the input, this approach is more effective than data augmentation, particularly when these symmetries are well understood, e.g., translation, rotation, scaling [415].

- **Reliable predictions under input symmetries.** A key advantage of symmetry-preserving models is their ability to provide consistent predictions under symmetrical transformations of the input, both during training and inference [65, 424]. These guarantees carry immense significance, especially for tasks where maintaining prediction stability under symmetric transformations is essential.

A core example is medical imaging. In tasks like identifying cancerous cells in tissue slice images [404], it is evident that geometric transformations such as rotations, translations, and scaling should *not* alter the predictions of the network. However, networks that do not consider symmetry preservation can yield unexpected inconsistent and potentially hazardous variations in their prediction, which is a major barrier to the use of neural networks in critical applications [154, 390].

By ensuring prediction consistency under symmetrical conditions, symmetry preserving models bolster their utility and reliability in real-world scenarios, addressing a primary limitation of unconstrained neural networks.

- **Powerful neural networks with fewer parameters.** A closer look at the parameterization of symmetry-preserving networks reveals that symmetry preservation is effectively encoded in the weights of the neural network by means of weight sharing [87, 304]. Symmetry preservation implicitly ties together several weights that would otherwise be independently parameterized in unconstrained networks.

A great example of this can be seen in CNNs, when compared to fully-connected networks (Fig. 1.1). In CNNs, weights are tied to be identical across all spatial positions, unlike in fully connected networks where weights at each position can have completely different values. Symmetry preservation effectively leads to networks with fewer parameters, thereby making these models more parameter efficient.

In summary, symmetry preservation embodies neural networks with a better understanding of the data being processed. This ability grants them improved data and parameter efficiency, as well as enhanced explainability, robustness and generalization. These advancements render symmetry preserving networks highly valuable for a wide range of practical and critical applications.

### 1.3 Research Objective

The primary objective of this dissertation is the study of continuous modeling and symmetry preservation to improve the efficiency of Deep Learning techniques. Based on this objective, we define the following two research questions:

- **RQ 1.** Can continuous modeling be used to improve Deep Learning efficiency? If so, which specific efficiency aspects as per Sec. 1.2.1 does it improve?
- **RQ 2.** Can symmetry preservation be used to improve Deep Learning efficiency? If so, which specific efficiency aspects as per Sec. 1.2.1 does it improve?

The body of this dissertation is divided in two parts devoted to the study of each of the inductive biases under consideration. The following section provides detailed insights into the dissertation structure and the contributions of each chapter. The efficiency contributions of the techniques proposed in each chapter are summarized in Tab. 1.1.

## 1.4 Dissertation Structure and Contributions

This dissertation is structured in two main parts, devoted to each of the inductive biases under study. In accordance with the guidelines set forth by the Doctorate Regulations of the Vrije Universiteit Amsterdam, this section lists the papers on which each chapter is based, along with a brief summary and an account of my contributions to each paper. Table 1.1 summarizes the efficiency contributions of each chapter.

**Table 1.1:** Efficiency contributions as per Sec. 1.2.1 of each chapter.<sup>a</sup>

|                                      | Compute<br>Efficiency | Data<br>Efficiency | Parameter<br>Efficiency | Design<br>Efficiency |
|--------------------------------------|-----------------------|--------------------|-------------------------|----------------------|
| PART I:<br>CONTINUOUS<br>MODELLING   | Chapter 2             | ✓                  | -                       | ✓                    |
|                                      | Chapter 3             | ✓                  | -                       | ✓                    |
|                                      | Chapter 4             | ✓                  | -                       | -                    |
|                                      | Chapter 5             | ✓                  | -                       | ✓                    |
|                                      | Chapter 6             | ✓                  | -                       | ✓                    |
| PART II:<br>SYMMETRY<br>PRESERVATION | Chapter 7             | -                  | ✓                       | -                    |
|                                      | Chapter 8             | -                  | ✓                       | -                    |
|                                      | Chapter 9             | -                  | ✓                       | -                    |
|                                      | Chapter 10            | -                  | ✓                       | -                    |
|                                      | Chapter 11            | ✓                  | ✓                       | ✓                    |

<sup>a</sup>Note that Tab. 1.1 does not depict the environmental and financial efficiency aspects defined in Sec. 1.2.1. From a technical standpoint, enhancements in environmental and financial efficiency are implicitly driven by advancements across the remaining efficiency axes. Consequently, we do not explicitly add them here. However, we consider important to acknowledge the importance of these efficiency aspects, even more as Deep Learning continues to permeate everyday tools and applications.

### PART I: EFFICIENCY THROUGH CONTINUOUS MODELING

**Chapter 2. Continuous Kernel Convolutions for Sequential Data.** This chapter is based on the paper:

David W. Romero, Anna Kuzina, Erik J Bekkers, Jakub M Tomczak and Mark

Hoogendoorn. CKConv: Continuous kernel convolution for sequential data. In *International Conference on Learning Representations*, 2022.

*Summary:* This chapter introduces a continuous parameterization of convolutional kernels through the use of MLPs that map coordinates to the value of convolutional kernels at those coordinates. This approach facilitates: (i) the creation of large, parameter-efficient convolutional kernels able to capture global context at a much lower cost than Transformers; (ii) the construction of resolution agnostic neural architectures; and (iii) an effective way to process irregularly-sampled data.

*Personal contributions:* Original idea to use MLPs to model convolutional kernels, theory, implementation, ablations, experiments and writing.

### Chapter 3. Modeling Long Context in ND: From Task-Specific to a General-Purpose CNN.

This chapter is based on the papers:

David W. Romero\*, David M. Knigge\*, Albert Gu, Erik J. Bekkers, Efstratios Gavves, Jakub M. Tomczak and Mark Hoogendoorn. Towards a General Purpose CNN for Long Range Dependencies in ND. In *ICML Workshop on Continuous Time Methods for Machine Learning*, 2022.

David M. Knigge\*, David W. Romero\*, Albert Gu, Efstratios Gavves, Erik J. Bekkers, Jakub M. Tomczak, Mark Hoogendoorn and Jan-Jakob Sonke. modeling long range dependencies in ND: From task-specific to a general purpose CNN. In *International Conference on Learning Representations*, 2023.

*Summary:* This chapter presents a convolutional framework able to handle data of varying dimensionality, lengths and resolutions without architectural changes. Its core insight is that this can be achieved through the use of layers able to model global context regardless of these factors in a resolution agnostic way. The resulting architecture shows high versatility, delivering good results across diverse modalities and datasets without requiring customization for each use-case.

*Personal contribution:* This work was carried out in equal contribution with David M. Knigge. We co-developed the ideas and insights presented in these papers. We both contributed to theory, implementation, ablations, experiments and writing.

### Chapter 4. Learned Gridification for Efficient Point-Cloud Processing.

This chapter is based on the paper:

Putri A. van der Linden\*, David W. Romero\* and Erik J. Bekkers. Learned Gridification for Efficient Point Cloud Processing. In *Proceedings of the 2<sup>nd</sup> Annual Workshop on Topology, Algebra, and Geometry in Machine Learning (ICML)*, 2023.

*Summary:* This chapter introduces a method to reduce the compute and memory complexity of point-cloud processing significantly. It involves converting point-clouds into regular grid representations through a learnable “gridification” step, which is then efficiency processed with grid-based operations, e.g., Conv3D. After

processing the grid representation, it can be converted back to point-cloud form for tasks such as segmentation. Gridified networks match and even surpass state of the art methods while being remarkably faster and more scalable.

*Personal contribution:* This work was carried out in equal contribution with Putri A. van der Linden. We co-developed the idea and formulation of learnable gridification, contributed to theory, implementation, ablations, experiments and writing.

**Chapter 5. Continuous Kernel Convolutions with Differentiable Kernel Sizes.** This chapter is based on the paper:

David W. Romero\*, Robert-Jan Bruintjes\*, Jakub M. Tomczak, Erik J. Bekkers, Mark Hoogendoorn and Jan C. van Gemert. Flexconv: Continuous kernel convolutions with differentiable kernel sizes. In *International Conference on Learning Representations*, 2022.

*Summary:* This chapter improves the parameterization of Continuous Kernel Convolutions to make their kernel sizes learnable by backpropagation. This allows CNNs to autonomously find the optimal kernel sizes that leads to the best accuracy, therefore releasing users from the need to pre-specify kernel sizes, and the need to rely on simple network configurations, e.g. with  $3 \times 3$  kernels at all layers.

*Personal contributions:* This work was carried out in equal contribution with Robert-Jan Bruintjes. We co-developed the idea of parameterizing convolutional kernels as the combination of an MLP with a learnable Gaussian, and of regularizing against aliasing through the use of Multiplicative Filter Networks [104]. We both contributed to theory, implementation, ablations, experiments and writing.

**Chapter 6. Learning Convolutional Neural Architectures by Backpropagation.** This chapter is based on the paper:

David W. Romero and Neil Zeghidour. DNArch: Learning convolutional neural architectures by backpropagation. In *Differentiable Almost Everything: Differentiable Relaxations, Algorithms, Operators, and Simulators Workshop (ICML)*, 2023.<sup>2</sup>

*Summary:* This chapter introduces an scalable method able to learn several architectural components of CNNs by propagation: (i) the size of all convolutional kernels, (ii) the width of all layers in the network, (iii) the position and value of all downsampling layers, and (iv) the depth of the network. This is achieved in an inexpensive way by viewing neural architectures as multidimensional continuous entities and using learnable differentiable masks to learn their sizes. Using a general-purpose CNN as backbone, e.g., a CCNN (Sec. 3), our method automatically finds high-performant architectures across several tasks and modalities.

*Personal contribution:* Original idea to use differentiable masks to learn the whole architecture of CNNs, theory, implementation, ablations, experiments and writing.

---

<sup>2</sup>This paper was carried out during a Research Internship at Google Research.

---

PART II: EFFICIENCY THROUGH SYMMETRY PRESERVATION

**Chapter 7. Focusing Equivariance on Transformations Co-Ocurring in Data.** This chapter is based on the paper:

David W. Romero and Mark Hoogendoorn. Co-attentive equivariant neural networks: Focusing equivariance on transformations co-occurring in data. In *International Conference on Learning Representations*, 2020.

*Summary:* This chapter proposes the use of a lightweight block cyclic permutation equivariant self-attention module to explicitly accentuate the responses of group convolutions whose relative geometrical poses align with geometric patterns seen in data. Due to this explicit system for the accentuation of co-occurring symmetries, resulting models that capture geometrical dependencies better, therefore enhancing the generalization of group equivariant models.

*Personal contribution:* Original idea to use block cyclic equivariant self-attention to accentuate geometric patterns co-occurring in data, formulation, theory, implementation, ablations, experiments and writing.

**Chapter 8. Attentive Group Equivariant Convolutional Neural Networks.** This chapter is based on the paper:

David W. Romero, Erik J. Bekkers, Jakub M. Tomczak and Mark Hoogendoorn. Attentive group equivariant convolutional networks. In *International Conference on Machine Learning*, pp. 8188-8199. PMLR, 2020.

*Summary:* This chapter proposes the use of a generalized form of visual attention that is symmetry preserving. This mechanism allows for the accentuation of complex geometrical templates as well as the generation of attention maps that behave equivariantly, leading to improved generalization and interpretability.

*Personal contribution:* Original idea of symmetry-preserving visual attention, formulation, theory, implementation, ablations, experiments and writing.

**Chapter 9. Group Equivariant Stand-Alone Self-Attention.** This chapter is based on the paper:

David W. Romero and Jean-Baptiste Cordonnier. Group equivariant stand-alone self-attention for vision. In *International Conference on Learning Representations*, 2021.

*Summary:* Based on the success of Transformers, we provide a symmetry-preserving formulation of the self-attention operation. Since self-attention relies on positional encodings to handle geometrical information, our formulation is steerable, and thus can operate with groups whose action lives outside the data grid. Thanks to symmetry preservation, the resulting group equivariant Transformers exhibit improved data-efficiency and generalization than non-equivariant counterparts.

*Personal contribution:* Original idea of symmetry-preserving self-attention, formulation, theory, implementation, ablations, experiments and writing.

**Chapter 10. Scale-Translation Equivariant Learning From Raw Time-Series.** This chapter is based on the paper:

David W. Romero, Erik J. Bekkers, Jakub M. Tomczak and Mark Hoogendoorn. Wavelet networks: Scale-translation equivariant learning from raw time-series. In *Transactions on Machine Learning Research*, 2023.

*Summary:* This chapter provides a translation scale equivariant formulation of convolution operations for the processing of raw time-series. The resulting mappings share an interesting relationship with the well-known Wavelet transform, providing interesting insights about the modus operandi of the resulting architectures. The preservation of translation and scale symmetries leads to improved data efficiency and generalization over existing CNNs that work on raw data.

*Personal contribution:* Original idea of using scale translation equivariance for the processing of time-series, its connection to the Wavelet transform, formulation, theory, implementation, ablations, experiments and writing.

**Chapter 11. Learning Equivariances and Partial Equivariances from Data.** This chapter is based on the paper:

David W. Romero and Suhas Lohit. Learning equivariances and partial equivariances from data. In *Advances in Neural Information Processing Systems 35*, pp. 36466-36478, 2022.<sup>3</sup>

*Summary:* While symmetry preservation is highly effective when the relevant symmetries exist in data, it can become too constraining if these symmetries are misspecified or only partially present. This chapter presents a framework that allows for the automatic adjustment of symmetry preservation during training to match to the actual symmetries present in data. The resulting models benefit from the data efficiency offered by symmetry preservation, while maintaining the flexibility to adjust when it becomes overly restrictive. This leads to more adaptable, data-efficient models that effectively self-manage the level in which this inductive bias affects the learning process.

*Personal contribution:* Original idea and formulation of partially equivariant models, theory, implementation, ablations, experiments and writing.

## OTHER CONTRIBUTIONS

In addition to the papers explicitly considered in this dissertation, I was also involved in a number of projects for which I served in an advisory role. The papers associated with these projects are:

---

<sup>3</sup>This paper was carried out during a research internship at Mitsubishi Electric Research Laboratories.

- David M. Knigge, **David W. Romero** and Erik J. Bekkers. Exploiting redundancy: Separable group convolutional networks on lie groups. In *International Conference on Machine Learning* pp. 11359-11386. PMLR, 2022.
- Tycho van der Ouderaa, **David W. Romero** and Mark van der Wilk. Relaxing equivariance constraints with non-stationary continuous filters. In *Advances in Neural Information Processing Systems 35*, pp. 33818-33830, 2022.
- Stefano Massaroli, Michael Poli, Daniel Y. Fu, Hermann Kumbong, **David W. Romero**, Rom N. Parnichkun, Arman Timalsina, Quinn McIntyre, Beidi Chen, Atri Rudra, Ce Zhang, Christopher Ré, Stefano Ermon and Yoshua Bengio. Laughing hyena distillery: Extracting compact recurrences from convolutions. In *Advances in Neural Information Processing Systems*, 2023.
- Erik J. Bekkers, Sharvaree Vadgama, Rob Hesselink, Putri A. van der Linden and **David W. Romero**. Fast, Expressive SE( $n$ ) Equivariant Networks through Weight Sharing in Position-Orientation Space. In *International Conference on Learning Representations*, 2024.
- Alonso Urbano and **David W. Romero**. Self-Supervised Detection of Perfect and Partial Input-Dependent Symmetries. In *ICML Workshop on Geometry-grounded Representation Learning and Generative Modeling*, 2024.

We briefly discuss these papers in Chapter 12.1, making explicit their connection to efficiency aspects of deep learning and the remaining limitations of this work.

#### 1.4.1 How to read this dissertation

Each chapter in this dissertation is based on one or more papers, which are either published or currently under review. To maintain the self-contained nature of each chapter, we have chosen to keep them as consistent as possible with the original publications, only making modifications to align with the formatting of this dissertation. This approach enables each chapter to function as an independent unit, at the cost of needing possible repetitions of key concepts across some chapters. For readers who plan to go through the dissertation sequentially, we recommend skimming past concepts previously introduced in the Preliminaries and Background sections of earlier chapters.

Each chapter is accompanied by a corresponding Appendix situated after the References section. These appendixes serve to complement the main text by including, among other things, complementary theoretical definitions, proofs and detailed descriptions of the datasets used in the experimental section of each chapter. Readers are advised to skim past the concepts and datasets introduced in the Appendixes of previous chapters.

## Part I

# EFFICIENCY THROUGH CONTINUOUS MODELING



# 2

## Continuous Kernel Convolutions for Sequential Data

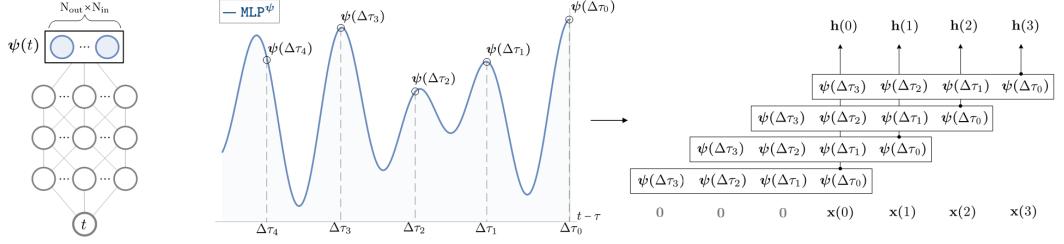
*Based on the paper:*

*CKConv: Continuous Kernel Convolution for Sequential Data [321]*

### 2.1 Introduction

Recurrent Neural Networks (RNNs) have long governed tasks with sequential data [63, 148, 326]. Their main ingredient are *recurrent units*: network components with a recurrence formulation which grants RNNs the ability to be unrolled for arbitrarily many steps and handle sequences of arbitrary size. In practice, however, the effective *memory horizon* of RNNs, i.e., the number of steps the network retains information from, has proven to be surprisingly small, most notably due to the *vanishing gradients problem* [22, 147]. Interestingly, it is the very recurrent nature of RNNs that allows them to be unrolled for arbitrarily many steps which is responsible for vanishing gradients [282]. This, in turn, hinders learning from the far past and induces a small effective memory horizon.

Convolutional Neural Networks (CNNs) [208] have proven a strong alternative to recurrent architectures as long as relevant input dependencies fall within their memory horizon [11, 70, 75, 80, 277]. CNNs avoid the training instability and vanishing / exploding gradients characteristic of RNNs by avoiding *back-propagation through time* [426] altogether. However, these architectures model convolutional kernels as a sequence of independent weights. As a result, their memory horizon must be defined *a-priori*, and larger memory horizons induce a proportional growth of the model size.



**Figure 2.1:** Continuous Kernel Convolution (CKConv). CKConv views a convolutional kernel as a vector-valued continuous function  $\psi : \mathbb{R} \rightarrow \mathbb{R}^{N_{\text{out}} \times N_{\text{in}}}$  parameterized by a small neural network  $\text{MLP}^\psi$ .  $\text{MLP}^\psi$  receives a time-step and outputs the value of the convolutional kernel at that position. We sample convolutional kernels by passing a set of relative positions  $\{\Delta\tau_i\}$  to  $\text{MLP}^\psi$ , and perform convolution with the sampled kernel next. Since  $\text{MLP}^\psi$  is a continuous function, CKConvs can (i) construct arbitrarily large kernels, (ii) generate kernels at different resolutions, and (iii) handle irregular data.

In this work, we provide a solution to these limitations. We propose to view a convolutional kernel as a continuous function parameterized by a small neural network instead of a sequence of independent weights. The resulting *Continuous Kernel Convolution* (CKConv) enjoys the following properties:

- CKConvs can define arbitrarily large memory horizons within a single operation. Consequently, *Continuous Kernel Convolutional Neural Networks* (CKCNNs) detach their memory horizon from (i) the depth of the network, (ii) the dilation factors used, and (iii) the size of the network.
- CKConvs do not rely on any form of recurrence. As a result, CKCNNs (i) can be trained in parallel, and (ii) do not suffer from vanishing / exploding gradients or small effective memory horizons.
- Continuous convolutional kernels can be evaluated at arbitrary positions. Consequently, CKConvs and CKCNNs can be readily used on irregularly sampled data, and data at different resolutions.

We observe that continuous kernel parameterizations previously used to handle irregular data *locally*, e.g., Schütt et al. [343], Wu et al. [436], are not adequate to model long-term dependencies. This is due to the inability of their kernels to model long spatial complex functions (Sec. 2.4.2). Contrarily, CKConvs perfectly describe long complex non-linear, non-smooth functions by parameterizing their kernels as SIRENs [358]: implicit neural representations with Sine nonlinearities. Shallow CKCNNs match or outperform state-of-the-art approaches on several tasks comprising stress tests, continuous, discrete and irregular data, as well as resolution changes. To the best of our knowledge, we are first to observe the potential of continuous convolutional kernels to model long-term dependencies, and to provide an useful parameterization to this end.

## 2.2 Related Work

**Continuous kernel formulation.** Continuous formulations for convolutional kernels were introduced to handle irregularly sampled 3D data *locally* [343, 355, 417, 436]. As discrete convolutions learn independent weights for specific relative positions, they cannot handle irregularly sampled data effectively. Following work focuses on point-cloud applications [111, 158, 352, 382]. Other approaches include Monte Carlo approximations of continuous operations [105]. Our work proposes a new broad flavor of applications for which continuous kernels are advantageous.

**Implicit neural representations.** Implicit neural representations construct continuous data representations by encoding the input in the weights of a neural network [261, 278, 358]. This leads to numerous advantages over conventional (discrete) data representations, e.g., memory efficiency, analytic differentiability, with interesting properties for several applications, e.g., generative modelling [94, 344].

Since we model convolutional kernels as continuous functions and parameterize them via neural networks, our approach can be understood as *implicitly representing the convolutional kernels of a conventional CNN*. Different is the fact that these convolutional kernels are not known a-priori, but learned as a part of the optimization task of the CNN. Making the connection between implicit neural representations and continuous kernel formulations explicitly brings substantial insights for the construction of these kernels. In particular, it motivates the use of Sine nonlinearities [358] to parameterize them, which leads to significant improvements over the ReLU, LeakyReLU, and Swish nonlinearities used so far for this purpose (Sec. 2.4.2).

## 2.3 The Convolution and Common Kernel Parameterizations

**Notation.**  $[n]$  denotes the set  $\{0, 1, \dots, n\}$ . Bold capital and lowercase letters depict vectors and matrices, e.g.,  $\mathbf{x}, \mathbf{W}$ , sub-indices index vectors, e.g.,  $\mathbf{x}=\{x_c\}_{c=1}^{N_{\text{in}}}$ , parentheses index time, e.g.,  $\mathbf{x}(\tau)$  is the value of  $\mathbf{x}$  at time-step  $\tau$ , and calligraphic letters depict sequences, e.g.,  $\mathcal{X}=\{\mathbf{x}(\tau)\}_{\tau=0}^{N_X}$ .

**Centered and causal convolutions.** Let  $\mathbf{x} : \mathbb{R} \rightarrow \mathbb{R}^{N_{\text{in}}}$  and  $\boldsymbol{\psi} : \mathbb{R} \rightarrow \mathbb{R}^{N_{\text{in}}}$  be a vector valued signal and kernel on  $\mathbb{R}$ , such that  $\mathbf{x}=\{x_c\}_{c=1}^{N_{\text{in}}}$  and  $\boldsymbol{\psi}=\{\psi_c\}_{c=1}^{N_{\text{in}}}$ . The convolution is defined as:

$$(\mathbf{x} * \boldsymbol{\psi})(t) = \sum_{c=1}^{N_{\text{in}}} \int_{\mathbb{R}} x_c(\tau) \psi_c(t - \tau) d\tau. \quad (2.1)$$

In practice, the input signal  $\mathbf{x}$  is gathered via some sampling procedure. Resultantly, the convolution is effectively performed between the sampled input signal described as a sequence of finite length  $\mathcal{X}=\{\mathbf{x}(\tau)\}_{\tau=0}^{N_X}$  and a convolutional kernel  $\mathcal{K}=\{\boldsymbol{\psi}(\tau)\}_{\tau=0}^{N_X}$

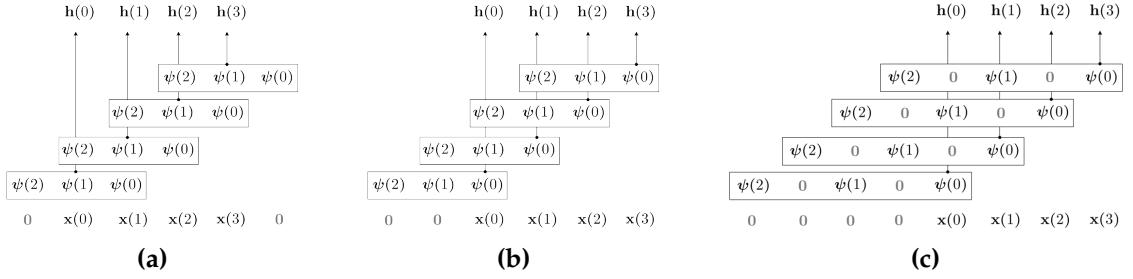


Figure 2.2: Discrete centered, causal, and dilated causal convolutions.

described the same way:

$$(x * \psi)(t) = \sum_{c=1}^{N_{in}} \sum_{\tau=-N_{X/2}}^{N_{X/2}} x_c(\tau) \psi_c(t - \tau). \quad (2.2)$$

Values  $x(\tau)$  falling outside of  $\mathcal{X}$  are often *padded* by a constant value of zero (Fig. 2.2a).

The convolutional kernel is commonly *centered* around the point of calculation  $t$ . For sequence modeling this can be undesirable as future input values  $\{x(t - \tau)\}_{\tau=-N_{X/2}}^{-1}$  are considered during the operation. This is solved by providing a *causal formulation to the convolution*: a formulation in which the convolution at time-step  $t$  only depends on input values at time-steps  $(t - \tau) \leq t$  (Fig. 2.2b):

$$(x * \psi)(t) = \sum_{c=1}^{N_{in}} \sum_{\tau=0}^t x_c(\tau) \psi_c(t - \tau). \quad (2.3)$$

In practice, causal convolutions are easily implemented via asymmetrical padding. In this work, we consider causal convolutions as default, but our analyses are valid for centered convolutions as well.

**Discrete convolutional kernels.** By a large margin, most convolutional kernels  $\psi$  in literature are parameterized as a finite sequence of  $N_K + 1$  independent learnable weights  $\mathcal{K} = \{\psi(\tau)\}_{\tau=0}^{N_K}$  (Fig. 2.2). As these weights are independent of one another,  $N_K$  must be kept small to keep the parameter count of the model tractable. Hence, the kernel size is often much smaller than the input length:  $N_K \ll N_X$ . This parameterization presents important limitations:

- The memory horizon  $N_K$  must be defined a priori.
- Since  $N_K \ll N_X$ , this parameterization implicitly assumes the convolution  $(x * \psi)$  at position  $t$  only depends on input values at positions up to  $\tau=N_K$  steps in the past. Hence, no functions depending on inputs  $x(t - \tau)$  for  $\tau > N_K$  can be modeled.
- The most general selection of  $N_K$  is given by a *global memory horizon*:  $N_K=N_X$ . Unfortunately, as discrete convolutional kernels are modeled as a sequence of independent weights, this incurs an extreme growth of the model size and rapidly becomes statistically unfeasible.

**Dilated convolutional kernels.** To alleviate these limitations, previous works propose to interleave kernel weights with zeros in order to cover larger memory horizons without additional weights (Fig. 2.2c). This formulation alleviates some of the previous limitations, but introduces additional ones:

- Dilated kernels are unable to model dependencies of input values falling in the interleaved regions.
- Several authors use dilated convolutions with varying dilation factors as a function of depth, e.g., Bai et al. [11], Dai et al. [75], Oord et al. [277]. By carefully selecting layer-wise dilation factors, one can assure that some kernel hits each input within the memory horizon of the network. However, due to the extreme sparsity of the formulation, it is difficult to estimate the effective amount of processing applied to the input. In addition, this layout ties together (i) the memory horizon, (ii) the depth, and (iii) the layer-wise dilation factors of the network, which effectively constraints the flexibility of the neural architecture design.

In contrast to the (dilated) discrete convolutions presented in this section, our proposed formulation allows handling arbitrarily long sequences with arbitrarily large, dense memory horizons in a single layer and under a fixed parameter budget.

## 2.4 Continuous Kernel Convolution

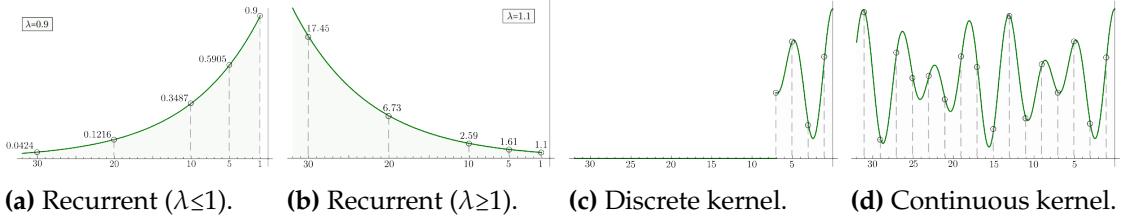
In this section, we introduce our approach. First, we define it formally, analyze its properties, illustrate its connection to recurrent units, and elaborate on the functional family they can describe. Next, we discuss concrete parameterizations of continuous convolutional kernels, illustrate their connection to implicit neural representations, and show that our final kernels are able to fit complex functions.

### 2.4.1 Formulation and Properties

**Arbitrarily large convolutional kernels.** We formulate the convolutional kernel  $\psi$  as a continuous vector-valued function parameterized by a small neural network  $\text{MLP}^\psi : \mathbb{R} \rightarrow \mathbb{R}^{N_{\text{out}} \times N_{\text{in}}}$  (Fig. 2.1, left).  $\text{MLP}^\psi$  receives a relative position  $(t-\tau)$  and outputs the value of the convolutional kernel at that position  $\psi(t-\tau)$ . As a result, an arbitrarily large convolutional kernel  $\mathcal{K} = \{\psi(t-\tau)\}_{\tau=0}^{N_K}$  can be constructed by providing an equally large sequence of relative positions  $\{t-\tau\}_{\tau=0}^{N_K}$  to  $\text{MLP}^\psi$ . For  $N_K = N_X$ , the size of the resulting kernel is equal to that of the input sequence  $\mathcal{X}$ , and thus it is able to model (global) long-term dependencies. The *Continuous Kernel Convolution* (CKConv) is given by:

$$(\mathbf{x} * \psi)(t) = \sum_{c=1}^{N_{\text{in}}} \sum_{\tau=0}^t x_c(\tau) \text{MLP}_c^\psi(t-\tau). \quad (2.4)$$

**Irregularly sampled data.** CKConvs are able to handle irregularly-sampled and partially observed data. To this end, it is sufficient to sample  $\text{MLP}^\psi$  at positions for which



**Figure 2.3:** Functional family of recurrent units, discrete convolutions and CKConvs. For max. eigenvalues of  $\mathbf{W}$ ,  $\lambda \neq 1$ , recurrent units are restricted to exponentially decreasing ( $\lambda \leq 1$ ) or increasing ( $\lambda \geq 1$ ) functions (Figs. 2.3a, 2.3b). Discrete convolutions can describe arbitrary functions within their memory horizon but are zero otherwise (Fig. 2.3c). Conversely, CKConvs define arbitrary long memory horizons, and thus are able to describe arbitrary functions upon the entire input sequence (Fig. 2.3d).

the input signal is known and perform the convolution operation with the sampled kernel. For very non-uniformly sampled inputs, an inverse density function over the samples can be incorporated in order to provide an unbiased estimation of the convolution response (see Appx. A.1.1, Wu et al. [436] for details).

**Data at different resolutions.** CKConvs can also process data at different resolutions. Consider the convolution  $(\mathbf{x} * \psi)_{sr_1}$  between an input signal  $\mathbf{x}$  and a continuous convolutional kernel  $\psi$  sampled at a sampling rate  $sr_1$ . Now, if the convolution receives the same input signal sampled at a different sampling rate  $sr_2$ , it is sufficient to sample the convolutional kernel at the sampling rate  $sr_2$  in order to perform an “equivalent” operation:  $(\mathbf{x} * \psi)_{sr_2}$ . As shown in Appx. A.1.2, it holds that:

$$(\mathbf{x} * \psi)_{sr_2}(t) \approx \frac{sr_2}{sr_1} (\mathbf{x} * \psi)_{sr_1}(t). \quad (2.5)$$

That is, convolutions calculated at different resolutions  $sr_1$  and  $sr_2$  are approximately equal up to a factor given by the resolution change. As a result, CKCNNs (i) can be trained in datasets with data at varying resolutions, and (ii) can be deployed at resolutions other than those seen during training.

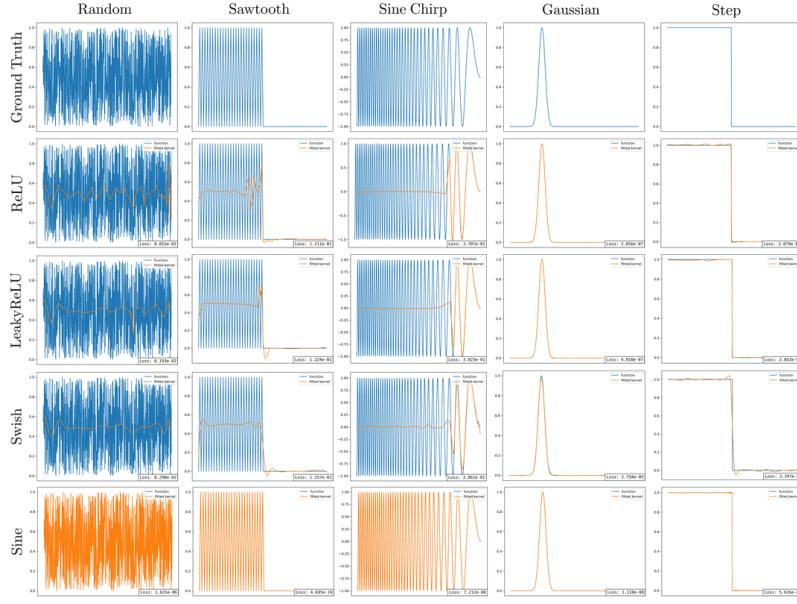
We note that, except for RNNs with continuous-time interpretations, e.g., Gu et al. [126], Kidger et al. [183], the previous features are difficult to obtain.

**(Linear) recurrent units are continuous kernel convolutions.** Consider a recurrent unit

$$\mathbf{h}(\tau) = \sigma(\mathbf{Wh}(\tau - 1) + \mathbf{Ux}(\tau)) \quad (2.6)$$

$$\tilde{\mathbf{y}}(\tau) = \text{softmax}(\mathbf{Vh}(\tau)), \quad (2.7)$$

where  $\mathbf{U}, \mathbf{W}, \mathbf{V}$  depict the *input-to-hidden*, *hidden-to-hidden* and *hidden-to-output* connections of the unit,  $\mathbf{h}(\tau)$ ,  $\tilde{\mathbf{y}}(\tau)$  the hidden representation and the output at time-step  $\tau$ , and  $\sigma$  a pointwise nonlinearity. As shown in Appx. A.1.3, we can express the hidden representation  $\mathbf{h}$  of a linear recurrent unit, i.e., with  $\sigma = \text{Id}$ , as a convolution between the input  $\mathbf{x}$  and a convolutional kernel  $\psi(\tau) = \mathbf{W}^\top \mathbf{U}$  of size equal to the input. That is, as



**Figure 2.4:** Approximation quality of MLPs with ReLU, LeakyReLU, Swish, and Sine nonlinearities. Networks with (smooth) piece-wise nonlinearities are unable to approximate non-smooth, non-linear functions. Sine networks, on the other hand, quickly approximate all target functions to near perfection. All networks share the same structure and vary only in the nonlinearity used.

a continuous kernel convolution with an exponentially increasing or decreasing kernel (Fig. 2.3). Different authors show that nonlinear recurrent units are also restricted to the same functional family [7, 282, 473].

**The functional family of continuous kernel convolutions.** From the previous observation, we can conclude that CKConvs are not only more general than discrete convolutions, but that the functional family they describe is also more general than that of (linear) recurrent units (Fig. 2.3).

## 2.4.2 The Continuous Convolutional Kernel $\text{MLP}^\psi$

**Convolutional kernels as point-wise MLPs.** Let  $\{\Delta\tau_i = (t - \tau_i)\}_{i=0}^N$  be a sequence of relative positions. The continuous vector-valued convolutional kernel  $\psi : \mathbb{R} \rightarrow \mathbb{R}^{N_{\text{out}} \times N_{\text{in}}}$  is parameterized as a neural network  $\text{MLP}^\psi$  which maps each relative position  $\Delta\tau_i$  to the value of the convolutional kernel at that position (Fig. 2.1, left). We refer to the nonlinearity used in  $\text{MLP}^\psi$  as  $\sigma$ .

**What kind of kernels can  $\text{MLP}^\psi$  generate?** Our method relies on the assumption that the neural network  $\text{MLP}^\psi$  is able to model complex dependencies densely among all elements within the memory horizon. That is, it assumes that  $\text{MLP}^\psi$  is able to generate arbitrary convolutional kernels.

To test this hypothesis, we fit existing MLP parameterizations, i.e., with ReLU, LeakyReLU and Swish nonlinearities, to long target functions of varying level of smoothness and non-linearity (Fig. A.1). We observe that existing parameterizations can approximate simple functions, e.g., Gaussian, step functions, but fail by a large marking for increasing levels of non-linearity and non-smoothness. For our analysis, this means that CKConvs with ReLU, LeakyReLU and Swish parameterizations are *not* able to represent complex input dependencies. In our ablation studies (Appx. A.4) we verify that CKCNNs with these kernels consistently perform worse than our proposed parameterization.

**Convolutional kernels as implicit neural representations.** We notice that parameterizing a convolutional kernel with a neural network is equivalent to constructing an implicit neural representation of the kernel, with the subtle difference that our target objective is not known a-priori, but learned as part of the optimization task. Implicit neural representations study generic ways to represent data living in low-dimensional spaces, e.g.,  $\mathbb{R}^2$ , via neural networks, and thus, despite this difference, constitute an interesting starting point for the parameterization of continuous convolutional kernels. In particular, recent works noticed that neural networks with piece-wise activation functions are unable to model high-frequencies. To alleviate this limitation, they introduce random Fourier features [377] and Sine nonlinearities [358].

Based on these observations, we repeat the fitting experiment for a SIREN [358]: a MLP with hidden layers of the form  $\mathbf{y} = \text{Sine}(\omega_0[\mathbf{W}\mathbf{x} + \mathbf{b}])$ . That is with Sine nonlinearities, and a non-learnable value  $\omega_0$  that serves as a prior for the oscillations of the output. We observe that a SIREN quickly approximates *all* target functions to near perfection regardless of their grade of smoothness or nonlinearity. Even a sequence of random noise. This implies that, contrary to other parameterizations, CKConvs with SIREN kernels have the ability to model complex input dependencies across large memory horizons. Our experimental results verify this statement. Our ablation studies in Appx. A.4 show that SIREN kernels consistently outperform all other variants. In addition, our experimental results in Sec. 2.5 show that *shallow* CKCNNs with SIREN kernels achieve state-of-the-art across datasets of different nature, i.e., with continuous and discrete data.

**The success of Sine nonlinearites: A spline basis interpretation.** Sitzmann et al. [358] motivates the usage of Sine nonlinearities for implicit neural representations. However, it is not clear *why* Sine nonlinearities are better suited for this task than (smooth) piece-wise nonlinearities. For the interested reader, we provide an interpretation to this phenomenon from a spline function approximation perspective in Appx. A.2.

Of most practical relevance from this analysis is the observation that proper initialization of the network parameters, particularly of the bias term  $\{\mathbf{b}^{(l)}\}$ , is important to create a well-spread set of basis functions suited for function approximation. For SIRENs, this is achieved by initializing the bias term uniformly across the period of each of the Sine components:  $\mathbf{b}_i \sim \mathcal{U}(-\pi\|\mathbf{W}_{i,:}\|^{-1}, \pi\|\mathbf{W}_{i,:}\|^{-1})$ . We observe that this initialization leads to better results and faster convergence for all tasks considered.

## 2.5 Experiments

We validate our approach against several existing models and across several tasks selected from the corresponding papers. Specifically, we benchmark its ability to handle long-term dependencies, data at different resolutions and irregularly-sampled data. A complete description of the datasets used as well as additional experiments and ablation studies can be found in the Appendix (Appx. A.3, A.4).

**Network details.** We parameterize our convolutional kernels as 3-layer SIRENs. Weight normalization [333] leads to better and faster convergence when applied to the layers in MLP, and we use it across all experiments. All our CKCNNs follow the structure shown in Fig. A.4 and vary only in the number of blocks and channels. We use two residual blocks for all experiments reported in this section. Specifications on the architectures and hyperparameters used are given in Appx. A.5. We speed up the convolutions by using the *convolution theorem* ( $f * \psi = \mathcal{F}^{-1}\{\mathcal{F}\{f\} \cdot \overline{\mathcal{F}\{\psi\}}\}$ , with  $\mathcal{F}$  the Fourier transform).

**Stress experiments.** First, we validate that CKCNNs can readily model memory horizons of different lengths. To this end, we evaluate if a shallow CKCNN is able to solve the *Copy Memory* and *Adding Problem* tasks [148] for sequences of sizes in the range [100, 6000]. Success is achieved if 100% accuracy, or a loss  $\leq 1e-4$  are obtained, for the copy memory and adding problem, respectively. Random predictions for the adding problem lead to a loss of approx. 0.17.

Our results show that a shallow CKCNN is able to solve both problems for all sequence lengths considered without requiring structural modifications (Tab. 2.2). Recurrent architectures are not able to solve the copy problem at all and could solve the sum problem up to 200 steps. TCNs with  $k=7$ ,  $n=7$  were able to solve both tasks for up to 1000 steps. However, larger lengths were out of reach as their memory horizon is constrained a priori. To handle larger sequences, TCNs must modify their network structure based on prior knowledge regarding the expected length of the input sequence.

**Discrete sequences.** The continuous nature of our kernels might give the impression that CKCNNs are only suited for continuous data, i.e., time-series. However, Sine non-linearities allow our convolutional kernels to model complex non-smooth non-linear functions (Fig. 2.4). Consequently, we validate whether CKCNNs can be applied for discrete sequence modeling on the following tasks: sMNIST, pMNIST [203], sCIFAR10 [386] and Char-level PTB [251].

Shallow CKCNNs outperform recurrent, self-attention and convolutional models on sMNIST and pMNIST (Tab. 2.1). On sMNIST, a small CKCNN (100K params.) achieves state-of-the-art results with a model 80 $\times$  smaller than the current state-of-the-art. A wider CKCNN (1M params.) slightly increases this result further. On pMNIST, we see an improvement of 0.8% over the best model of size  $\leq 100K$ , and our wider shallow CKCNN achieves state-of-the-art on this dataset. For sCIFAR10, our small CKCNN obtains similar results to a self-attention model 5 $\times$  bigger, and our wider variant im-

**Table 2.1:** Test results on discrete sequential datasets.

| MODEL           | SIZE | S-MNIST<br>Acc (%) | P-MNIST<br>Acc (%) | S-CIFAR10<br>Acc (%) | CHAR-PTB<br>bpc          |
|-----------------|------|--------------------|--------------------|----------------------|--------------------------|
| TCN [11]        | 70K  | <b>99.0</b>        | <b>97.2</b>        | -                    | <b>1.31<sup>†</sup></b>  |
| LSTM [11]       | 70K  | 87.2               | 85.7               | -                    | 1.36 <sup>†</sup>        |
| GRU [11]        | 70K  | 96.2               | 87.3               | -                    | 1.37 <sup>†</sup>        |
| IndRNN [218]    | 83K  | <b>99.0</b>        | 96.0               | -                    | -                        |
| DilRNN [44]     | 44K  | 98.0               | 96.1               | -                    | -                        |
| HiPPO [126]     | 0.5M | -                  | <b>98.30</b>       | -                    | -                        |
| r-LSTM [386]    | 0.5M | 98.4               | 95.2               | 72.2                 | -                        |
| Self-Att. [386] | 0.5M | 98.9               | 97.9               | 62.2                 | -                        |
| TrellisNet [12] | 8M   | <b>99.20</b>       | 98.13              | <b>73.42</b>         | <b>1.158<sup>†</sup></b> |
| CKCNN           | 98K  | <b>99.31</b>       | <b>98.00</b>       | 62.25                | -                        |
| CKCNN-Big       | 1M   | <b>99.32</b>       | <b>98.54</b>       | 63.74                | <b>1.045<sup>†</sup></b> |

<sup>†</sup> Model sizes are 3M for TCN, LSTM and GRU, 13.4M for TrellisNet and 1.8M for CKCNN-Big.

**Table 2.2:** Evaluation on stress tasks. ✓ marks if the problem has been solved.

| MODEL                 | SIZE | SEQ. LENGTH |     |      |      |      |
|-----------------------|------|-------------|-----|------|------|------|
|                       |      | 100         | 200 | 1000 | 3000 | 6000 |
| COPY MEMORY           |      |             |     |      |      |      |
| GRU                   | 16K  | -           | -   | -    | -    | -    |
| TCN                   | 16K  | ✓           | ✓   | ✓    | -    | -    |
| CKCNN                 | 16K  | ✓           | ✓   | ✓    | ✓    | ✓    |
| ADDING PROBLEM (LOSS) |      |             |     |      |      |      |
| GRU                   | 70K  | ✓           | ✓   | -    | -    | -    |
| TCN                   | 70K  | ✓           | ✓   | ✓    | -    | -    |
| CKCNN                 | 70K  | ✓           | ✓   | ✓    | ✓    | ✓    |

**Table 2.3:** Test results on CT, SC and SC\_raw.

| MODEL                 | SIZE | CT           | SC           | SC_RAW       |
|-----------------------|------|--------------|--------------|--------------|
| GRU-ODE [81]          | 89K  | 96.2         | 44.8         | ~ 10.0       |
| GRU- $\Delta t$ [183] | 89K  | 97.8         | 20.0         | ~ 10.0       |
| GRU-D [46]            | 89K  | 95.9         | 23.9         | ~ 10.0       |
| ODE-RNN [325]         | 88K  | 97.1         | 93.2         | ~ 10.0       |
| NCDE [183]            | 89K  | 98.8         | 88.5         | ~ 10.0       |
| CKCNN                 | 100K | <b>99.53</b> | <b>95.27</b> | <b>71.66</b> |

proves performance by an additional 1%. Our best results are obtained with an even wider model (2.5M params) with which an accuracy of 65.59% is obtained. On Char-level PTB a CKCNN with 3M parameters outperforms all models considered as well as the state-of-the-art: Mogrifier LSTMs [260], while being 13.3× smaller.

**Time-series modeling.** Next, we evaluate CKCNNs on time-series data. To this end, we consider the *Character Trajectories* (CT) [9] and the *Speech Commands* (SC) [421] datasets. We follow Kidger et al. [183] to obtain a balanced classification dataset with precomputed mel-frequency cepstrum coefficients. In addition, we evaluate the ability of CKCNNs to model long-term dependencies by training on the raw SC dataset (*SC\_raw*), whose records have length 16k.

We compare CKCNNs to representative sequential models with continuous-time interpretations: GRU-ODE [81], GRU- $\Delta t$  [183], ODE-RNN [325], and NCDE [183]. Sequential continuous-time models were selected as they are only sequential methods also able to handle irregularly-sampled data, and data at different resolutions. Our results show that shallow CKCNNs outperform all continuous-time models considered for both the CT and SC datasets (Tab. 2.3). In addition, CKCNNs obtain promising results on *SC\_raw*, which validates their ability to handle very-long-term dependencies. In fact, CKCNNs

**Table 2.4:** Test results on irregular data.

| MODEL           | PHYSIONET<br>AUC | CHARACTERTRAJECTORIES |              |              |       | SPEECHCOMMANDS_RAW |              |              |              |
|-----------------|------------------|-----------------------|--------------|--------------|-------|--------------------|--------------|--------------|--------------|
|                 |                  | (0%)                  | (30%)        | (50%)        | (70%) | (0%)               | (30%)        | (50%)        | (70%)        |
| GRU-ODE         | 0.852            | 96.2                  | 92.6         | 86.7         | 89.9  | ~ 10.0             | ~ 10.0       | ~ 10.0       | ~ 10.0       |
| GRU- $\Delta t$ | 0.878            | 97.8                  | 93.6         | 91.3         | 90.4  |                    |              |              |              |
| GRU-D           | 0.871            | 95.9                  | 94.2         | 90.2         | 91.9  | :                  | :            | :            | :            |
| ODE-RNN         | 0.874            | 97.1                  | 95.4         | 96.0         | 95.3  |                    |              |              |              |
| NCDE            | 0.880            | 98.8                  | 98.7         | 98.8         | 98.6  |                    |              |              |              |
| CKCNN           | <b>0.895</b>     | <b>99.53</b>          | <b>99.30</b> | <b>98.83</b> | 98.14 | <b>71.66</b>       | <b>63.46</b> | <b>60.55</b> | <b>57.50</b> |

trained on SC\_raw are able outperform several Neural ODE models trained on the pre-processed data (SC).

In addition, we observed that neural ODE methods considered in Tab. 2.3 were prohibitively slow for long sequences. For instance, NCDEs were  $228\times$  slower than a CK-CNN of equivalent size on SC\_raw, taking 17 hours per epoch to train. Consequently, training a NCDE on SC\_raw for a matching number of epochs would take more than 212 days to conclude. In order to provide results for these models, we train them under the same computational budget than CKCNNs. This is enough to train them for a single epoch. All obtained results are at best only marginally better than random.

**Testing at different sampling rates.** We now consider the case where a network is trained with data at a sampling rate  $sr_1$ , and tested with data at a different sampling rate  $sr_2$ . Our results show that the performances of CKCNNs remains stable for large sampling rate fluctuations (Tab. 2.5). This behaviour contrasts with most previous continuous-time models, whose performance rapidly decays upon these changes. CKCNNs outperform HiPPO [126] and set a new state-of-the-art in this setting. Importantly, depending on the sampling, additional care may be needed to account for spatial displacements and high-frequencies of our kernels (see Appx. A.5.2 for details).

**Irregularly-sampled data.** To conclude, we validate CKCNNs for irregularly-sampled data. To this end, consider the PhysioNet sepsis challenge [307] as well as the CT dataset with drops of 30%, 50% and 70% of the data as in Kidger et al. [183]. In addition, we provide results under the same methodology for the SC\_raw dataset.

Our results show that CKCNNs outperform NCDEs and obtain state-of-the-art performance on the PhysioNet dataset. In addition, CKCNNs exhibit stable performance for varying quantities of missing data, and perform better than several models explicitly developed to this end (Tab. 2.4). On the CT dataset, NCDEs perform slightly better than CKCNNs for large data drop rates. However, we argue that our method is still advantageous due to the gains in training speed –see Section 2.6 for details–.

**Table 2.5:** Results for different train and test resolutions. Fractions depict resolutions proportional to the original one of the dataset. The accuracy of all models on the original resolution surpasses 90%.

| CKCNN - SIZE=100K |             |              |              |              |              |       |
|-------------------|-------------|--------------|--------------|--------------|--------------|-------|
| DATASET           | TRAIN FREQ. | TEST FREQ.   |              |              |              |       |
|                   |             | 1            | 1/2          | 1/4          | 1/8          | 1/16  |
| CT                | 1           | <b>99.53</b> | 99.30        | 99.30        | 95.80        | 76.45 |
|                   | 1/2         | 98.83        | <b>99.07</b> | 98.37        | 96.97        | 80.42 |
|                   | 1/4         | 96.74        | 96.97        | <b>99.30</b> | 98.83        | 84.85 |
|                   | 1/8         | 96.97        | 97.44        | 97.20        | <b>99.30</b> | 73.43 |
| SC_RAW            | 1           | <b>71.66</b> | 65.96        | 52.11        | 40.33        | 30.87 |
|                   | 1/2         | 72.09        | <b>72.06</b> | 69.03        | 63.00        | 29.67 |
|                   | 1/4         | 68.25        | 68.40        | <b>69.47</b> | 67.09        | 37.91 |
|                   | 1/8         | 40.48        | 42.00        | 54.91        | <b>66.44</b> | 22.29 |

| MODEL COMPARISON - CHARACTER TRAJECTORIES |       |         |      |      |       |              |
|---|-------|---------|------|------|-------|--------------|
| MODEL                                     | GRU-D | ODE-RNN | LMU  | NCDE | HIPPO | CKCNN        |
| 1 → 1/2                                   | 23.1  | 41.8    | 44.7 | 6.0  | 88.8  | <b>99.30</b> |
| 1/2 → 1                                   | 25.5  | 31.5    | 11.3 | 13.1 | 90.1  | <b>98.83</b> |

## 2.6 Discussion and Limitations

**Parameter-efficient large convolutional kernels.** CKConvs construct large complex kernels with a fixed parameter budget. For large input sequences, this results in large savings in the number of parameters required to construct global kernels with conventional CNNs. For sequences from the pMNIST (length = 784) and SC\_raw (length = 16000) datasets, a conventional CNN with global kernels would require 2.14M and 46.68M of parameters, respectively, for a model equivalent to our CKCNN (100K). In other words, our kernel parameterization allows us to construct CKCNNs that are 21,84 and 445,71 times smaller than corresponding conventional CNNs for these datasets. Detailed exploration on the effect of our efficient continuous kernel parameterizations in optimization, overfitting and generalization is an interesting direction for future research.

**Is depth important? Shallow global memory horizons.** Our results are obtained with CKCNNs built with two residual blocks only. Additional experiments (Appx. A.4.2) indicate that our models do not benefit from larger depth, and suggest that CKCNNs do not rely on very deep features. Though further analysis is required to draw consistent conclusions, it is intriguing to explore if it is sufficient to equip neural networks with global memory horizons even if this happens in a shallow manner.

**High-frequency components.** Interestingly, our kernels often contain frequency components higher than the resolution of the grid used during training (Fig. A.5). As a result, transitions to finer resolutions benefit from smoothing (see Appx. A.5.3). Nevertheless, we believe that, if tuned properly, these high-frequency components might prove advantageous for tasks such as super-resolution and compression.

**Faster continuous-time models.** CKCNNs rely on convolutions, and thus can be executed in parallel. As a result, CKCNNs can be trained faster than recurrent architectures. This difference becomes more pronounced with concurrent continuous-time models for sequential data, which are based on neural ODEs and require at least 5 $\times$  slower than RNNs [183]. At the cost of larger memory costs, CKCNNs can be further sped up by using the convolution theorem.

**Neural networks parameterizing spatial functions should be able to model high-frequencies.** Our findings indicate that, common nonlinearities do not provide MLPs modelling spatial continuous functions the ability to model high-frequencies. Consequently, architectures that model continuous spatial functions via neural networks should transition towards models endowed with this ability, e.g., MLPs with Sine nonlinearities. These models encompass convolutional networks with continuous kernels [343, 382, 436], positional encodings in transformers [163, 314], and graph neural networks [82]. Sine nonlinearities can be used to reduce the number of parameters needed to model local functions, or to extend the receptive field of the operations efficiently.

**Memory requirements.** Although, CKCNNs can be deployed and trained in parallel, CKCNNs must store the convolution responses at each layer and for all input positions. This induces a linear memory complexity with regard to the sequence length, and largely contrasts with recurrent continuous-time models, whose memory complexity is constant. The memory consumption of the operation is further incremented if the convolution theorem is applied because it requires multiplying the Fourier transform of the convolution and the kernel, and taking them back to the temporal representation. On the other hand, large convolutional kernels seem to allow CNNs to perform well without using many layers, which has a positive effect on memory consumption.

**Selection of  $\omega_0$ .** We observe that CKCNNs are very susceptible to the selection of  $\omega_0$ . For instance, performance on pMNIST may vary from 98.54 to 65.22 for values of  $\omega_0$  in [1, 100]. Consequently, finding a good value of  $\omega_0$  induces an important cost in hyperparameter search (see Appx. A.5.4).  $\omega_0$  acts as a prior on the variability of the target function. However, it is not obvious which value of  $\omega_0$  is optimal for the internal (unknown) features of a network. Learning layer-wise  $\omega_0$  values yielded sub-optimal results and the best results were obtained with a predefined  $\omega_0$  value across all layers.

## 2.7 Conclusion and Future Work

We introduced the Continuous Kernel Convolution (CKConv), a simple, yet powerful approach able to model global long-term dependencies effectively in a parameter-efficient manner. Aside from the ability to get good accuracy, CKConvs are readily able to handle irregularly-sampled data, and data at different resolutions. CKCNNs achieve state-of-the-art results on multiple datasets, and often surpass neural architectures designed for particular settings, e.g., for irregularly-sampled data.

We are intrigued about the potential of CKCNNs for tasks in which (global) long-term dependencies play a crucial role, e.g., audio, video, reinforcement learning, (autoregressive) generative modeling. The usage of CKConvs to model long-term interactions in images is also very promising. In addition, CKConvs provide a convenient way to study the effect of the receptive field size of convolutional architectures, as no network modifications are required for different sizes. Our findings may also be useful for specific problems with irregularly-sampled data, e.g., medical, point clouds. We are also excited about structural advances of CKConvs. For instance, attentive versions of CKCNNs, or formulations that further improve computation and parameter efficiency

**Alleviating limitations.** Reducing the memory consumption of CKConvs is vital for its application on a broad range of scenarios, e.g., embedded devices. Moreover, finding kernel parameterizations more stable to hyperparameter changes is desirable to reduce the need for hyperparameter search.

**What is the best implicit kernel parameterization for convolutional kernels?** Despite the success of SIRENs, we believe that better kernel parameterizations might still be constructed, e.g., with Random Fourier Features [377]. Aside from improvements in implicit neural representations, which are directly transferable to CKConvs, we consider important to analyze the effect that having unknown, changing target objectives has on the approximation. A thorough empirical study of possible kernel parameterizations is an important direction for future research. A parameterization with which additional desiderata, e.g., smoothness, can be imposed is also desirable.

# 3

## Modelling Long Context in ND: From Task-Specific to a General Purpose CNN

*Based on the papers:*

*Towards a General Purpose CNN for Long Range Dependencies in ND [320]*  
*Modeling Long Context in ND: From Task-Specific to a General Purpose CNN [191]*

### 3.1 Introduction

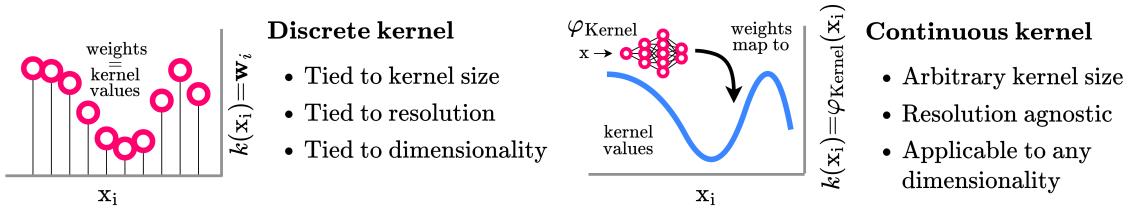
The vast popularity of Convolutional Neural Networks [208] (CNNs) is a result of their high performance and efficiency, which has led them to achieve state-of-the-art in applications across sequential [1, 392], visual [196, 356] and high-dimensional data [343, 436]. Nevertheless, a major limitation of CNNs –and other neural networks– is that their architectures must be tailored to particular applications in order to consider the length, resolution and dimensionality of the input data. This has led to a plethora of task-specific architectures [11, 143, 277, 297, 323, 356, 372, 436] which (*i*) hampers the selection of the most appropriate architecture for a particular task, and (*ii*) obscures the transfer and generalization of insights across applications. In this work, we tackle the need for problem-specific CNN architectures and propose a generic CNN architecture that can be used independent of the length, resolution and dimensionality of the data.

**CNN architectures are data dependent.** Current CNN architectures are task-specific because they are tied to the *length*, *resolution*, and *dimensionality* of the input. The *length* of the data varies from task to task, e.g. audio fragments may span milliseconds to minutes. This requires carefully chosen strides and pooling to capture relevant dependen-

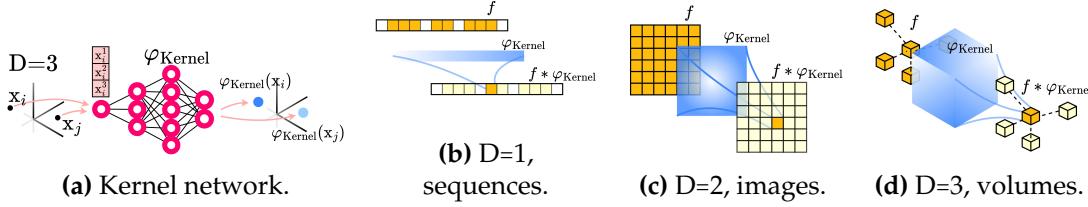
cies across the entire input [210, 392]. In addition, physical signals, e.g., audio, images, are continuous in nature. As such, their semantic meaning is independent of the *resolution* at which they are sampled, e.g., the same audio may be expressed at different resolutions. Nevertheless, current CNN architectures are resolution-bound, and thus different resolutions require different CNNs. These limitations aggravate when considering *multi-dimensional* data. Each input dimension can be defined at different lengths and resolutions, e.g., video, rectangular images, and each data modality brings its own conventions for each of these properties, e.g., the resolution of a second of audio (16kHz) [421] strongly contrasts with that of images ( $32 \times 32$ ) [195].

**Towards a unified CNN architecture.** As discussed in Sec. 3.3, the core component that makes CNNs data-dependent are their *discrete convolutional kernels*. Convolutional kernels are implemented via a one-to-one mapping between kernel values and model parameters (Fig. 3.1 left), which (i) binds them to the input resolution and length, and (ii) makes them ill suited to model long-range dependencies. The latter results from the large number of parameters needed to construct large kernels. This is why standard CNNs favour local kernels in combination with task-dependent depths and pooling layers to model long-range dependencies, at the cost of making them task-dependent.

**The need for a continuous parameterization.** To overcome task-dependent architectures, it is crucial to define a kernel parameterization that decouples parameter count from kernel size. Following Romero et al. [321], Schütt et al. [343], we use a small neural network to define a continuous mapping from positions to the value of the kernel at those positions. The resulting *Continuous Convolutional Kernels* (Fig. 3.2), allow for the construction of convolutional kernels of arbitrary size in a parameter efficient manner. Consequently, the same convolutional layers –and thus the same CNN– can be used regardless of the input length, resolution and dimensionality. We leverage this formulation to construct the *Continuous Convolutional Neural Network* (CCNN): a single CNN architecture that can be applied regardless of input length, resolution or dimensionality.



**Figure 3.1:** Discrete and continuous convolutional kernels. Discrete convolutional kernels assign a weight  $w_i$  out of a discrete set of weights  $\mathbf{W}$  to a relative offset  $x - \tilde{x}$ . This ties the kernel to the length, resolution and dimensionality of the input, limiting the general applicability of the CNN architectures. Instead, our *Continuous Convolutional Neural Network* parameterizes kernel values as a continuous function  $\varphi_{\text{Kernel}}$  over the input domain  $\mathbb{R}^d$ , which decouples it from data characteristics.



**Figure 3.2:** Continuous convolutional kernels: the key to a unified CNN architecture. The continuous parameterization of convolutional kernels used in this work consists of a small kernel network  $\varphi_{\text{Kernel}}$  that receives coordinates as input and outputs the value of the convolutional kernel at that position (3.2a). By changing the dimensionality of the coordinates  $x_i$ , the same kernel network can render convolutional kernels for sequential (3.2b), visual (3.2c), and higher dimensional data (3.2d).

**Empirical results.** To showcase the proposed CCNN, we deploy the same CCNN for several tasks on sequential (1D), visual (2D) and point-cloud (3D) data. Our CCNN matches and often outperforms the current state-of-the-art across all tasks considered. Importantly, the continuous parameterization of our CCNN allows it to handle irregularly sampled data natively. As a result, the CCNN is not restricted to grid data, e.g., 3D voxels, and can be used on point-clouds directly.

#### Contributions:

- We propose the *Continuous Convolutional Neural Network*: a general purpose CNN architecture able to process data of arbitrary resolution, dimensionality and length without structural changes.
- We study the layers of CNNs, and demonstrate that the ability to model long-term dependencies on ND without the need of input dependent downsampling and depth values is *necessary and sufficient* for the construction of a general purpose CNN architecture.
- In order to model long-term dependencies on ND without input dependent down-sampling and depth values, we utilize and improve the Continuous Kernel Convolutions of Romero et al. [321]. Our proposed improvements allow the proposed Continuous CNN to achieve good empirical results on the tasks considered in 1D, 2D and 3D without structural changes.

## 3.2 Related Work

A section with extended comparisons to related works is provided in Appx. B.1.

**General purpose architectures.** To the best of our knowledge, the only existing method aiming for a general purpose architecture is the Perceiver [168], which uses a Transformer to lift restrictions regarding data characteristics and modalities. However, (i) it must map inputs –regardless of their size– to a small latent representation to reduce the

quadratic complexity of self-attention, (ii) decouples the depth of the network from its parameter count via recurrence, which requires tuning the number of unrolling steps per task, and (iii) must use absolute positional encodings to encode the data structure, which break the translation equivariance of the self-attention operation [314]. In contrast, CCNNs provide a general purpose architecture that: (i) scales much more favorably than self-attention, (ii) does not require a constant small latent representation, (iii) does not require task-dependent depths, and (iv) preserves translation equivariance.

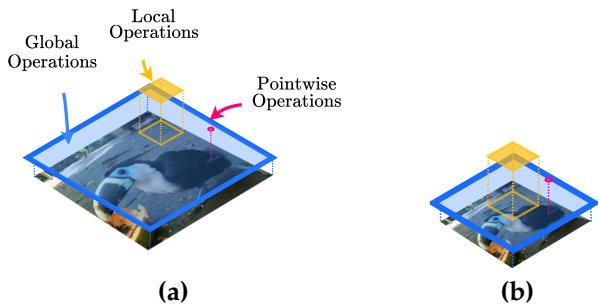
**Long range dependencies in ND.** Based on our analysis (Sec. 3.3), any architecture able to model long range dependencies in ND without the need of input-dependent pooling or depth could be used as a general purpose architecture. To our best knowledge, the only existing convolutional methods able to construct global convolutional kernels are CKConvs [319, 321] and state-spaces [128, 129]. However, state-spaces rely on complex dynamical systems that are not easily defined in ND –aside from the combination of 1D systems, equivalent to representing ND kernels as combinations of  $N$  independent 1D kernels– [272]. Consequently, we select CKConvs as the building block of our approach given their advantages in terms of expressivity and simplicity.

**Convolutional kernels as neural networks.** Modelling convolution kernels with small neural networks that map kernel positions to kernel values showed promising results for small kernels [171, 343, 436]. Subsequently, [321] realized that this parameterization decouples the size of the convolutional kernel from the number of parameters required to construct it, and thus can be used to construct arbitrarily large convolutional kernels in a parameter efficient manner. In this work, we show that this parameterization allows for the construction of a single CNN that can be used regardless of the input length, resolution and dimensionality.

Romero et al. [321] realized that the piece-wise MLPs used so far to parameterize convolutional kernels were unable to model complex long range dependencies due to their spectral bias [377], and showed that implicit neural representations –specifically SIRENs [358]– could be used to solve the issue. Subsequently, Romero et al. [319] parameterized their convolutional kernels with Multiplicative Anisotropic Gabor Nets (MAGNets), which provided them control over frequencies admitted in the convolutional kernel, thus preventing aliasing and aiding generalization across resolutions. In our work, we use FlexConvs parameterized by MAGNets. However, we observe that neural networks used to parameterize convolutional kernels are not correctly initialized for that purpose, and propose an initialization that solves the issue.

### 3.3 Toward Data-Independent CNN Architectures

In this section, we study the components of CNN architectures and pinpoint the changes required in order to construct a CNN architecture independent of input lengths, resolutions and dimensionalities.



**Figure 3.3:** Operation types: global, pointwise and local. Local operations are resolution dependent. Transferring a local operation from (3.3a) to a lower resolution (3.3b) leads to an increased receptive field.

### 3.3.1 Pointwise Operations: Linear Layers, Dropout, Pointwise Nonlinearities and Residual Connections

Pointwise operations are operations applied to each spatial element of the input separately (Fig. 3.3), e.g., pointwise linear layers, dropout and pointwise nonlinearities. As such pointwise operations do not depend on the input shape and model the same function regardless of input length, resolution and dimensionality. Learnable parameters of pointwise operations, e.g., in pointwise linear layers and PReLU [141], are shared over the spatial domain; the same set of parameters is applied to inputs of any spatial shape.

**Conclusion.** Based on the previous observations, we can conclude that pointwise operations can be used without changes within a general-purpose CNN architecture.

### 3.3.2 Global Operations: Normalization Layers and Global Pooling

Global operations aggregate all spatial elements of the input for their processing, e.g., normalization layers, global pooling. As such, common global operations do not depend on the specific shape of the input signal and their effect is equivalent regardless of the input length, resolution and dimensionality. It is important to note, however, that common global CNN operations only define learnable parameters along the channel axes e.g., the scale and mean parameters of a normalization layer have shape  $\mathbf{w} \in \mathbb{R}^{N_{\text{in}}}$ . Nevertheless, if one were to define a global operation for which spatial positions are also assigned weights, the shape of the parameters would be dependent on the input length, resolution and dimensionality, and thus the previous statement would no longer hold.

**Conclusion.** The previous observations indicate that global operations that only define channel-wise learnable parameters can be used without changes in order to construct a unified CNN architecture. Luckily, this is the case for all common global operations used in CNN architectures.<sup>1</sup>

### 3.3.3 Local Operations: Convolutional Layers and Subsampling

Local operations are operations that rely on spatial portions of the input and are applied across its spatial dimensions, e.g., convolution, subsampling. In practice, the neighbor-

<sup>1</sup>Global discrete convolutional kernels can be interpreted as global operations for which parameters along the spatial dimensions of the input are defined. Therefore, these require special treatment (Sec. 3.3.3).

hoods on which these operations are applied are hyperparameters, e.g.,  $3 \times 3$  kernels or pooling over  $2 \times 2$  regions, and are selected based on the input size. Unfortunately, if the resolution of the input changes then the portion of the input that the operation considers changes and the effect of the operation changes (Fig. 3.3). Similarly, if the length of the input changes, then larger neighborhoods are required in order to model dependencies across the input. These effects exacerbate if one considers multi-dimensional inputs, as different dimensions might require different neighborhood sizes. Consequently, we can say that local operations *depend on the resolution, length and dimensionality of the input*.

A possible solution would be to adjust the size of the operations proportional to resolution and length changes. Unfortunately, if the local operation defines learnable parameters over its spatial dimensions –as in (discrete) convolutional kernels (Sec. 3.3.3)–, then increasing the size of the convolutional kernel is tied to a proportional increase in the number of parameters required to construct it. This, in turn changes the learning dynamics of the network and easily becomes prohibitive.

**The need for a continuous parameterization.** A better solution results from using a parameterization in which the number of parameters is *independent* from the size of the kernel. By doing so, the same parameterization can be used independently from the input length, resolution and dimensionality.

### From Data-Dependent to Data-Independent Convolutional Layers

Conventional CNNs implement a discrete version of the convolution operation

$$(f * k)^o(\mathbf{x}) = \int_{\mathbb{R}^D} f(\mathbf{x} - \tilde{\mathbf{x}})k^o(\tilde{\mathbf{x}})d\tilde{\mathbf{x}}, \quad o \in [1, \dots, N_{\text{out}}], \quad (3.1)$$

where  $f : \mathbb{R}^D \rightarrow \mathbb{R}^{N_{\text{in}}}$  is a D-dimensional input signal with  $N_{\text{in}}$  channels, and  $k : \mathbb{R}^D \rightarrow \mathbb{R}^{N_{\text{in}} \times N_{\text{out}}}$  is a set of  $N_{\text{out}}$  convolutional kernels. For the types of data CNNs are commonly used for, e.g. images, the signal  $f$  is generally sampled on a discrete grid of equidistant points, and thus it can be described as a function  $f : \mathbb{Z}^D \rightarrow \mathbb{R}^{N_{\text{in}}}$ . In practice, the signal  $f$  is non-zero only on a finite subset of the grid  $\Omega(f) \subset \mathbb{Z}^D$  with limits given by the range of sampling, e.g., height and width of an image. Accordingly, the convolutional kernel  $k$  is defined over the same grid of coordinates, of which generally a subset  $\Omega(k) \subset \Omega(f)$  maps to nonzero values. Since both  $\Omega(k)$  and  $\Omega(f)$  are discrete and finite, the convolution is computed as the inner product of function and kernel values at each position:

$$(f * k)^o(\mathbf{x}) = \sum_{\tilde{\mathbf{x}} \in \Omega(k)} f(\mathbf{x} - \tilde{\mathbf{x}})k^o(\tilde{\mathbf{x}}), \quad \mathbf{x} \in \Omega(f). \quad (3.2)$$

**Discrete convolutional kernels tie convolutional layers to data characteristics.** Conventionally, the convolutional kernel  $k$  is implemented through a discrete set of randomly initialized weights  $\mathbf{W}$ , of which each entry  $\mathbf{w}_i \in \mathbb{R}^{N_{\text{in}} \times N_{\text{out}}}$  corresponds to a point  $\mathbf{x}_i \in \Omega(k)$ . Consequently, an increased kernel size is reflected in a larger set  $\Omega(k)$ , and thus in a correspondingly larger weight matrix  $\mathbf{W}$ . This directly ties a model’s parameter count to its kernel size.

In order to model the long-range dependencies needed to extract high-level features in a parameter-efficient way, we must then resort to pooling operations and the stacking of layers that implicitly increase the receptive field of the kernel. This in turn, makes CNNs effective only on inputs of a certain size. For example, if we apply a network created to model long-range dependencies over images of size  $256 \times 256$  to images of size  $32 \times 32$ , then intermediary pooling layers would make the spatial extent of the feature maps collapse long before all convolutional layers are applied. Similarly, if we use a network designed to model long-range dependencies on  $32 \times 32$  on images of size  $256 \times 256$ , the network will not be able to model long-range dependencies in the input.

Additionally, the definition of the kernel  $k$  through a discrete set of weights  $\mathbf{W}$  ties the model to a given input resolution. Yet, for many applications, the input  $f$  is a discretization of an underlying continuous function. Therefore, we would like our model to provide the same responses regardless of the resolution at which  $f$  is provided. As discrete convolutional kernels live in a discrete domain, they cannot be easily represented at other resolutions. In fact, one can show that discrete CNNs do not generalize to unseen resolutions [272, 319]. Consequently, it is not possible to reliably apply trained discrete CNNs across resolutions.

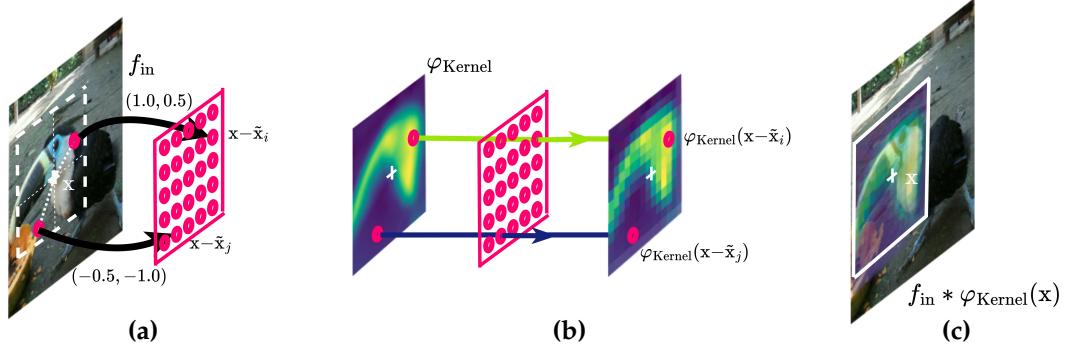
In addition, note that in Eq. 3.2, the same discrete convolutional kernel  $k$  can be used at every location  $\mathbf{x} \in \Omega(f)$  only because the input signal  $f$  is defined over an equidistant grid, and the values of the discrete kernel  $k(\tilde{\mathbf{x}})$  align with the features  $f(\mathbf{x})$ ,  $\forall \mathbf{x} \in \Omega(f)$ . This is not the case for irregular data. Consequently, discrete kernels are ill-suited to handle irregular data. With weights fixed to relative positions, an infinite number of weights would be needed to cover any continuous domain.

These limitations suggest a better approach to model convolutional kernels: *using the model weights to parameterize  $k$  as a continuous function over the data domain  $\mathbb{R}^d$* .

**A data independent parameterization.** To obtain a formulation for a CNN applicable to arbitrary resolutions and sizes, we require a parameterization for convolutional layers that is invariant to the set  $\Omega(f)$  over which  $f$  is sampled. In other words, we must find a parameterization with which the kernel can be modelled over the underlying continuous domain of the input signal, i.e.,  $\mathbb{R}^d$ . Moreover, to avoid models with different parameter count for different resolutions, it is necessary that the parameterization of the kernel decouples its parameter count from the size of the kernel.

Such a parameterization is provided by *Continuous Kernel Convolutions* (CKConvs) [319, 321]. CKConvs provide a continuous parameterization for convolutional kernels by using a small neural network  $\varphi_{\text{Kernel}} : \mathbb{R}^D \rightarrow \mathbb{R}^{N_{\text{out}} \times N_{\text{in}}}$  as a kernel generator network. This network maps coordinates in the domain of the kernel  $\mathbf{x}_i \in \mathbb{R}^D$  to the values of the convolutional kernel at that position:  $k(\mathbf{x}) \in \mathbb{R}^{N_{\text{out}} \times N_{\text{in}}}$  (Fig. 3.1). A Continuous Kernel Convolution is illustrated in Fig. 3.4.

Since the parameter count of the kernel generator network is independent from the num-



**Figure 3.4:** Applying a Continuous Kernel Convolution. Given a pixel-position in the input image  $x \in \Omega(f)$ , we obtain relative offsets to surrounding pixels  $\{x - \tilde{x}\}_{\tilde{x} \in \Omega(f)}$  (3.4a). Next, we pass each relative position to the kernel generator network in order to generate the kernel:  $\{\varphi_{Kernel}(x - \tilde{x})\}_{\tilde{x} \in \Omega(f)}$  (3.4b). Finally, we apply the convolution between the generated convolutional kernel and the input (3.4c).

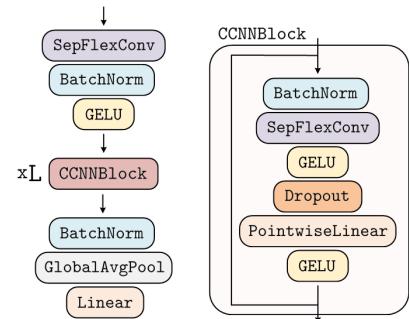
ber of points in the neighborhood that determines the size of the kernel, CKConvs allow for construction of arbitrarily large kernels without increasing the parameter count of the layer. [321] shows that large kernels allow CNNs to model long range spatial dependencies at every layer, thus removing the need for downsampling and stacking of layers to increase receptive fields. This in turn allows us to build an architecture which does not contain resolution-, dimensionality-, and size-dependent layers.

**Conclusion.** The previous observations indicate that local operations equipped with existing parameterizations are tied to the length, resolution and dimensionality of the input. Nevertheless, this limitation can be lifted if an alternative parameterization is used that detaches the neighborhood of action of the operation from the length, resolution and dimensionality of the input.

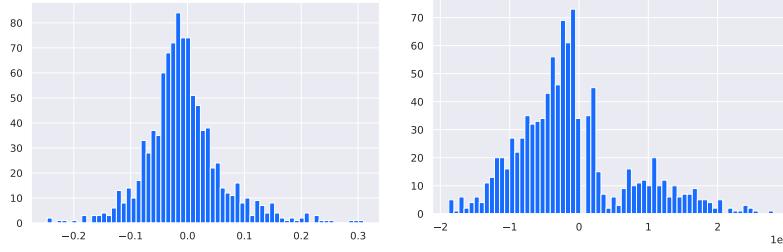
### 3.4 A General Purpose CNN Architecture

**Starting point.** In principle, the CNN architectures with CKConv [321] and FlexConv [319] introduced previously fulfill the requirements posited in Sec. 3.3. Nevertheless, as depicted in these papers, these architectures still must be tailored to specific applications and domains in order to perform well, e.g., TCN [11] and ResNet [143] backbones for sequential and visual tasks, respectively in Romero et al. [319].

In order to construct a single performant general purpose architecture that works well across all



**Figure 3.5:** The CCNN architecture.



**Figure 3.6:** Histogram of the output of a CCNN with and without our proposed kernel initialization.

tasks considered, we start from a FlexNet [319], and propose several structural changes (Sec. 3.4.1). The resulting CCNN architecture is shown in Fig. 3.5.

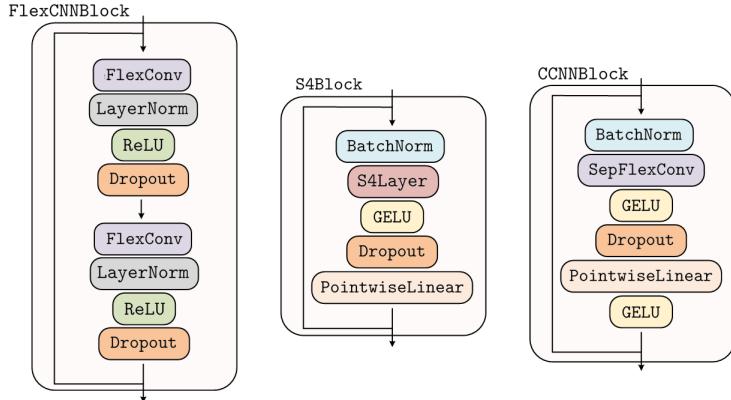
### 3.4.1 Modifications and Improvements

**Proper initialization of the kernel generator network  $\varphi_{\text{Kernel}}$ .** First, we analyze the kernel generator network  $\varphi_{\text{Kernel}}$ , and observe that it is not initialized properly in previous works [319, 321, 343, 436] for the purpose of parameterizing convolutional kernels.

Recall that, in order to ensure training stability it is desirable to retain a constant (unitary) variance throughout the activations of a neural network [119]. Hence, the discrete weights  $\mathbf{W}$  conventionally used to construct a convolutional kernel are initialized to have variance inversely proportional to the number of elements over which the convolution is computed, i.e., the number of pixels  $|\Omega(k)|$  times the number of channels  $N_{\text{in}}$ . For instance, He initialization [141] initializes  $\mathbf{W}$  s.t.  $\text{Var}(\mathbf{W})=g^2/(N_{\text{in}}|\Omega(k)|)$ , with a gain  $g$  that depends on the nonlinearity used.

In related works, the networks used to parameterize convolutional kernels are themselves initialized to *preserve a constant (unitary) variance throughout the network*. Consequently, when used as a kernel generator network, standard initializations lead the generated kernel to have unitary variance, i.e.,  $\text{Var}(k)=1$ . This in turn, make CNNs using neural networks to parameterize their convolutional kernels experience a layer-wise growth in the variance of the feature maps proportional to  $N_{\text{in}} \cdot |\Omega(k)|$ . This growth is of particular importance for kernel generator networks that generate large convolutional kernels, i.e., with large  $|\Omega(k)|$ . For instance, we observe that the logits of CNNs with CK-Convs [321] and FlexConvs [319] lie in the order of  $10^{19}$  on initialization: an undesirable property that might lead to unstable training and a need for low learning rates.

To address this issue, we must ensure that the variance at the output of the kernel generator network is inversely proportional to  $N_{\text{in}} \cdot |\Omega(k)|$ . Inspired by Chang et al. [43], we therefore re-weight the last linear layer of the kernel generator network  $\varphi_{\text{Kernel}}$  by  $g^2/\sqrt{N_{\text{in}} \cdot |\Omega(k)|}$ . With this modification, we observe that the variance of the generated convolutional kernels satisfies the desired constraints, and consequently, the logits of our CCNN show unitary variance upon initialization (Fig. 3.6).



**Figure 3.7:** Residual blocks used in FlexCCNNs [319], S4 [129] and CCNNs (ours).

**Depthwise Separable Continuous Convolutions.** Separable convolutions have long been used to improve the parameter and computational efficiency of CNNs [311, 354]. Recent architectures have leveraged separability, and found CNNs with separable kernels to perform better than CNNs with conventional convolutions [190, 236]. This phenomenon results from the separation of spatial and channel dimensions, which allows for wider networks without additional computational and parameter complexity.

Based on these observations, we construct a depth-wise separable version of FlexConv [319], in which a channel-wise convolution is computed with a kernel generated by a kernel generator network  $\varphi_{\text{Kernel}} : \mathbb{R}^D \rightarrow \mathbb{R}^{N_{\text{in}}}$ , followed by a pointwise linear layer from  $N_{\text{in}}$  to  $N_{\text{out}}$  dimensions. Separable FlexConvs allow constructing a much wider CCNN—from 30 to 140 hidden channels—with the same parameter and computation complexity.

**An improved residual block.** Residual connections [143] provide training stability and improved performance. A residual block  $\mathcal{R}(f) = \psi(f) + f$  is defined as the sum between the input and a so-called *residual connection*  $\psi$  composed of multiple layers: convolutional, normalization, etc.

Recent studies found improvements over the residual block of He et al. [143] by changing the nonlinearities as well as the position and type of normalization layers within the blocks [236, 441]. Based on these advances and empirical evidence, we modify the residual blocks of Romero et al. [319] with residual blocks similar to those of S4 [129] complemented with a nonlinearity at the end of the block. A comparison of our residual block and those in [129] and [319] is given in Fig. 3.7.

**$L_2$  regularization of continuous kernels.** Weight decay penalizes the  $L_2$  norm of learned kernel values  $k$  by adding the norm of the weights  $\mathbf{W}$  as an additional loss term [197]. Since continuous kernels are not directly parameterized by weights  $\mathbf{W}$ , applying  $L_2$  regularization on the parameters of  $\varphi_{\text{Kernel}}$  directly would not have the intended effect. Consequently, in order to produce the same effect, we amend the  $L_2$  regularization term to penalize the *generated convolution kernel* instead. Given  $\varphi_{\text{Kernel}}^{:,l}$  the set of generated

kernel for all channels at a layer  $l$ ,  $\mathcal{L}_{\text{obj}}$  the objective loss function, and  $\lambda$  the regularizing parameter, we define a regularized loss as:

$$\mathcal{L} = \mathcal{L}_{\text{obj}} + \mathcal{L}_{\text{reg}} = \mathcal{L}_{\text{obj}} + \lambda \cdot \frac{1}{2} \sum_{l=1}^L \|\varphi_{\text{Kernel}}^{:,l}\|^2. \quad (3.3)$$

## 3.5 Experiments

We aim to define an architecture that can be applied regardless of the specific data characteristics. To this end, we construct a CCNN and validate it on sequential (1D), visual (2D) and point-cloud (3D) datasets –see Appx. B.2 for a detailed description of each dataset–. We show that the *same* CCNN obtains state-of-the-art results on several sequential tasks, competitive performance on image tasks, and surpasses the Perceiver on point-cloud processing. In addition, we show that learned CCNNs generalize from regular to irregular data by transferring a trained CCNN from voxels to point-clouds.

**Architecture specifications.** To study the scalability of our method, we create two CCNNs with differing numbers of hidden layers sizes: CCNN<sub>4,140</sub> (4 blocks, 140 channels, 200K params) and CCNN<sub>6,380</sub> (6 blocks, 380 channels, 2M params). The kernel network  $\varphi_{\text{Kernel}}$  is chosen to be a 3-layer MAGNet [319], with 32 hidden units for CCNN<sub>4,140</sub>, and 64 hidden units for the larger CCNN<sub>6,380</sub>. In each task, the input dimension of the kernel network  $\varphi_{\text{Kernel}}$  corresponds to the dimensionality of the data –1 for sequences, 2 for images, and 3 for point-clouds–. Additional details regarding hyperparameters, training regimes and experimental settings are given in Appx. B.3. An empirical assessment of the computational efficiency of our architectures is given in Appx B.4.1.

**CCNNs on sequential datasets.** In 1D we consider Sequential MNIST, Permuted MNIST [208], Sequential CIFAR10 [195], the Long Range Arena [379] and the Speech Commands dataset [421]. Our results (Tabs. 3.1, 3.2), demonstrate that CCNNs obtain state-of-the-art across several tasks, surpassing the performance of tailored architectures such as Recurrent Neural Networks and Transformers. The performance of CCNNs is explained by their ability to model long term dependencies with higher capabilities than tailored models. Interestingly, we observe that on tasks defined over flattened 2-dimensional signals, e.g., sMNIST, sCIFAR10, CCNNs learn to construct very large periodic kernels that model flattened local dependencies in 2D (Fig. B.3). This showcases the ability of CCNNs to model meaningful long range dependencies.

Additionally, we evaluate the capacity of CCNNs to classify speech [421] both from pre-possessed (Speech-MFCC) and raw signals (Speech-Raw). Our results (Tab. 3.1) demonstrate state-of-the-art results on both preprocessed and raw signals –of length 16000–, thus demonstrating the ability of CCNNs to process very heterogeneous data types with a unified architecture.

*Generalization across resolutions.* CCNNs are defined on a continuous space. Consequently, it is possible to train a CCNN at one resolution and deploy it at other resolutions –a

**Table 3.1:** Experimental results on sequence, image and point-cloud datasets.  $\times$  - *unable to apply, either due to the model’s inability to handle the specified data, or to computational complexity.*

|                       | SIZE  | SEQUENCE |          |          |             |            | IMAGE       |          |          | POINT-CLOUD |            |
|-----------------------|-------|----------|----------|----------|-------------|------------|-------------|----------|----------|-------------|------------|
|                       |       | sMNIST   | pMNIST   | sCIFAR10 | SPEECH-MFCC | SPEECH-RAW | SPEECH-0.5x | CIFAR10  | CIFAR100 | STL10       | ModelNet40 |
| Transformer           | 500K  | 98.90    | 97.90    | 62.20    | 90.75       | $\times$   | $\times$    | $\times$ | $\times$ | $\times$    | $\times$   |
| LSSL                  | 7.8M  | 99.53    | 98.76    | 84.65    | $\times$    | $\times$   | $\times$    | $\times$ | $\times$ | $\times$    | $\times$   |
| S4                    | 7.8M  | 99.63    | 98.70    | 91.13    | $\times$    | $\times$   | $\times$    | $\times$ | $\times$ | $\times$    | $\times$   |
| LSSL                  | 300k  | N.A.     | N.A.     | N.A.     | 93.58       | $\times$   | $\times$    | $\times$ | $\times$ | $\times$    | $\times$   |
| S4                    | 300k  | N.A.     | N.A.     | N.A.     | 93.96       | 98.32      | 96.30       | $\times$ | $\times$ | $\times$    | $\times$   |
| CKCNN-Seq             | 98k   | 99.31    | 98.00    | 62.25    | 95.30       | 71.66      | 65.96       | $\times$ | $\times$ | $\times$    | $\times$   |
| CKCNN-Seq-Big         | 1M    | 99.32    | 98.54    | 63.74    | $\times$    | $\times$   | $\times$    | $\times$ | $\times$ | $\times$    | $\times$   |
| FlexTCN-6             | 375k  | 99.62    | 98.63    | 80.82    | 97.67       | 91.73      | $\times$    | $\times$ | $\times$ | $\times$    | $\times$   |
| ResNet-44             | 660k  | $\times$ | $\times$ | $\times$ | $\times$    | $\times$   | $\times$    | 92.90    | 71.15    | NA          | $\times$   |
| ResNet-18             | 11.2M | $\times$ | $\times$ | $\times$ | $\times$    | $\times$   | $\times$    | 94.92    | 77.50    | 81.04       | $\times$   |
| ViT                   | 6.3M  | $\times$ | $\times$ | $\times$ | $\times$    | $\times$   | $\times$    | 90.92    | 66.54    | N.A.        | $\times$   |
| Swin-T/1              | 27.5M | $\times$ | $\times$ | $\times$ | $\times$    | $\times$   | $\times$    | 94.46    | 78.07    | N.A.        | $\times$   |
| Parabolic CNN         | 502k  | $\times$ | $\times$ | $\times$ | $\times$    | $\times$   | $\times$    | 88.5     | 64.8     | 77.0        | $\times$   |
| Hamiltonian CNN       | 264k  | $\times$ | $\times$ | $\times$ | $\times$    | $\times$   | $\times$    | 89.3     | 64.9     | 78.3        | $\times$   |
| CKCNN                 | 630k  | $\times$ | $\times$ | $\times$ | $\times$    | $\times$   | $\times$    | 86.8     | N.A.     | N.A.        | $\times$   |
| FlexNet-6             | 670k  | $\times$ | $\times$ | $\times$ | $\times$    | $\times$   | $\times$    | 92.2     | N.A.     | N.A.        | $\times$   |
| SpiderCNN             | N.A.  | $\times$ | $\times$ | $\times$ | $\times$    | $\times$   | $\times$    | 77.97    | N.A.     | N.A.        | 92.4       |
| PointConv             | N.A.  | $\times$ | $\times$ | $\times$ | $\times$    | $\times$   | $\times$    | 89.13    | N.A.     | N.A.        | 92.5       |
| DGCNN                 | 1.84M | $\times$ | $\times$ | $\times$ | $\times$    | $\times$   | $\times$    | $\times$ | $\times$ | $\times$    | 89.0       |
| PointNet              | 3.48M | $\times$ | $\times$ | $\times$ | $\times$    | $\times$   | $\times$    | $\times$ | $\times$ | $\times$    | 89.2       |
| PointNet++            | 1.99M | $\times$ | $\times$ | $\times$ | $\times$    | $\times$   | $\times$    | $\times$ | $\times$ | $\times$    | 90.0       |
| RepSurf-U             | 1.48M | $\times$ | $\times$ | $\times$ | $\times$    | $\times$   | $\times$    | $\times$ | $\times$ | $\times$    | 94.7       |
| CCNN <sub>4,140</sub> | 200k  | 99.72    | 98.82    | 90.30    | 95.01       | 98.34      | 96.22       | 92.78    | 66.86    | 81.80       | 84.44      |
| CCNN <sub>6,380</sub> | 2M    | 99.72    | 98.84    | 93.08    | 97.98       | 98.44      | 96.44       | 95.20    | 73.16    | 83.00       | 85.70      |

feat not achievable with discrete models [272, 319]–. To showcase this ability, we train CCNNs on Speech-Raw, and test them on a subsampled version of the dataset (Speech-0.5x). CCNNs not only generalize, but outperform existing models on zero-shot prediction over resolution changes.

**CCNNs on image datasets.** In 2D, we consider the CIFAR10, CIFAR100 [195] and the STL10 datasets [64]. CCNNs outperform existing continuous convolutional vision architectures [319, 321, 330, 384] and are competitive with existing large scale discrete models, while being (much) smaller (Tab. 3.1).

*The importance of modelling long range dependencies in ND.* In principle, all tasks can be treated as sequential tasks ignoring the ND structure –as done in S4 [129] for images due to the complexity of defining state spaces on 2D–, but this sacrifices structural information. Contrarily, CCNNs can be easily defined on multidimensional spaces simply by changing the dimension of the input coordinates of the kernel generator networks. We observe that by considering the 2D structure of the Image and Pathfinder tasks of the LRA benchmark, much better results can be obtained (Tab. 3.2, right). In Pathfinder with 2D images, the CCNN<sub>6,380</sub> obtains an accuracy of 96.00, outperforming the previous state-of-the-art by almost 10% points and performing remarkably better than on flattened images. Additionally, we observe that models trained on the original 2D data converge faster than their sequential counterparts (Fig. 3.9). Finally, we remark that discrete 2D CNNs with small convolutional kernels, e.g., ResNet-18 [143], are unable to solve Pathfinder due to the lack of fine-grained global context resulting from intermediate pooling layers. This was also seen by Gu et al. [129].

**CCNNs on point-cloud datasets.** An additional advantage that comes from the continuous nature of CCNNs, is that –contrary to discrete CNNs– CCNNs can be seamlessly

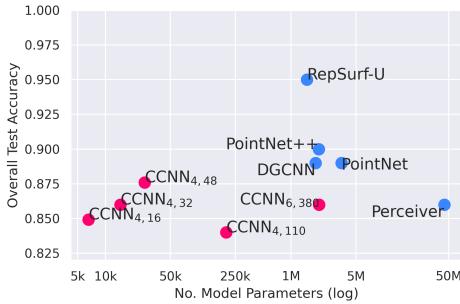
**Table 3.2:** Experimental results on the Long Range Arena benchmark.  $\times$  - unable to apply due to the model’s inability to handle the specified data.

|                       | LISTOPS      | TEXT         | IMAGE        | PATHFINDER   | AVG.         | 2DIMAGE      | 2DPATHFINDER |
|-----------------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|
| Transformer           | 36.37        | 64.27        | 42.44        | 71.40        | 53.66        | $\times$     | $\times$     |
| Reformer              | 37.27        | 56.10        | 38.07        | 68.50        | 50.56        | $\times$     | $\times$     |
| BigBird               | 36.05        | 64.02        | 40.83        | 74.87        | 57.17        | $\times$     | $\times$     |
| Linear Trans.         | 16.13        | 65.90        | 42.34        | 75.30        | 50.46        | $\times$     | $\times$     |
| Performer             | 18.01        | 65.40        | 42.77        | 77.05        | 51.18        | $\times$     | $\times$     |
| FNet                  | 35.33        | 65.11        | 38.67        | 77.80        | 54.52        | $\times$     | $\times$     |
| Nystromförmér         | 37.15        | 65.52        | 41.58        | 70.94        | 57.46        | $\times$     | $\times$     |
| Luna-256              | 37.25        | 64.57        | 47.38        | 77.72        | 59.37        | $\times$     | $\times$     |
| S4                    | <b>58.35</b> | 76.02        | 87.26        | 86.05        | <b>80.48</b> | $\times$     | $\times$     |
| CCNN <sub>4,140</sub> | 44.85        | 83.59        | 87.62        | 91.36        | 76.86        | 89.48        | 94.80        |
| CCNN <sub>6,380</sub> | 43.60        | <b>84.08</b> | <b>88.90</b> | <b>91.51</b> | 77.02        | <b>91.12</b> | <b>96.00</b> |

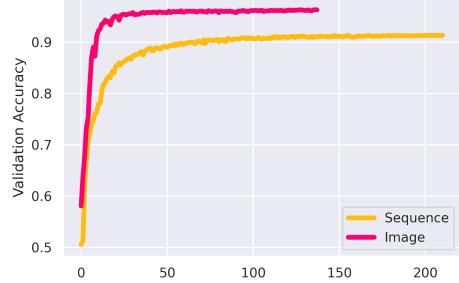
applied on irregular data, e.g., point-clouds. To assess the performance of CCNNs in 3D, we consider the ModelNet40 dataset [438], which consists of 3D meshes of objects often treated as point-clouds [436]. Our results show that CCNNs are able to achieve a decent overall classification accuracy in comparison to point-cloud specific architectures with a relatively low parameter count (200 K). In particular, CCNNs outperform the Perceiver [168] while being much smaller (Fig. 3.8).

*From regular to irregular data.* Interestingly, the continuous nature of CCNNs allows us to use CCNNs trained on regular data to process irregular data. We showcase this ability by training a CCNN<sub>4,140</sub> on a version of ModelNet10 [438] voxelized onto a grid of 40×40×40 voxels –Fig. B.2b shows an example of the point-cloud and voxelized datasets–. After trained, we deploy the CCNN on the original (point-cloud) and voxelized test sets and observe respective test accuracies of 90.99 and 90.64. This result shows that CCNNs learn representations that reflect the continuous nature of the data, showcasing CCNNs as a truly general-purpose architecture even able to generalize across different data representations –a feat hardly obtainable by existing architectures–.

*Large CCNNs overfit on point-clouds, but tiny CCNNs perform surprisingly well.* In contrast to 1D and 2D, we observe that larger CCNNs perform worse than smaller ones on 3D point-clouds (Fig. 3.8). We hypothesize that this is a result of the sparsity of the task. Continuous convolutional kernels define a kernel function over the entire domain even if the number of kernel values sampled is sparse. As a result, sparse supervision makes it hard for the network to learn a kernel function that generalizes to unseen data, as sparsity exposes the kernel function to aliasing –high frequencies between sampled points–. To corroborate this hypothesis, we train extremely small CCNNs –with 6K, 14K and 48K parameters– on ModelNet40. We observe that these models achieve surprising results and even outperform the 200K and 2M CCNNs (Fig. 3.8). These results supports our previous hypothesis, but also show that the continuous kernel paradigm can be used to construct extremely small, performant CNNs.



**Figure 3.8:** Parameter count versus performance on ModelNet40. Very small CCNN models obtain remarkable results.



**Figure 3.9:** Performance of the CCNN<sub>6,380</sub> on the LRA Image task, where samples are either input as sequence or with the original image structure.

### 3.6 Limitations and Future Work

**Computational efficiency.** Although convolutions with large convolutional kernels scale better than self-attention, e.g., Perceiver [168], they can still be expensive. Luckily, the Fourier convolution can be used to strongly reduce their cost, e.g., [129, 319, 321]. Nevertheless, the Fourier convolutions by themselves are not sufficient to scale CCNNs to very large inputs. This problem is exacerbated if irregular data is considered, as it (i) prevents the usage Fourier convolutions, and (ii) requires rendering a different convolutional kernel for each spatial position of each sample in the batch. An important avenue for future research may look into reducing computational requirements, either via separability or self-adjusting architectures, e.g., downsampling [308], for regular data, and “gridifying” techniques for irregular data.

**Larger models on point-cloud data.** Although CCNNs performs remarkably well on point-clouds with extremely small models, the current CCNN formulation is unable to achieve better performance with larger models. We hypothesize that this is due to increasing aliasing in the learned kernel generator networks. Although anti-aliasing techniques for kernel generator networks exist [319], this assume an underlying grid and a corresponding Nyquist frequency. An additional avenue for future research may look into the aliasing issue on irregular data either by generalizations of [319], or properly-defined smoothing techniques.

**Cross-modal training and data fusion.** CCNNs present a potential solution for cross-modal training –currently challenging due to dissimilar per-modality architectures–. Additionally, the continuous properties of CCNNs allow them to be trained on a wide range of data sources, e.g., multiple resolutions, regular, irregular data, etc, which is interesting for several machine learning application.

# 4

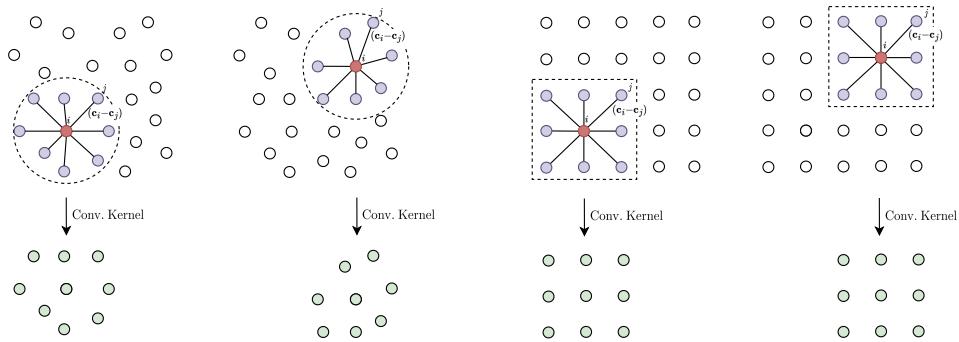
## Learnable Gridification for Efficient Point-Cloud Processing

*Based on the paper:  
Learnable Gridification for Efficient Point-Cloud Processing [394]*

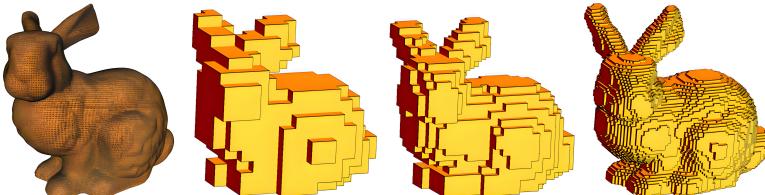
### 4.1 Introduction

Point clouds provide sparse geometric representations of objects or surfaces equipped with signals defined over their structure, e.g., the surface normals of an underlying object [297, 438] or the chemical properties of a molecule [302, 343]. Several neural operators have been developed that can be applied to such sparse representations provided by point clouds. These methods can be broadly understood as continuous generalizations of neural operators originally defined over regular discrete grids, e.g., convolution [436] and self-attention [472].

**The problem of learning on raw point clouds.** Unfortunately, the flexibility required from neural operators to accommodate irregular sparse representations like point clouds brings about important increases in time and memory consumption. This is especially prominent in neural operations that construct feature representations based on neighborhood information, e.g., convolution. In the case of point clouds, the irregular distances between points make these neural operations significantly more computationally demanding compared to regular grid representations like images or text. For instance, for convolution, the convolutional kernel needs to be recalculated for each point in a point cloud to account for irregular distances from the query point to other points in



**Figure 4.1:** Convolution on point clouds and grids. Due to the irregular nature of point clouds, convolutional kernels –and other operations based on neighborhood information– must be re-rendered for every query point in the point cloud (left). In contrast, grid data is regularly arranged, and thus pairwise distances are equal for any query point in the grid (right). As a result, the convolutional kernel can be computed once and reused for all query points.



**Figure 4.2:** Voxelization of the Stanford Bunny [388] for different resolutions [179].

its neighborhood (Fig. 4.1 left). In contrast, grid representations standardize pairwise distances following a grid structure (Fig. 4.1 right). As a result, the distances from a point to all other points in its neighborhood are fixed for all points queried in the grid. Therefore, it is possible to compute the kernel once, and reuse it across all query positions. This difference illustrates that operations relying on neighborhood information scale much worse in terms of memory and time for point clouds than for grid data, specially for large inputs and large neighborhoods.

**A potential solution: Voxelization.** A potential solution to address the challenges posed by point clouds lies in treating the point cloud as a continuous density that can be sampled on a dense regular grid: a process called *voxelization* [257, 438]. The idea of voxelization is to create a grid that overlaps with the domain of the point cloud (Fig. 4.2). Although voxelization methods create grid representations on which neural operations defined on grids can act, e.g., Conv3D, the grids resulting from voxelization are often-times much larger than the number of points in the original point cloud. This is a consequence of (*i*) the high-resolution grids required to describe fine details from the point cloud, and (*ii*) the low occupancy of the grid resulting from the sparse nature of point clouds which generally leads to many more points to process in the resulting grid than

in the original point cloud.

**Our proposed solution: Gridification.** In this paper, we propose an alternative solution to address the memory and computational scalability of point cloud methods by addressing its root cause: *the irregularity of the data*. We propose *learnable gridification* as the first step in a point cloud processing pipeline to transform the point cloud into a *compact, regular grid* (Fig. 4.3). Thanks to gridification, subsequent layers can use operations defined on grids, e.g., Conv3D, which scale much better than native point cloud methods. In a nutshell, gridification can be understood as a *convolutional message passing* layer acting on a *bipartite graph* that establishes connections between points in the point cloud to points in the grid given by a *bilateral k-nearest neighbor connectivity*. The proposed bilateral *k*-nearest neighbor connectivity guarantees that all points both in the point cloud and in the grid are connected, therefore allowing for the construction of expressive yet compact grid representations.

In contrast to voxelization, gridification produces expressive compact grid representations in which the number of points in the resulting compact regular grid is roughly equal to the number of points in the original point cloud, yet the grid is able to preserve fine geometric details from the original point cloud. For instance, we observe that point clouds with  $N=1000$  points can be effectively mapped to a compact dense  $10 \times 10 \times 10$  grid without significant information loss. We show through theoretical and empirical analysis that the resulting grid representations scale much better in terms of memory and time than native point cloud methods. This is verified on several comparison studies for increasing number of points in the point cloud and increasing neighborhood sizes in the construction of convolutional kernels.

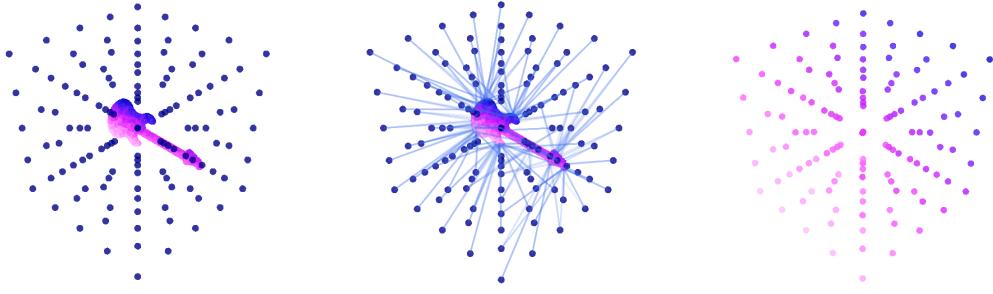
We demonstrate that gridification can also be used for tasks from point clouds to point clouds, e.g., segmentation. To this end, we introduce a *learnable de-gridification* step at the end of the point cloud processing pipeline, which can be seen as an inverted gridification step that maps the compact, regular grid back to its original point cloud form. This extension allows for the construction of *gridified networks* –networks that operate on grids– to solve global prediction tasks, e.g., classification, as well as dense prediction tasks, e.g., segmentation and regression, on point cloud data.

## 4.2 Method

### 4.2.1 Point cloud and grid representations

**Point cloud.** A point cloud  $\mathcal{P}=\{(\mathbf{c}_i^{\mathcal{P}}, \mathbf{x}_i^{\mathcal{P}})\}_{i=1}^{N_{\mathcal{P}}}$  is an *unstructured* set of  $N_{\mathcal{P}}$  pairs of coordinate-feature values  $(\mathbf{c}_i^{\mathcal{P}}, \mathbf{x}_i^{\mathcal{P}})$  scattered in space without any predefined pattern or connectivity. Point clouds sparsely represent geometric structures through pairs of coordinate vectors  $\mathbf{c}_i^{\mathcal{P}} \in \mathbb{R}^D$  and corresponding function values over that geometric structure  $\mathbf{x}_i^{\mathcal{P}} \in \mathbb{R}^{F_{\mathcal{P}}}$ , e.g., surface normals, RGB-values, electric potentials, etc.

**Grid.** A grid  $\mathcal{G}=\{(\mathbf{c}_i^{\mathcal{G}}, \mathbf{x}_i^{\mathcal{G}})\}_{i=1}^{N_{\mathcal{G}}}$  can be interpreted as a point cloud on which the coordinate-



**Figure 4.3:** Gridification. Gridification maps a point cloud  $\mathcal{P}$  onto a compact regular grid  $\mathcal{G}$ . The method first constructs a D-dimensional grid (left) that overlaps the point cloud. Then, it connects points on the point cloud to points in the grid given by a connectivity scheme  $\mathcal{E}_{\mathcal{P} \rightarrow \mathcal{G}}$ , i.e., a set of edges from points in the point cloud to points in the grid, determined by *bilateral k-nearest neighbors connectivity* (middle). Finally, gridification propagates information from the point cloud onto the grid through a *convolutional message passing* layer acting over the bipartite graph  $(\mathcal{P}, \mathcal{G}, \mathcal{E}_{\mathcal{P} \rightarrow \mathcal{G}})$ . By carefully selecting the different components of the gridification module, gridification is able to construct compact rich grid representations that can be subsequently processed with grid operations such as Conv3D.

feature pairs  $(\mathbf{c}_i^{\mathcal{G}}, \mathbf{x}_i^{\mathcal{G}})$  are arranged in a regular pattern that form a lattice. In contrast to general point clouds, points in a grid are evenly spaced and align along predefined axes, e.g.,  $x, y, z$ . The regular spacing between points leads to regular pairwise distances for all query points in the grid. As a result, we can calculate pairwise attributes once, and reuse them for all query points.

#### 4.2.2 Gridification: From a point cloud to a dense grid

We seek to map the sparse point cloud  $\mathcal{P} = \{(\mathbf{c}_i^{\mathcal{P}}, \mathbf{x}_i^{\mathcal{P}})\}_{i=1}^{N_{\mathcal{P}}}$  onto a compact regular grid  $\mathcal{G} = \{(\mathbf{c}_i^{\mathcal{G}}, \mathbf{x}_i^{\mathcal{G}})\}_{i=1}^{N_{\mathcal{G}}}$  in  $\mathbb{R}^D$ . We formalize this process as an operation over a *bipartite graph* that establishes connections between points in the point cloud  $\mathcal{P}$  to points in the grid  $\mathcal{G}$  given by a *connectivity scheme*  $\mathcal{E}_{\mathcal{P} \rightarrow \mathcal{G}}$  defined as a set of edges  $\mathbf{e}_{j \rightarrow i} \in \mathcal{E}_{\mathcal{P} \rightarrow \mathcal{G}}$ .

**Learnable gridification as message passing.** We aim to learn a mapping from  $\mathcal{P}$  to  $\mathcal{G}$  such that the grid representation  $\mathcal{G} = \{(\mathbf{c}_i^{\mathcal{G}}, \mathbf{x}_i^{\mathcal{G}})\}_{i=1}^{N_{\mathcal{G}}}$ ,  $\mathbf{c}_i \in \mathbb{R}^D$ ,  $\mathbf{x}_i^{\mathcal{G}} \in \mathbb{R}^{F_{\mathcal{G}}}$ , adequately represents the source point cloud  $\mathcal{P}$  for the downstream task. Given a source point cloud  $\mathcal{P}$ , a target grid  $\mathcal{G}$  and a connectivity scheme  $\mathcal{E}_{\mathcal{P} \rightarrow \mathcal{G}}$ , we define gridification as a *convolutional message passing* layer [117] on the bipartite graph  $(\mathcal{P}, \mathcal{G}, \mathcal{E}_{\mathcal{P} \rightarrow \mathcal{G}})$  defined as:

$$\mathbf{x}_i^{\mathcal{G}} = \phi_{\text{upd}} \left( \bigoplus_{\mathbf{e}_{j \rightarrow i} \in \mathcal{E}_{\mathcal{P} \rightarrow \mathcal{G}}} \phi_{\text{msg}} \left( \phi_{\text{node}} \left( \mathbf{x}_j^{\mathcal{P}} \right), \phi_{\text{pos}} \left( \mathbf{c}_i^{\mathcal{G}} - \mathbf{c}_j^{\mathcal{P}} \right) \right) \right). \quad (4.1)$$

It consists of a node embedding network  $\phi_{\text{node}} : \mathbb{R}^{F_{\mathcal{P}}} \rightarrow \mathbb{R}^H$  that processes the point

cloud features  $\mathbf{x}_i^G$ , a positional embedding network  $\phi_{\text{pos}} : \mathbb{R}^D \rightarrow \mathbb{R}^H$  that creates feature representations based on the pairwise distances between coordinates in  $G$  and  $\mathcal{P}$  – thus resembling a convolutional kernel–, a message embedding network  $\phi_{\text{msg}} : \mathbb{R}^{2H} \rightarrow \mathbb{R}^H$  that receives both the node embedding and the relative position embedding to create the so-called *message*. After the messages are created for all nodes described by connectivity of the node, these features are aggregated via the aggregation function  $\oplus$ , e.g., max, mean. Finally, the aggregated message is passed through the update network  $\phi_{\text{upd}} : \mathbb{R}^H \rightarrow \mathbb{R}^{F_G}$  to produce the grid feature representations  $\mathbf{x}_i^G \in \mathbb{R}^{F_{\mathcal{P}}}$ .

### 4.2.3 De-gridification: From a dense grid to a point cloud

To extend the use of gridification to tasks from the point cloud  $\mathcal{P}$  to the point cloud  $G$ , e.g., segmentation, regression, we define a *de-gridification* step that sends a grid representation  $G$  back to its original point cloud form  $\mathcal{P}$ . Formally, the de-gridification step is defined as:

$$\mathbf{x}_i^{\mathcal{P}} = \phi_{\text{upd}} \left( \bigoplus_{e_{j \rightarrow i} \in \mathcal{E}_{G \rightarrow \mathcal{P}}} \phi_{\text{msg}} \left( \phi_{\text{node}} \left( \mathbf{x}_j^G \right), \phi_{\text{pos}} \left( \mathbf{c}_i^{\mathcal{P}} - \mathbf{c}_j^G \right) \right) \right). \quad (4.2)$$

Intuitively, de-gridification can be interpreted as a gridification step from  $G$  to  $\mathcal{P}$  given by an inverted connectivity scheme  $\mathcal{E}_{G \rightarrow \mathcal{P}} = (\mathcal{E}_{\mathcal{P} \rightarrow G})^{-1}$ . Note that, it is not necessary to calculate the connectivity scheme for the de-gridification step. Instead, we can obtain it simply by taking the connectivity scheme from the gridification step  $\mathcal{E}_{G \rightarrow \mathcal{P}}$  and inverting the output and input nodes of the edges.

### 4.2.4 Requirements and properties of gridification

We desire to construct a compute and memory efficient grid representation  $G$  that captures all aspects of the point cloud  $\mathcal{P}$  as good as possible. That is, a compact, yet rich grid representation  $G$  that preserves the structure of the point cloud  $\mathcal{P}$  with as low loss of information as possible. With this goal in mind, we identify the following requirements:

- (i) The number of points in the grid  $N_G$  should be at least as large as the number of points in the point cloud  $N_{\mathcal{P}}$ .
- (ii) The width of all hidden representations of the node embedding network  $\phi_{\text{node}}$  should be *at least as large* as the width of the point cloud features  $\mathbf{x}_i^{\mathcal{P}}$ , i.e.,  $F_{\mathcal{P}}$ .
- (iii) The width of all hidden representations of the position embedding network  $\phi_{\text{pos}}$  should be at least as large as the dimension of the domain  $D$ .
- (iv) The width of all hidden representations of the embedding networks  $\phi_{\text{upd}}$ ,  $\phi_{\text{msg}}$  should be *at least as large* as the width of the point cloud features  $\mathbf{x}_i^{\mathcal{P}}$  plus the dimension of the domain  $D$ .
- (v) Each point-cloud point  $\mathbf{c}^{\mathcal{P}}$  should be connected to *at least* one point  $\mathbf{c}^G$  in the grid.
- (vi) The positional embedding network  $\phi_{\text{pos}}$  should describe high frequencies.

(vii) Each grid point  $\mathbf{c}^G$  should be connected to *at least* one point  $\mathbf{c}^P$  in the point cloud.

**Preventing information loss.** To prevent information loss, we want to avoid any kind of compression either in the grid representation or in any intermediary representation during the gridification process. Consequently, we restrict the number of points as well as the width of all representations to be at least as big as the corresponding dimensions in the source point cloud  $P$  –items (i)–(iv)–. In addition, we must make sure that all points in the point cloud are connected to points in the grid to prevent points from being disregarded during gridification –item (v)–. Finally, we must also make sure that the positional embedding network  $\phi_{\text{pos}}$  is able to represent high frequencies –item (vi)–. This is important as multilayer perceptrons (MLPs) with piecewise nonlinearities, e.g., ReLU, have been shown to have an implicit bias towards smooth functions [358, 377]. In the context of gridification, this means that using conventional MLPs for the positional embedding network  $\phi_{\text{pos}}$  could result in over-smooth grid representations unable to represent fine details from the source point cloud. We circumvent this issue by using parameterizations for  $\phi_{\text{pos}}$  able to model high frequencies (Sec. 4.2.5).

**Encouraging compact representations.** In addition to encouraging no information loss, we also identify requirements that encourage the resulting grid representation to be compact and expressive. First, we note that item (v) is important for this end as well, as over-smooth representations implicitly require higher resolutions to be able to encode fine-grained details. Additionally, we impose all points in the grid to be connected to points in the point cloud –item (vii)– to prevent the grid representation from having low occupancy. This restriction allows us to make sure that all the spatial capacity of the grid is being used. This in turn allows us to construct compact rich grid representations.

#### 4.2.5 Materializing the gridification module

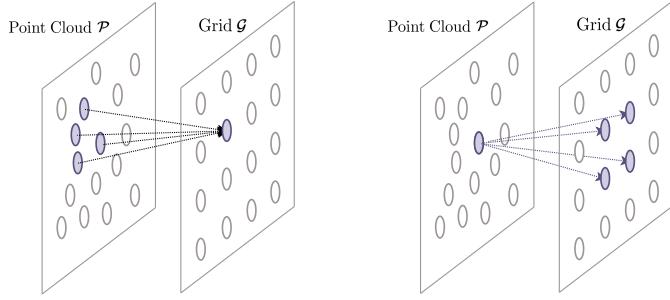
Based on the previous requirements and properties, we define the components of the gridification module as follows:

##### The grid $G$

Let  $[a, b]^D$  be the domain of the point cloud  $P$ , i.e.,  $\mathbf{c}_i^P \in [a, b]^D, \forall \mathbf{c}_i^P \in P$ . Then, we define the regular grid  $G$  over the same domain  $[a, b]^D$  with  $\sqrt[D]{N^G}$  points along each dimension. By doing so, we guarantee that the grid  $G$  is uniformly spaced over the domain of the point cloud, therefore (i) preserving the statistics of the input point cloud, and (ii) being able to represent the underlying signal in the same range. In practice, point clouds are normalized during preprocessing. As a result, we often have that  $a = -1, b = 1$ , leading to a point cloud and a grid defined on  $[-1, 1]^D$ .

##### The connectivity scheme $\mathcal{E}_{P \rightarrow G}$

Motivated by the requirements in Sec. 4.2.4, we opt for *bilateral k-nearest neighbor connectivity* over common alternatives such as radius connectivity [297, 298] or one-way



**Figure 4.4:** Bilateral  $k$ -nearest neighbor connectivity for  $k=4$ .

$k$ -nearest neighbor connectivity [14, 71] for the construction of the connectivity scheme  $\mathcal{E}_{\mathcal{P} \rightarrow \mathcal{G}}$  to guarantee that no points either in the grid  $\mathcal{G}$  nor the point cloud  $\mathcal{P}$  are disconnected. Bilateral  $k$ -nearest neighbor connectivity consists of a two-way  $k$ -nearest neighbor approach in which first each point  $\mathbf{c}_i^{\mathcal{G}}$  in the grid is linked to the  $k$  nearest points  $\mathbf{c}_j^{\mathcal{P}}$  in the point-cloud. Subsequently, connections are established from each point  $\mathbf{c}_i^{\mathcal{P}}$  in the point cloud to its nearest  $k$  points  $\mathbf{c}_j^{\mathcal{G}}$  in the grid (Fig. 4.4). By following this procedure, bilateral  $k$ -nearest neighbor connectivity creates a *complete* connectivity scheme from  $\mathcal{P}$  to  $\mathcal{G}$  with at least  $k$  and at most  $2k$  edges per point.

#### The positional embedding network $\phi_{\text{pos}}$

In literature, the positional embedding network  $\phi_{\text{pos}}$  is often parameterized as an MLP with piecewise nonlinearities, e.g., ReLU, that receives relative positions  $(\mathbf{c}_i - \mathbf{c}_j)$  as input and retrieves the value of an spatial function at that position  $\phi_{\text{pos}}(\mathbf{c}_i - \mathbf{c}_j)$  [298, 343, 436]. However, previous studies have shown that MLPs with piecewise nonlinearities suffer from an spectral bias towards low frequencies, which limits their ability to represent functions with high frequencies [358, 377]. In the context of modelling spatial neural operators such as  $\phi_{\text{pos}}$ , this implies that using piecewise MLPs to parameterize spatial neural operators leads to inherently smooth operators. Consequently, applying such an operator over an input function, e.g., via a convolution operation, would implicitly perform a low-pass filtering of the input, causing the output representations to lack information regarding fine-grained details of the input.

To overcome this issue, we rely on the insights from *Continuous Kernel Convolutions* [321] and parameterize the positional embedding network as a *Neural Field* [358, 377]. In contrast to piecewise MLPs, neural fields easily model high frequencies, and thus allow for powerful parameterizations of spatial neural operators that do not perform smoothing. In the context of gridification, using neural fields to parameterize  $\phi_{\text{pos}}$  allows gridification to project fine-grained geometric information from the point cloud onto the grid.

#### 4.2.6 Gridified networks for global and dense prediction

Gridification and de-gridification allow for the construction of *gridified networks* able to

process point clouds both for global and dense prediction tasks (Fig. 4.5). For global prediction tasks, e.g., classification, we construct a point cloud processing pipeline consisting of gridification, followed by a *grid network*, i.e., a neural network that operates on grid data, designed for global prediction, e.g., a ResNet [143] or a ViT [191]. For dense prediction tasks, e.g., segmentation, our proposed point cloud pipeline consists of gridification, followed by a grid network designed for dense predictions, e.g., a U-Net [323] or a CCNN [191]. After the processed grid representation is obtained, we utilize the de-gridification step to map back the grid representation to a point cloud with the output node predictions.

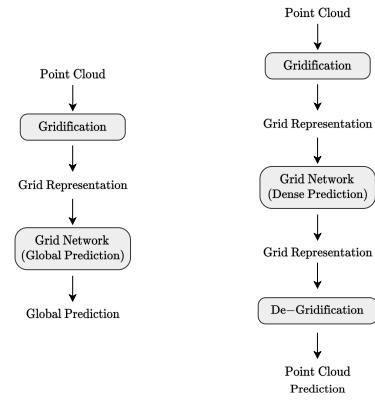
### 4.3 Related Work

Deep learning approaches for point cloud processing can be broadly classified in two main categories: (i) native point cloud methods and (ii) voxelization methods.

**Native point cloud methods.** Native point cloud methods operate directly on the raw, irregular point cloud data without any preprocessing steps such as voxelization. These methods leverage the inherent spatial distribution of the points to extract meaningful features. PointNet [297] introduced a pioneering framework for point cloud processing by employing shared multilayer perceptrons and symmetric functions to learn global and local features from unordered point sets. PointNet++ [298] extended this work with hierarchical neural networks to capture hierarchical structures in point clouds. PointConv [436] introduced a convolution operation specifically designed for point clouds, incorporating local coordinate systems to capture local geometric structures. PointGNN [353] utilized graph neural networks to model interactions between neighboring points.

Despite the flexibility in handling irregular data that native point cloud methods provide, they suffer from scalability issues due to the increased computational and memory complexity of processing unstructured point sets.

**Voxelization methods.** Voxelization methods aim to convert the irregular point cloud data into a regular grid structure, enabling the utilization of neural architectures designed for regular grid data. VoxNet [257] introduced the concept of voxelization for point clouds and employed 3D convolutions on the resulting grid representations. Volumetric CNN [296] extended this approach with an occupancy grid representation and achieved impressive performance on 3D shape classification tasks. Other works, such as



**Figure 4.5:** Point cloud pipeline for global (left) and dense prediction tasks (right).

VoxSegNet [420] explore variations of voxelization techniques to improve performance on tasks like object detection and segmentation.

While voxelization methods offer a well-founded solution to the computational and memory complexity of native point cloud methods, in practice, they suffer from high memory consumption and information loss due to the discretization process. This is due to the inherent trade-off between the need to capture fine geometric details –which requires high resolution grids–, and the need for efficiency –which favors low resolution grids–. As a result, conventional voxelization methods struggle to strike a balance between resolution and speed. In contrast, gridification is able to generate compact yet expressive grid representations able to preserve fine geometric details on a low resolution grid with roughly the same number of points as the source point cloud.

**Hybrid methods.** Aside from pure point cloud and voxelization methods, there exist works that attempt combine the advantages of both categories. Their main idea is to combine point-wise and grid-wise operations to perform effective feature extraction while maintaining scalability and efficiency. PointGrid [204] uses a hybrid representation by voxelizing the point cloud and employing a combination of point-wise and grid-wise operations at each layer. Point-Voxel CNN [235] combines grid convolutions with point-wise feature extraction. It uses low-resolution voxelization to aggregate neighborhoods with regular 3D convolutions and MLPs to generate point-wise features that preserve fine-grained structure. These features are then fused through interpolation. Point-Voxel Transformer [462] follows a similar two-branch structure, but replaces convolutions with windowed self-attention.

Although hybrid methods reduce the computational and memory complexity of native point cloud methods, their explicit use of voxelization still leads to a trade-off between information loss and efficiency on that branch. To compensate for the information lost during voxelization, they require a parallel raw point cloud branch, which does not scale well. In contrast, gridification does not make use of raw point cloud branches but instead focuses on the creation of descriptive compact grid representations that preserve the geometric information of the source point cloud. Hence, gridification offers a solution with better scalability properties than existing hybrid methods.

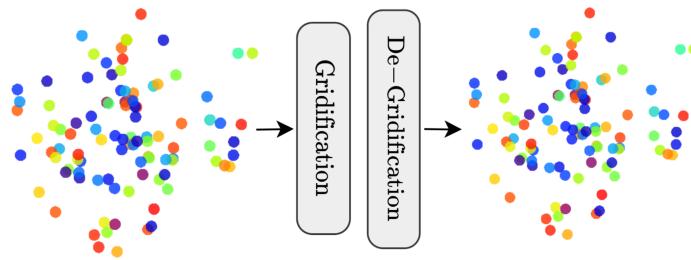
## 4.4 Experiments

To evaluate our approach, we first analyze the expressive capacity of gridification and de-gridification on a toy point cloud reconstruction task. Next, we construct gridified networks and evaluate them on classification and segmentation tasks. In addition, we provide empirical analyses on the computational and memory complexity of gridified networks which we then corroborate with theoretical analyses.

**Experimental setup.** For the position embedding function  $\phi_{\text{pos}}$  we use an Random Fourier Feature Network [377], due to explicit control over the smoothness through the

**Table 4.1:** Classification performance on ModelNet40 benchmark.

| MODEL                           | INPUT                                      | TYPE         | ACCURACY | PARAMETERS |
|---------------------------------|--|--------------|----------|------------|
| PointNet++ [298]                | $32 \times 1000$                           | native       | 89.64    | 1.5M       |
| VoxNet [257]                    | $32 \times 30^3$                           | voxelization | 83.00    | 0.92M      |
| PointGrid [204]                 | $32 \times 16^3$                           | voxelization | 92.00    | -          |
| Point Voxel Transformer [462]   | $32 \times 1024$                           | hybrid       | 94.00    | 2.76M      |
| Gridified Networks 3x3x3 (Ours) | $32 \times 1000 \rightarrow 32 \times 3^3$ | voxelization | 90.86    | 0.28M      |
| Gridified Networks 9x9x9 (Ours) | $32 \times 1000 \rightarrow 32 \times 9^3$ | voxelization | 92.28    | 0.47M      |

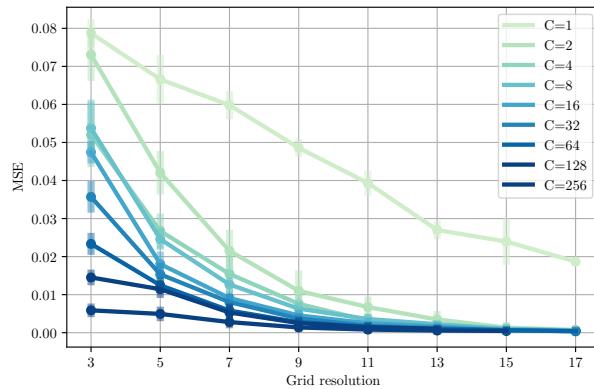
**Figure 4.6:** Random point clouds with random scalar node features are mapped to a grid representation and back via (de)-gridification.

initial frequency parameter  $\Omega$ . The practical setup and instantiation of the convolution blocks can be found in Appendix C.1. We train our models without data augmentation using AdamW [239] and a cosine scheduler [238] with 10 epochs of linear warm-up. We follow the standard procedure and preprocess all objects in the datasets to be centered and normalized. For each dataset, we choose the grid resolution such that its number of points is roughly equal to the size of the original point cloud. For ModelNet40 we use surface normals in addition to positions as node features. Dataset specific hyperparameters can be found in Appendix C.2.

#### 4.4.1 Random point cloud reconstruction

First, we evaluate the expressivity of our proposed gridification and de-gridification procedure. To this end, we construct a dataset with 1000 synthetic random graphs –800 for training and 200 for validation– consisting of a predefined number of nodes  $N^\mathcal{P}=1000$  randomly sampled on the unit cube, i.e.,  $c_i^\mathcal{P} \sim \mathcal{U}([-1, 1]^3)$ , accompanied with a random scalar feature  $f_i^\mathcal{P} \sim \mathcal{U}(-1, 1)$  at each position.

**Experimental setup.** To evaluate the expressiveness of our method, we set up a network consisting only of a gridification and a de-gridification step, i.e., no intermediary layers, in a point cloud reconstruction pipeline. In other words, the task consists of propagating the point cloud into a grid representation, and mapping the grid representation back to the original point cloud (see Fig. 4.6). Therefore, to successfully reconstruct the original point cloud from the grid representation, the grid representation must be able to retain sufficient information from the input point cloud.



**Figure 4.7:** Random point cloud reconstruction error for varying grid resolution and number of channels on the grid representation.

**Results.** Fig. 4.7 shows reconstruction errors for different resolutions and different number of channels in the intermediary grid representation. We observe that it is possible to obtain good reconstructions by increasing the resolution of the grid or its number of channels. From an efficiency perspective, it is preferred to utilize low resolution representations with a larger number of channels due to the exponential growth in computational demands associated with higher grid resolutions, which instead scale linearly with the number of channels of the representation. Our experiments show that gridification is able to obtain compact grid representations that preserve the structure of the input point cloud. Furthermore, the quality of the grid representations can be efficiently improved by scaling the number of channels used.

#### 4.4.2 ModelNet40 classification

Next, we evaluate gridification on point cloud classification. We deploy gridified networks on ModelNet40 [438]: a synthetic dataset for 3D shape classification, consisting of 12,311 3D meshes of objects belonging to 40 classes. ModelNet40 is broadly used as a point cloud benchmark in which points are uniformly sampled from meshes.

**Results.** Our results (Tab. 4.1) show that gridified networks achieve competitive performance while being significantly more efficient in terms of parameters, compute and memory. Interestingly, and in contrast to voxelization methods, we observe that gridified networks operate well even on extremely low resolution grids. For instance, on a  $3 \times 3 \times 3$  grid, gridified networks attain an accuracy of 90.86%.

#### 4.4.3 ShapeNet part segmentation

Next, we evaluate gridification on point cloud segmentation. To this end, we deploy gridified networks on ShapeNet [452]: a synthetic dataset with 16,000 point clouds of objects from 16 categories, each of which contains 2 to 6 parts. The objective of the task

**Table 4.2:** Segmentation performance on ShapeNet-part benchmark.

| MODEL                | Gridified Networks | PointNet++ | PointGrid |
|----------------------|--------------------|------------|-----------|
| TYPE                 | voxelization       | native     | hybrid    |
| instance average IoU | 87.07              | 85.1       | 86.4      |
| class average IoU    | 81.68              | 81.9       | 82.2      |
| airplane             | 88.52              | 82.4       | 85.7      |
| bag                  | 86.54              | 79.0       | 82.5      |
| cap                  | 74.09              | 87.7       | 81.8      |
| car                  | 80.46              | 77.3       | 77.9      |
| chair                | 91.44              | 90.8       | 92.1      |
| earphone             | 51.81              | 71.8       | 82.4      |
| guitar               | 92.61              | 91.0       | 92.7      |
| knife                | 89.44              | 85.9       | 85.8      |
| lamp                 | 82.07              | 83.7       | 84.2      |
| laptop               | 96.07              | 95.3       | 95.3      |
| motor                | 65.36              | 71.6       | 65.2      |
| mug                  | 92.99              | 94.1       | 93.4      |
| pistol               | 86.72              | 81.3       | 81.7      |
| rocket               | 58.57              | 58.7       | 56.9      |
| skateboard           | 75.70              | 76.4       | 73.5      |
| table                | 85.66              | 82.6       | 84.6      |

is to segment the point clouds into one of 50 possible part annotations.

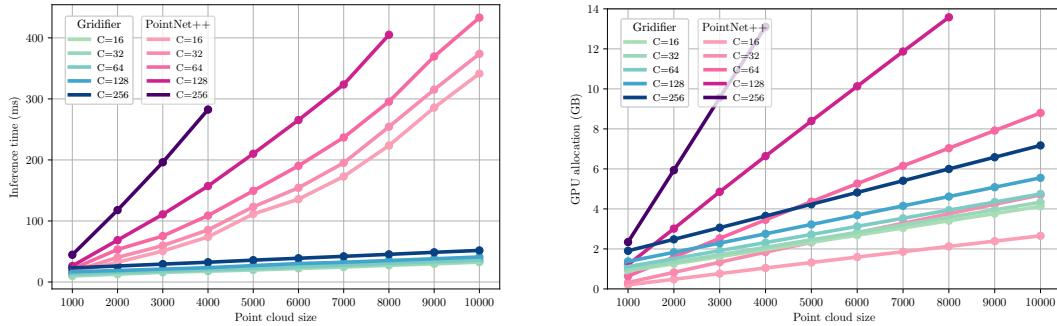
**Results.** Our results (Tab. 4.2) demonstrate that gridified networks are also able to achieve competitive performance in segmentation tasks, while being significantly more efficient in terms of parameters, compute and memory. This result validates the ability of gridification to handle dense prediction tasks via gridification and de-gridification.

#### 4.4.4 Efficiency analysis of gridification

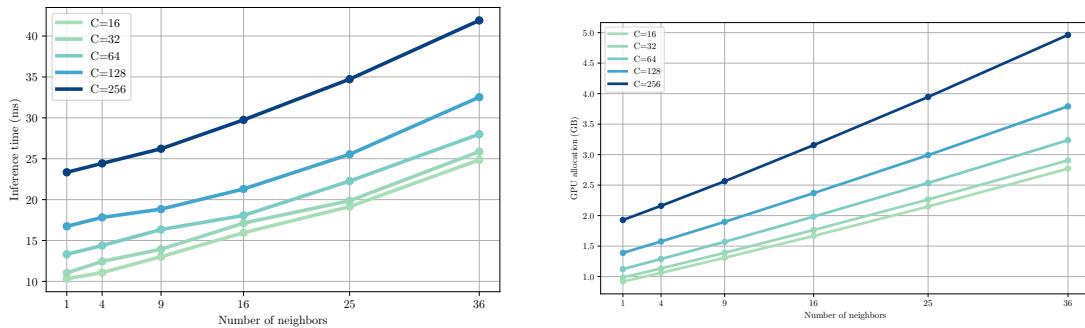
Finally, we investigate the scalability properties of gridification. Specifically, we analyze the time and memory consumption of gridified networks during inference on ModelNet40 for point clouds with increasing size, and compare the computation and memory complexity of convolutional operations on grid and point cloud data.

**Scaling gridified networks to large point clouds.** Fig 4.8 shows the average time and memory consumption during inference on ModelNet40 for gridified networks and PointNet++. We observe that gridified networks exhibit a much more favorable scalability both in terms of inference time and GPU allocation –linear vs. quadratic– as the input size and number of channels increase. This demonstrates that gridified networks scale much better than native point cloud methods both for larger point clouds and networks.

**Scaling the receptive field of neural operations.** Furthermore, we analyze the scalability properties of gridified networks relative to the size of its receptive fields. As illustrated in Fig. 4.1), for native point cloud methods the convolutional kernel must be recomputed for all query points in the point cloud. As a consequence, the construction of the convolutional kernels of size K for all query points in a point cloud with N points



**Figure 4.8:** Average time (left) and GPU allocation (right) during inference on ModelNet40 for a batch size of 32.



**Figure 4.9:** Average time (left) and GPU allocation (right) on ModelNet40 validation set per batch  $B = 32$  for various number of neighbors and number of channels  $C$  on the grid representation.

incurs in  $\mathcal{O}(KD)$  memory and time complexity. In contrast, on grid data, we can compute the kernel once and reuse it at all positions. As a result, on a grid, this operation incurs in  $\mathcal{O}(D)$  time and memory complexity.

Fig. 4.9 show the methods' potential to scale up the receptive field of the gridification module without introducing significant computational overhead.

## 4.5 Limitations and future work

**The resolution of gridification depends on the size of the point cloud.** The main limitation of gridification is that the resolution on the grid is directly proportional to the size of point cloud in order to preserve information. This in turn means that the whole gridified architecture must be changed for point clouds of different sizes, even if they represent the same underlying signal. This is in contrast to native point cloud methods, which, due to their continuous nature, are, in principle, able to generalize to point clouds of different sizes as long as these exhibit the same structures.

**Towards no information loss.** While gridification aims to produce compact grid repre-

sentations with minimal information loss, our experiments reveal that some information still gets lost in the process. Loosely speaking, it should be possible to create grid representations that do not lose any information by ensuring that the grid representation has at least as many points and as many channels as the source point cloud representation. Gaining richer theoretical understanding of gridification, could therefore lead to grid representations with no information loss either by imposing other requirements on gridification, or by considering different functional families in the gridification process.

**Large scale point clouds and global context.** While we verify the scalability and efficiency of gridified networks for increasing point cloud sizes, we only carry on experiments on relatively small datasets. In future work, we aim to deploy gridification to large scale datasets. Furthermore, recent works have shown that using global receptive fields in convolutional operations consistently leads to better results across several tasks, even outperforming well-established Transformer architectures [129, 191, 292]. Due to the computational complexity of native point cloud methods, networks with global context have not been explored for point cloud processing. With gridification this ability becomes computationally feasible. Exploring the effect of global context for point cloud processing is an exciting research direction.

**Generative tasks.** Gridification opens up the possibility of performing scalable generative tasks on large point clouds. Gridification can directly be extended to generative tasks if we assume that the point-cloud structure is preserved, i.e., if the coordinates of the output and input point clouds are equal. If this is not the case, e.g., for the generation of molecules [153, 447], de-gridification module must be modified to predict both the features and positions of the new point cloud.

**Equivariant gridification.** In its current form, gridified networks do not respect symmetries which might be important for some applications, e.g., equivariance to 3D rotations for the prediction and generation of molecules [153, 343]. In future work, we aim to extend gridification to respect these symmetries by taking inspiration from equivariant graph neural networks [111, 336]. It is important to note that not only gridification and de-gridification must be equivariant, but also that the grid operations in between. This can be achieved in an efficient yet expressive manner through the use of continuous Monte-Carlo convolutions on the regular representations of the group [105, 316].

## 4.6 Conclusion

This work presents gridification, a method that strongly reduces the computational requirements of point cloud processing pipelines by mapping input point clouds to a grid representation, and performing neural operations in there. We demonstrate that gridified networks are able to match the accuracy of native point cloud methods, while being much faster and memory efficient. Through empirical and theoretical analyses, we also show that gridified networks scale much more favorably than native point cloud methods to larger point clouds and larger neighborhoods.

# 5

## Continuous Kernel Convolutions with Differentiable Kernel Sizes

*Based on the paper:*

*FlexConv: Continuous Kernel Convolutions with Differentiable Kernel Sizes* [319]

### 5.1 Introduction

The kernel size of a convolutional layer defines the region from which features are computed, and is a crucial choice in their design. Commonly, small kernels (up to 7px) are used almost exclusively and are combined with pooling to model long term dependencies [143, 356, 372, 375]. Recent works indicate, however, that CNNs benefit from using convolutional kernels (*i*) of varying size at different layers [291, 384], and (*ii*) at the same resolution of the data [72, 286, 321]. Unfortunately, most CNNs represent convolutional kernels as tensors of discrete weights and their size must be fixed prior to training. This makes exploring different kernel sizes at different layers difficult and time-consuming due to (*i*) the large search space, and (*ii*) the large number of weights required to construct large kernels.

A more efficient way to tune different kernel sizes at different layers is to *learn* them during training. Existing methods define a *discrete* weighted set of basis functions, e.g., shifted Delta-Diracs (Fig. 5.2b, Dai et al. [74]) or Gaussian functions (Fig. 5.2c, Jacobsen et al. [167], Pintea et al. [291], Shelhamer et al. [349]). During training they learn dilation factors over the basis functions to increase the kernel size, which crucially limits the bandwidth of the resulting kernels.

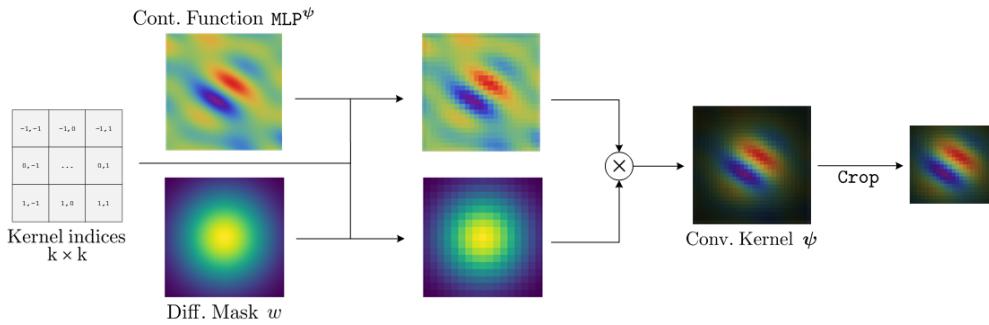
In this work, we present the *Flexible Size Continuous Kernel Convolution* (FlexConv), a convolutional layer able to learn *high bandwidth* convolutional kernels of varying size during training (Fig. 5.1). Instead of using discrete weights, we provide a *continuous parameterization* of convolutional kernels via a small neural network [321]. This parameterization allows us to model continuous functions of arbitrary size with a fixed number of parameters. By multiplying the response of the neural network with a Gaussian mask, the size of the kernel can be learned during training (Fig. 5.2a). This allows us to produce detailed kernels of small sizes (Fig. 5.3), and tune kernel sizes efficiently.

FlexConvs can be deployed at higher resolutions than those observed during training, simply by using a more densely sampled grid of kernel indices. However, the high bandwidth of the kernel can lead FlexConv to learn kernels that show aliasing at higher resolutions, if the kernel bandwidth exceeds the Nyquist frequency. To solve this problem, we propose to parameterize convolutional kernels as *Multiplicative Anisotropic Gabor Networks* (MAGNets). MAGNets are a new class of Multiplicative Filter Networks [104] that allows us to analyze and control the frequency spectrum of the generated kernels. We use this analysis to regularize FlexConv against aliasing. With this regularization, FlexConvs can be directly deployed at higher resolutions with minimal accuracy loss. Furthermore, MAGNets provide higher descriptive power and faster convergence speed than existing continuous kernel parameterizations [105, 321, 343]. This leads to important improvements in classification accuracy (Sec. 5.4).

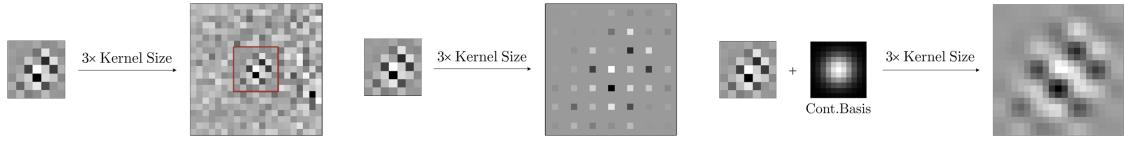
Our experiments show that CNNs with FlexConvs, coined *FlexNets*, achieve state-of-the-art across several sequential datasets, match performance of recent works with learnable kernel sizes with less compute, and are competitive with much deeper ResNets [143] when applied on image benchmark datasets. Thanks to the ability of FlexConvs to generalize across resolutions, FlexNets can be efficiently trained at low-resolution to save compute, e.g.,  $16 \times 16$  CIFAR images, and be deployed on the original data resolution with marginal accuracy loss, e.g.,  $32 \times 32$  CIFAR images.

In summary, our **contributions** are:

- We introduce the *Flexible Size Continuous Kernel Convolution* (FlexConv), a convolution operation able to learn *high bandwidth* convolutional kernels of varying size end-to-end.
- Our proposed *Multiplicative Anisotropic Gabor Networks* (MAGNets) allow for analytic control of the properties of the generated kernels. This property allows us to construct analytic alias-free convolutional kernels that generalize to higher resolutions, and to train FlexNets at low resolution and deploy them at higher resolutions. Moreover, MAGNets show higher descriptive power and faster convergence speed than existing kernel parameterizations.
- CNN architectures with FlexConvs (FlexNets) obtain state-of-the-art across several sequential datasets, and match recent works with learnable kernel size on CIFAR-



**Figure 5.1:** The Flexible Size Continuous Kernel Convolution (FlexConv). FlexConv defines convolutional kernels as the multiplication of a continuous convolutional kernel  $MLP^\psi$ , with a Gaussian mask of local support  $w_{\text{gauss}}$ :  $\psi(x, y) = w_{\text{gauss}}(x, y; \theta_{\text{mask}}) \cdot MLP^\psi(x, y)$ . By learning the parameters of the mask, the size of the convolutional kernel can be optimized during training. See also Fig. D.1.



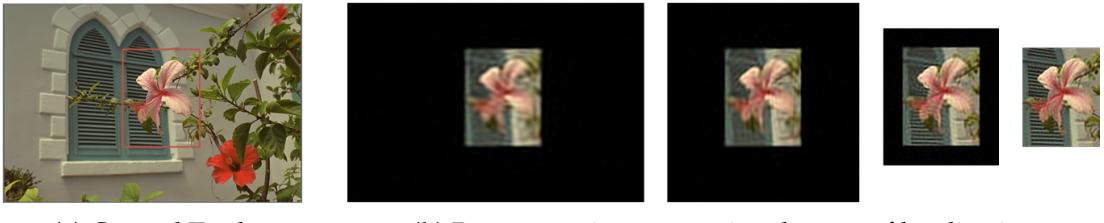
**Figure 5.2:** Existing approaches increase the size of convolutional kernels via (learnable) parametric dilations, e.g., by deformation [74] (b) or by Gaussian blur [291] (c). However, dilation limits the bandwidth of the dilated kernel and with it, the amount of detail it can describe. Contrarily, FlexNets extend their kernels by passing a larger vector of positions to the neural network parameterizing them. As a result, FlexConvs are able to learn *high bandwidth* convolutional kernels of varying size end-to-end (a).

10 with less compute.

## 5.2 Related Work

**Adaptive kernel sizes.** Loog and Lauze [237] regularize the scale of convolutional kernels for filter learning. For image classification, adaptive kernel sizes have been proposed via learnable pixel-wise offsets [74], learnable padding operations [136], learnable dilated Gaussian functions [271, 349, 373, 442] and scalable Gaussian derivative filters [225, 291, 384]. These approaches either dilate discrete kernels (Fig. 5.2b), or use discrete weights on dilated basis functions (Fig. 5.2c). Using dilation crucially limits the bandwidth of the resulting kernels. In contrast, FlexConvs are able to construct high bandwidth convolutional kernels of varying size with a fixed parameter count. Larger kernels are obtained simply by passing more positions to the kernel network (Fig. 5.1).

**Continuous kernel convolutions.** Discrete convolutional kernel parameterizations assign an independent weight to each specific position in the kernel. Continuous convo-



**(a) Ground Truth**      **(b) Reconstructions at varying degrees of localization**

**Figure 5.3:** The importance of dynamic sizes in continuous kernel convolutions. Consider a neural network predicting pixel values at each position. If the entire image is considered, the network must use part of its capacity to learn to predict zeros outside of the flower region, which in turn degrades the quality of the approximation in the region of interest (b). Importantly, the better the localization of the flower, the higher the approximation fidelity becomes. FlexNets learn the size of their convolutional kernels at each layer during training, and thus (i) use the capacity of the kernel efficiently, (ii) converge faster to good approximations, and (iii) are faster in execution.

lutional kernels, on the other hand, view convolutional kernels as continuous functions parameterized via a small neural network  $\text{MLP}^\psi: \mathbb{R}^D \rightarrow \mathbb{R}^{N_{\text{out}} \times N_{\text{in}}}$ , with  $D$  the data dimensionality. This defines a convolutional kernel for which arbitrary input positions can be queried. Continuous kernels have primarily been used to handle irregularly-sampled data *locally*, e.g., molecular data [343, 355] and point-clouds [352, 382, 417].

Recently, Romero et al. [321] introduced the Continuous Kernel Convolution (CKConv) as a tool to model long-term dependencies. CKConv uses a continuous kernel parameterization to construct convolutional kernels as big as the input signal with a constant parameter cost. Contrarily, FlexConvs jointly learn the convolutional kernel as well as its size. This leads to important advantages in terms of expressivity (Fig. 5.3), convergence speed and compute costs of the operation.

**Implicit neural representations.** Parameterizing a convolutional kernel via a neural network can be seen as learning an implicit neural representation of the underlying convolutional kernel [321]. Implicit neural representations construct continuous data representations by encoding data in the weights of a neural network [104, 278, 358].

We replace the SIREN [358] kernel parameterization used in Romero et al. [321] by our *Multiplicative Anisotropic Gabor Networks*: a new class of Multiplicative Filter Networks [104]. MFNs allow for analytic control of the resulting representations, and allow us to construct analytic alias-free convolutional kernels. The higher expressivity and convergence speed of MAGNets lead to accuracy improvements in CNNs using them as kernel parameterization.

## 5.3 Method

In this section, we introduce our approach. First, we introduce FlexConv and the Gaussian mask. Next, we introduce our Multiplicative Anisotropic Gabor Networks (MAGNets) and provide a description of our regularization technique used to control the spectral components of the generated kernel.

### 5.3.1 Flexible Size Continuous Kernel Convolution (FlexConv)

To learn the kernel size during training, FlexConvs define their convolutional kernels  $\psi$  as the product of the output of a neural network  $\text{MLP}^\psi$  with a Gaussian mask of local support. The neural network  $\text{MLP}^\psi$  parameterizes the kernel, and the Gaussian mask parameterizes its size (Fig. 5.1).

**Anisotropic Gaussian mask.** Let  $G(x; \mu_X, \sigma_X^2) := \exp\left\{-\frac{1}{2}\sigma_X^{-2}(x - \mu_X)^2\right\}$  be a Gaussian function parameterized by a mean-variance tuple  $(\mu_X, \sigma_X^2)$ . The anisotropic Gaussian mask is defined as:

$$w_{\text{gauss}}(x, y; \{\mu_X, \sigma_X^2, \mu_Y, \sigma_Y^2\}) = G(x; \mu_X, \sigma_X^2) \cdot G(y; \mu_Y, \sigma_Y^2). \quad (5.1)$$

By learning  $(\mu_X, \sigma_X^2)$  and  $(\mu_Y, \sigma_Y^2)$ , anisotropic non-centered windows are learned.

### 5.3.2 Multiplicative Anisotropic Gabor Networks (MAGNets)

In this section, we formalize our proposed parameterization for the kernel  $\text{MLP}^\psi$ . We introduce Multiplicative Filter Networks [104] and present MAGNets next.

**Multiplicative Filter Networks (MFNs).** Recently, Fathony et al. [104] proposed to construct implicit neural representations as the linear combination of exponentially many basis functions  $\mathbf{g}$ :

$$\mathbf{h}^{(1)} = \mathbf{g}([x, y]; \theta^{(1)}) \quad \mathbf{g}: \mathbb{R}^2 \rightarrow \mathbb{R}^{N_{\text{hid}}} \quad (5.2)$$

$$\mathbf{h}^{(l)} = (\mathbf{W}^{(l)} \mathbf{h}^{(l-1)} + \mathbf{b}^{(l)}) \cdot \mathbf{g}([x, y]; \theta^{(l)}) \quad \mathbf{W}^{(l)} \in \mathbb{R}^{N_{\text{hid}} \times N_{\text{hid}}}, \mathbf{b}^{(l)} \in \mathbb{R}^{N_{\text{hid}}} \quad (5.3)$$

$$\psi(x, y) = \mathbf{W}^{(L)} \mathbf{h}^{(L-1)} + \mathbf{b}^{(L)} \quad \mathbf{W}^{(L)} \in \mathbb{R}^{N \times N_{\text{hid}}}, \mathbf{b}^{(L)} \in \mathbb{R}^N \quad (5.4)$$

where  $\{\theta^{(l)}, \mathbf{W}^{(l)}, \mathbf{b}^{(l)}\}$  depict the learnable parameters of the bases and the affine transformations, and  $N, N_{\text{hid}}$  depict the number of output and hidden channels, respectively. Depending on the selection of  $\mathbf{g}$ , MFNs obtain approximations comparable to those of SIRENs [358] with faster convergence rate. The most successful instantiation of MFNs are the *Multiplicative Gabor Network* (MGN): MFNs constructed with isotropic Gabor functions as basis  $\mathbf{g}$  (in Eq. 5.2):

$$\mathbf{g}([x, y]; \theta^{(l)}) = \exp\left(-\frac{\gamma^{(l)}}{2}[(x - \mu^{(l)})^2 + (y - \mu^{(l)})^2]\right) \text{Sin}(\mathbf{W}_g^{(l)} \cdot [x, y] + \mathbf{b}_g^{(l)}), \quad (5.5)$$

$$\theta^{(l)} = \{\gamma^{(l)} \in \mathbb{R}^{N_{\text{hid}}}, \mu^{(l)} \in \mathbb{R}^{N_{\text{hid}}}, \mathbf{W}_g^{(l)} \in \mathbb{R}^{N_{\text{hid}} \times 2}, \mathbf{b}_g^{(l)} \in \mathbb{R}^{N_{\text{hid}}}\}. \quad (5.6)$$

Note that, by setting  $N=N_{\text{out}} \times N_{\text{in}}$ , an MFN can parameterize a convolutional kernel with  $N_{\text{in}}$  input and  $N_{\text{out}}$  output channels. Fathony et al. [104] show that MFNs are equivalent to a linear combination of exponentially many basis functions  $\mathbf{g}$ . This allows us to analytically derive properties of MFN representations, and plays a crucial role in the derivation of alias-free MAGNets (Sec. 5.3.3).

**Multiplicative Anisotropic Gabor Networks (MAGNets).** Our MAGNet formulation is based on the observation that isotropic Gabor functions, i.e., with equal  $\gamma$  for the horizontal and vertical directions, are undesirable as basis for the construction of MFNs. Whenever a frequency is required along a certain direction, an isotropic Gabor function automatically introduces that frequency in both directions. As a result, other bases must counteract this frequency in the direction where the frequency is not required, and thus the capacity of the MFN is not used optimally [79].

Following the original formulation of the 2D Gabor functions [79], we alleviate this limitation by using anisotropic Gabor functions instead:

$$\mathbf{g}([x, y]; \boldsymbol{\theta}^{(l)}) = \exp \left( -\frac{1}{2} \left[ \left( \gamma_X^{(l)} (x - \mu_X^{(l)}) \right)^2 + \left( \gamma_Y^{(l)} (y - \mu_Y^{(l)}) \right)^2 \right] \right) \sin(\mathbf{W}_g^{(l)}[x, y] + \mathbf{b}_g^{(l)}) \quad (5.7)$$

$$\boldsymbol{\theta}^{(l)} = \left\{ \gamma_X^{(l)} \in \mathbb{R}^{N_{\text{hid}}}, \gamma_Y^{(l)} \in \mathbb{R}^{N_{\text{hid}}}, \mu_X^{(l)} \in \mathbb{R}^{N_{\text{hid}}}, \mu_Y^{(l)} \in \mathbb{R}^{N_{\text{hid}}}, \mathbf{W}_g^{(l)} \in \mathbb{R}^{N_{\text{hid}} \times 2}, \mathbf{b}_g^{(l)} \in \mathbb{R}^{N_{\text{hid}}} \right\}. \quad (5.8)$$

The resulting *Multiplicative Anisotropic Gabor Network* (MAGNet) obtains better control upon frequency components introduced to the approximation, and demonstrates important improvements in terms of descriptive power and convergence speed (Sec. 5.4).

**MAGNet initialization.** Fathony et al. [104] proposes to initialize MGNs by drawing the size of the Gaussian envelopes, i.e., the  $\gamma^{(l)}$  term, from a  $\text{Gamma}(\alpha \cdot L^{-1}, \beta)$  distribution at every layer  $l \in [1, \dots, L-1]$ . We observe however that this initialization does not provide much variability on the initial extension of the Gaussian envelopes and in fact, most of them cover a large portion of the space at initialization. To stimulate diversity, we initialize the  $\{\gamma_X^{(l)}, \gamma_Y^{(l)}\}$  terms by a  $\text{Gamma}(\alpha l^{-1}, \beta)$  distribution at the  $l$ -th layer. We observe that our proposed initialization consistently leads to better accuracy than the initialization of [104] across all tasks considered. (Sec. 5.4).

### 5.3.3 Analytic Alias-free MAGNets

FlexConvs can be deployed at higher resolutions than those observed during training, simply by sampling the underlying continuous representation of the kernel more densely, and accounting for the change in sampling rate. Consider a D-dimensional input signal  $f_{r^{(1)}}$  with resolution  $r^{(1)}$ . FlexConv learns a kernel  $\psi_{r^{(1)}}$  that can be inferred at a higher resolution  $r^{(2)}$  [321]:

$$\left( f_{r^{(2)}} * \psi_{r^{(2)}} \right) \approx \left( \frac{r^{(1)}}{r^{(2)}} \right)^D \left( f_{r^{(1)}} * \psi_{r^{(1)}} \right). \quad (5.9)$$

Note however, that Eq. 5.9 holds *approximately*. This is due to aliasing artifacts which can appear if the frequencies in the learned kernel surpass the Nyquist criterion of the target resolution. Consequently, an anti-aliased parameterization is vital to construct kernels that generalize well to high resolutions.

**Towards alias-free implicit neural representations.** We observe that SIRENs as well as unconstrained MFNs and MAGNets exhibit aliasing when deployed on resolutions higher than the training resolution, which hurts performance of the model. An example kernel with aliasing is shown in Fig. D.2.

To combat aliasing, we would like to control the representation learned by MAGNets. MAGNets –and MFNs in general– construct implicit neural representations that can be seen as a *linear combination of basis functions*. This property allows us to analytically derive and study the properties of the resulting neural representation. Here, we use this property to derive the maximum frequency of MAGNet-generated kernels, so as to regularize MAGNets against aliasing during training. We analytically derive the maximum frequency of a MAGNet, and penalize it whenever it exceeds the Nyquist frequency of the training resolution. We note that analytic derivations are difficult for other implicit neural representations, e.g., SIRENs, due to stacked layer-wise nonlinearities.

**Maximum frequency of MAGNets.** The maximum frequency of a MAGNet is given by:

$$f_{\text{MAGNet}}^+ = \sum_{l=1}^L \max_{i_l} \left( \left( \max_j \frac{\mathbf{W}_{g,i_l,j}^{(l)}}{2\pi} \right) + \frac{\sigma_{\text{cut}} \min\{\gamma_{X,i_l}^{(l)}, \gamma_{Y,i_l}^{(l)}\}}{2\pi} \right), \quad (\text{D.12})$$

where  $L$  corresponds to the number of layers,  $\mathbf{W}_g^{(l)}$ ,  $\gamma_X^{(l)}$ ,  $\gamma_Y^{(l)}$  to the MAGNet parameters as defined in Eq. 5.8, and  $\sigma_{\text{cut}}=2 \cdot \text{stdev}$  to the cut-off frequency of the Gaussian envelopes in the Gabor filters. The derivation of this term is provided in Appx. D.1.1.

**Effect of the FlexConv mask.** The Gaussian mask used to localize the response of the MAGNet also has an effect on the frequency spectrum. Hence, the maximum frequency of a FlexConv kernel is:

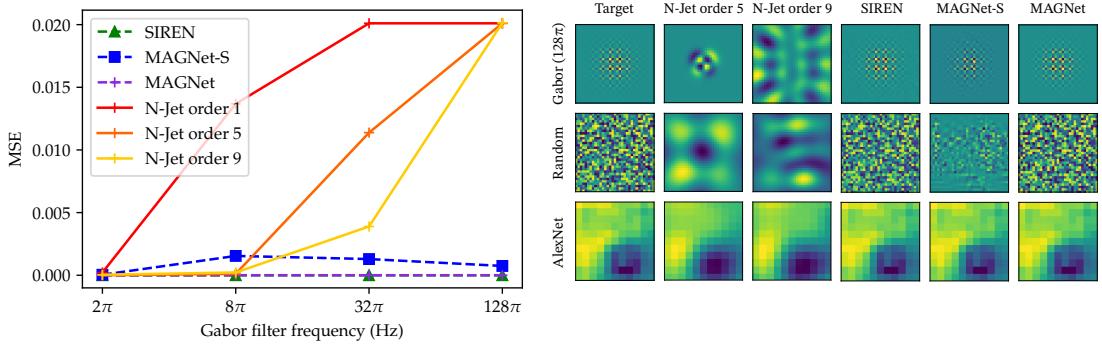
$$f_{\text{FlexConv}}^+ = f_{\text{MAGNet}}^+ + f_{w_{\text{gauss}}}^+, \quad \text{with } f_{w_{\text{gauss}}}^+ = \frac{\sigma_{\text{cut}}}{\max\{\sigma_X, \sigma_Y\}2\pi}. \quad (\text{D.13})$$

Here,  $\sigma_X, \sigma_Y$  correspond to the mask parameters (Eq. 5.1). Intuitively, multiplication with the mask blurs in the frequency domain, as it is equivalent to convolution with the Fourier transform of the mask.

**Aliasing regularization of FlexConv kernels.** With the analytic derivation of  $f_{\text{FlexConv}}^+$  we penalize the generated kernels to have frequencies smaller or equal to their Nyquist frequency  $f_{\text{Nyq}}(k)$  via:

$$\mathcal{L}_{\text{HF}} = \|\max\{f_{\text{FlexConv}}^+, f_{\text{Nyq}}(k)\} - f_{\text{Nyq}}(k)\|^2, \quad \text{with } f_{\text{Nyq}}(k) = \frac{k-1}{4}. \quad (\text{D.16})$$

Here,  $k$  depicts the size of the FlexConv kernel before applying the Gaussian mask, and is equal to the size of the input signal. In practice, we implement Eq. D.16 by regularizing



**Figure 5.4:** *Left:* Final MSE after fitting each model to Gabor filters of different frequencies. N-Jets cannot fit high frequencies. *Right:* Kernels learned by each model. SIREN and MAGNet can fit all targets. MAGNet-S: a small MAGNet of size akin to N-Jets, still does well on the Gabor and AlexNet targets.

the individual MAGNet layers, as is detailed in Appx. D.1.2. As verification, Fig. D.2 shows that FlexNet kernels are properly regularized against aliasing.

## 5.4 Experiments

We evaluate FlexConv across classification tasks on sequential and image benchmark datasets, and validate the ability of MAGNets to approximate complex functions. A complete description of the datasets used is given in Appx. D.2. The hyperparameters used in our experiments are reported in Appx. D.4.2.

### 5.4.1 What kind of functions can MAGNets approximate?

**Bandwidth of methods with learnable sizes.** First, we compare the bandwidth of MAGNet against N-Jet [291] by optimizing each to fit simple targets: (i) Gabor filters of known frequency, (ii) random noise and (iii) an  $11 \times 11$  AlexNet kernel from the first layer [196]. Fig. 5.4 shows that, even with 9 orders of Gaussian derivatives, N-Jets cannot fit high frequency signals in large kernels. Crucially, N-Jet models require many Gaussian derivative orders to model high frequency signals in large kernels: a hyperparameter which proportionally increases their inference time and parameter count. MAGNets, on the other hand, accurately model large high frequency signals. This allows FlexNets to learn large kernels with high frequency components.

**Expressivity of MLP parameterizations.** Next, we compare the descriptive power and convergence speed of MAGNets, Gabor MFNs, Fourier MFNs and SIRENs for image approximation. To this end, we fit the images in the Kodak dataset [193] with each of these methods. Our results (Tab. D.1) show that MAGNets outperform all other methods, and converge faster to good approximations.

**Table 5.1:** Test accuracy and ablation on sMNIST, pMNIST, sCIFAR10 and npCIFAR10.

| MODEL                         | SIZE | sMNIST       | pMNIST       | sCIFAR10     | npCIFAR10    |
|-------------------------------|------|--------------|--------------|--------------|--------------|
| DilRNN [44]                   | 44K  | 98.0         | 96.1         | -            | -            |
| IndRNN [218]                  | 83K  | 99.0         | 96.0         | -            | -            |
| TCN [11]                      | 70K  | 99.0         | 97.2         | -            | -            |
| r-LSTM [386]                  | 0.5M | 98.4         | 95.2         | 72.2         | -            |
| Self-Att. [386]               | 0.5M | 98.9         | 97.9         | 62.2         | -            |
| TrellisNet [12]               | 8M   | 99.20        | 98.13        | 73.42        | -            |
| URLSTM [127]                  | -    | 99.28        | 96.96        | 71.00        | -            |
| URGRU + Zoneout [127]         | -    | 99.27        | 96.51        | <b>74.40</b> | -            |
| HiPPO [126]                   | 0.5M | -            | <b>98.30</b> | -            | -            |
| Lipschitz RNN [97]            | 158K | 99.4         | 97.3         | 64.2         | 59.0         |
| coRNN [327]                   | 134K | <b>99.4</b>  | 97.3         | -            | 59.0         |
| UnICORNN [328]                | 135K | -            | 98.4         | -            | <b>62.4</b>  |
| pLMU [55]                     | 165K | -            | 98.49        | -            | -            |
| CKCNN-2                       | 98K  | 99.31        | 98.00        | 62.25        | 60.5         |
| CKCNN-2-Big                   | 1M   | 99.32        | 98.54        | 63.74        | 62.2         |
| CKTCN <sub>FOURIER</sub> -2   | 105K | 99.44        | 98.40        | 68.28        | 66.26        |
| CKTCN <sub>GABOR</sub> -2     | 106K | 99.52        | 98.38        | 69.26        | 67.37        |
| CKTCN <sub>MAGNet</sub> -2    | 105K | <b>99.55</b> | <b>98.57</b> | <b>74.58</b> | <b>67.52</b> |
| FlexTCN-2                     | 108K | <b>99.60</b> | <b>98.61</b> | <b>78.99</b> | <b>67.11</b> |
| FlexTCN-4                     | 241K | <b>99.60</b> | <b>98.72</b> | <b>80.26</b> | <b>67.42</b> |
| FlexTCN-6                     | 375K | <b>99.62</b> | <b>98.63</b> | <b>80.82</b> | <b>69.87</b> |
| FlexTCN <sub>SIREN</sub> -6   | 343K | 99.03        | 95.36        | 69.24        | 57.27        |
| FlexTCN <sub>Fourier</sub> -6 | 370K | 99.49        | 97.97        | 74.79        | 67.35        |
| FlexTCN <sub>Gabor</sub> -6   | 373K | 99.50        | 98.37        | 78.36        | 67.56        |
| FlexTCN <sub>MAGNet</sub> -6  | 375K | <b>99.62</b> | <b>98.63</b> | <b>80.82</b> | <b>69.87</b> |

#### 5.4.2 Classification Tasks

**Network specifications.** Here, we specify our networks for all our classification experiments. We parameterize all our convolutional kernels as the superposition of a 3-layer MAGNet and a learnable anisotropic Gaussian mask. We construct two network instances for sequential and image datasets respectively: FlexTCNs and FlexNets. Both are constructed by taking the structure of a baseline network –TCN [11] or CIFARResNet [143]–, removing all internal pooling layers, and replacing convolutional kernels by FlexConvs. The FlexNet architecture is shown in Fig. D.4 and varies only in the number of channels and blocks, e.g., FlexNet-16 has 7 blocks. Akin to [321] we utilize the Fourier theorem to speed up convolutions with large kernels.

**Mask initialization.** We initialize the FlexConv masks to be small. Preliminary experiments show this leads to better performance, faster execution, and faster training convergence. For sequences, the mask center is initialized at the last kernel position to prioritize the last information seen.

**Time-series and sequential data.** First we evaluate FlexTCNs on sequential classification datasets, for which long-term dependencies play a crucial role. We validate our approach on intrinsic discrete data: *sequential MNIST*, *permuted MNIST* [203], *sequential CIFAR10* [44], *noise-padded CIFAR10* [42], and time-series: *CharacterTrajectories* (CT) [9], *SpeechCommands* [421] with raw waveform (SC.raw) and MFCC representations (SC).

Our results are shown in Tables 5.1 and 5.2. FlexTCNs with two residual blocks ob-

**Table 5.2:** Test accuracy on CT, SC and SC\_raw

| MODEL                         | SIZE  | CT           | SC           | SC_RAW       |
|-------------------------------|-------|--------------|--------------|--------------|
| GRU-ODE                       | 89K   | 96.2         | 44.8         | ~10.0        |
| GRU- $\Delta t$               | 89K   | 97.8         | 20.0         | ~10.0        |
| GRU-D                         | 89K   | 95.9         | 23.9         | ~10.0        |
| ODE-RNN                       | 89K   | 97.1         | 93.2         | ~10.0        |
| NCDE                          | 89K   | 98.8         | 88.5         | ~10.0        |
| CKCNN                         | 100K  | <b>99.53</b> | 95.27        | 71.66        |
| CKTCN <sub>Fourier</sub>      |       | -            | 95.65        | 74.90        |
| CKTCN <sub>Gabor</sub>        |       | -            | 96.66        | 78.10        |
| CKTCN <sub>MAGNet</sub>       | 105K  | <b>99.53</b> | <b>97.01</b> | <b>80.69</b> |
| FlexTCN-2                     | 105SK | <b>99.53</b> | <b>97.10</b> | <b>88.03</b> |
| FlexTCN-4                     | 239K  | <b>99.53</b> | <b>97.73</b> | <b>90.45</b> |
| FlexTCN-6                     | 373K  | <b>99.53</b> | <b>97.67</b> | <b>91.73</b> |
| FlexTCN <sub>SIREN</sub> -6   | 370K  | -            | 95.83        | 85.73        |
| FlexTCN <sub>Fourier</sub> -6 | 342K  | -            | 97.62        | 91.02        |
| FlexTCN <sub>Gabor</sub> -6   | 373K  | -            | 97.35        | 91.50        |
| FlexTCN <sub>MAGNet</sub> -6  | 373K  | -            | <b>97.67</b> | <b>91.73</b> |

**Table 5.3:** Results on CIFAR-10. Results from \*original works and † single run.

| MODEL                                       | SIZE  | CIFAR-10 ACC.   | TIME (SEC/EPOCH) |
|---|-------|-----------------|------------------|
| CIFARResNet-44                              | 0.66M | 92.9*†          | 22               |
| DCN- $\sigma^{ji}$                          | 0.47M | 89.7 $\pm$ 0.3* | -                |
| N-Jet-CIFARResNet32                         | 0.52M | 92.3 $\pm$ 0.3* | -                |
| N-Jet-ALLCNN                                | 1.07M | 92.5 $\pm$ 0.1* | -                |
| FlexNet-16 w/ conv. ( $k = 3$ )             | 0.17M | 89.5 $\pm$ 0.3  | 41               |
| FlexNet-16 w/ conv. ( $k = 33$ )            | 20.0M | 78.0 $\pm$ 0.3  | 242              |
| FlexNet-16 w/ N-Jet                         | 0.70M | 91.7 $\pm$ 0.1  | 409              |
| CKCNN-16                                    | 0.63M | 72.1 $\pm$ 0.2  | 68               |
| CKCNN <sub>MAGNet</sub> -16                 | 0.67M | 86.8 $\pm$ 0.6  | 102              |
| FlexNet <sub>SIREN</sub> -16                | 0.63M | 89.0 $\pm$ 0.3  | 89               |
| FlexNet <sub>Gabor</sub> -16                | 0.67M | 91.9 $\pm$ 0.2  | 161              |
| FlexNet <sub>Gabor</sub> -16 + anis. Gauss. | 0.67M | 92.0 $\pm$ 0.1  | 147              |
| FlexNet <sub>Gabor</sub> -16 + Gabor init.  | 0.67M | 92.0 $\pm$ 0.2  | 150              |
| FlexNet-16                                  | 0.67M | 92.2 $\pm$ 0.1  | 127              |

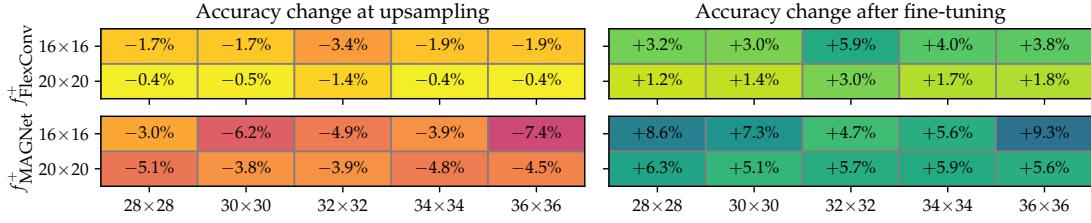
tain state-of-the-art results on all tasks considered. In addition, depth further improves performance. FlexTCN-6 improves the current state-of-the-art on sCIFAR10 and npCIFAR10 by more than 6%. On the difficult SC\_raw dataset –with sequences of length 16000–, FlexTCN-6 outperform the previous state-of-the-art by 20.07%.

Furthermore, we conduct ablation studies by changing the parameterization of  $\text{MLP}^\psi$ , and switching off the learnable kernel size ("CKTCNs") and considering global kernel sizes instead. CKTCNs and FlexTCNs with MAGNet kernels outperform corresponding models with all other kernel parameterizations: SIRENs [358], MGNs and MFNs [104]. Moreover, we see a consistent improvement with respect to CKCNNs [321] by using learnable kernel sizes. This shows that both MAGNets and learnable kernel sizes contribute to the performance of FlexTCNs. Note that in 1D, MAGNets are equivalent to MGNs. However, MAGNets consistently perform better than MGNs. This improvement in accuracy is a result of our MAGNet initialization.

**Image classification.** Next, we evaluate FlexNets on CIFAR-10 [195]. Additional experiments on ImageNet-32, MNIST and STL-10 are given in Appx. D.3.

Table 5.3 shows our results on CIFAR-10. FlexNets are competitive with pooling-based methods such as CIFARResNet [143] and outperform learnable kernel size method DCNs [384]. In addition, we compare using N-Jet layers of order three (as in Pintea et al. [291]) in FlexNets against using MAGNet kernels. We observe that N-Jet layers lead to worse performance, and are significantly slower than FlexConv layers with MAGNet kernels. The low accuracy of N-Jet layers is likely to be linked to the fact that FlexNets do not use pooling. Consequently, N-Jets are forced to learn large kernels with high-frequencies, which we show N-Jets struggle learning in Sec. 5.4.1.

To illustrate the effect of learning kernel sizes, we also compare FlexNets against FlexNets



**Figure 5.5:** Alias-free FlexNet-16 on CIFAR-10. We report change in accuracy between source and target resolutions, directly after upsampling (left) and after fine-tuning (right) (means over five runs).

with large and small discrete convolutional kernels (Tab. 5.3). Using small kernel sizes is parameter efficient, but is not competitive with FlexNets. Large discrete kernels on the other hand require a copious amount of parameters and lead to significantly worse performance. These results indicate that the best solution is somewhere in the middle and varying kernel sizes can learn the optimal kernel size for the task at hand.

Similar to the sequential case, we conduct ablation studies on image data with learnable, non-learnable kernel sizes and different kernel parameterizations. Table 5.3 shows that FlexNets outperform CKCNNs with corresponding kernel parameterizations. In addition, a clear difference in performance is apparent for MAGNets with respect to other parameterizations. These results corroborate that both MAGNets and FlexConvs contribute to the performance of FlexNets. Moreover, Tab. 5.3 illustrates the effect of the two contributions of MAGNet over MGN: anisotropic Gabor filters, and our improved initialization. Our results are in line with our results for sequential data (Tabs. 5.1, 5.2), illustrating the value of the proposed improvements of MAGNets.

#### 5.4.3 Alias-free FlexNets

**Regularizing the FlexConv mask.** Though including  $f_{w_{\text{gauss}}}^+$  in the frequency analysis of MAGNets is crucial for the accuracy of the derivation, including the FlexConv mask in aliasing regularization is undesirable, as it steers the model to learn large kernels in order to minimize the loss (see Eq. D.16). However, excluding the mask from regularization could compromise the ability of FlexNet to generalize to higher resolutions. Here, we experiment with this trade-off.

Figure 5.5 shows accuracy change between ten source and target resolution combinations on CIFAR-10, both for including and excluding the FlexConv mask in the aliasing regularization. We train at the source resolution for 100 epochs, before testing the model at the target resolution with the upsampling described in Sec. 5.3.3. Next, we adjust  $f_{\text{Nyq}}(k)$  to the target resolution, and finetune each model for 100 epochs at the target resolution.

We find that regularizing just  $f_{\text{MAGNet}}^+$  yields a trade-off. It increases the accuracy differ-

**Table 5.4:** Alias-free FlexNets on CIFAR-10.

| MODEL                                      | SIZE  | CIFAR-10 ACC.                    |                                      |
|--|-------|----------------------------------|--------------------------------------|
|  |       | 16 px                            | $\Delta_{16\text{px}} 32 \text{ px}$ |
| CIFARResNet-44                             | 0.66M | $85.8 \pm 0.2$                   | $-31.6 \pm 1.3$                      |
| FlexNet-16 w/ conv. ( $k = 3$ )            | 0.17M | $85.3 \pm 0.2$                   | $-21.2 \pm 1.0$                      |
| FlexNet-16 w/ conv. ( $k = 33$ )           | 20.0M | $67.7 \pm 0.6$                   | $-57.1 \pm 1.6$                      |
| FlexNet-16 w/ N-Jets                       | 0.70M | <b><math>86.4 \pm 0.2</math></b> | $-5.5 \pm 1.3$                       |
| CKCNN-16 <sub>SIREN</sub>                  | 0.63M | $45.9 \pm 1.0$                   | $-15.8 \pm 1.2$                      |
| FlexNet-16 <sub>SIREN</sub>                | 0.63M | $70.4 \pm 0.8$                   | $-50.0 \pm 16.9$                     |
| FlexNet-16 w/o reg.                        | 0.67M | <b><math>86.4 \pm 0.4</math></b> | $-34.4 \pm 14.3$                     |
| FlexNet-16 w/ reg. $f_{\text{MAGNet}}^+$   | 0.67M | <b><math>86.5 \pm 0.1</math></b> | $-3.8 \pm 2.0$                       |
| FlexNet-16 w/ reg. $f_{\text{FlexConv}}^+$ | 0.67M | $85.1 \pm 0.3$                   | <b><math>-3.3 \pm 0.3</math></b>     |

ence between low and high resolution inference, but also increases the fine-tune accuracy at the target resolution. We therefore regularize  $f_{\text{MAGNet}}^+$  only by default.

Results of our alias-free FlexNet training on CIFAR-10 are in Table 5.4. We observe that the performance of a FlexNet trained without aliasing regularization largely breaks down when the dataset is upscaled. However, with our aliasing regularization most of the performance is retained.

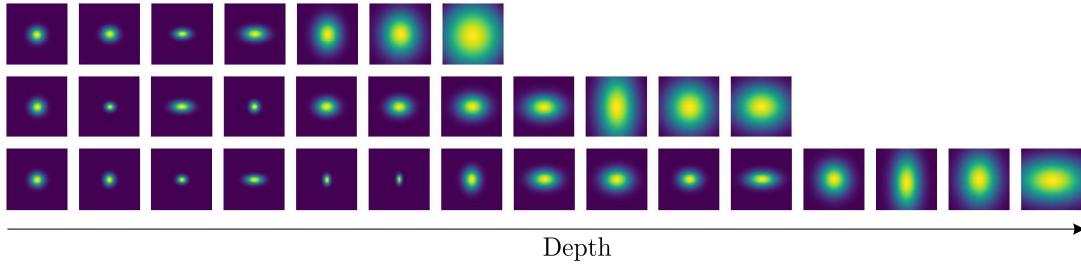
Comparatively, FlexNet retains more of the source resolution performance than FlexNets with N-Jet layers, while baselines degrade drastically at the target resolution. Fig. D.2 shows the effect of aliasing regularization on the frequency components of FlexConv.

**Training at lower resolutions saves compute.** We can train alias-free FlexNets at lower resolutions. To verify that this saves compute, we time the first 32 batches of training a FlexNet-7 on CIFAR-10. We compare against training on  $16 \times 16$  images (downsampled before training). On  $16 \times 16$  images, each batch takes 179ms ( $\pm 7$ ms). On  $32 \times 32$  images, each batch takes 222ms ( $\pm 9$ ms). Therefore, we save 24% training time when training FlexNets alias-free at half the native CIFAR-10 resolution.

## 5.5 Discussion

**Learned kernel sizes match conventional priors.** Commonly, CNNs use architectures of small kernels and pooling layers. This allows convolutions to build a progressively growing receptive field. With learnable kernel sizes, FlexNet could learn a different prior over receptive fields, e.g., large kernels first, and small kernels next. However, FlexNets learn to increase kernel sizes progressively (Fig. 5.6), and match the network design that has been popular since AlexNet [196].

**Mask initialization as a prior for feature importance.** The initial values of the FlexConv mask can be used to prioritize information at particular input regions. For instance, initializing the center of mask on the first element of sequential FlexConvs can be used to prioritize information from the far past. This prior is advantageous for tasks such



**Figure 5.6:** Learned masks for FlexNets with 3, 5 and 7 residual blocks. FlexNets learn very small kernels at shallow layers, which become larger as a function of depth.

as npCIFAR10. We observe that using this prior on npCIFAR10 leads to much faster convergence and better results (68.33% acc. w/ FlexTCN-2).

**MAGNet regularization as prior induction.** MAGNets allow for analytic control of the properties of the resulting representations. We use this property to generate alias-free kernels. However, other desiderata could be induced, e.g., smoothness, for the construction of implicit neural representations.

**Benefits of cropping and the influence of PyTorch.** Dynamic cropping adjust the computational cost of the convolutions on the fly. For a signal of size  $M$  and a cropped kernel size  $k$ , this incurs in savings from  $O(M^{2^D})$  to  $O(M^D k^D)$  relative to using global kernel sizes ( $O(M^4)$  to  $O(M^2 k^2)$  in 2D). We test this theoretical speed up in a controlled environment for the Speech Commands and CIFAR-10 datasets. Cropping reduces the per-epoch run time by a factor of 11.8x and 5.5x for Speech Commands and CIFAR-10, respectively. Interestingly, however, both run times become similar if the flag `torch.backends.cudnn.benchmark` is activated, with global kernel sizes being sometimes faster. This is because this flag tells PyTorch to optimize the convolution algorithms used under the hood, and some of these CUDA algorithms seem to be faster than our masking strategy on Python.

## 5.6 Limitations

**Dynamic kernel sizes: computation and memory cost of convolutions with large kernels.** Performing convolutions with large convolutional kernels is a compute-intensive operation. FlexConvs are initialized with small kernel sizes and their inference cost is relatively small at the start of training. However, despite the cropping operations used to improve computational efficiency (Figs. 5.1, 5.3, Tab. 5.3), the inference time may increase to up to double as the learned masks increase in size. At the cost of more memory, convolutions can be sped up by performing them in the frequency domain. However, we observe that this does not bring gains on image data considered as FFT convolutions are only faster for very large convolutional kernels –in the order of hundreds of pixels.

**Remaining accuracy drop in alias-free FlexNets.** Some drop in accuracy is still ob-

served when using alias-free FlexNets at a higher test resolutions (Tab. 5.4). Although more evidence is needed, this may be caused by aliasing effects introduced by ReLU [401] or by changes in the statistics of the feature maps passed to global pooling [385].

## 5.7 Conclusion

We propose FlexConv, a convolutional operation able to learn high bandwidth convolutional kernels of varying size during training at a fixed parameter cost. We demonstrate that FlexConvs are able to model long-term dependencies without the need of pooling, and shallow pooling-free FlexNets achieve state-of-the-art performance on several sequential datasets, match performance of recent works with learned kernel sizes with less compute, and are competitive with much deeper ResNets on image benchmark datasets. In addition, we show that our alias-free convolutional kernels allow FlexNets to be deployed at higher resolutions than seen during training with minimal loss.

**Future work.** MAGNets give control over the bandwidth of the kernel. We anticipate that this control has more uses, such as fighting sub-sampling aliasing [180, 182, 468]. With the ability to upscale FlexNets to different input image sizes comes the possibility of transfer learning representations between previously incompatible datasets, such as CIFAR-10 and ImageNet. In a similar vein, the automatic adaptation of FlexConv to the kernel sizes required for the task at hand may make it possible to generalize the FlexNet architecture across different tasks and datasets. Neural architecture search [482] could see benefits from narrowing the search space to exclude kernel size and pooling layers. In addition, we envisage additional improvements from structural developments of FlexConvs such as attentive FlexNets.

# 6

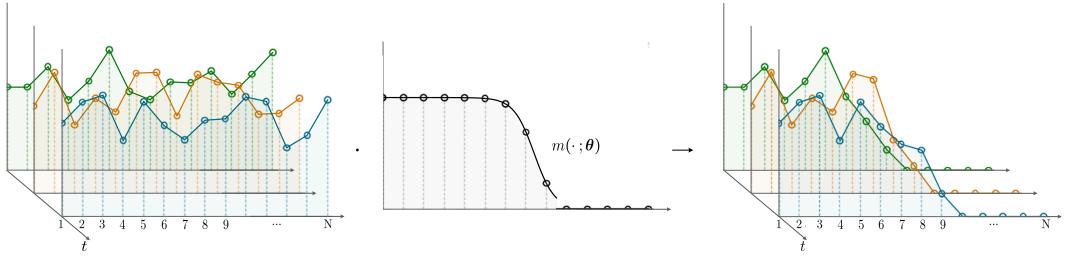
## Learning Convolutional Neural Architectures by Backpropagation

*Based on the paper:  
DNArch: Learning Convolutional Neural Architectures by Backpropagation [317]*

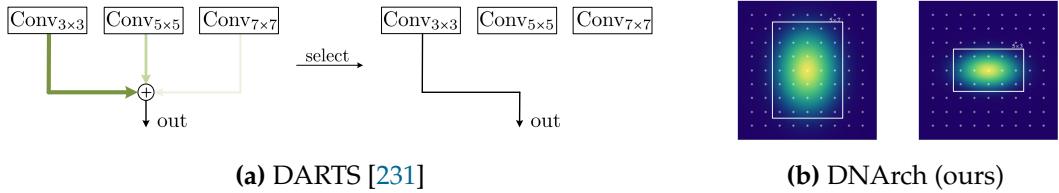
### 6.1 Introduction

Convolutional Neural Networks (CNNs) [208] are widely used for tasks such as image classification [143, 196], speech recognition [346, 459], text classification [70] and generative modeling [85, 277] due to their performance and efficiency. However, tailoring a CNN architecture to a specific task or dataset typically requires substantial human intervention and cross-validation to design the architectures, e.g. to determine appropriate kernel sizes, width, depth, etc. This has motivated exploring the space of architectures in an automatic fashion, by developing architecture search algorithms [230, 477, 482].

While these methods can find good architectures, they must solve an expensive discrete optimization problem that involves training and evaluating candidate architectures in each iteration, e.g. to optimize a reward with reinforcement learning [482], or to evolve the model through a genetic algorithm [230]. Differentiable Architecture Search (DARTS) [231] addresses this issue by allowing the network to consider a set of *predefined* possible components in parallel, e.g., convolutions with kernels of size  $3 \times 3$ ,  $5 \times 5$ ,  $7 \times 7$ , and adjusting their contribution using learnable weights (Fig. 6.2a). Although DARTS is able to *select* components via backpropagation, it requires (*i*) defining a (small) set of



**Figure 6.1:** DNAArch views neural architectures as entities in a continuous multidimensional space, and uses differentiable masks to learn their length by backpropagation. In this example, DNAArch learns the width of a layer by applying a differentiable mask  $m$  with learnable parameters  $\theta$  to the channel dimension. As a result, changes in the value of  $\theta$  effectively results in changes in the layer’s width.



**Figure 6.2:** DARTS vs. DNAArch. DARTS learns the size of convolutional kernels using backpropagation to select among predefined options, e.g., DASH [350]. DNAArch, on the other hand, learns the size of convolutional kernels by modifying the parameters of the differentiable mask  $m(\cdot; \theta)$ . Different  $\theta$  values lead to different sizes.

possible components beforehand, (ii) computing and keeping their responses in memory during training, and (iii) retraining the found architecture from scratch to remove the effect of other components in the output.

In this paper, we introduce *Differentiable Neural Architectures* (DNAArch), a method that simultaneously learns the weights and the entire architecture of a CNN during training by backpropagation. Specifically, DNAArch learns the weights as well as (i) the size of convolutional kernels at each layer, (ii) the number of channels at each layer, (iii) the position and resolution of downsampling layers, and (iv) the number of layers of the network. To this end, DNAArch takes a novel approach to learning neural architectures by viewing them as entities defined in a multidimensional continuous space with dimensions corresponding to network attributes, e.g., depth, width, etc., and using differentiable masks with learnable parameters along each dimension to control their length (Fig. 6.1). Unlike DARTS methods, e.g., [231, 350], DNAArch does *not* require a predefined set of components to choose from, but instead is able to explore among *all* feasible values, e.g., all kernel sizes between  $1 \times 1$  and  $N \times N$  for a  $N \times N$  image. This is a result of the truly continuous nature of DNAArch, which, unlike DARTS, does not require multiple instantiations of the same layer for different parameter values (Fig. 6.2a). Instead, DNAArch explores the parameter space by modifying the learnable parameters of the differentiable masks (Fig. 6.2b), making it a much more scalable NAS method. Since both the architecture

and the weights are optimized in a single run, no retraining is needed after training.

**Results.** We empirically show that DNArc is able to find performant CNN architectures across several classification and dense prediction tasks on sequential and image datasets. The architectures found by DNArc consistently surpass the general-purpose convolutional architecture on which DNArc is applied, and often outperform specialized task-specific architectures. Moreover, we show that DNArc can be easily combined with a regularization term that controls the computational complexity of candidate networks. By doing so, DNArc explores among neural architectures that respect a predefined computational budget during the entire training process. As a result, finding architectures with DNArc is roughly as expensive as a single training loop of the underlying baseline.

## 6.2 Method

DNArc has two key components: Differentiable Masking and Continuous Kernel Convolutions. Here, we introduce these concepts and show how they can be used to learn CNN architectures next.

### 6.2.1 Differentiable Masking

Let us consider an arbitrary function  $f : [a, b] \rightarrow \mathbb{R}$ , which we want to be non-zero only in a subset  $[c, d] \subseteq [a, b]$ . To this end, we can multiply  $f$  with a mask  $m$  whose values are non-zero only on  $[c, d]$ , e.g., a rectangular mask  $\Pi_{[c,d]}(x) = \mathbb{1}_{[c,d]}$ . However, as its gradient is either zero or non-defined, it is not possible to learn the interval in which it is non-zero by backpropagation. To overcome this limitation, we can instead use a parametric differentiable mask  $m(\cdot; \theta)$  whose interval of non-zero values is defined by its parameters  $\theta$ . As the mask  $m(\cdot; \theta)$  is differentiable with regard to its parameters  $\theta$ , we can learn the interval on which it is non-zero using backpropagation.

In this work, we consider two types of masks: a Gaussian mask  $m_{\text{gauss}}(\cdot; \{\mu, \sigma^2\})$  parameterized by its mean and variance  $\theta = \{\mu, \sigma^2\}$ ; and a Sigmoid mask  $m_{\text{sigm}}(\cdot; \{\mu, \tau\})$  parameterized by its offset and its temperature  $\theta = \{\mu, \tau\}$  defined as:

$$m_{\text{gauss}}(x; \{\mu, \sigma^2\}) = \left\{ \exp\left(-\frac{1}{2} \frac{(x-\mu)^2}{\sigma^2}\right) \text{ if } \exp\left(-\frac{1}{2} \frac{(x-\mu)^2}{\sigma^2}\right) \geq T_m; 0 \text{ otherwise} \right\}, \quad (6.1)$$

$$m_{\text{sigm}}(x; \{\mu, \tau\}) = \left\{ 1 - \text{sigm}(\tau(x - \mu)) \text{ if } 1 - \text{sigm}(\tau(x - \mu)) \geq T_m; 0 \text{ otherwise} \right\}, \quad (6.2)$$

where  $T_m$  is a predefined threshold below which the mask is zero. These masks are illustrated in Fig. 6.3. To avoid clutter, in the rest of the document we will refer to these masks as  $m_{\text{gauss}}$  and  $m_{\text{sigm}}$ , and will provide specific instantiations when needed.

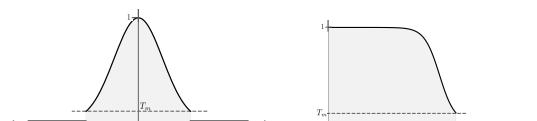


Figure 6.3: Gaussian and sigmoid masks.

**Multidimensional masks.** N-dimensional masks can be constructed by combining N 1D masks, each with their own parameters. For example, the Gaussian mask used to learn the size of convolutional kernels in Fig. 6.2 is constructed as:

$$m_{\text{gauss}}(x, y; \{\{\mu_X, \mu_Y\}, \{\sigma_X^2, \sigma_Y^2\}\}) = m_{\text{gauss}}(x; \{\mu_X, \sigma_X^2\}) \cdot m_{\text{gauss}}(y; \{\mu_Y, \sigma_Y^2\}) \quad (6.3)$$

### Materializing parameters only for non-zero mask values

Parts of differentiable masks will map to zero based on the value of the parameters  $\theta$ . Therefore, it would be a waste of compute and memory to materialize the mask –and the corresponding network parameters, e.g., channels  $\text{ch} \in [10, N]$  in Fig. 6.1– to zero them out next. Luckily, we can take advantage of the invertible form of the Gaussian and Sigmoid masks to materialize parameters only for values for which the mask is non-zero. To this end, we find the value  $x_{T_m}$  for which the mask is equal to the threshold  $T_m$ , i.e.,  $x_{T_m} = x$  such that  $m(x; \theta) = T_m$ , and only materialize the mask and the corresponding network parameters for values of  $x$  for which the value of the mask is greater than  $T_m$ . By inverting the mask equations (Eqs. 6.1, 6.2), we obtain  $x_{T_m}$  as:

$$\pm x_{T_m} = \mu \pm \sqrt{-2\sigma^2 \log(T_m)}, \quad (6.4) \quad x_{T_m} = \mu - \frac{1}{\tau} \log\left(\frac{1}{1-T_m} - 1\right), \quad (6.5)$$

for Gaussian and Sigmoid masks, respectively. Consequently, we can make sure that all rendered values will be used by only materializing the mask and related network parameters for values of  $x$  within the range  $[-x_{T_m}, x_{T_m}]$  for Gaussian masks and  $[x_{\min}, x_{T_m}]$  for Sigmoid masks, where  $x_{\min}$  depicts the lowest coordinate indexing the mask.

### 6.2.2 Continuous Kernel Convolutions

To prevent finding poor architectures due to insufficiently large receptive fields, it is important for a network to be able to model the global context regardless of specific architectural choices. We rely on Continuous Kernel Convolutions (CKConvs) [321] to model global dependencies on inputs of arbitrary length, resolution and dimensionality regardless of the network architecture [191]. CKConvs view convolutional kernels  $\psi$  as continuous functions parameterized by a small neural network  $\text{MLP}_\psi : \mathbb{R}^D \rightarrow \mathbb{R}^{N_{\text{in}} \times N_{\text{out}}}$  that receives coordinates  $\mathbf{c}_i \in \mathbb{R}^D$  as input and predicts the value of the convolutional kernel at those coordinates:  $\mathbf{c}_i \mapsto \text{MLP}_\psi(\mathbf{c}_i) = \psi(\mathbf{c}_i)$ . To construct a kernel of size  $K_x \times K_y$ , a CK-Conv layer constructs a grid of  $K_x \times K_y$  coordinates  $[\mathbf{c}_{(1,1)}, \mathbf{c}_{(1,2)}, \dots, \mathbf{c}_{(K_x, K_y)}]$ , and passes each coordinate through  $\text{MLP}_\psi$  (Fig. 6.5). As a result, CKConvs construct large kernels with few parameters by detaching the size of the kernel from its parameter count.

### 6.2.3 The need for learnable architectures

General-purpose architectures like Perceiver [168] and the Continuous CNN (CCNN) [320] make few assumptions about their input signals, and thus require few architectural changes to handle different tasks. However, the architectures of general-purpose models are static, and thus they are likely not optimal among all the tasks the model

might need to solve. For instance, Perceiver maps inputs to a hidden representation of constant size regardless of the complexity of the task and the input length, resolution and dimensionality. Consequently, it will likely not be able to represent tasks on large inputs correctly. CCNNs, on the other hand, avoid pooling and always perform (global) convolutions on the original input resolution. While this addresses the issue of having a hidden representation of fixed size for inputs with different characteristics, CCNNs can lead to unnecessarily high computational complexity by always performing convolutions at the input resolution. In addition, both the architectures of Perceivers and CCNNs are controlled by non-differentiable hyperparameters, and thus adapting them to a new task requires hyperparameter search across many configurations. To address these limitations, we propose to construct neural architectures that tune themselves to fit the requirements of a particular task in a single training run.

#### 6.2.4 Learning CNN architectures by backpropagation

Most components of DNArc, such as the learning of the network's width and depth, are not limited to convolutional architectures. However, this research aims to show *how DNArc can be used to learn as many components of a neural architecture as possible*. To that end, we use a general-purpose convolutional architecture: the CCNN [191], and make all its architectural components learnable.

**The Continuous CNN [191].** The Continuous CNN (CCNN) is a general-purpose convolutional model able to handle inputs of arbitrary dimension, length and resolution without changes. It consists of an Encoder, a Decoder, and many residual blocks (Fig. 6.4). We refer to the branch on which residual blocks modify the input as *residual branch*, and to the branch connecting the input and the output directly as the *identity branch*. The Encoder and Decoder adapt the input and output of the model to the goal of the task, e.g., dense / global predictions. Importantly, CCNN's ability to model global context on inputs of any resolution, length and dimensionality makes it an ideal base network for DNArc as (i) it prevents the formation of poor architectures due to insufficient receptive fields, and (ii) it allows DNArc to be used on tasks on data of arbitrary length and dimensionality without changing the base network –which is needed in existing methods (see NAS-Bench-360 [387] for several examples).

##### Learning the size of convolutional kernels

First introduced in FlexConv [319], differentiable masking can be combined with CK-Convs to learn the size of convolutional kernels by backpropagation. This is done by

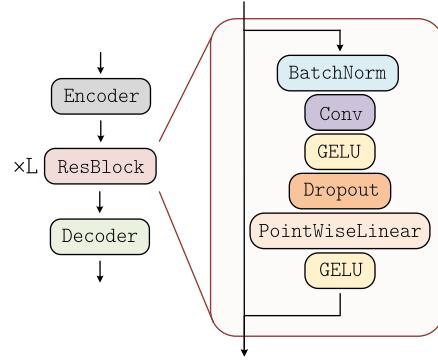
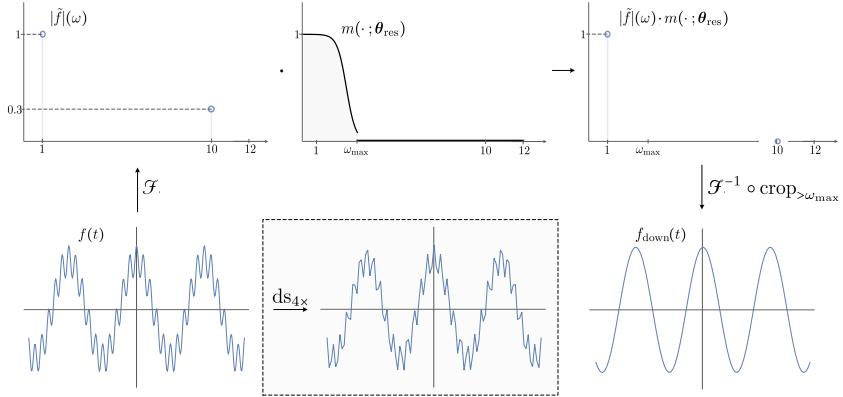
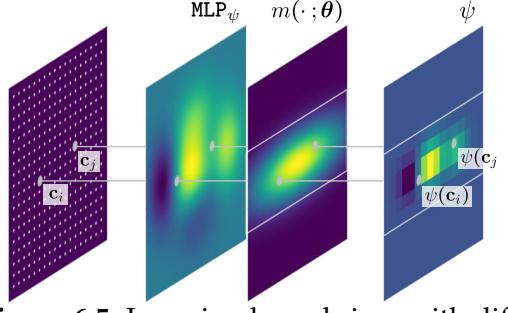


Figure 6.4: The CCNN architecture [320].



**Figure 6.6:** Learning downsampling with differentiable masking on the Fourier domain.

modelling convolutional kernels  $\psi$  as the product of a small neural network  $\text{MLP}_\psi$ , i.e., a Continuous Kernel Convolution, and a differentiable mask  $m(\cdot; \theta)$  with learnable parameters, i.e.,  $\psi(c_i) = \text{MLP}_\psi(c_i) \cdot m(c_i; \theta)$  (Fig. 6.5). Note that, it is possible to construct the convolutional kernel only for non-zero values of the mask  $m(c_i; \theta)$  by following the method outlined in Sec. 6.2.1.



**Figure 6.5:** Learning kernel sizes with differentiable masking and CKConvs [321].

### Learning downsampling layers

We can also use differentiable masking to learn downsampling by applying a differentiable mask on the Fourier domain. The Fourier transform  $\mathcal{F}$  represents a function  $f : \mathbb{R}^D \rightarrow \mathbb{R}$  in terms of its spectrum  $\tilde{f} : \mathbb{R}^D \rightarrow \mathbb{C}$ , which map frequencies  $\omega$  to the amount of that frequency in the input  $f(\omega)$ . A useful identity in this context is that cropping high frequencies in the Fourier domain equals downsampling in the spatial domain.

To learn downsampling, we use a learnable sigmoid mask  $m_{\text{sigm}}$  to perform a *learnable low-pass filtering* on the input. This is achieved by multiplying the spectrum of the input with the mask  $m_{\text{sigm}}$ . By doing so, all frequencies above the mask's cutoff frequency  $\omega_{\max} = T_m$  becomes zero (Fig. 6.6). An important consequence of low-pass filtering is that as the spectrum of the signal becomes zero above  $\omega_{\max}$ , the low-passed signal can be faithfully represented at a lower resolution determined by  $\omega_{\max}$ . Letting  $\mathcal{F}, \mathcal{F}^{-1}$  be the Fourier and inverse Fourier transform,  $\text{crop}_{>\omega_{\max}}$  be an operator that crops all values above  $\omega_{\max}$ , and  $f_{\text{down}}$  represent the downsampled signal  $f$ , we have that:

$$f_{\text{down}} = \mathcal{F}^{-1} [\text{crop}_{>\omega_{\max}} (\mathcal{F}[f] \cdot m_{\text{sigm}}(\cdot; \theta))]. \quad (6.6)$$

Fig. 6.6 shows an example of downsampling by a factor of 4x. Unlike conventional downsampling, e.g., max-pooling, *spectral downsampling* [308, 312] considers the spectral

content of the input during downsampling, and thus prevents *aliasing* –where the output resolution is insufficient to accurately represent the underlying signal (Fig. 6.6, middle down). This is important since it has been shown that aliasing has negative effects on robustness [468], generation [180] and generalization [401].

**Combining learnable downsampling and convolution.** The previous method requires mapping inputs to the Fourier domain and back to learn downsampling. Fortunately, CCNNs as well as most methods that rely on global convolutions, e.g., CKConv [321], S4 [129], rely on the *Fourier convolution theorem*:  $(f * \psi) = \mathcal{F}^{-1}[\mathcal{F}[f] \cdot \mathcal{F}[\psi]]$  to compute convolutions with large kernels efficiently. This means that CCNNs already use a Fourier and inverse Fourier transforms in each residual block to compute convolutions. Hence, we can avoid recomputing these steps by placing the learnable downsampling operation *within the Fourier convolution*. Specifically, we simultaneously compute downsampling and convolution by applying the differentiable mask  $m_{\text{sigm}}$  and the cropping operations  $\text{crop}_{>\omega_{\max}}$  before returning from the Fourier domain back to the spatial domain:<sup>1</sup>

$$(f * \psi)_{\text{down}} = \mathcal{F}^{-1} [\text{crop}_{>\omega_{\max}}(m_{\text{sigm}}(\cdot; \theta) \cdot \mathcal{F}[f] \cdot \mathcal{F}[\psi])]. \quad (6.7)$$

**Materializing functions only on the output resolution.** Note that Eq. 6.7 computes the convolution on the resolution of the input and downsamples next. This incurs in an unnecessary overhead as the output of the convolution will be downsampled directly after. A more efficient approach comes from inverting the order of these operations to compute the convolution at the downsampled resolution. Luckily, this can be achieved by using the method outlined in Sec. 6.2.1. Since the cutoff frequency of the mask corresponds to the coordinate at which the mask equals the threshold, i.e.,  $\omega_{\max} = x_{T_m}$ , it can be calculated using Eq. 6.5. Next, since the cutoff frequency defines the minimum resolution required to faithfully represent the input, we can downsample the input and the kernel to that resolution before the convolution to compute it on the output resolution.

**Learning subsampling for dense tasks.** Riad et al. [308] apply learned downsampling on both the identity and the residual branches of a residual block to limit resolution of all representations after a specific residual block. In the context of DNArch, this is undesirable for two reasons: First, as we learn the whole network architecture during training, it is not known *a priori* what resolution mappings will require at each layer. However, forcing the identity branch to have the same resolution as the corresponding residual branch restricts all subsequent mappings to be of maximum that resolution. Secondly, dense prediction tasks, e.g., segmentation, require the learned architecture to produce outputs that share the same resolution as the input. However, if the identity branch is also downsampled, the output of the network would be of lower resolution even for a single level of downsampling in the network. This in turn, would result in over-smoothed predictions.

---

<sup>1</sup>We note that the Fourier transform is not strictly necessary to learn downsampling, e.g., for CNNs with local kernels. Leveraging the Fourier convolution theorem, equivalent downsampling can be achieved by convolving the input with the inverse Fourier transform of the mask in the spatial domain (see Appx. E.3).

Based on these observations, we use downsampling *only* on the residual branch and upsample features at the end of each residual block back to the resolution of the input. This allows us to (i) have features at same resolution as the input in the last layer, and (ii) learn U-Net [323] like architectures.

### Learning the width of the network at each layer

We learn the width of a layer by applying a differentiable mask  $m(\cdot; \theta)$  along the channel dimension of feature representations (see Fig. 6.1). The network can then adjust the width of its representations by changing the value of  $\theta$ . By using the method in Sec. 6.2.1 only non-masked channels are rendered.

**Positioning of the width masks.** We aim to learn the width of all layers in a network. To this end, we apply differentiable masks with independent learnable parameters along the channel dimensions of all the network components that change the network’s width, i.e., all Conv and PWLinear layers. This corresponds to learning three independent masks for each residual block in the network, which correspond to the input ( $N_{in}$ ), the middle ( $N_{mid}$ ) and the output ( $N_{out}$ ) channels of the residual block (Fig. 6.7). Components that do not change the width, e.g., BatchNorm, GELU, have their width determined by the preceding mask.

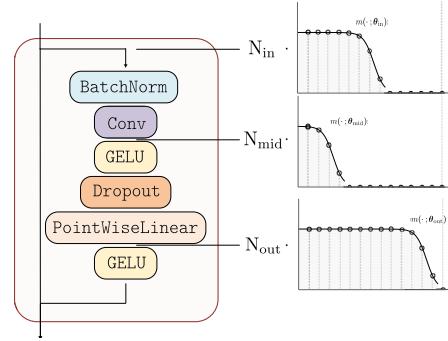


Figure 6.7: Positioning of width masks.

**The advantage of avoiding masks on the identity branch.** Applying a differentiable mask on the output of the entire residual block to constrain its width, i.e., after the sum of the residual and identity branches, would accumulate the effect of all masks applied before that block. As a result, the  $l$ -th block would be effectively masked by the combination of all masks before that block, i.e.,  $\prod_{i \leq l} m_i$ , with  $m_i$  the mask after the  $i$ -th block. Since the values of the masks live in the  $[0, 1]$  interval, this would result in an exponential decrease in the magnitude of the hidden feature presentations. To avoid this, we apply the masks on the residual branch only. In addition, keeping the identity branch intact allows DNArc to construct DenseNet-like architectures [161], where blocks can reuse channels that only have been modified by some –or none– of the previous blocks.

### Learning the depth of the network

We learn the network’s depth by viewing the number of residual blocks as a continuous axis with values  $[1, 2, \dots, D]$  corresponding to the index of each block, and using a differentiable mask  $m(\cdot, \theta)$  along this axis to dynamically mask out blocks based on the value of the mask parameters  $\theta$  (Fig. 6.8).

**Positioning of the depth mask.** To ensure that information flows from the input to the output of the network regardless of the value of the mask parameters, we *only* apply the mask on the residual branch. If the mask were also applied on the identity branch, feature representations at the end of the network could become zero, and the network would only be able to output random predictions.

### Learning entire convolutional architectures by backpropagation

By simultaneously using the methods outlined in this section, DNArc uses backpropagation to learn the weights, the size of convolutional kernels at each layer, the number of channels at each layer, the position and resolution of downsampling layers, and the depth of a convolutional network.

#### 6.2.5 Learning neural architectures under computational constraints

We can ensure that the architectures searched by DNArc respect a predefined computational complexity by including an additional regularization term  $\mathcal{L}_{\text{comp}}$  that reflects the complexity of the current candidate architecture based on its mask parameters. To this end, we define the optimization loss  $\mathcal{L}$  as the sum of the task objective loss  $\mathcal{L}_{\text{obj}}$  and the complexity loss  $\mathcal{L}_{\text{comp}}$  weighted by a factor  $\lambda$ :

$$\mathcal{L} = \mathcal{L}_{\text{obj}} + \lambda \mathcal{L}_{\text{comp}}. \quad (6.8)$$

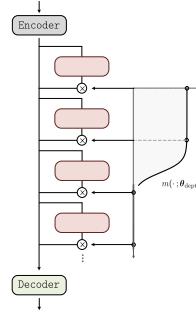
By minimizing this loss, DNArc is encouraged to find architectures that meet the desired computational budget while still achieving good performance on the end task.

##### Defining the complexity loss $\mathcal{L}_{\text{comp}}$

The purpose of  $\mathcal{L}_{\text{comp}}$  is to use the size of the learned masks to estimate the total computation needed for a forward pass of the network. Its construction is outlined below.

**Layer-wise complexities.** Let  $C_{\text{layer}}(L, N_{\text{in}}, N_{\text{out}})$  be the number of operations required in a given layer with an input of length  $L$  and  $N_{\text{in}}$  and  $N_{\text{out}}$  input and output channels. To estimate the number of computations required based on the current size of the masks, we can substitute the length of each dimension with the size of the corresponding masks:  $C_{\text{layer}}(\text{size}(m_{\text{res}}), \text{size}(m_{N_{\text{in}}}), \text{size}(m_{N_{\text{out}}}))$ . As an example, consider a pointwise linear layer  $\text{lin} : \mathbb{R}^{N_{\text{in}}} \rightarrow \mathbb{R}^{N_{\text{out}}}$ . It takes an input  $f$  of length  $L$  and  $N_{\text{in}}$  channels and multiplies each element along the spatial dimensions of the input with a matrix of dimensions  $[N_{\text{in}}, N_{\text{out}}]$  to produce an output of the same length, but with  $N_{\text{out}}$  number of channels. The total operations required in this layer is given by  $C_{\text{lin}}(f) = L \cdot N_{\text{in}} \cdot N_{\text{out}}$ .

Now, let us use three differentiable masks  $m_{\text{res}}$ ,  $m_{N_{\text{in}}}$  and  $m_{N_{\text{out}}}$  to mask the resolution,



**Figure 6.8:** Learning network's depth with differential masking.

input and output channels of the linear layer. The total number of computations is now:

$$\mathcal{C}_{\text{lin},\text{masked}} = \text{size}(m_{\text{res}}) \cdot \text{size}(m_{N_{\text{in}}}) \cdot \text{size}(m_{N_{\text{out}}}).$$

Since the size of the masks is now involved in the computation of the operations required, we can utilize it as an additional source of feedback to update the masks by making the function size differentiable with regard to the mask parameters. The same concept is used to calculate the cost of other layers based on the size of the masks. A summary of these costs can be found in Appx. E.4.

**Effect of the depth mask.** To take into account the effect of the depth mask, we use it to determine the number of residual blocks in the network. If the number of operations of a network with  $D$  residual blocks is denoted as  $\mathcal{C}_{\text{net},D}$ , the complexity of a network with masked depth is given by  $\mathcal{C}_{\text{net},\text{size}(m_{\text{depth}})}$  with  $\text{size}(m_{\text{depth}})$  the size of the depth mask.

**Computing the size of the masks.** The size of a mask can be calculated in a differentiable manner by determining the length of the mask in continuous space and using that length to estimate the change in size of the corresponding network dimension. Specifically, the length at a time  $t$  is  $2x_{T_m}^t$  and  $x_{T_m}^t - x_{\min}$ , for Gaussian and Sigmoid masks, respectively (see Fig. 6.3). For some initial  $x_{T_m}^0$  and corresponding initial length  $N$ , the size of a Gaussian and a Sigmoid mask at time  $t$  is respectively:

$$\text{size}(m_{\text{gauss}}) = \frac{2x_{T_m}^t}{2x_{T_m}^0} N, \quad (6.9) \quad \text{size}(m_{\text{sigm}}) = \frac{x_{T_m}^t - x_{\min}}{x_{T_m}^0 - x_{\min}} N. \quad (6.10)$$

**Computational constraints as an additional loss.** Let  $\mathcal{C}_{\text{curr}}$  be the current complexity of the network and  $\mathcal{C}_{\text{target}}$  be the desired target complexity. We define the computational loss  $\mathcal{L}_{\text{comp}}$  as the relative  $\ell^2$  difference between the relative complexity of the current network and the target:

$$\ell^2 \left( \frac{\mathcal{C}_{\text{curr}}}{\mathcal{C}_{\text{target}}}, 1 \right) = \left\| \frac{\mathcal{C}_{\text{curr}}}{\mathcal{C}_{\text{target}}} - 1.0 \right\|_2^2. \quad (6.11)$$

This form has two advantages over the alternative form  $\ell^2(\mathcal{C}_{\text{curr}}, \mathcal{C}_{\text{target}})$ . It (i) prevents overflow that might occur when comparing large values  $-\mathcal{C}_{\text{curr}}$  and  $\mathcal{C}_{\text{target}}$  may easily be of order  $10^{-10}$ , and (ii) allows for consistent tuning of  $\lambda$  for different tasks and complexities. In the alternative form  $\ell^2(\mathcal{C}_{\text{curr}}, \mathcal{C}_{\text{target}})$ ,  $\lambda$  might need to be tuned independently for different complexity regimes.

### 6.3 Experiments

We evaluate DNArc on sequential and image datasets for classification and dense prediction tasks. On 1D, we use the Long Range Arena (LRA) benchmark [379], which includes six sequence modelling tasks with sequence lengths ranging from 1024 to over 16000. On 2D, we perform image classification on the CIFAR10 and CIFAR100 datasets [195] and report results on two dense prediction tasks from the NAS-Bench-360 benchmark [387]: DarcyFlow [220] and Cosmic [466]. A detailed description of the datasets

**Table 6.1:** Performance on the LRA benchmark.  $\times$  denotes random guessing. Highest per-section scores are in bold and the overall best scores are underlined. For DNArch, values in parenthesis indicate the computational cost of the architecture relative to the target complexity.

| MODEL  | LISTOPS                              | TEXT                                 | RETRIEVAL                            | IMAGE                                | PATHFINDER                           | PATH-X                               | AVG.         |
|--|--------------------------------------|--------------------------------------|--------------------------------------|--------------------------------------|--------------------------------------|--------------------------------------|--------------|
| Transformer [403]                                  | <b>36.37</b>                         | 64.27                                | 57.46                                | 42.44                                | 71.40                                | $\times$                             | 53.66        |
| Reformer [189]                                     | 37.27                                | 56.10                                | 53.40                                | 38.07                                | 68.50                                | $\times$                             | 50.56        |
| Performer [59]                                     | 18.01                                | <b>65.40</b>                         | 53.82                                | <b>42.77</b>                         | <b>77.05</b>                         | $\times$                             | 51.18        |
| BigBird [458]                                      | 36.05                                | 64.02                                | <b>59.29</b>                         | 40.83                                | 74.87                                | $\times$                             | <b>54.17</b> |
| Mega ( $\mathcal{O}(L^2)$ ) [244]                  | <b>63.14</b>                         | <b>90.43</b>                         | <b>91.25</b>                         | <b>90.44</b>                         | <b>96.01</b>                         | <b>97.98</b>                         | <b>88.21</b> |
| Mega-chunk ( $\mathcal{O}(L)$ ) [244]              | 58.76                                | 90.19                                | 90.97                                | 85.80                                | 94.41                                | 93.81                                | 85.66        |
| S4D [130]  | 60.47                                | 86.18                                | 89.46                                | 88.19                                | 93.06                                | 91.95                                | 84.89        |
| S4 [129]   | 59.60                                | 86.82                                | 90.90                                | <b>88.65</b>                         | 94.20                                | 96.35                                | 86.09        |
| S5 [360]   | <b>61.50</b>                         | <b>89.31</b>                         | <b>91.40</b>                         | 88.00                                | <b>95.33</b>                         | <b>98.58</b>                         | <b>87.35</b> |
| FNet [213]   | 35.33                                | 65.11                                | 59.61                                | 38.67                                | 77.80                                | $\times$                             | 54.42        |
| Luna-256 [243]                                     | 37.25                                | 64.57                                | 79.29                                | 47.38                                | 77.72                                | $\times$                             | 59.37        |
| CCNN <sub>4,140</sub> [320]                        | <b>44.85</b>                         | <b>83.59</b>                         | $\times$                             | <b>87.62</b>                         | <b>91.36</b>                         | $\times$                             | <b>76.86</b> |
| CCNN <sub>4,140</sub> (Global Kernels)             | 55.65                                | 87.80                                | 90.55                                | 85.51                                | 94.26                                | 91.15                                | 84.15        |
| DNArch <sub>K</sub> (CCNN <sub>4,140</sub> )       | 59.90                                | 88.28                                | 90.66                                | 86.07                                | 93.46                                | 89.93                                | 84.72        |
| DNArch <sub>K,R</sub> (CCNN <sub>4,140</sub> )     | 60.15 <sub>(0.80\times)</sub>        | 88.50 <sub>(0.75\times)</sub>        | 91.08 <sub>(0.78\times)</sub>        | 86.55 <sub>(0.82\times)</sub>        | 94.05 <sub>(0.89\times)</sub>        | 91.15 <sub>(0.82\times)</sub>        | 85.25        |
| DNArch <sub>K,R,W,D</sub> (CCNN <sub>4,140</sub> ) | <b>60.55</b> <sub>(1.01\times)</sub> | <b>89.03</b> <sub>(1.00\times)</sub> | <b>91.22</b> <sub>(1.02\times)</sub> | <b>87.20</b> <sub>(1.02\times)</sub> | <b>94.95</b> <sub>(1.00\times)</sub> | <b>91.71</b> <sub>(1.01\times)</sub> | <b>85.78</b> |

used can be found in Appx. E.5.

**Experimental setup.** We use two CNNs of different capacity as base networks: a CCNN<sub>4,140</sub> –4 blocks, 140 channels, 200K parameters–, and a CCNN<sub>6,380</sub> –6 blocks, 380 channels, 2M parameters–, and use DNArch to learn their architectures. To understand the impact of learning each network component, we also report results learning some and none of the neural architecture components.

**Mask configurations.** We initialize all the masks to match the architecture of the baseline CNNs at the beginning of training. We use a Gaussian mask to learn kernel sizes as in FlexConv [319], and Sigmoid masks to learn width, depth and downsampling. All masks use a threshold of  $T_m=0.1$ . All kernel masks are centered, i.e.,  $\mu=0$ , and initialized to either be small or global, i.e.,  $\sigma \in [0.0325, 0.5]$ . Resolution masks are initialized to weight the highest input frequency by 0.85, and the width and depth masks are initialized to match the size of the base network’s architecture. More information on hyperparameters, training regimes, and experimental settings can be found in Appx. E.6.

**Notations.** We use DNArch as an operator acting on a base network and specify the learned components with indices K, R, W, D representing kernel sizes, downsampling, width and depth. DNArch<sub>K</sub>(CCNN<sub>4,140</sub>) indicates using DNArch to learn only the kernel sizes of a CCNN<sub>4,140</sub>.

### 6.3.1 Using DNArch without computational constraints

First, we use DNArch to improve the expressiveness and computational efficiency of a CCNN<sub>4,140</sub>. We start using DNArch to learn the receptive field of all convolutional

**Table 6.2:** Dense prediction results.

| MODEL  | DARCYFLOW<br>rel. $\ell^2$ loss    | COSMIC<br>1 - AUROC             |
|--|------------------------------------|---------------------------------|
| Expert*  | <b>0.008</b>                       | <b>0.13</b>                     |
| WRN [456]  | 0.073                              | 0.24                            |
| DenseNAS [103]                                     | 0.100                              | 0.38                            |
| DARTS [231]  | <b>0.026</b>                       | 0.229                           |
| Auto-DL [229]                                      | 0.049                              | 0.495                           |
| DASH [350]   | 0.060                              | <b>0.190</b>                    |
| CCNN <sub>4,140</sub> (Global Kernels)             | 0.002989                           | 0.059                           |
| DNArch <sub>K</sub> (CCNN <sub>4,140</sub> )       | 0.002970                           | 0.058                           |
| DNArch <sub>K,R</sub> (CCNN <sub>4,140</sub> )     | 0.002929 <sub>(0.79×)</sub>        | 0.056 <sub>(0.82×)</sub>        |
| DNArch <sub>K,R,W,D</sub> (CCNN <sub>4,140</sub> ) | <b>0.002285</b> <sub>(1.01×)</sub> | <b>0.055</b> <sub>(1.01×)</sub> |
| CCNN <sub>6,380</sub> (Global Kernels)             | 0.004521                           | 0.059                           |
| DNArch <sub>K,R,W,D</sub> (CCNN <sub>6,380</sub> ) | <b>0.001763</b> <sub>(1.00×)</sub> | <b>0.048</b> <sub>(1.00×)</sub> |

\* FNO [220] and deepCR [466].

**Table 6.3:** Image classification results.

| MODEL  | CIFAR10                         | CIFAR100                        |
|--|---------------------------------|---------------------------------|
| WRN [456]  | -                               | <b>76.65</b>                    |
| DenseNAS [103]                                     | -                               | 74.51                           |
| DARTS [231]  | -                               | 75.98                           |
| Auto-DL [229]                                      | -                               | -                               |
| DASH [350]   | -                               | 75.63                           |
| CCNN <sub>4,140</sub> (Global Kernels)             | 90.52                           | 64.72                           |
| DNArch <sub>K</sub> (CCNN <sub>4,140</sub> )       | 92.51                           | 69.01                           |
| DNArch <sub>K,R</sub> (CCNN <sub>4,140</sub> )     | 92.77 <sub>(0.82×)</sub>        | 68.96 <sub>(0.85×)</sub>        |
| DNArch <sub>K,R,W,D</sub> (CCNN <sub>4,140</sub> ) | <b>93.47</b> <sub>(1.01×)</sub> | <b>72.98</b> <sub>(1.03×)</sub> |
| CCNN <sub>6,380</sub> (Global Kernels)             | 94.18                           | 72.29                           |
| DNArch <sub>K,R,W,D</sub> (CCNN <sub>6,380</sub> ) | <b>95.03</b> <sub>(1.00×)</sub> | <b>76.37</b> <sub>(1.02×)</sub> |

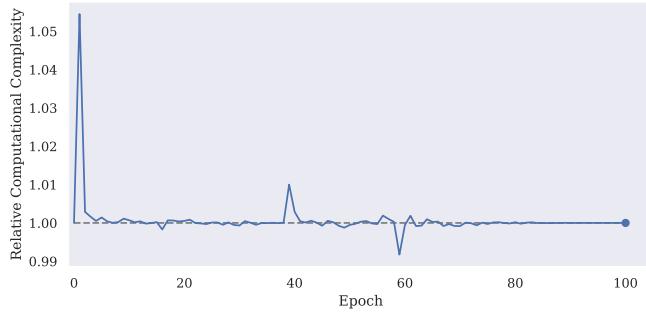
layers, and then we learn both the kernel sizes and downsampling layers to simultaneously improve the expressiveness and the computational efficiency of the CCNN<sub>4,140</sub>. It is worth noting that in this scenario learning is solely driven by the objective loss  $\mathcal{L}_{\text{obj}}$ , i.e., the regularization term  $\mathcal{L}_{\text{comp}}$  is not used. In addition, as we use Fourier convolutions, the learned kernel sizes do not impact computational efficiency.

**Results.** Except for PathFinder and Path-X, we find that using DNArch to learn kernel sizes consistently improves the accuracy of the base architecture (DNArch<sub>K</sub> models in Tabs. 6.1-6.3). Interestingly, found DNArch architectures perform on par, and even surpass, architectures specifically designed for each tasks, e.g., S4 [129] for sequential tasks and NFOs [220] for PDEs on 2D with a remarkably lower number of trainable parameters. In contrast to DARTS methods, e.g., DASH [350], DNArch can be applied across *all tasks* without the need to *manually change* the base architecture. When additionally learning downsampling, we observe that DNArch finds high-performant architectures with improved computational efficiency (DNArch<sub>K,R</sub> models in Tabs. 6.1-6.3). Interestingly, we observe that the found models often exhibit slight accuracy improvements. This is explained by low resolution kernels being easier to model and construct than higher resolution ones.

### 6.3.2 Using DNArch under computational constraints

Next, we utilize DNArch to learn entire convolutional architectures that respect a predefined computational budget. To this end, we start with base CCNN<sub>4,140</sub> and CCNN<sub>6,380</sub> networks, and allow DNArch to learn their width, depth, kernel sizes and downsampling. We define the target complexity  $\mathcal{L}_{\text{comp}}$  as the complexity of the base CCNN networks. In other words, we use DNArch to find better convolutional architectures of computational complexity roughly equal to that of the base networks.

**Results.** Our results (DNArch<sub>K,R,W,D</sub> models in Tabs. 6.1-6.3) show that DNArch finds



**Figure 6.9:** Relative complexity during the course of training on the Text task. This behavior is consistent across all tasks.

neural architectures that achieve higher accuracy than the base CNN networks while keeping the same computational complexity. In addition, we observe that learning more neural architecture components consistently leads to better results, therefore supporting the claim that using gradient-steered architectures can be more beneficial than using handcrafted ones. Furthermore, we observe that using base architectures with larger complexity and capacity consistently leads to better results. This result is encouraging for the application of DNArc to large architectures, e.g., LLMs [34, 60].

**Computational complexity of DNArc.** To assess the applicability of DNArc, it is important to analyze its computational overhead. To this end, we analyze the behavior of the relative complexity ( $C_{\text{curr}}/C_{\text{target}}$ ) during training (Fig. 6.9). Interestingly, we observe that the theoretical complexity of candidate architectures  $C_{\text{curr}}$  stays close to the target complexity  $C_{\text{target}}$  *during the whole training*. This indicates that: (i) DNArc only searches among architectures that share the target computational complexity, and that (ii) the computational overhead of DNArc is *negligible*. As a result, the cost of using DNArc on top of a CNN is comparable to the cost of training the base CNN network. Note that, for the experiments in Sec. 6.3.1, the cost of training can be even lower than that of the base network since the complexity of found architectures are up to 25% faster.

### 6.3.3 Architectures found by DNArc

The architectures found by DNArc are listed in Tabs. E.1-E.3. Interestingly, we observe that found architectures are very diverse, even within each architecture. For instance, some residual blocks have a bottleneck structure, some an expanded structure, and others have monotonically decreasing or increasing widths. Interestingly, the resolution of found architectures for classification tasks, e.g., Text, often follow the style of U-Nets, and not the monotonically decreasing pattern commonly seen in handcrafted networks. On dense tasks, we observe architectures that resemble U-Nets, and even concatenated U-Nets, e.g., the 1.5× U-Net like architecture found for the Cosmic task.

The kernel sizes found by DNArc are also very diverse. In 1D tasks, found kernels are often large, which would make them parameter intensive with traditional parameteri-

zations. In 2D tasks, we often see rectangular kernels that do not follow a monotonic pattern of increasing or decreasing sizes. Instead, found architectures often perform interleaved low-level and high-level feature extraction.

## 6.4 Limitations

**Training on TPU requires static shapes.** We train our models on TPUs, a type of accelerator that requires a static computational graph derived for specific input and network shapes via the XLA (Accelerated Linear Algebra) compiler. As a result, TPUs do not support operations that change the shapes of arrays during training. This means that on TPUs, DNArch can only perform masking modifications to the network during training, i.e., setting certain channels to zero but still computing their outputs. At inference, however, the masks are fixed. Consequently, we can effectively trim unused values to remove useless computations in a way that is compatible with XLA. It is important to note that this limitation is solely an implementation issue caused by nature of TPUs' hardware and can be avoided by using libraries and hardware that support dynamic computational graphs, e.g., PyTorch and GPUs. While our results were obtained using TPUs, we also provide a PyTorch implementation that avoids this issue, making it more flexible and accessible, especially in scenarios where one needs to keep candidate networks close to the target complexity  $C_{\text{target}}$  during training.

**DNArch requires instantiating the largest possible architecture.** While masking weights through a gradient update is straightforward, increasing the number of active weights requires those weights to be instantiated in memory. This means that even with dynamic computational graphs, it is necessary to instantiate the largest possible architecture learnable by DNArch in memory. To overcome this limitation, we set the maximum kernel size to the size of the input, and limit the maximum network size along the depth and width dimensions to double the number of blocks and channels of the base network. While this trick allows DNArch to easily shrink and grow representations within that range, this restricts the potential sizes of optimal architectures and can restrict the applicability of DNArch to very large models, e.g., LLMs [34, 60].

## 6.5 Outlook and future work

**Input-dependent neural architectures.** In this work, the mask parameters are constant for all inputs within a task. An alternative approach could use an additional neural network  $\text{MLP}_{\text{mask}}$  to predict the mask parameters based on context, e.g., the current input, current task, etc. This would enable the creation of context-dependent neural architectures such as early-exit systems [116, 342, 380], but where the whole network architecture is context-dependent. Consequently, resulting architectures would provide finer control of per-sample / per-modality complexity than existing methods.

**Dynamic weighting of  $\mathcal{L}_{\text{comp}}$  during training.** DNArch explores architectures with

complexity similar to target complexity throughout training. This results from using a constant  $\lambda$  in Eq. 6.8. Alternatively, one could use a dynamic value of  $\lambda$  during training to induce a different training behavior. For example, gradually increasing  $\lambda$  would allow DNArch to explore architectures with larger complexity at first, and progressively encourage it to converge to networks with the desired target complexity. Such a weighting scheduling of  $\lambda$  could lead DNArch to find better architectures.

**Training DNArch with additional / multiple constraints.** Here, we only consider computational complexity as a constraint when training with DNArch. However, other properties such as memory efficiency, hardware-awareness and robustness are equally important. Designing regularization terms that encourage other properties in DNArch as well as exploring how different properties can be optimized in unison are important directions for further research.



## Part II

# EFFICIENCY THROUGH SYMMETRY PRESERVATION



# 7

## Focusing Equivariance on Transformations Co-Occurring in Data

*Based on the paper:*

*Co-Attentive Equivariant Neural Networks: Focusing Equivariance On Transformations Co-Occurring In Data* [315]

### 7.1 Introduction

Thorough experimentation in the fields of psychology and neuroscience has provided support to the intuition that our visual perception and cognition systems are able to identify familiar objects despite modifications in size, location, background, viewpoint and lighting [35]. Interestingly, we are not just able to recognize such modified objects, but are able to characterize which modifications have been applied to them as well. As an example, when we see a picture of a cat, we are not just able to tell that there is a cat in it, but also its position, its size, facts about the lighting conditions of the picture, and so forth. Such observations suggest that the human visual system is *equivariant* to a large *transformation group* containing translation, rotation, scaling, among others. In other words, the mental representation obtained by seeing a transformed version of an object, is equal to that of seeing the original object and transforming it mentally next.

These fascinating abilities exhibited by biological visual systems have inspired a large field of research towards the development of neural architectures able to replicate them. Among these, the most popular and successful approach is the Convolutional Neural Network (CNN) [206], which incorporates equivariance to translation via convolution.

Unfortunately, in counterpart to the human visual system, CNNs do not exhibit equivariance to other transformations encountered in visual data (e.g., rotations). Interestingly, however, if an ordinary CNN happens to learn rotated copies of the same filter, the stack of feature maps becomes equivariant to rotations even though individual feature maps are not [65]. Since ordinary CNNs must learn such rotated copies independently, they effectively utilize an important number of network parameters suboptimally to this end (see Fig. 3 in Krizhevsky et al. [196]). Based on the idea that equivariance in CNNs can be extended to larger transformation groups by stacking convolutional feature maps, several approaches have emerged to extend equivariance to, e.g., planar rotations [89, 216, 252, 424], spherical rotations [67, 69, 432], scaling [253, 433] and general transformation groups [65], such that transformed copies of a single entity are not required to be learned independently.

Although incorporating equivariance to arbitrary transformation groups is conceptually and theoretically similar<sup>1</sup>, evidence from real-world experiences motivating their integration might strongly differ. Several studies in neuroscience and psychology have shown that our visual system does not react equally to all transformations we encounter in visual data. Take, for instance, translation and rotation. Although we easily recognize objects independently of their position of appearance, a large corpus of experimental research has shown that this is not always the case for in-plane rotations. Yin [453] showed that *mono-oriented objects*, i.e., complex objects such as faces which are customarily seen in one orientation, are much more difficult to be accurately recognized when presented upside-down. This behaviour has been reproduced, among others, for magazine covers [77], symbols [145] and even familiar faces (e.g., from classmates) [33]. Intriguingly, Schwarzer [345] found that this effect exacerbates with age (adults suffer from this effect much more than children), but, adults are much faster and accurate in detecting mono-oriented objects in usual orientations.

Based on these studies, we draw the following conclusions:

- The human visual system does not perform (fully) equivariant feature transformations to visual data. Consequently, it does not react equally to all possible input transformations encountered in visual data, even if they belong to the same transformation group (e.g., in-plane rotations).
- The human visual system does not just encode familiarity to objects but seems to learn through experience the poses in which these objects customarily appear in the environment to assist and improve object recognition [110, 310, 357].

Complementary studies [276, 378] suggest that our visual system encodes orientation atypicality relative to the context rather than on an absolute manner (Fig. 7.1). Motivated by the aforementioned observations we state *the co-occurrence envelope hypothesis*:

---

<sup>1</sup>It is achieved by developing feature mappings that use the transformation group in the feature mapping itself, e.g., translating a filter during a feature transformation is used to obtain translation equivariance.

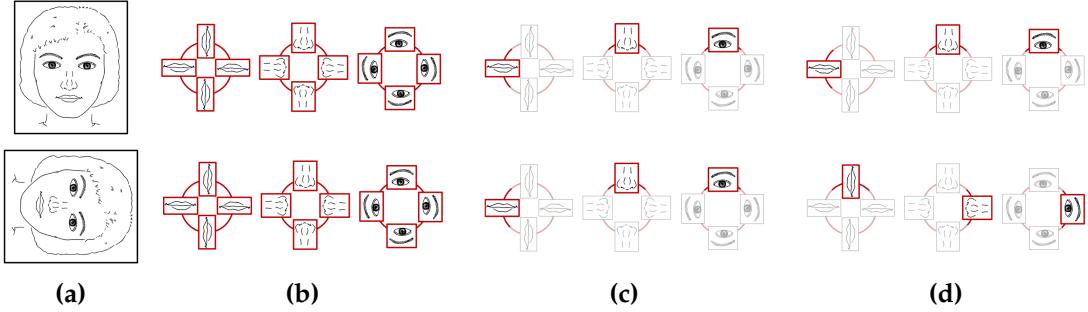


**Figure 7.1:** Our visual system infers object identities according to their size, location and orientation. In this blurred picture, observers describe the scene as containing a car and a pedestrian in the street. However, the pedestrian is in fact the *same shape* as the car, except for a 90° rotation. The atypicality of this orientation *within the context defined by the street scene* causes the car to be recognized as a pedestrian. Taken from [276].

**The Co-occurrence Envelope Hypothesis.** *By allowing equivariant feature mappings to detect transformations that co-occur in the data and focus learning on the set formed by these co-occurrent transformations (i.e., the co-occurrence envelope of the data), one is able to induce learning of more representative feature representations of the data, and, resultantly, enhance the descriptive power of neural networks utilizing them. We refer to one such feature mapping as co-attentive equivariant.*

**Identifying the co-occurrence envelope.** Consider a rotation equivariant network receiving two copies of the same face (Fig. 7.2a). A conventional rotation equivariant network is required to perform inference and learning on the set of all possible orientations of the visual patterns constituting a face regardless of the input orientation (Fig. 7.2b). However, by virtue of its rotation equivariance, it is able to recognize rotated faces even if it is trained on upright faces only. A possible strategy to simplify the task at hand could be to restrict the network to react exclusively to upright faces (Fig. 7.2c). In this case, the set of relevant visual pattern orientations becomes much smaller, at the expense of disrupting equivariance to the rotation group. Resultantly, the network would risk becoming unable to detect faces in any other orientation than those it is trained on. A better strategy results from restricting the set of relevant pattern orientations by defining them relative to one another (e.g., mouth orientation relative to the eyes) as opposed to absolutely (e.g., upright mouth) (Fig. 7.2d). In such a way, we are able to exploit information about orientation co-occurrences in the data without disrupting equivariance. The set of co-occurrent orientations in Fig. 7.2d corresponds to the co-occurrence envelope of the samples in Fig. 7.2a for the transformation group defined by rotations.

In this work, we introduce *co-attentive equivariant feature mappings* and apply them on existing equivariant neural architectures. To this end, we leverage the concept of *at-*



**Figure 7.2:** Effect of multiple attention strategies for the prioritization of relevant pattern orientations in rotation equivariant networks for the task of face recognition. Given that all attention strategies are learned exclusively from upright faces, we show the set of relevant directions for the recognition of faces in two orientations (Fig. 7.2a) obtained by: no attention (Fig. 7.2b), attending to the pattern orientations of appearance independently (Fig. 7.2c) and, attending to the pattern orientations of appearance relative to one another (Fig. 7.2d). Built upon Figure 1 from Schwarzer [345].

tention [10] to modify existing mathematical frameworks for equivariance, such that co-occurring transformations can be detected. It is critical not to disrupt equivariance in the attention procedure as to preserve it across the entire network. To this end, we introduce *cyclic equivariant self-attention*, a novel attention mechanism able to preserve equivariance to cyclic groups.

**Experiments and results.** We explore the effects of co-attentive equivariant feature mappings for single and multiple symmetry groups. Specifically, we replace conventional rotation equivariant mappings in  $p4$ -CNNs [65] and DRENs [216] with co-attentive ones. We show that *co-attentive rotation equivariant neural networks* consistently outperform their conventional counterparts in fully (rotated MNIST) and partially (CIFAR-10) rotational settings. Subsequently, we generalize cyclic equivariant self-attention to multiple similarity groups and apply it on  $p4m$ -CNNs [65] (equivariant to rotation and mirror reflections). Our results are in line with those obtained for single symmetry groups and support our stated hypothesis.

### Contributions.

- We propose the *co-occurrence envelope hypothesis* and demonstrate that conventional equivariant mappings are consistently outperformed by our proposed *co-attentive equivariant* ones.
- We generalize co-attentive equivariant mappings to multiple symmetry groups and provide, to the best of our knowledge, the first attention mechanism acting generally on symmetry groups.

## 7.2 Preliminaries

**Equivariance.** We say that a feature mapping  $f : \mathcal{X} \rightarrow \mathcal{Y}$  is equivariant to a (transformation) group  $\mathcal{G}$  (or  $\mathcal{G}$ -equivariant) if it commutes with actions of the group  $\mathcal{G}$  acting on its domain and codomain:

$$f(T_g^{\mathcal{X}}(x)) = T_g^{\mathcal{Y}}(f(x)) \quad \forall g \in \mathcal{G}, x \in \mathcal{X}, \quad (7.1)$$

where  $T_g^{(\cdot)}$  denotes a *group action* in the corresponding space. In other words, the ordering in which we apply a group action  $T_g$  and the feature mapping  $f$  is inconsequential. There are multiple reasons as of why equivariant feature representations are advantageous for learning systems. Since group actions  $T_g^{\mathcal{X}}$  produce predictable and interpretable transformations  $T_g^{\mathcal{Y}}$  in the feature space, the *hypothesis space of the model* is reduced [424] and the learning process simplified [434]. Moreover, equivariance allows the construction of  $L$ -layered networks by stacking several equivariant feature mappings  $\{f^{(1)}, \dots, f^{(l)}, \dots, f^{(L)}\}$  such that the input structure as regarded by the group  $G$  is preserved (e.g., CNNs and input translations). As a result, any intermediate network representation  $(f^{(l)} \circ \dots \circ f^{(1)})(x), l \in L$ , is able to take advantage of the structure of  $x$  as well. *Invariance* is a special case of equivariance in which  $T_g^{\mathcal{Y}} = \text{Id}_{\mathcal{Y}}$ , the identity, and thus all group actions in the input space are mapped to the same feature representation.

**Equivariant neural networks.** In neural networks, the integration of equivariance to arbitrary groups  $G$  has been achieved by developing feature mappings  $f$  that utilize the actions of the group  $G$  in the feature mapping itself. Interestingly, *equivariant feature mappings* encode equivariance as *parameter sharing* with respect to  $G$ , i.e., the same weights are reused for all  $g \in G$ . This makes the inclusion of larger groups extremely appealing in the context of parameter efficient networks.

Conventionally, the  $l$ -th layer of a neural network receives a signal  $x^{(l)}(u, \lambda)$  (where  $u \in \mathbb{Z}^2$  is the spatial position and  $\lambda \in \Lambda_l$  is the unstructured channel index, e.g., RGB channels in a color image), and applies a feature mapping  $f^{(l)} : \mathbb{Z}^2 \times \Lambda_l \rightarrow \mathbb{Z}^2 \times \Lambda_{l+1}$  to generate the feature representation  $x^{(l+1)}(u, \lambda)$ . In CNNs, the feature mapping  $f^{(l)} := f_T^{(l)}$  is defined by a *convolution*<sup>2</sup> ( $\star_{\mathbb{R}^2}$ ) between the input signal  $x^{(l)}$  and a learnable convolutional filter  $W_{\lambda', \lambda}^{(l)}, \lambda' \in \Lambda_l, \lambda \in \Lambda_{l+1}$ :

$$x^{(l+1)}(u, \lambda) = [x^{(l)} \star_{\mathbb{R}^2} W_{\lambda', \lambda}^{(l)}](u, \lambda) = \sum_{\lambda', u'} x^{(l)}(u + u', \lambda') W_{\lambda', \lambda}^{(l)}(u'). \quad (7.2)$$

By sliding  $W_{\lambda', \lambda}^{(l)}$  across  $u$ , CNNs preserve the spatial structure of the input through the mapping  $f_T^{(l)}$  and successfully provide equivariance to the translation group  $T = (\mathbb{Z}^2, +)$ .

The underlying idea for the extension of equivariance to larger groups in CNNs is conceptually equivalent to the strategy utilized by [206] for translation equivariance. Con-

---

<sup>2</sup>Formally it is as a correlation. However, we hold on to the standard deep learning terminology.

sider, for instance, the inclusion of equivariance to the set of rotations by  $\theta_r$  degrees<sup>3</sup>:  $\Theta = \{\theta_r = r \frac{2\pi}{r_{\max}}\}_{r=1}^{r_{\max}}$ . To this end, we modify the feature mapping  $f_R^{(l)} := f_R^{(l)} : \mathbb{Z}^2 \times \Theta \times \Lambda_l \rightarrow \mathbb{Z}^2 \times \Theta \times \Lambda_{l+1}$  to include the rotations defined by  $\Theta$ . Let  $x^{(l)}(u, r, \lambda)$  and  $W_{\lambda', \lambda}^{(l)}(u, r)$  be the input and the convolutional filter with an affixed index  $r$  for rotation. The *rotational convolution* ( $\star_{\mathbb{R}^2 \rtimes \Theta}$ )  $f_R^{(l)}$  is defined as:

$$x^{(l+1)}(u, r, \lambda) = [x^{(l)} \star_{\mathbb{R}^2 \rtimes \Theta} W_{\lambda', \lambda}^{(l)}](u, r, \lambda) = \sum_{\lambda', r', u'} x^{(l)}(u + u', r', \lambda') W_{\lambda', \lambda}^{(l)}(\theta_r u', r' - r). \quad (7.3)$$

Since  $f_R^{(l)}$  produces ( $\dim(\Theta) = r_{\max}$ ) times more output feature maps than  $f_T^{(l)}$ , we need to learn fewer convolutional filters  $W_{\lambda', \lambda}^{(l)}$  to produce the same number of output channels.

**Learning equivariant neural networks.** Consider the change of variables  $g = u$ ,  $G = \mathbb{Z}^2$ ,  $g \in G$  and  $g = (u, r)$ ,  $G = \mathbb{Z}^2 \rtimes \Theta$ ,  $g \in G$  in Eq. 7.2 and Eq. 7.3, respectively. In general, neural networks are learned via backpropagation [206] by iteratively applying the chain rule of derivation to update the network parameters. Intuitively, the networks outlined in Eq. 7.2 and Eq. 7.3 obtain feedback from all  $g \in G$  and, resultantly, are inclined to learn feature representations that perform optimally on the entire group  $G$ . However, as outlined in Fig. 7.2 and Section 7.1, several of those feature combinations are not likely to simultaneously appear. Resultantly, the *hypothesis space of the model* as defined by Weiler et al. [424] might be further reduced.

Note that this reasoning is tightly related to existing explanations for the large success of *spatial* [431, 444, 465] and *temporal* [242, 264, 403, 465] attention in neural architectures.

## 7.3 Co-Attentive Equivariant Neural Networks

In this section we define co-attentive feature mappings and apply them in the context of equivariant neural networks (Figure 7.3). To this end, we introduce cyclic equivariant self-attention and utilize it to construct co-attentive rotation equivariant neural networks. Subsequently, we show that cyclic equivariant self-attention is extendable to larger symmetry groups and make use of this fact to construct co-attentive neural networks equivariant to rotations and mirror reflections.

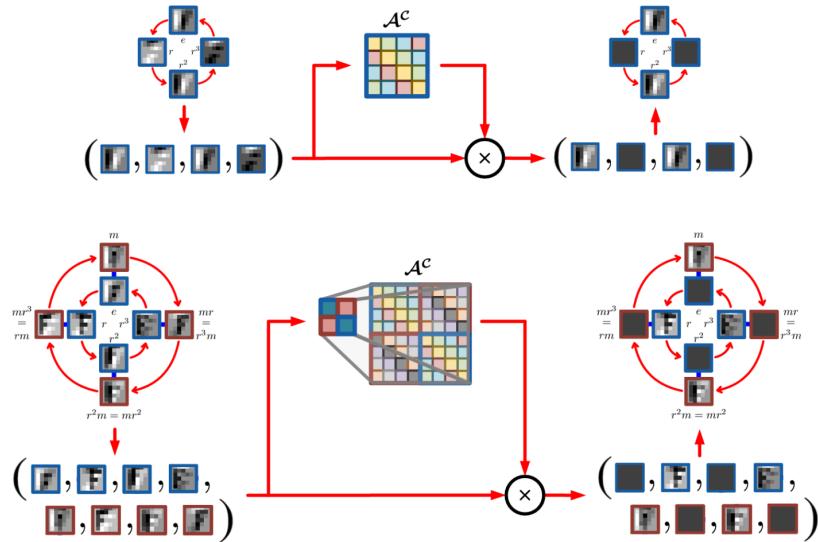
### 7.3.1 Co-Attentive Rotation Equivariant Neural Networks

To allow rotation equivariant networks to utilize and learn *co-attentive equivariant representations*, we introduce an attention operator  $\mathcal{A}^{(l)}$  on top of the roto-translational convolution  $f_R^{(l)}$  with which discernment along the rotation axis  $r$  of the generated feature responses  $x^{(l)}(u, r, \lambda)$  is possible. Formally, our *co-attentive rotation equivariant feature mapping*  $f_{\mathcal{R}}^{(l)}$  is defined as follows:

$$x^{(l+1)} = f_{\mathcal{R}}^{(l)}(x^{(l)}) = \mathcal{A}^{(l)}(f_R^{(l)}(x^{(l)})) = \mathcal{A}^{(l)}([x^{(l)} \star_{\mathbb{R}^2 \rtimes \Theta} W_{\lambda', \lambda}^{(l)}]). \quad (7.4)$$

---

<sup>3</sup>The reader may easily verify that  $\Theta$  (and hence  $\mathbb{Z}^2 \rtimes \Theta$ , with  $(\rtimes)$  the semi-direct product) forms a group.



**Figure 7.3:** Co-attentive equivariant feature mappings acting on the groups  $p4$  (top) and  $p4m$  (bottom). In order to learn co-attentive equivariant representations, *cyclic equivariant self-attention*  $\mathcal{A}^C$  is applied on top of the output of a conventional equivariant feature mapping (here  $p4$  and  $p4m$  group convolutions, respectively). Resultantly, the group convolution responses are modulated based on their assessed relevance. For multiple symmetry groups, the group convolution responses must be rearranged in a vector structure so that the permutation laws of  $\mathcal{A}^C$  correspond to those of the composing group symmetries. Same colors in  $\mathcal{A}^C$  denote equal weights. The *circulant (block) structure* of  $\mathcal{A}^C$  ensures that group equivariance is preserved through the course of attention. Therefore, if the input is rotated –or mirrored in  $p4m$ –, the attention mask will transform accordingly. Built upon Figures 1 and 2 of Cohen and Welling [65].

Theoretically,  $\mathcal{A}^{(l)}$  could be defined globally over  $f_R^{(l)}(x^{(l)})$  (i.e., simultaneously along  $u, r, \lambda$ ) as depicted in Eq. 7.4. However, we apply attention locally to: (1) grant the algorithm enough flexibility to attend locally to the co-occurrence envelope of feature representations and, (2) utilize attention exclusively along the rotation axis  $r$ , such that our contributions are clearly separated from those possibly emerging from spatial attention. To this end, we apply attention pixel-wise on top of  $f_R^{(l)}(x^{(l)})$  (Eq. 7.5). Furthermore, we assign a single attention instance  $\mathcal{A}_\lambda^{(l)}$  to each learned feature representation and utilize it across the spatial dimension of the output feature maps<sup>4</sup>:

$$x^{(l+1)}(u, r, \lambda) = \mathcal{A}_\lambda^{(l)}(\{x^{(l+1)}(u, \hat{r}, \lambda)\}_{\hat{r}=1}^{r_{\max}})(r). \quad (7.5)$$

**Attention and self-attention.** Consider a source vector  $x = (x_1, \dots, x_n)$  and a target vector  $y = (y_1, \dots, y_m)$ . In general, an attention operator  $\mathcal{A}$  leverages information from the source vector  $x$  (or multiple feature mappings thereof) to estimate an attention matrix  $A \in [0, 1]^{n \times m}$ , such that: (1) the element  $A_{i,j}$  provides an importance assessment of the source element  $x_i$  with reference to the target element  $y_j$  and (2) the sum of importance over all  $x_i$  is equal to one:  $\sum_i A_{i,j} = 1$ . Subsequently, the matrix  $A$  is utilized to modulate the original source vector  $x$  as to *attend* to a subset of relevant source positions with regard to  $y_j$ :  $\tilde{x}^j = (A_{:,j})^T \odot x$  (where  $\odot$  is the Hadamard product). A special case of attention is that of *self-attention* [52], in which the target and the source vectors are equal ( $y := x$ ). In other words, the attention mechanism estimates the influence of the sequence  $x$  on the element  $x_j$  for its weighting.

In general, the attention matrix<sup>5</sup>  $A \in [0, 1]^{n \times m}$  is constructed via nonlinear space transformations  $f_{\tilde{A}} : \mathbb{R}^n \rightarrow \mathbb{R}^{n \times m}$  of the source vector  $x$ , on top of which the softmax function is applied:  $A_{:,j} = \text{softmax}(f_{\tilde{A}}(x)_{:,j})$ . This ensures that the properties previously mentioned hold. Typically, the mappings  $f_{\tilde{A}}$  found in literature take feature transformation pairs of  $x$  as input (e.g.,  $\{s, H\}$  in RNNs [242],  $\{Q, K\}$  in self-attention networks [403]), and perform (non)-linear mappings on top of it, ranging from multiple feed-forward layers [10] to several operations between the transformed pairs [242, 264, 403, 465]. Due to the computational complexity of these approaches and the fact that we do extensive pixel-wise usage of  $f_{\tilde{A}}$  on every network layer, their direct integration in our framework is computationally prohibitive. To circumvent this problem, we modify the usual self-attention formulation as to enhance its descriptive power in a much more compact setting.

**Compact local self-attention.** Initially, we relax the range of values of  $A$  from  $[0, 1]^{n \times n}$  to  $\mathbb{R}^{n \times n}$ . This allows us to encode much richer relationships between element pairs  $(x_i, x_j)$  at the cost of less interpretability. Subsequently, we define  $A = x^T \odot \tilde{A}$ , where  $\tilde{A} \in \mathbb{R}^{n \times n}$  is a matrix of learnable parameters. Furthermore, instead of directly applying softmax on the columns of  $A$ , we first sum over the contributions of each element  $x_i$  to obtain a vector  $a = \{\sum_i A_{i,j}\}_{j=1}^n$ , which is then passed to the softmax function. Following [403],

---

<sup>4</sup>For a more meticulous discussion on how Eq. 7.5 attains co-occurrent attention, see Appendix F.1.

<sup>5</sup>Technically, each column of  $A$  is restricted to a simplex and hence  $A$  lives in a subspace of  $[0, 1]^{n \times m}$ .

we prevent the softmax function from reaching regions of low gradient by scaling its argument by  $(\sqrt{\dim(A)})^{-1} = (1/n)$ :  $\tilde{a} = \text{softmax}((1/n)a)$ . Lastly, we counteract the contractive behaviour of the softmax function by normalizing  $\tilde{a}$  before weighting  $x$  as to preserve the magnitude range of its argument. This allows us to use  $\mathcal{A}$  in deep architectures. Our *compact self-attention mechanism* is summarized as follows:

$$a = \{\sum_i A_{i,j}\}_{j=1}^n = \sum_i (x^T \odot \tilde{A})_{i,j} = x \tilde{A} \quad (7.6)$$

$$\tilde{a} = \text{softmax}((1/n)a) \quad (7.7)$$

$$\hat{x} = \mathcal{A}(x) = (\tilde{a} / \max(\tilde{a})) \odot x. \quad (7.8)$$

**The cyclic equivariant self-attention operator  $\mathcal{A}^C$ .** Consider  $\{x(u, r, \lambda)\}_{r=1}^{r_{\max}}$ , the vector of responses generated by a roto-translational convolution  $f_R$  stacked along the rotation axis  $r$ . By applying self-attention along  $r$ , we are able to generate an importance matrix  $A \in \mathbb{R}^{r_{\max} \times r_{\max}}$  relating all pairs of  $(\theta_i, \theta_j)$ -rotated responses in the rotational group  $\Theta$  at a certain position. We refer to this attention mechanism as *full self-attention* ( $\mathcal{A}^F$ ). Although  $\mathcal{A}^F$  is able to encode arbitrary linear source-target relationships for each target position, it is not restricted to conserve equivariance to  $\Theta$ . Resultantly, we risk incurring into the behavior outlined in Fig. 7.2c. Before we further elaborate on this issue, we introduce the *cyclic permutation operator*  $\mathcal{P}^i$ , which induces a cyclic shift of  $i$  positions on its argument:  $\sigma^{\mathcal{P}^i}(x_j) = x_{(j+i) \bmod (\dim(x))}$ ,  $\forall x_j \in x$ .

Consider a full self-attention operator  $\mathcal{A}^F$  acting on top of a roto-translational convolution  $f_R$ . Let  $p$  be an input pattern to which  $f_R$  only produces a strong activation in the feature map  $x(\hat{r}) = f_R(p)(\hat{r})$ ,  $\hat{r} \in \{r\}_{r=1}^{r_{\max}}$ . Intuitively, during learning, only the corresponding attention coefficients  $\tilde{A}_{:, \hat{r}}$  in  $\mathcal{A}^F$  would be significantly increased. Now, consider the presence of the input pattern  $\theta_i p$ , a  $\theta_i$ -rotated variant of  $p$ . By virtue of the rotational equivariance property of the feature mapping  $f_R$ , we obtain (locally) an exactly equal response to that of  $p$  up to a cyclic permutation of  $i$  positions on  $r$ , and thus, we obtain a strong activation in the feature map  $\mathcal{P}^i(x(\hat{r})) = x(\sigma^{\mathcal{P}^i}(\hat{r}))$ . We encounter two problems in this setting:  $\mathcal{A}^F$  is not able to detect that  $p$  and  $\theta_i p$  correspond to the exact same input pattern and, as each but the attention coefficients  $\tilde{A}_{:, j}$  is small, the network might considerably damp the response generated by  $\theta_i p$ . As a result, the network might (1) squander important feedback information during learning and (2) induce learning of repeated versions of the same pattern for different orientations. In other words,  $\mathcal{A}^F$  does not behave equivariantly as a function of  $\theta_i$ .

Interestingly, we can introduce prior-knowledge into the attention model by restricting the structure of  $\tilde{A}$ . By leveraging *equivariance to the cyclic group  $C_n$* , we are able to solve the problems exhibited by  $\mathcal{A}^F$  and simultaneously reduce the number of additional parameters required by the self-attention mechanism (from  $r_{\max}^2$  to  $r_{\max}$ ). Consider again the input patterns  $p$  and  $\theta_i p$ . We incorporate the intuition that  $p$  and  $\theta_i p$  are one and the same entity, and thus,  $f_R$  (locally) generates the same output feature map up to a cyclic permutation  $\mathcal{P}^i$ :  $f_R(\theta_i p) = \mathcal{P}^i(f_R(p))$ . Consequently, the attention mechanism

should produce the *exact same* output for both  $p$  and  $\theta_i p$  up to the same cyclic permutation  $\mathcal{P}^i$ . In other words,  $\mathcal{A}$  (and thus  $\tilde{A}$ ) should be *equivariant to cyclic permutations*.

A well-known fact in mathematics is that a matrix  $A$  is equivariant with respect to cyclic permutations of the domain if and only if it is *circulant* [3, 5]. We make use of this fact and leverage *circulant matrices* to impose cyclic equivariance to the structure of  $\tilde{A}$ . Formally, a circulant matrix  $C \in \mathbb{R}^{n \times n}$  is composed of  $n$  cyclic permutations of its defining vector  $c = \{c_i\}_{i=1}^n$ , such that its  $j$ -th column is a cyclic permutation of  $j - 1$  positions of  $c$ :  $C_{:,j} = \mathcal{P}^{j-1}(c)^T$ . We construct our *cyclic equivariant self-attention operator*  $\mathcal{A}^C$  by defining  $\tilde{A}$  as a circulant matrix specified by a learnable attention vector  $a^C = \{a_i^C\}_{i=1}^{r_{\max}}$ :

$$\tilde{A} = \{\mathcal{P}^{j-1}(a^C)^T\}_{j=1}^n, \quad (7.9)$$

and subsequently applying Eqs. 7.6-7.8. Resultantly,  $\mathcal{A}^C$  is able to assign the responses generated by  $f_R$  for rotated versions of an input pattern  $p$  to a unique entity:  $f_R(\theta_i p) = \mathcal{P}^i(f_R(p))$ , and dynamically adjust its output to the angle of appearance  $\theta_i$ , such that the attention operation does not disrupt its propagation downstream the network:  $\mathcal{A}^C(f_R(\theta_i p)) = \mathcal{P}^i(\mathcal{A}^C(f_R(p)))$ . Consequently, the attention weights  $a^C$  are updated equally regardless of specific values of  $\theta_i$ . Due to these properties,  $\mathcal{A}^C$  does not incur in any of the problems outlined earlier in this section. Conclusively, our *co-attentive rotation equivariant feature mapping*  $f_R^{(l)}$  is defined as follows:

$$x^{(l+1)}(u, r, \lambda) = f_R^{(l)}(x^{(l)})(u, r, \lambda) = \mathcal{A}_\lambda^{C(l)}([x^{(l)} *_{\mathbb{R}^2 \rtimes \Theta} W_{\lambda', \lambda}^{(l)}])(u, r, \lambda). \quad (7.10)$$

Note that a co-attentive equivariant map  $f_R$  is approximately equal –up to a normalized softmax operation (Eq. 7.8)– to a conventional equivariant one  $f_R$ , if  $\tilde{A} = \alpha I$  for any  $\alpha \in \mathbb{R}$ .

### 7.3.2 Extending $\mathcal{A}^C$ To Multiple Symmetry Groups

The self-attention mechanisms outlined in the previous section are easily extendable to larger groups consisting of multiple symmetries. Consider, for instance, the group  $\theta_r m$  of rotations by  $\theta_r$  degrees and mirror reflections  $m$  defined analogously to the group  $p4m$  in Cohen and Welling [65]. Let  $p(u, r, m, \lambda)$  be an input signal with an affixed index  $m \in \{m_0, m_1\}$  for mirror reflections ( $m_1$  indicates mirrored) and  $f_{\theta_r m}$  be a *group convolution* [65] on the  $\theta_r m$  group. The group convolution  $f_{\theta_r m}$  produces two times as many output channels ( $2r_{\max} : m_0 r_{\max} + m_1 r_{\max}$ ) as those generated by the roto-translational convolution  $f_R$  (Eq. 7.3, Fig. 7.3).

Full self-attention  $\mathcal{A}^F$  can be integrated directly by modulating the output of  $f_{\theta_r m}$  as depicted in Sec. 7.3.1 with  $\tilde{A} \in \mathbb{R}^{2r_{\max} \times 2r_{\max}}$ . Here,  $\mathcal{A}^F$  relates each of the group convolution responses with one another. However, just as for  $f_R$ ,  $\mathcal{A}^F$  disrupts the equivariance property of  $f_{\theta_r m}$  to the  $\theta_r m$  group.

Similarly, the cyclic equivariant self-attention operator  $\mathcal{A}^C$  can be extended to multiple symmetry groups as well. Before we continue, we introduce the *cyclic permutation operator*  $\mathcal{P}^{i,t}$ , which induces a cyclic shift of  $i$  positions on its argument along the transforma-

tion axis  $t$ . Consider the input patterns  $p$  and  $\theta_i p$  outlined in the previous section and  $mp$ , a mirrored instance of  $p$ . Let  $x(u, r, m, \lambda) = f_{\theta_r m}(p)(u, r, m, \lambda)$  be the response of the group convolution  $f_{\theta_r m}$  for the input pattern  $p$ . By virtue of the rotation equivariance property of  $f_{\theta_r m}$ , the generated response for  $\theta_i p$  is equivalent to that of  $p$  up to a cyclic permutation of  $i$  positions along the rotation axis  $r$ :  $f_{\theta_r m}(\theta_i p)(u, r, m, \lambda) = \mathcal{P}^{i,r}(f_{\theta_r m}(p))(u, r, m, \lambda) = x(u, \sigma^{\mathcal{P}^i}(r), m, \lambda)$ . Similarly, by virtue of the mirror equivariance property of  $f_{\theta_r m}$ , the response generated by  $mp$  is equivalent to that of  $p$  up to a cyclic permutation of one position along the mirroring axis  $m$ :  $f_{\theta_r m}(mp)(u, r, m, \lambda) = \mathcal{P}^{1,m}(f_{\theta_r m}(p))(u, r, m, \lambda) = x(u, r, \sigma^{\mathcal{P}^1}(m), \lambda)$ . Note that if we take two elements from a group  $g, h$ , their composition  $(gh)$  is also an element of the group. Resultantly,  $f_{\theta_r m}((m\theta^i)p)(u, r, m, \lambda) = (\mathcal{P}^{1,m} \circ \mathcal{P}^{i,r})(f_{\theta_r m}(p))(u, r, m, \lambda) = \mathcal{P}^{1,m}(\mathcal{P}^{i,r}(x))(u, r, m, \lambda) = \mathcal{P}^{1,m}(x)(u, \sigma^{\mathcal{P}^i}(r), m, \lambda) = x(u, \sigma^{\mathcal{P}^i}(r), \sigma^{\mathcal{P}^1}(m), \lambda)$ .

In other words, in order to extend  $\mathcal{A}^C$  to the  $\theta_r m$  group, it is necessary to restrict the structure of  $\tilde{A}$  such that it respects the *permutation laws* imposed by the equivariant mapping  $f_{\theta_r m}$ . Let us rewrite  $x(u, r, m, \lambda)$  as  $x(u, g, \lambda)$ ,  $g = (mr) \in \{m_0, m_1\} \times \{\hat{r}\}_{\hat{r}=1}^{r_{\max}}$ . In this case, we must impose a *circulant block matrix* structure on  $\tilde{A}$  such that: (1) the composing blocks permute internally as defined by  $\mathcal{P}^{i,r}$  and (2) the blocks themselves permute with one another as defined by  $\mathcal{P}^{1,m}$ . Formally,  $\tilde{A}$  is defined as:

$$\tilde{A} = \begin{bmatrix} \tilde{A}_1 & \tilde{A}_2 \\ \tilde{A}_2 & \tilde{A}_1 \end{bmatrix}, \quad (7.11)$$

where  $\{\tilde{A}_i \in \mathbb{R}^{r_{\max} \times r_{\max}}\}$ ,  $i \in \{1, 2\}$  are circulant matrices (Eq. 7.9). Importantly, the ordering of the permutation laws in  $\tilde{A}$  is interchangeable if the input vector is modified accordingly, i.e.,  $g = (rm)$ .

In summary, cyclic equivariant self-attention  $\mathcal{A}^C$  can be directly extended to act on any  $G$ -equivariant feature mapping  $f_G$ , and for any symmetry group  $G$ , if the group actions  $T_g^Y$  produce cyclic permutations on the codomain of  $f_G$ . To this end, one must restrict the structure of  $\tilde{A}$  to that of a circulant block matrix, such that *all* permutation laws of  $T_g^Y$  hold:  $T_g^Y(\mathcal{A}^C(f_G)) = \mathcal{A}^C(T_g^Y(f_G)), \forall g \in G$ .

## 7.4 Experiments

**Experimental Setup.** We validate our approach by exploring the effects of co-attentive equivariant feature mappings for single and multiple symmetry groups on existing equivariant architectures. Specifically, we replace conventional rotation equivariant mappings in  $p4$ -CNNs [65] and DRENs [216] with co-attentive equivariant ones and evaluate their effects in fully (rotated MNIST) and partially (CIFAR-10) rotational settings. Similarly, we evaluate co-attentive equivariant maps acting on multiple similarity groups by replacing equivariant mappings in  $p4m$ -CNNs [65] (equivariant to rotation and mirror reflections) likewise. Unless otherwise specified, we replicate as close as possible the same data processing, initialization strategies, hyperparameter values and evaluation

strategies utilized by the baselines in our experiments. Note that the goal of this paper is to study and evaluate the relative effects obtained by co-attentive equivariant networks with regard to their conventional counterparts. Accordingly, we do not perform any additional tuning relative to the baselines. We believe that improvements to our reported results are feasible by further parameter tuning of the proposed co-attentive equivariant networks.

The additional learnable parameters, i.e., those associated to the cyclic self-attention operator ( $\tilde{A}$ ) are initialized identically to the rest of the layer. Subsequently, we replace the values of  $\tilde{A}$  along the diagonal by 1 (i.e.,  $\text{diag}(\tilde{A}_{\text{init}}) = 1$ ) such that  $\tilde{A}_{\text{init}}$  approximately resembles the identity  $I$  and, hence, co-attentive equivariant layers are initially approximately equal to equivariant ones.

**Rotated MNIST.** The rotated MNIST dataset [201] contains 62000 gray-scale 28x28 handwritten digits uniformly rotated on the entire circle  $[0, 2\pi]$ . The dataset is split into training, validation and tests sets of 10000, 2000 and 50000 samples, respectively. We replace rotation equivariant layers in  $p4$ -CNN [65], DREN and DRENMaxPooling [216] with co-attentive ones. Our results show that co-attentive equivariant networks consistently outperform conventional ones (see Table 7.1).

**CIFAR-10.** The CIFAR-10 dataset [195] consists of 60000 real-world 32x32 RGB images uniformly drawn from 10 classes. Contrarily to the rotated MNIST dataset, this dataset does not exhibit rotation symmetry. The dataset is split into training, validation and tests sets of 40000, 10000 and 10000 samples, respectively. We replace equivariant layers in the  $p4$  and  $p4m$  variations of the All-CNN [365] and the ResNet44 [143] proposed by [65] with co-attentive ones. Likewise, we modify the r\_x4-variations of the NIN [223] and ResNet20 [143] models proposed by [216] in the same manner. Our results show that co-attentive equivariant networks consistently outperform conventional ones in this setting as well (see Table 7.1).

**Training convergence of equivariant networks.** [216] reported that adding too many rotational equivariant (isotonic) layers decreased the performance of their models on CIFAR-10. As a consequence, they did not report results on fully rotational equivariant networks for this setting and attributed this behaviour to the non-symmetry of the data. We noticed that, with equal initialization strategies, rotational equivariant CNNs were much more prone to divergence than ordinary CNNs. This behaviour can be traced back to the additional feedback resulting from roto-translational convolutions (Eq. 7.3) compared to ordinary ones (Eq. 7.2). After further analysis, we noticed that the data preprocessing strategy utilized by Li et al. [216] leaves some very large outlier values in the data ( $|x| > 100$ ), which strongly contribute to the behaviour outlined before.

In order to evaluate the relative contribution of co-attentive equivariant neural networks we constructed fully equivariant DREN architectures based on their implementation. Although the obtained results were much worse than those originally reported in Li et al. [216], we were able to stabilize training by clipping input values to the 99 per-

**Table 7.1:** Comparison of conventional equivariant and co-attentive equivariant neural networks. Values between parenthesis correspond to relevant results obtained from our own experiments.

| Rotated MNIST             |                      |           | CIFAR-10                                 |                |           |
|---------------------------|----------------------|-----------|--|----------------|-----------|
| Model                     | Test Error (%)       | # Params. | Model                                    | Test Error (%) | # Params. |
| Z2CNN                     | 5.03 ± 0.002         | 21.75k    | All-CNN                                  | 9.44           | 1.372M    |
| <i>p</i> 4-CNN            | 2.28 ± 0.0004        | 19.88k    | <i>p</i> 4-All-CNN                       | 8.84           | 1.371M    |
| <i>a</i> - <i>p</i> 4-CNN | <b>2.06 ± 0.0429</b> | 20.76k    | <i>a</i> - <i>p</i> 4-All-CNN            | <b>7.68</b>    | 1.373M    |
| DREN                      | 1.78 (1.99)          | 19.88k    | <i>p</i> 4 <i>m</i> -All-CNN             | 7.59           | 1.219M    |
| <i>a</i> -DREN            | <b>1.674</b>         | 20.76k    | <i>a</i> - <i>p</i> 4 <i>m</i> -All-CNN  | <b>6.92</b>    | 1.223M    |
| DRENMaxPool.              | 1.56 (1.60)          | 24.68k    | ResNet44                                 | 9.45 (9.85)    | 2.639M    |
| <i>a</i> -DRENMaxPool.    | <b>1.34</b>          | 25.68k    | <i>p</i> 4 <i>m</i> -ResNet44            | 6.46 (9.47)    | 2.623M    |
|                           |                      |           | <i>a</i> - <i>p</i> 4 <i>m</i> -ResNet44 | <b>9.12</b>    | 2.632M    |
|                           |                      |           | NIN                                      | 10.41 (15.92)  | 0.967M    |
|                           |                      |           | r-NINx4                                  | 14.96          | 0.958M    |
|                           |                      |           | <i>a</i> -r-NINx4                        | <b>13.67</b>   | 0.968M    |
|                           |                      |           | ResNet20                                 | 9.00 (12.32)   | 0.335M    |
|                           |                      |           | r-ResNet20x4                             | 12.31          | 0.333M    |
|                           |                      |           | <i>a</i> -r-ResNet20x4                   | <b>11.32</b>   | 0.339M    |

centile of the data ( $|x| \leq 2.3$ ) and reducing the learning rate to 0.01, such that the same hyperparameters could be used across all network types. The obtained results (see Table 7.1) signalize that DREN networks are *comparatively better than CNNs both in fully and partially rotational settings*, contradictorily to the conclusions drawn in Li et al. [216].

This behaviour elucidates that although the inclusion of equivariance to larger transformation groups is beneficial both in terms of accuracy and parameter efficiency, one must be aware that such benefits are directly associated with an increase of the network susceptibility to divergence during training.

## 7.5 Discussion and Future Work

Our results show that co-attentive equivariant feature mappings can be utilized to enhance conventional equivariant ones. Interestingly, co-attentive equivariant mappings are beneficial both in partially and fully rotational settings. We attribute this to the fact that a set of co-occurring orientations between patterns can be easily defined (and exploited) in both settings. It is important to note that we utilized attention independently over each spatial position  $u$  on the codomain of the corresponding group convolution. Resultantly, we were restricted to mappings of the form  $xA$ , which, in turn, constraint our attention mechanism to have a circulant structure in order to preserve equivariance (since group actions acting in the codomain of the group convolution involve cyclic permutations and cyclic self-attention is applied in the codomain of the group convolution).

In future work, we want to extend the idea presented here to act on the entire group simultaneously (i.e., along  $u$  as well). By doing so, we lift our current restriction to map-

pings of the form  $xA$  and therefore, may be able to develop attention instances with enhanced descriptive power. Following the same line of thought, we want to explore incorporating attention in the convolution operation itself. Resultantly, one is not restricted to act exclusively on the codomain of the convolution, but instead, is able to impose structure in the domain of the mapping as well. Naturally, such an approach could lead to enhanced descriptiveness of the incorporated attention mechanism. Moreover, we want to utilize and extend more complex attention strategies (e.g., Bahdanau et al. [10], Luong et al. [242], Mishra et al. [264], Vaswani et al. [403]) such that they can be applied to large transformation groups without disrupting equivariance. As outlined earlier in Section 7.3.1, this becomes very challenging from a computational perspective as well, as it requires extensive usage of the corresponding attention mechanism. Resultantly, an efficient implementation thereof is mandatory. Furthermore, we want to extend co-attentive equivariant feature mappings to continuous (e.g., Worrall et al. [434]) and 3D space (e.g., Cohen et al. [67, 69], Worrall and Brostow [432]) groups, and for applications other than visual data (e.g., speech recognition).

Finally, we believe that our approach could be refined and extended to a first step towards dealing with the enumeration problem of large groups [115], such that functions acting on the group (e.g., group convolution) are approximated by evaluating them on the set of co-occurring transformations as opposed to on the entire group. Such approximations are expected to be very accurate, as non-co-occurrent transformations are rare. This could be thought of as sharpening up co-occurrent attention to co-occurrent restriction.

## 7.6 Conclusion

We have introduced the concept of co-attentive equivariant feature mapping and applied it in the context of equivariant neural networks. By attending to the co-occurrence envelope of the data, we are able to improve the performance of conventional equivariant ones on fully (rotated MNIST) and partially (CIFAR-10) rotational settings. We developed cyclic equivariant self-attention, an attention mechanism able to attend to the co-occurrence envelope of the data without disrupting equivariance to a large set of transformation groups (i.e., all transformation groups  $G$ , whose action in the codomain of a  $G$ -equivariant feature mapping produce cyclic permutations). Our obtained results support the proposed co-occurrence envelope hypothesis.

# 8

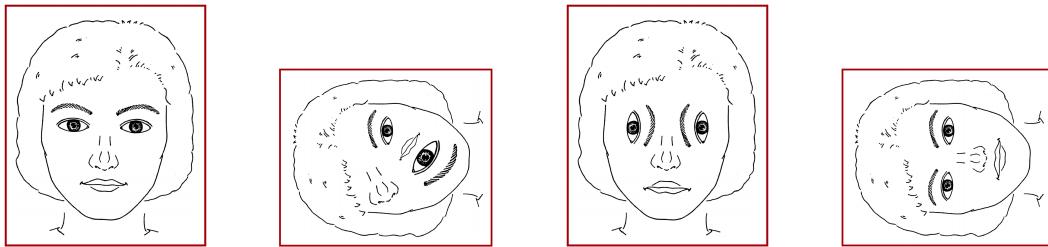
## Attentive Group Equivariant Convolutional Neural Networks

*Based on the paper:  
Attentive Group Equivariant Convolutional Networks [318]*

### 8.1 Introduction

Convolutional Neural Networks (CNNs) [206] have shown impressive performance in a wide variety of domains. The developments of CNNs as well as of many other machine learning approaches have been fueled by intuitions and insights into the composition and *modus operandi* of multiple biological systems [28, 30, 84, 84, 427, 474]. Though CNNs have achieved remarkable performance increases on several benchmark problems, their training efficiency as well as generalization capabilities are still open for improvement. One concept being exploited for this purpose is that of *equivariance*, again drawing inspiration from human beings.

Humans are able to identify familiar objects despite modifications in location, size, viewpoint, lighting conditions and background [35]. In addition, we do not just recognize them but are able to describe in detail the type and amount of modification applied to them as well [40, 340, 407]. Equivariance is strongly related to the idea of *symmetry*. As these modifications do not modify the essence of the underlying object, they should be treated (and learned) as a single concept. Recently, several approaches have embraced these ideas to preserve symmetries including translations [206], planar rotations



**Figure 8.1:** Meaningful relationships among object symmetries. Though every figure is composed by the same elements, only the outermost examples resemble faces. The relative positions, orientations and scales of elements in the innermost examples do not match any meaningful face composition and hence, should not be labelled as such. Built upon Fig. 1 from Schwarzer [345].

[18, 53, 89, 150, 215, 216, 252, 359, 404, 424, 434], spherical rotations [67, 69, 382, 423, 432], scaling [253, 362, 433] and general symmetry groups [17, 65, 68, 194, 315, 406, 422].

While group convolutional networks are able to learn powerful representations based on symmetry patterns, they lack any explicit means to learn meaningful relationships among them, e.g., relative positions, orientations and scales (Fig. 8.1). In this paper, we draw inspiration from another promising development in the machine learning domain driven by neuroscience and psychology, e.g., Pashler [283], *attention*, to learn such relationships. The notion of attention is related to the idea that not all components of an input signal are *per se* equally relevant for a particular task. As a consequence, given a task and a particular input signal, task-relevant components of the input should be focused during its analysis while irrelevant, possibly misleading ones should be suppressed. Attention has been broadly applied to fields ranging from natural language processing [10, 52, 403], visual understanding [164, 279, 444], and graph analysis [405, 469].

Specifically, we present *attentive group convolutions*, a generalization of the group convolution, in which attention is applied during convolution to accentuate meaningful symmetry combinations and suppress non-plausible, possibly misleading ones. We indicate that prior work on visual attention can be described as special cases of our proposed framework and show empirically that our *attentive group equivariant group convolutional networks* consistently outperform conventional group equivariant ones on rot-MNIST and CIFAR-10 for the SE(2) and E(2) groups. In addition, we provide means to interpret the learned concepts through the visualization of the equivariant attention maps.

### Contributions:

- We propose a general group theoretical framework for equivariant visual attention, *the attentive group convolution*, and show that prior works on visual attention are special cases of our framework.
- We introduce a specific type of network referred to as *attentive group convolutional networks* as an instance of this theoretical framework.

- We show that our *attentive group convolutional networks* consistently outperform plain group equivariant ones.
- We provide means to interpret the learned concepts via visualization of the predicted equivariant attention maps.

## 8.2 Preliminaries

Before describing our approach, we first define crucial prior concepts: (group) convolutions and attention mechanisms.

### 8.2.1 Spatial Convolution and Translation Equivariance

Let  $f, \psi : \mathbb{R}^d \rightarrow \mathbb{R}^{N_{\tilde{c}}}$  be a vector valued signal and filter on  $\mathbb{R}^d$ , such that  $f = \{f_{\tilde{c}}\}_{\tilde{c}=1}^{N_{\tilde{c}}}$  and  $\psi = \{\psi_{\tilde{c}}\}_{\tilde{c}=1}^{N_{\tilde{c}}}$ . The spatial convolution ( $\star_{\mathbb{R}^d}$ ) is defined as:

$$[f \star_{\mathbb{R}^d} \psi](y) = \sum_{\tilde{c}=1}^{N_{\tilde{c}}} \int_{\mathbb{R}^d} f_{\tilde{c}}(x) \psi_{\tilde{c}}(x - y) dx \quad (8.1)$$

Intuitively, Eq. 8.1 resembles a collection of  $\mathbb{R}^d$  inner products between the input signal  $f$  and  $y$ -translated versions of  $\psi$ . Since the continuous integration in Eq. 8.1 is usually performed on signals and filters captured in a discrete grid  $\mathbb{Z}^d$ , the integral on  $\mathbb{R}^d$  is reduced to a sum on  $\mathbb{Z}^d$ . In our derivations, however, we stick to the continuous case as to guarantee the validity of our theory for techniques defined on continuous spaces, e.g., steerable and Lie group convolutions [18, 66, 434].

To study (and generalize) the properties of the convolution, we rewrite Eq. 8.1 using the translation operator  $\mathcal{L}_y$ :

$$[f \star_{\mathbb{R}^d} \psi](y) = \sum_{\tilde{c}=1}^{N_{\tilde{c}}} \int_{\mathbb{R}^d} f_{\tilde{c}}(x) \mathcal{L}_y[\psi_{\tilde{c}}](x) dx \quad (8.2)$$

where  $\mathcal{L}_y[\psi_{\tilde{c}}](x) = \psi_{\tilde{c}}(x - y)$ . Note that the translation operator  $\mathcal{L}_y$  is indexed by an amount of translation  $y$ . Resultantly, we actually consider a set of operators  $\{\mathcal{L}_y\}_{y \in \mathbb{R}^d}$  that indexes the set of all possible translations  $y \in \mathbb{R}^d$ . A fundamental property of the convolution is that it commutes with translations:

$$\mathcal{L}_y[f \star_{\mathbb{R}^d} \psi](x) = [\mathcal{L}_y[f] \star_{\mathbb{R}^d} \psi](x), \quad x, y \in \mathbb{R}^d. \quad (8.3)$$

In other words, convolving a  $y$ -translated signal  $\mathcal{L}_y[f]$  with a filter is equivalent to first convolving the original signal  $f$  with the filter  $\psi$ , and  $y$ -translating the obtained response next. This property is referred to as *translation equivariance* and, in fact, convolution (and reparametrizations thereof) is the *only* linear *translation equivariant* mapping [17, 68, 194].

### 8.2.2 Group Convolution and Group Equivariance

The convolution operation can be extended to general transformations by utilizing a larger set of transformations  $\{\mathcal{L}_g\}_{g \in G}$ , s.t.  $\{\mathcal{L}_y\}_{y \in \mathbb{R}^d} \subseteq \{\mathcal{L}_g\}_{g \in G}$ . However, in order to preserve equivariance, we must restrict the class of transformations allowed in  $\{\mathcal{L}_g\}_{g \in G}$ . To formalize this intuition, we first present some important concepts from *group theory*.

**Groups.** A *group* is a tuple  $(G, \cdot)$  consisting of a set  $G$ ,  $g \in G$ , and a binary operation  $\cdot : G \times G \rightarrow G$ , referred to as the *group product*, that satisfies the following axioms:

- *Closure:* For all  $h, g \in G$ ,  $h \cdot g \in G$ .
- *Identity:* There exists an  $e \in G$ , such that  $e \cdot g = g \cdot e = g$ .
- *Inverse:* For all  $g \in G$ , there exists an element  $g^{-1} \in G$ , such that  $g \cdot g^{-1} = g^{-1} \cdot g = e$ .
- *Associativity:* For all  $g, h, k \in G$ ,  $(g \cdot h) \cdot k = g \cdot (h \cdot k)$ .

**Group actions.** Let  $G$  and  $X$  be a group and a set, respectively. The (left) *group action* of  $G$  on  $X$  is a function  $\odot : G \times X \rightarrow X$  that satisfies the following axioms:

- *Identity:* If  $e$  is the identity of  $G$ , then, for any  $x \in X$ ,  $e \odot x = x$ .
- *Compatibility:* For all  $g, h \in G$ ,  $x \in X$ ,  $g \odot (h \odot x) = (g \cdot h) \odot x$ .

In other words, the action of  $G$  on  $X$  describes how the elements  $x \in X$  are transformed by  $g \in G$ . For brevity, we omit the operations  $\cdot$  and  $\odot$  and refer to the set  $G$  as a group, to elements  $g \cdot h$  as  $gh$  and to actions  $(g \odot x)$  as  $gx$ .

**Semi-direct product and affine groups.** In practice, one is mainly interested in the analysis of data (and hence convolutions) defined on  $\mathbb{R}^d$ . Consequently, groups of the form  $G = \mathbb{R}^d \rtimes H$ , resulting from the *semi-direct product* ( $\rtimes$ ) between the translation group  $\mathbb{R}^d$  and an arbitrary (Lie) group  $H$  that acts on  $\mathbb{R}^d$  (e.g., rotation, scaling, mirroring), are of main interest. This family of groups is referred to as *affine groups* and their group product is defined as:

$$g_1 g_2 = (x_1, h_1)(x_2, h_2) = (x_1 + h_1 x_2, h_1 h_2) \quad (8.4)$$

where  $g_1 = (x_1, h_1)$ ,  $g_2 = (x_2, h_2) \in G$ ,  $x_1, x_2 \in \mathbb{R}^d$  and  $h_1, h_2 \in H$ . Some important affine groups are the roto-translation ( $\text{SE}(d) = \mathbb{R}^d \rtimes \text{SO}(d)$ ), the scale-translation ( $\mathbb{R}^d \rtimes \mathbb{R}^+$ ) and the euclidean ( $\text{E}(d) = \mathbb{R}^d \rtimes \text{O}(d)$ ) groups.

**Group representations.** Let  $G$  be a group and  $\mathbb{L}_2(X)$  be a space of functions defined on some vector space  $X$ . The (left) regular *group representation* of  $G$  on functions  $f \in \mathbb{L}_2(X)$  is a transformation  $\mathcal{L} : G \times \mathbb{L}_2(X) \rightarrow \mathbb{L}_2(X)$ ,  $(g, f) \mapsto \mathcal{L}_g[f]$ , such that it shares the group structure via:

$$\mathcal{L}_g \mathcal{L}_h[f](x) = \mathcal{L}_{gh}[f](x) \quad (8.5)$$

$$\mathcal{L}_g[f](x) := f(g^{-1}x) \quad (8.6)$$

for any  $g, h \in G$ ,  $f \in \mathbb{L}_2(X)$ ,  $x \in X$ . That is, concatenating two such transformations, parametrized by  $g$  and  $h$ , is equivalent to one transformation parametrized by  $gh \in G$ . Intuitively, the representation of  $G$  on a function  $f \in \mathbb{L}_2(X)$  describes how the function as a whole, i.e.,  $f(x)$ ,  $\forall x \in X$ , is transformed by the effect of group elements  $g \in G$ .

If the group  $G$  is affine, i.e.,  $G = \mathbb{R}^d \rtimes H$ , the (left) group representation  $\mathcal{L}_g$  can be split as:

$$\mathcal{L}_g[f](x) = \mathcal{L}_y \mathcal{L}_h[f](x) \quad (8.7)$$

with  $g = (y, h) \in G$ ,  $y \in \mathbb{R}^d$  and  $h \in H$ . This property is key for the efficient implementation of functions on groups.

### The Group Convolution

Let  $f, \psi : G \rightarrow \mathbb{R}^{N_c}$  be a vector valued signal and kernel on  $G$ . The group convolution ( $\star_G$ ) is defined as:

$$[f \star_G \psi](g) = \sum_{\tilde{c}=1}^{N_c} \int_G f_{\tilde{c}}(\tilde{g}) \psi_{\tilde{c}}(g^{-1}\tilde{g}) d\tilde{g} \quad (8.8)$$

$$= \sum_{\tilde{c}=1}^{N_c} \int_G f_{\tilde{c}}(\tilde{g}) \mathcal{L}_g[\psi_{\tilde{c}}](\tilde{g}) d\tilde{g} \quad (8.9)$$

Differently to Eq. 8.2, the domain of the signal  $f$ , the filter  $\psi$  and the group convolution itself  $[f \star_G \psi]$  are now defined on the group  $G$ .<sup>1</sup> Intuitively, the group convolution resembles a collection of inner products between the input signal  $f$  and  $g$ -transformed versions of  $\psi$ . A key property of the group convolution is that it generalizes equivariance (Eq. 8.3) to arbitrary groups, i.e., it commutes with  $g$ -transformations:

$$\mathcal{L}_{\bar{g}}[f \star_G \psi](g) = [\mathcal{L}_{\bar{g}}[f] \star_G \psi](g), \quad g, \bar{g} \in G. \quad (8.10)$$

In other words, group convolving a  $\bar{g}$ -transformed signal  $\mathcal{L}_{\bar{g}}[f]$  with a filter  $\psi$  is equivalent to first convolving the original signal  $f$  with the filter  $\psi$ , and  $\bar{g}$ -transforming the obtained response next. This property is referred to as *group equivariance* and, just as for spatial convolutions, the group convolution (or reparametrizations thereof) is the *only* linear  $G$ -equivariant map [17, 68, 194].

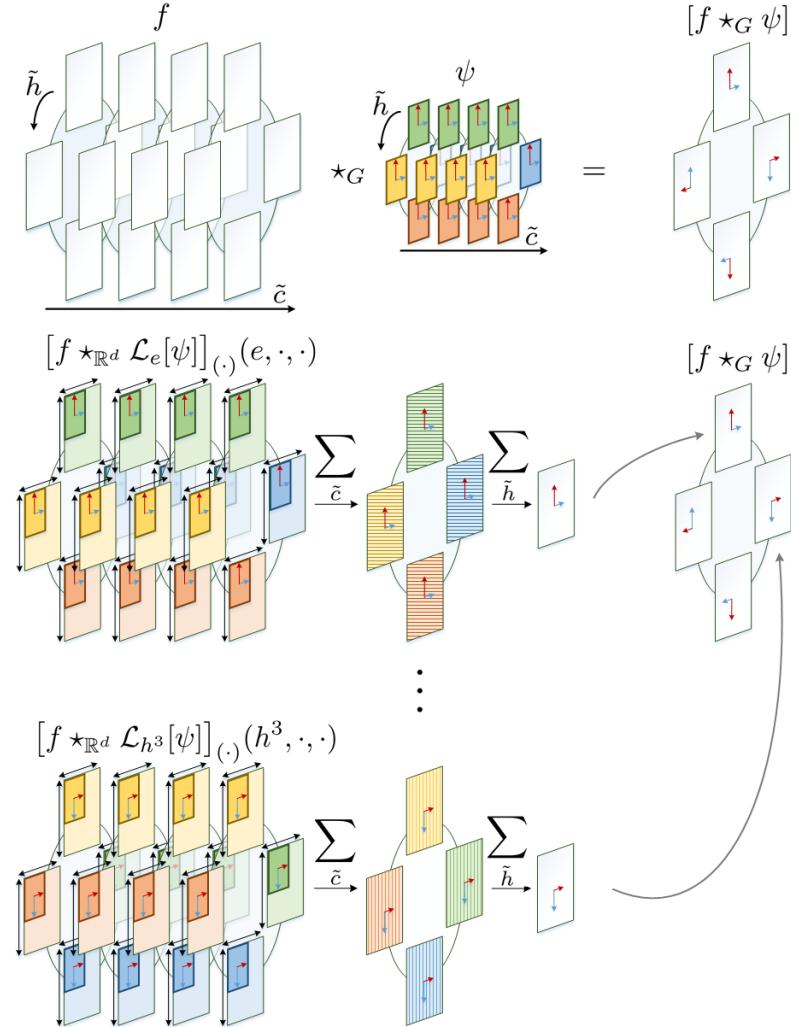
**Group convolution on affine groups.** For affine groups, the group convolution (Eq. 8.9) can be decomposed, without modifying its properties, by taking advantage of the group structure and the representation decomposition (Eq. 8.7) as:

$$[f \star_G \psi](g) = \sum_{\tilde{c}=1}^{N_c} \int_H \int_{\mathbb{R}^2} f_{\tilde{c}}(\tilde{x}, \tilde{h}) \mathcal{L}_g[\psi_{\tilde{c}}](\tilde{x}, \tilde{h}) d\tilde{x} d\tilde{h} \quad (8.11)$$

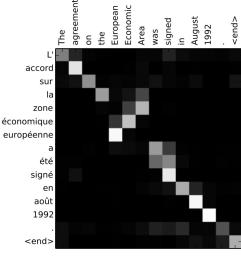
$$= \sum_{\tilde{c}=1}^{N_c} \int_H \int_{\mathbb{R}^2} f_{\tilde{c}}(\tilde{x}, \tilde{h}) \mathcal{L}_x \mathcal{L}_h[\psi_{\tilde{c}}](\tilde{x}, \tilde{h}) d\tilde{x} d\tilde{h} \quad (8.12)$$

---

<sup>1</sup>Note that Eq. 8.2 matches Eq. 8.9 with the substitution  $G = \mathbb{R}^d$ . It follows that  $\mathcal{L}_g[f](x) = f(g^{-1}x) = f(x - y)$ , where  $g^{-1} = -y$  is the inverse of  $g$  in the translation group  $(\mathbb{R}^d, +)$  for  $g = y$ .



**Figure 8.2:** Group convolution on the roto-translation group  $\text{SE}(2)$  for discrete rotations by 90 degrees (also called the  $p4$  group). The  $p4$  group is defined as  $H = \{e, h, h^2, h^3\}$ , with  $h$  depicting a  $90^\circ$  rotation. The group convolution corresponds to  $|H| = 4$  convolutions between the input  $f$  and  $h$ -transformations of the filter  $\psi$ ,  $\mathcal{L}_h[\psi]$ ,  $\forall h \in H$ . Each of these convolutions is equal to the sum over group elements  $\tilde{h} \in H$  and channels  $\tilde{c} \in [N_{\tilde{c}}]$  of the spatial channel-wise convolutions  $[f_{\tilde{c}} \star_{\mathbb{R}^2} \mathcal{L}_h[\psi_{\tilde{c}}]]$  among  $f$  and  $\mathcal{L}_h[\psi]$ .



**Figure 8.3:** English to French translation. Brighter depicts stronger influence. Note how relevant parts of the input sentence are highlighted as a function of the current output word during translation. Taken from Bahdanau et al. [10].

where  $g = (x, h)$ ,  $\tilde{g} = (\tilde{x}, \tilde{h}) \in G$ ,  $x, \tilde{x} \in \mathbb{R}^d$  and  $h, \tilde{h} \in H$ . By doing so, the group convolution can be separated into  $|H|$  spatial convolutions of the input signal  $f$  for each  $h$ -transformed filter  $\mathcal{L}_h[\psi]$  (Fig. 8.2):

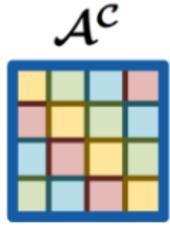
$$[f *_G \psi](x, h) = \sum_{\tilde{c}=1}^{N_c} \int_H [f_{\tilde{c}} *_{\mathbb{R}^2} \mathcal{L}_h[\psi_{\tilde{c}}]](x, \tilde{h}) d\tilde{h} \quad (8.13)$$

Resultantly, the computational cost of a group convolution is roughly equivalent to that of a spatial convolution with a filter bank of size  $N_c \times |H|$  [65, 69, 433].

### 8.2.3 Attention, Self-Attention and Visual Attention

Attention mechanisms find their roots in recurrent neural network (RNN) based machine translation. Let  $\varphi(\cdot)$  be an arbitrary non-linear mapping (e.g., a neural network),  $y = \{y_j\}_{j=1}^m$  be a sequence of target vectors  $y_i$ , and  $\underline{x} = \{x_i\}_{i=1}^n$  be a source sequence, whose elements influence the prediction of each value  $y_j \in \underline{y}$ . In early models (e.g., Cho et al. [56], Kalchbrenner and Blunsom [177]), features in the input sequence are aggregated into a context vector  $c = \sum_i \varphi(x_i)$  which is used to augment the hidden state in RNN layers. These models assume that source elements  $x_i$  contribute equally to *every* target element  $y_j$  and hence, that the same context vector  $c$  can be utilized for all target positions  $y_j$ , which does not generally hold (Fig. 8.3).

Bahdanau et al. [10] proposed the inclusion of *attention coefficients*  $\alpha_i = \{\alpha_{i,j}\}$ ,  $[n] = \{1, \dots, n\}$ ,  $i \in [n]$ ,  $j \in [m]$ ,  $\sum_i \alpha_{i,j} = 1$ , to modulate the contributions of the source elements  $x_i$  as a function of the current target element  $y_j$  by means of an adaptive context vector  $c_j = \sum_i \alpha_{i,j} \varphi(x_i)$ . Thereby, they obtained large improvements both in performance and interpretability. Recently, attention has been extended to several other machine learning tasks, e.g., Park et al. [279], Vaswani et al. [403], Veličković et al. [405]). The main development behind these extensions was *self-attention* [52], where, in contrast to conventional attention, the target and source sequences are equal, i.e.,  $\underline{x} = \underline{y}$ . Consequently, the attention coefficients  $\alpha_{i,j}$  encode correlations among input element pairs  $(x_i, x_j)$ . For vision tasks, self-attention has been proposed to encode visual co-occurrences in data [20, 39, 157, 279, 301, 315, 419, 431]. Unfortunately, its application on visual and, in general, on high-dimensional data is non-trivial.



**Figure 8.4:** Same colors depict equal weights. The first column of  $\mathcal{A}^c$  corresponds to  $\psi$  and the following ones to  $\mathcal{L}_h[\psi]$ , obtained via cyclic permutations. See how  $\{\mathcal{L}_h[\psi]\}_{h \in H}$  resembles a circulant matrix. Taken from Romero and Hoogendoorn [315].

### Visual Attention

In the context of visual attention, consider a feature map  $f : X \rightarrow \mathbb{R}^{N_c}$  to be the source “sequence”<sup>2</sup>. Self-attention then imposes the learning of a total  $n^2 = |X|^2$  attention vectors  $\alpha_{i,j} \in \mathbb{R}^{N_c}$ , which rapidly becomes unfeasible with increasing feature map size. Interestingly, Cao et al. [39] and Zhu et al. [479] empirically demonstrated that, for visual data, the attention coefficients  $\{\alpha_{i,j}\}$  are approximately invariant to changes in the target position  $x_j$ . Consequently, they proposed to approximate the attention coefficients  $\{\alpha_{i,j}\} \in \mathbb{R}^{|X|^2 \times N_c}$  by a single vector  $\{\alpha_i\} \in \mathbb{R}^{|X| \times N_c}$  which is independent of target position  $x_j$ . Despite this significant reduction in complexity, the dimensionality of  $\{\alpha_i\}$  is still very large and further simplifications are mandatory. To this end, existing works [157, 431] replace the input  $f$  with a much smaller vector of input statistics  $s$  that summarizes relevant information from  $f$ .

For instance, the SE-Net [157] utilizes global average pooling to produce a vector of channel statistics of  $f$ ,  $s^c \in \mathbb{R}^{N_c}$ ,  $s^c = \frac{1}{|\mathbb{R}^d|} \int_{\mathbb{R}^d} f_c(x) dx$ , which is subsequently passed to a small fully-connected network  $\varphi^c(\cdot)$  to compute channel attention coefficients  $\alpha^c = \{\alpha_{\tilde{c}}^c\}_{\tilde{c}=1}^{N_c} = \varphi^c(s^c)$ . These attention coefficients are then utilized to modulate the corresponding input channels  $f_{\tilde{c}}$ .

Complementary to channel attention akin to that of the SE-Net, Park et al. [279] utilize a similar strategy for spatial attention. Specifically, they utilize channel average pooling to generate a vector of spatial statistics of  $f$ ,  $s^x \in \mathbb{R}^d$ ,  $s^x = \frac{1}{N_c} \sum_{\tilde{c}=1}^{N_c} f_{\tilde{c}}(x)$ , which is subsequently passed to a small convolutional network  $\varphi^x(\cdot)$  to compute spatial attention coefficients  $\alpha^x = \{\alpha^x(x)\}_{x \in \mathbb{R}^2} = \varphi^x(s^x)$ . These attention coefficients are then utilized to modulate the corresponding spatial input positions  $f(x)$ . Recent works include more statistical information, e.g., max responses [431], or replace pooling by convolutions [39].

### 8.3 Attentive Group Equivariant Convolution

In this section, we propose our generalization of visual self-attention, discuss its properties and relations to prior work.

Let  $f, \psi : G \rightarrow \mathbb{R}^{N_c}$  be a vector valued signal and kernel on  $G$ , and let  $\alpha : G \times G \rightarrow [0, 1]^{N_c}$  be an *attention map* that takes target and source elements  $g, \tilde{g} \in G$ , respectively, as input.

<sup>2</sup>In the machine translation context, we can think of  $f$  as a sequence  $\underline{x} = \{f(x_i)\}_{i=1}^n$ , with  $n = |X|$  elements.

We define the *attentive group convolution* ( $\star_G^\alpha$ ) as:

$$[f \star_G^\alpha \psi](g) = \sum_{\tilde{c}=1}^{N_{\tilde{c}}} \int_G \alpha_{\tilde{c}}(g, \tilde{g}) f_{\tilde{c}}(\tilde{g}) \mathcal{L}_g[\psi_{\tilde{c}}](\tilde{g}) d\tilde{g} \quad (8.14)$$

with  $\alpha = \mathcal{A}[f]$  computed by some *attention operator*  $\mathcal{A}$ . As such, the attentive group convolution modulates the contributions of group elements  $\tilde{g} \in G$  at different channels  $\tilde{c} \in [N_{\tilde{c}}]$  during pooling.<sup>3</sup> The properties and conditions on  $\mathcal{A}$  are summarized in Thm. 8.3.1. An extensive motivation as well as its proof are provided in Appx. G.1.

**Theorem 8.3.1.** *The attentive group convolution is an equivariant operator if and only if the attention operator  $\mathcal{A}$  satisfies:*

$$\forall_{\bar{g}, g, \tilde{g} \in G} : \mathcal{A}[\mathcal{L}_{\bar{g}} f](g, \tilde{g}) = \mathcal{A}[f](\bar{g}^{-1}g, \bar{g}^{-1}\tilde{g}) \quad (8.15)$$

If, moreover, the maps generated by  $\mathcal{A}$  are invariant to one of its arguments, and, hence, exclusively attend to either the input or the output domain (Sec. 8.3.4), then  $\mathcal{A}$  satisfies Eq. 8.15 if and only if it is equivariant and thus, based on group convolutions.

### 8.3.1 Tying Equivariance and Visual Attention Together

Interestingly, and, perhaps in some cases unaware of it, all of the visual attention approaches outlined in Section 8.2.3, as well as all of those we are aware of [20, 39, 51, 86, 155, 157, 164, 224, 279, 301, 315, 419, 431, 444] exclusively utilize translation (or group) equivariance preserving maps for the generation of the attention coefficients and, hence, constitute altogether group equivariant networks by which they satisfy Thm. 8.3.1.

As will be explained in the following sections, all these works resemble special cases of Eq. 8.14 by substituting  $G$  with the corresponding group and modifying the specifications about how  $\alpha$  is calculated (Sec. 8.3.2 - 8.3.4).

#### Translation Equivariant Visual Attention

Since convolutions as well as popular pooling operations are translation equivariant, the visual attention approaches outlined in Sec. 8.2.3 are translation equivariant as well.<sup>4</sup> One particular case worth emphasising is that of SE-Nets. Here, a fully-connected network  $\varphi^C$ , a non-translation equivariant map, is used to generate the channel attention coefficients  $\alpha^C$ . However,  $\varphi^C$  is indeed translation equivariant. Recall that  $\varphi^C$  receives  $s^C$  as input, a signal obtained via global average pooling (a convolution-like operation). Resultantly,  $s^C$  can be interpreted as a  $\mathbb{R}^{N_{\tilde{c}} \times 1 \times 1}$  tensor and hence, applying a fully connected layer to  $s^C$  equals a pointwise convolution between  $s^C$  and a filter  $\psi_{\text{fully}} \in \mathbb{R}^{N_0 \times N_{\tilde{c}} \times 1 \times 1}$  with

---

<sup>3</sup>Note that Eq. 8.14 is equal to Eq. 8.9 up to a multiplicative factor  $\alpha_{\tilde{c}}(g, \tilde{g})^{-1}$ , if  $\alpha_{\tilde{c}}(g, \tilde{g})$  is constant for every  $g, \tilde{g} \in G, \tilde{c} \in [N_{\tilde{c}}]$ .

<sup>4</sup>In fact, conventional pooling operations (e.g., max, average) can be written as combinations of convolutions and pointwise non-linearities, which are translation equivariant, as well.

$N_o$  output channels.<sup>5</sup>

### Group Equivariant Visual Attention

To the best of our knowledge, the only work that provides a group theoretical approach towards visual attention is that of Romero and Hoogendoorn [315]. Here, the authors consider affine groups  $G$  with elements  $g = (x, h)$ ,  $x \in \mathbb{R}^d$ ,  $h \in H$  and cyclic permutation groups  $H$ . Consequently, they utilize a cyclic permutation equivariant map,  $\varphi^H(\cdot)$ , to generate attention coefficients  $\alpha^H(h)$ ,  $h \in H$ , with which the corresponding elements  $h$  are modulated. As a result, their proposed attention strategy is  $H$ -equivariant. To preserve translation equivariance, and hence,  $G$ -equivariance,  $\varphi^H$  is re-utilized at every spatial position  $x \in \mathbb{R}^d$ . This is equivalent to combining  $\varphi^H$  with a pointwise filter on  $\mathbb{R}^d$ . Romero and Hoogendoorn [315] found that equivariance to cyclic groups  $H$ , can *only* be achieved by constraining  $\varphi^H$  to have a *circulant structure*. This is equivalent to a convolution with a filter  $\psi$ , whose group representations  $\mathcal{L}_h$  induces cyclical permutations of itself (Fig. 8.4). Hence, it resembles a group convolution by which Thm. 8.3.1 is satisfied.

The work of Romero and Hoogendoorn [315] exclusively performs attention on the  $h$  component of the group elements  $g = (x, h) \in G$  and is only defined for (block) cyclic groups. Consequently, it does not consider spatial relationships during attention (Fig. 8.1) and is not applicable to general groups. Conversely, our proposed framework allows for simultaneous attention on both components of the group elements  $g = (x, h)$  in a  $G$  equivariance preserving manner.

### 8.3.2 Efficient Group Equivariant Attention Maps

Attentive group convolutions impose the generation of an additional attention map  $\alpha : G \times G \rightarrow [0, 1]^{N_c}$ , which is computationally demanding. To reduce this computational burden, we exploit the fact that visual data is defined on  $\mathbb{R}^d$  and, hence, relevant groups are affine, to provide an efficient factorization of the attention map  $\alpha$ .

In Sec. 8.2.3 we indicated that attention coefficients  $\alpha$  can be equivariantly factorized into spatial and channel components. We build upon this idea and factorize attention via:

$$\alpha_{\tilde{c}}(g, \tilde{g}) := \alpha^{\mathcal{X}}((x, h), (\tilde{x}, \tilde{h})) \alpha_{\tilde{c}}^{\mathcal{C}}(h, \tilde{h})$$

where  $\alpha^{\mathcal{X}}$  attends for spatial relations without considering channel characteristics and  $\alpha^{\mathcal{C}}$  attends for patterns in the channel- and  $H$ -axis, but ignores spatial patterns. We thus factorize  $\alpha$  into a *spatial attention map*  $\alpha^{\mathcal{X}} : G \times G \rightarrow [0, 1]$  and a *channel attention map*  $\alpha^{\mathcal{C}} : H \times H \rightarrow [0, 1]^{N_c}$ . Findings in literature have shown that, for visual data, attention maps are almost equivalent for different query positions and thus, only query-independent dependencies are learnt [39, 479]. Based on this observation, we further simplify  $\alpha^{\mathcal{X}}$  to be invariant over spatial positions either at the input or output space. Since separate

---

<sup>5</sup>This resembles a depth-wise separable convolution [58] with the first convolution given by global average pooling.

convolutional filters  $\psi$  could possibly benefit from different attention maps, we omit spatial positions in the input space (see Sec. 8.3.2 for details). In other words, we replace  $\alpha^{\mathcal{X}}(g, \tilde{g})$  with  $\alpha^{\mathcal{X}}(g, \tilde{h})$ , an spatial position invariant attention map over the input space:  $\alpha^{\mathcal{X}} : G \times H \rightarrow [0, 1]$ .

Conveniently, attention coefficients of type  $\alpha : \mathbb{R}^d \times H \rightarrow [0, 1]^{N_c}$  can be interpreted as functions on  $\mathbb{R}^d$  with pointwise visualizations  $\tilde{x} \mapsto \alpha(\tilde{x}, \tilde{h})$  for each  $\tilde{x} \in \mathbb{R}^d$ . Resultantly, we are able to aid the interpretability of the learned concepts and of the attended symmetries (e.g., Figs. 8.7, 8.8, G.3).

### The Attention Operator $\mathcal{A}$

Recall that the attention map  $\alpha$  is computed via an attention operator  $\mathcal{A}$ . In the most general case,  $\alpha$  and, hence  $\mathcal{A}$ , is a function of both the input signal  $f$  and the filter  $\psi$ . In order to define  $\mathcal{A}$  as such, we generalize the approach of Woo et al. [431] such that: (1) equivariance to general symmetry groups is preserved and (2) the attention maps depend on the filter  $\psi$  as well.

Let  $\phi^{\mathcal{C}} : \tilde{f} \mapsto s^{\mathcal{C}} = \{s_{\text{avg}}^{\mathcal{C}}, s_{\max}^{\mathcal{C}}\}$ ,  $s_i^{\mathcal{C}} : H \times H \rightarrow \mathbb{R}^{N_c}$  and  $\phi^{\mathcal{X}} : \tilde{f} \mapsto s^{\mathcal{X}} = \{s_{\text{avg}}^{\mathcal{X}}, s_{\max}^{\mathcal{X}}\}$ ,  $s_i^{\mathcal{X}} : G \times G \rightarrow \mathbb{R}$  be functions that generate channel ( $s^{\mathcal{C}}$ ) and spatial statistics ( $s^{\mathcal{X}}$ ), respectively, from an intermediary vector valued signal  $\tilde{f} : G \times G \rightarrow \mathbb{R}^{N_c}$  containing information both from the input and output spaces. Analogously to Woo et al. [431], we compute spatial and channel statistics to reduce the dimensionality of the input. However, in contrast to them, we compute these statistics from intermediary convolutional maps  $\tilde{f}$  rather than from the input signal  $f$  directly.<sup>6</sup> As a result, we take the influence of the filter  $\psi$  into account during the computation of the attention maps. Following the simplifications proposed in Sec. 8.3.2 for  $\alpha^{\mathcal{X}}$ , we can further reduce  $s_i^{\mathcal{X}}$  and  $\tilde{f}$  to functions of the form  $s_i^{\mathcal{X}} : G \times H \rightarrow \mathbb{R}$  and  $\tilde{f} : G \times H \rightarrow \mathbb{R}^{N_c}$ , respectively. Consequently, we define:

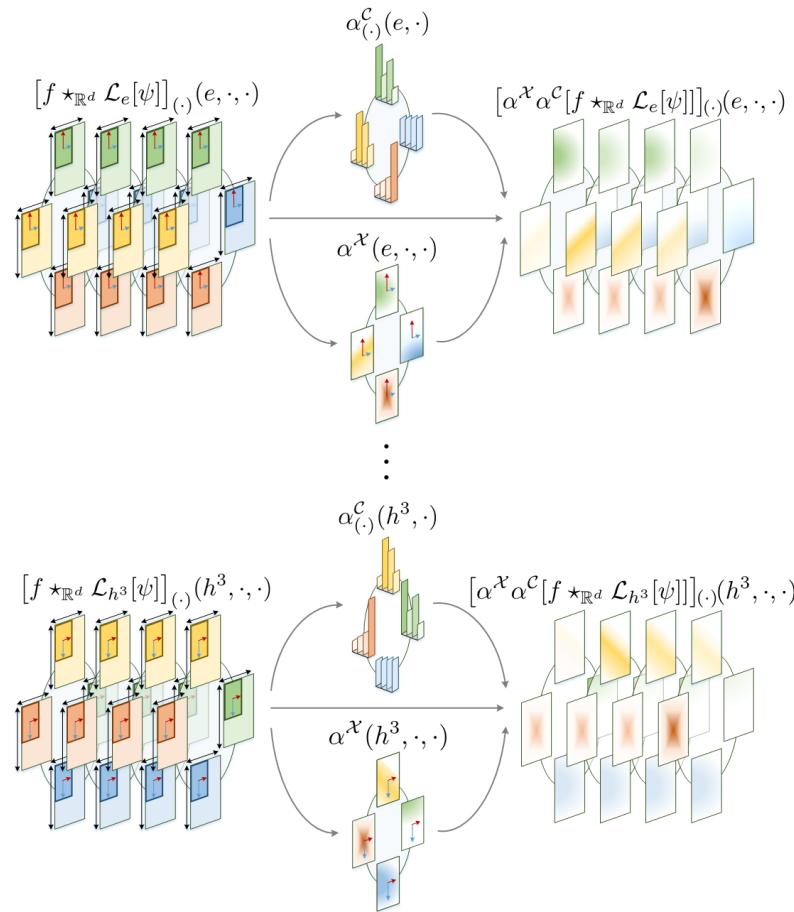
$$\tilde{f} = \{\tilde{f}_{\tilde{c}}\}_{\tilde{c}=1}^{N_c}, \quad \tilde{f}_{\tilde{c}}(x, h, \tilde{h}) := [\tilde{f}_{\tilde{c}} \star_{\mathbb{R}^d} \mathcal{L}_h[\psi_{\tilde{c}}]](x, \tilde{h}), \quad (8.16)$$

which is the intermediary result of the convolution between the input  $f$  and the  $h$ -transformation of the filter  $\psi$ ,  $\mathcal{L}_h[\psi]$  before pooling over  $\tilde{c}$  and  $\tilde{h}$  (Fig. 8.5, Eq. 8.13).

**Channel Attention.** Let  $\varphi^{\mathcal{C}} : s^{\mathcal{C}} \mapsto \alpha^{\mathcal{C}}$  be a function that generates a channel attention map  $\alpha^{\mathcal{C}} : H \times H \rightarrow [0, 1]^{N_c}$  from a vector of channel statistics  $s^{\mathcal{C}} : H \times H \rightarrow \mathbb{R}^{N_c}$  of the intermediate representation  $\tilde{f}$ . Our channel attention computation is analogous to that of Woo et al. [431] based on two fully connected layers. However, in our case, each linear layer is parametrized by a *matrix-valued kernel*  $\mathbf{W}_i : H \rightarrow \mathbb{R}^{N_{\text{out}} \times N_{\text{in}}}$ , which we shift via left-

---

<sup>6</sup>This is why the statistics  $s_i^{\mathcal{C}}, s_i^{\mathcal{X}}$  receive tuples  $(h, \tilde{h})$ ,  $(g, \tilde{g})$ , respectively, as input, as opposed to single argument inputs which often emerge in several prior works on visual attention.



**Figure 8.5:** Attentive group convolution on the roto-translation group  $SE(2)$ . In contrast to group convolutions (Fig. 8.2, Eq. 8.13), attentive group convolutions utilize channel  $\alpha^C$  and spatial  $\alpha^X$  attention to modulate the intermediary convolutional responses  $[f *_{\mathbb{R}^2} \mathcal{L}_h[\psi]]$  before pooling over the  $\tilde{c}$  and  $\tilde{h}$  axes.

regular representations  $\mathcal{L}_h[\mathbf{W}_i](\tilde{h}) = \mathbf{W}_i(h^{-1}\tilde{h})$  to guarantee equivariance (Thm. 8.3.1):

$$\begin{aligned} \alpha^C(h, \tilde{h}) &= \varphi^C[s^C](h, \tilde{h}) \\ &= \sigma\left(\left[\mathbf{W}_2(h^{-1}\tilde{h}) \cdot [\mathbf{W}_1(h^{-1}\tilde{h}) \cdot s_{\text{avg}}^C(h, \tilde{h})]^+\right] + \left[\mathbf{W}_2(h^{-1}\tilde{h}) \cdot [\mathbf{W}_1(h^{-1}\tilde{h}) \cdot s_{\text{max}}^C(h, \tilde{h})]^+\right]\right) \end{aligned} \quad (8.17)$$

with  $[\cdot]^+$  the ReLU function,  $\sigma$  the sigmoid function,  $r$  a reduction ratio and  $\mathbf{W}_1 : H \rightarrow \mathbb{R}^{\frac{N_c}{r} \times N_c}$ ,  $\mathbf{W}_2 : H \rightarrow \mathbb{R}^{N_c \times \frac{N_c}{r}}$  filters defined on  $H$ .

**Spatial Attention.** Let  $\varphi^X : s^X \mapsto \alpha^X$  be a function that generates a spatial attention map  $\alpha^X : G \times H \rightarrow [0, 1]$  from channel statistics  $s^X : G \times H \rightarrow \mathbb{R}^2$ , in which per input  $\tilde{h} \in H$  and output  $g \in G$ , the mean and max value is taken over the channel axis. Similarly to Woo et al. [431], spatial attention  $\alpha^X$  is then defined as:

$$\begin{aligned} \alpha^X(x, h, \tilde{h}) &= \varphi^X(s^X)(x, h, \tilde{h}) \\ &= \sigma\left([s^X \star_{\mathbb{R}^d} \mathcal{L}_h[\psi^X]]\right)(x, \tilde{h}) \end{aligned} \quad (8.18)$$

with  $\psi^X : G \rightarrow \mathbb{R}^2$  a group convolutional filter.

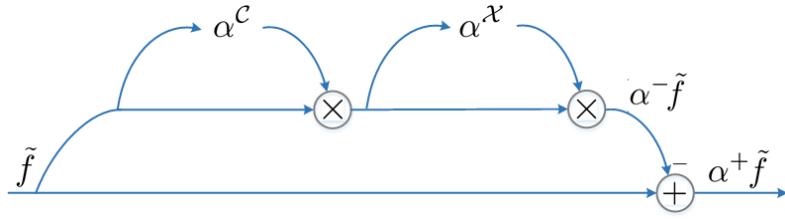
**Full Attention.** Woo et al. [431] carried out extensive experiments to find the best performing configuration to combine channel and spatial attention maps for the  $\mathbb{R}^d$  case, e.g., in parallel, serially starting with channel attention, serially starting with spatial attention. Based on their results we adopt their best performing configuration, i.e., *serially starting with channel attention*, for the  $G$  case (Fig. 8.6).

Recall that  $\tilde{f}$  is the intermediary result from the convolution between the input  $f$  and the  $h$ -transformation of the filter  $\psi$  before pooling over  $\tilde{c}$  and  $\tilde{h}$ . We perform attention on top of  $\tilde{f}$  (Fig. 8.6), where  $\alpha^C$  and  $\alpha^X$  are computed by Eqs. 8.17, 8.18, respectively. Resultantly, the attentive group convolution is computed as:

$$[f \star_G^\alpha \psi](x, h) = \sum_{\tilde{c}=1}^{N_c} \int_H \alpha^X(x, h, \tilde{h}) \alpha_{\tilde{c}}^C(h, \tilde{h}) \tilde{f}(x, h, \tilde{h}) d\tilde{h} \quad (8.19)$$

### 8.3.3 The Residual Attention Branch

Based on the findings of He et al. [143], several visual attention approaches propose to utilize residual blocks with direct connections during the course of attention to facilitate gradient flow [39, 157, 279, 419, 431]. However, these approaches calculate the final attention map  $\alpha^+$  as the sum of the direct connection  $\mathbf{1}$  and the attention map obtained from the attention branch  $\alpha$ , i.e.,  $\alpha^+ = \mathbf{1} + \alpha$ . Consequently, the obtained attention map  $\alpha^+ : \mathbb{R}^2 \rightarrow [1, 2]^{N_c}$  is *restricted* to the interval  $[1, 2]$  and the network loses its ability to suppress input components. Inspired by the aforementioned works, we propose to calculate attention in what we call a *residual attention branch* (Fig. 8.6). Specifically, we utilize the attention branch to calculate a *residual attention map* defined as  $\alpha^- = (\mathbf{1} - \alpha^+)$ ;  $\alpha^- : G \times G \rightarrow [0, 1]$ . Next, we subtract the residual attention map  $\alpha^-$  from the direct



**Figure 8.6:** Sequential channel and spatial attention performed on the residual attention branch (Sec. 8.3.3).

connection 1 to obtain the resultant attention map  $\alpha^+$ , i.e.,  $\alpha^+ = \mathbf{1} - \alpha^-$ . As a result, we are able to produce attention maps  $\alpha^+$  that span the  $[0, 1]$  interval while preserving the benefits of the direct connections of He et al. [143].

### 8.3.4 Attentive group convolution as a sequence of group convolutions and point-wise non-linearities

CNNs are usually organized in layers and hence, the input  $f$  is usually convolved in parallel with a set of  $N_o$  filters  $\{\psi_o\}_{o=1}^{N_o}$ . As outlined in the previous section, this implies that the attention maps can change as a function of the current filter  $\psi_o$ . One assumption broadly utilized in visual attention is that these maps do not depend on the filters  $\{\psi_o\}_{o=1}^{N_o}$ , and, hence, that  $\alpha$  is a sole function of the input signal  $f$  [86, 157, 279, 315, 431]. Consequently, the attention coefficients  $\alpha$  are reduced from a function  $\alpha : G \times G \rightarrow [0, 1]^{N_e}$  (c.f., Eq. 8.14) to a function  $\alpha : G \rightarrow [0, 1]^{N_e}$ . In other words, attention becomes only dependent on  $g$  (see Eqs. 8.17-8.19) and thus, the generation of the attention maps  $\alpha^C$ ,  $\alpha^X$  can be shifted to the input feature map  $f$ . Resultantly, the attentive group convolution is reduced to a sequence of conventional group convolutions and point-wise non-linearities (Thm. 8.3.1), which further reduces the computational cost of attention:

$$[f \star_G^\alpha \psi] = [f^\alpha \star_G \psi] = [( \alpha^X \alpha^C f ) \star_G \psi] \quad (8.20)$$

## 8.4 Experiments

We validate our approach by exploring the effects of using attentive group convolutions in contrast to conventional ones. We compare the conventional group equivariant networks  $p4$ - and  $p4m$ -CNNs of Cohen and Welling [65] on the rotated MNIST and CIFAR-10 datasets with their corresponding attentive counterparts:  $\alpha$ - $p4$ -CNNs and  $\alpha$ - $p4m$ -CNNs, respectively; and the  $p4$ - and  $p4m$ -DenseNets of Veeling et al. [404] on the PCam dataset with their corresponding attentive counterparts:  $\alpha$ - $p4$ -DenseNet and  $\alpha$ - $p4m$ -DenseNets, respectively. Additionally, we explore the effects of only applying channel attention (e.g.,  $\alpha_{CH}$ - $p4$ -CNNs), spatial attention (e.g.,  $\alpha_{SP}$ - $p4$ -CNNs) and applying attention directly on the input (e.g.,  $\alpha_F$ - $p4$ -CNNs).

We notice that the network architectures in Cohen and Welling [65] and Romero and Hoogendoorn [315] used for the CIFAR-10 experiments are equivariant only approximately. This results from using odd-sized convolutional kernels with stride  $\geq 1$  on even-sized feature maps (see Appx. G.3 for a complete discussion). Since this effect distorts the equivariance property of our equivariant attention maps, i.e., they also become equivariant only approximately (Figs. G.2, G.3), this issue must be fixed. We achieve this by replacing strided convolutions in such regimes by conventional convolutions followed by a max-pooling layer.

For all our experiments we replicate as close as possible the training and evaluation strategies of the corresponding baselines, replace approximately equivariant networks by exact equivariant ones, and initialize any additional parameter in the same way as the corresponding baseline. Extended implementation details are provided in Appx. G.2.

#### 8.4.1 rot-MNIST

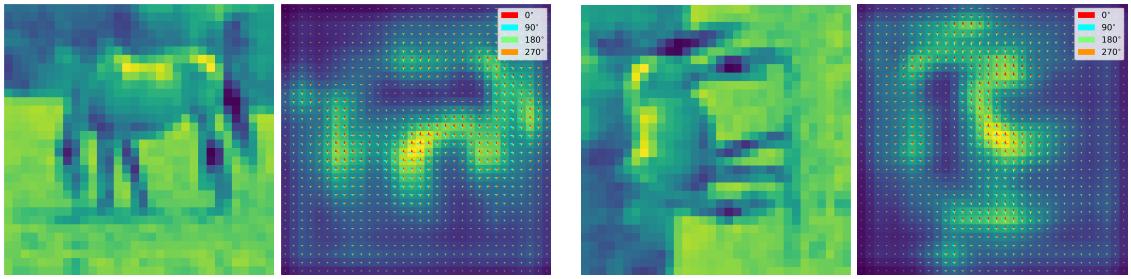
The rotated MNIST dataset [201] contains  $62k$  gray-scale 28x28 handwritten digits uniformly rotated for  $[0, 2\pi)$ . The dataset is split into training, validation and test sets of  $10k$ ,  $2k$  and  $50k$  images respectively. We compare  $p4$ -CNNs with all the corresponding attention variants previously mentioned. For our attention models, we utilize a filter size of 7 and a reduction ratio  $r$  of 2 on the attention branch. Since attentive group convolutions impose the learning of additional parameters, we also instantiate bigger  $p4$ -CNNs by increasing the number of channels uniformly at every layer to roughly match the number of parameters of the attentive versions. Furthermore, we compare our results with comparative attentive versions as defined in Romero and Hoogendoorn [315] ( $\alpha_{RH}$ ), which perform attention exclusively over the axis of rotations. Our results show that (1) attentive versions consistently outperform non-attentive ones, and that (2) using attention over the whole group leads further improves accuracy (Tab. 8.1).

#### 8.4.2 CIFAR-10

The CIFAR-10 dataset [195] consists of  $60k$  real-world 32x32 RGB images uniformly drawn from 10 classes. The dataset is split into training, validation and test sets of  $40k$ ,  $10k$  and  $10k$  images, respectively. We compare the  $p4$  and  $p4m$  versions of the All-CNN [365] and the Resnet44 [143] in Cohen and Welling [65] with attentive variations. For all our attention models, we utilize a filter size of 7 and a reduction ratio  $r$  of 16 on the attention branch. Unfortunately, attentive group convolutions impose an unfeasible increment on the memory requirements for this dataset.<sup>7</sup> Resultantly, we are only able to compare the  $\alpha_F$  variations of the corresponding networks. Our results show that attentive  $\alpha_F$  networks consistently outperform non-attentive ones (Tab. 8.2). Moreover, we

---

<sup>7</sup>the  $\alpha$ - $p4$  All-CNN requires approx. 72GB of CUDA memory, as opposed to 5GBs for the  $p4$ -All-CNN. This is due to the storage of the intermediary convolution responses required for the calculation of the attention weights (Eqs. 8.17- 8.19)



**Figure 8.7:** Equivariant attention maps on the roto-translation group  $SE(2)$ . The predicted attention maps behave equivariantly under group symmetries. The arrows depict the strength of the filter responses at the corresponding orientations.

**Table 8.1:** Test error rates on rot-MNIST (mean and std over 5 random seeds).

| NETWORK                   | TEST ERROR (%)                      | PARAM. |
|---------------------------|-------------------------------------|--------|
| $p4$ -CNN                 | $2.048 \pm 0.045$                   | 24.61K |
| $\alpha_{RH}$ - $p4$ -CNN | $1.980 \pm 0.032$                   | 24.85K |
| $BIG_{19}$ - $p4$ -CNN    | $1.796 \pm 0.035$                   | 77.54K |
| $\alpha$ - $p4$ -CNN      | <b><math>1.696 \pm 0.021</math></b> | 73.13K |
| $BIG_{15}$ - $p4$ -CNN    | $1.848 \pm 0.019$                   | 50.42K |
| $\alpha_{CH}$ - $p4$ -CNN | <b><math>1.825 \pm 0.048</math></b> | 48.63K |
| $\alpha_{SP}$ - $p4$ -CNN | <b><math>1.761 \pm 0.027</math></b> | 49.11K |
| $BIG_{11}$ - $p4$ -CNN    | $1.996 \pm 0.083$                   | 29.05K |
| $\alpha_F$ - $p4$ -CNN    | <b><math>1.795 \pm 0.028</math></b> | 29.46K |

**Table 8.2:** Test error rates on CIFAR10 and augmented CIFAR10+.

| NETWORK  | TYPE               | CIFAR10      | CIFAR10+     | PARAM. |
|----------|--------------------|--------------|--------------|--------|
| ALL-CNN  | $p4$               | 9.32         | 8.91         | 1.37M  |
|          | $\alpha_F$ - $p4$  | <b>8.8</b>   | <b>7.05</b>  | 1.40M  |
|          | $p4m$              | 7.61         | 7.48         | 1.22M  |
|          | $\alpha_F$ - $p4m$ | <b>6.93</b>  | <b>6.53</b>  | 1.25M  |
| RESNET44 | $p4m$              | 15.72        | 15.4         | 2.62M  |
|          | $\alpha_F$ - $p4m$ | <b>10.82</b> | <b>10.12</b> | 2.70M  |

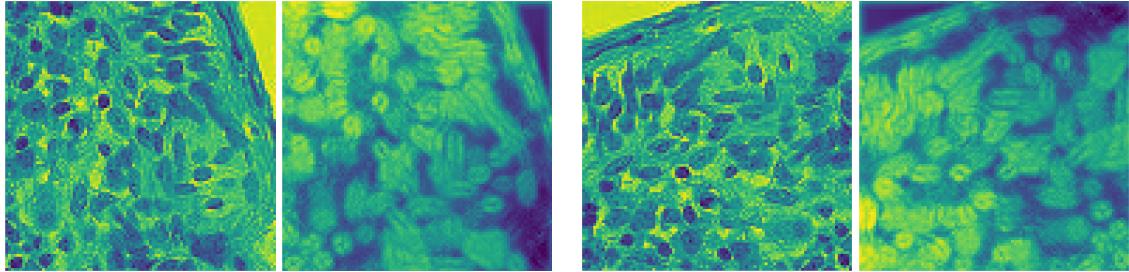
demonstrate that our proposed networks focus on relevant parts of the input and that the predicted attention maps behave equivariantly for group symmetries (Figs. 8.7, G.3).

#### 8.4.3 PCam

The PatchCamelyon dataset [404] consists of 327k 96x96 RGB image patches of tumorous / non-tumorous breast tissues extracted from the Camelyon16 dataset [16], where each patch was labelled as tumorous if the central region (32x32) contained at least one tumour pixel as given by the original annotation in Bejnordi et al. [16]. We compare the  $p4$  and  $p4m$  versions of the DenseNet [161] in Veeling et al. [404] with attentive variants. For all our attention models, we utilize a filter size of 7 and a reduction ratio  $r$  of 16 on the attention branch. Similarly to the CIFAR-10 case, we restrict our experiments to  $\alpha_F$  attentive networks due to computational constraints. Our results show that attentive  $\alpha_F$

**Table 8.3:** Test error rates on PCam.

| NETWORK  | TYPE           | TEST ERROR (%) | PARAM.  |
|----------|----------------|----------------|---------|
| DENSENET | $\mathbb{Z}^2$ | 15.93          | 130.60K |
|          | $p4$           | 12.45          | 129.65K |
|          | $\alpha_F-p4$  | <b>11.34</b>   | 140.45K |
|          | $p4m$          | 11.64          | 124.21K |
|          | $\alpha_F-p4m$ | <b>10.88</b>   | 141.22K |



**Figure 8.8:** Equivariant attention maps on the PCam dataset. The predicted attention maps behave equivariantly for group symmetries. In addition, the network learns to focus on the nuclei of the cells and to remove background elements during training.

consistently outperform non-attentive ones (Tab. 8.3). Interestingly, the  $\alpha_F-p4$ -DenseNet is already able to outperform the  $p4m$ -DenseNet without attention. Furthermore, our equivariant attention maps show that the network learns to focus on the nuclei of the cells and to remove the background in a group equivariant fashion (Fig. 8.8).

## 8.5 Discussion and future work

Our results show that attentive group convolutions can be utilized as a drop-in replacement for standard and group equivariant convolutions that simultaneously facilitates the interpretability of the network decisions. Similarly to convolutional and group convolutional networks, attentive group convolutional networks also benefit of data augmentation. Interestingly, however, we also see that including additional symmetries reduces the effect of augmentations given by group elements. This finding supports the intuition that symmetry variants of the same concept are learned independently for non-equivariant networks (see Fig. 2 in [196]). The main shortcoming of our approach is its computational burden. As a result, the application of  $\alpha$ -networks is computationally unfeasible for networks with several layers or channels. We believe, however, by extrapolation of our results on rot-MNIST, that further performance improvements are to be expected for  $\alpha$  variations, should hardware requirements suffice.

Group convolutional networks have recently been proven very successful in medical imaging applications [18, 199, 429]. Since explainability plays a crucial role here, we believe that our attentive maps could be of high relevance to aid the explainability of the network decisions. Moreover, since our attention maps are guaranteed to be equivari-

ant to transformations in the considered group, it is ensured that the predicted attention maps will be consistent across group symmetries. We believe this to be of crucial importance for rotation invariant tasks. Illustratively, in contrast to vanilla attentive CNNs, a malignant tissue will be ensured to generate consistent attention maps regardless of the orientation at which it has been provided to the network.

In future work, we want to explore ways to reduce the computational cost of full attention networks. If successful, we consider feasible to obtain a direct performance boost over our CIFAR-10 and PCam experimental results, without extensive additional memory requirements. Furthermore, we want to extend our work to symmetry groups defined on 3D. By doing so, we expect the range of possible applications of our work to reach several other important applications such as 3D medical imaging applications like CT-scans and other voxel-based representations.

## 8.6 Conclusion

We introduced attentive group convolutions, a generalization of the group convolution in which attention is utilized to explicitly highlight meaningful relationships among symmetries. We provided a general mathematical framework for group equivariant visual attention and indicated that prior work on visual attention can be perfectly described as special cases of the attentive group convolution. Our experimental results indicate that attentive group equivariant networks consistently outperform conventional group equivariant ones. Furthermore, attentive group equivariant networks provide equivariant attention maps that behave predictively for symmetries of the group and with which the learned concepts can be visualized.

# 9

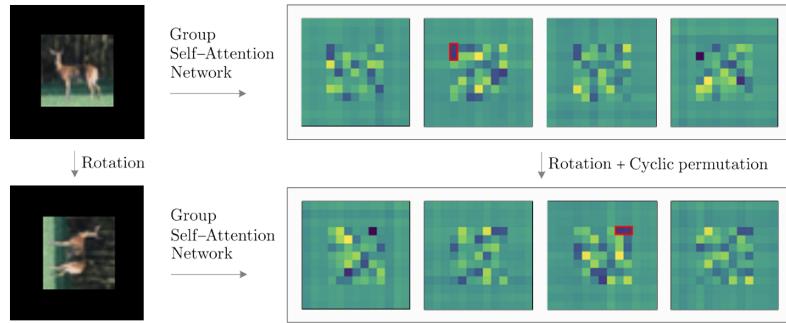
## Group Equivariant Stand-Alone Self-Attention

*Based on the paper:  
Group Equivariant Stand-Alone Self-Attention [314]*

### 9.1 Introduction

Recent advances in Natural Language Processing have been largely attributed to the rise of the *Transformer* [403]. Its key difference with previous methods, e.g., recurrent neural networks, convolutional neural networks (CNNs), is its ability to query information from all the input words simultaneously. This is achieved via the *self-attention operation* [10, 52], which computes the similarity between representations of words in the sequence in the form of *attention scores*. Next, the representation of each word is updated based on the words with the highest attention scores. Inspired by the capacity of transformers to learn meaningful inter-word dependencies, researchers have started applying self-attention in vision tasks. It was first adopted into CNNs by channel-wise attention [157] and non-local spatial modeling [419]. More recently, it has been proposed to replace CNNs with Transformers either partially [20] or entirely [301]. Contrary to discrete convolutional kernels, weights in self-attention are not tied to particular positions (Fig. H.1), yet self-attention layers can express any convolutional layer [72]. This flexibility allows leveraging long-range dependencies under a fixed parameter budget.

An orthogonal advancement to deep learning architectures is the incorporation of symmetries into the model itself. The seminal work by Cohen and Welling [65] provides



**Figure 9.1:** Behavior of feature representations in group self-attention networks. An input rotation induces a rotation plus a cyclic permutation to the intermediary feature representations of the network. Additional examples for all groups in this work and their usage are provided at [github.com/dwromero/g\\_selfatt/demo](https://github.com/dwromero/g_selfatt/demo).

a recipe to extend the *translation equivariance* of CNNs to other symmetry groups to improve generalization and sample- efficiency further (see Sec. 9.2). *Translation equivariance* is key to the success of CNNs. It describes the property that if a pattern is translated, its numerical descriptors are also translated, but not modified.

In this work, we introduce *group self-attention*, a self-attention formulation that grants equivariance to arbitrary symmetry groups. This is achieved by defining positional encodings invariant to the action of the group considered. In addition to generalization and sample-efficiency improvements provided by group equivariance, group equivariant self-attention networks (**GSA-Nets**) bring important benefits over group convolutional architectures: (i) *Parameter efficiency*: contrary to conventional discrete group convolutional kernels, where weights are tied to particular positions of neighborhoods on the group, group equivariant self-attention leverages long-range dependencies on group functions under a fixed parameter budget, yet it is able to express any group convolutional kernel. This allows for very expressive networks with low parameter count. (ii) *Steerability*: since the group acts directly on the positional encoding, **GSA-Nets** are *steerable* [424] by nature. This allows us to go beyond group discretizations that live in the grid without introducing interpolation artifacts.

### Contributions:

- We provide an extensive analysis on the equivariance properties of self-attention.
- We provide a general formulation to impose group equivariance to self-attention.
- We provide instances of self-attention equivariant to several symmetry groups.
- Our results show consistent improvements of **GSA-Nets** over non-equivariant ones.

## 9.2 Related Work

Several approaches exist which provide equivariance to various symmetry groups. The translation equivariance of CNNs has been extended to additional symmetries ranging from planar rotations [18, 53, 89, 123, 150, 215, 216, 252, 404, 424, 434] to spherical rotations [67, 69, 98–100, 423, 432], scaling [253, 322, 362, 433] and more general symmetry groups [17, 65, 68, 194, 374, 406, 422]. Importantly, all these approaches utilize discrete convolutional kernels, and thus, tie weights to particular positions in the neighborhood on which the kernels are defined. As group neighborhoods are (much) larger than conventional ones, the number of weights discrete group convolutional kernels require proportionally increases. This phenomenon is further exacerbated by *attentive group equivariant networks* [315, 318]. Since attention is used to leverage non-local information to aid local operations, non-local neighborhoods are required. However, as attention branches often rely on discrete convolutions, they effectively tie specific weights to particular positions on a large non-local neighborhood on the group. As a result, attention is bound to growth of the model size, and thus, to negative statistical efficiency. Differently, group self-attention is able to attend over arbitrarily large group neighborhoods under a fixed parameter budget. In addition, group self-attention is steerable by nature (Sec. 9.5.1) a property primarily exhibited by works carefully designed to that end.

Other way to detach weights from particular positions comes by parameterizing convolutional kernels as (constrained) neural networks [105, 382]. Introduced to handle irregularly-sampled data, e.g., point-clouds, networks parameterizing convolutional kernels receive relative positions as input and output their values at those positions. In contrast, our mappings change as a function of the input content. Most relevant to our work are the SE(3) and *Lie Transformers* [111, 163]. However, we obtain group equivariance via a generalization of positional encodings, whereas Hutchinson et al. [163] does so via operations on the Lie algebra and Fuchs et al. [111] does so via irreducible representations. In addition, our work prioritizes applications on visual data and extensively analyses theoretical aspects and properties of group equivariant self-attention.

## 9.3 Stand-Alone Self-Attention

In this section, we recall the mathematical formulation of self-attention and emphasize the role of the positional encoding. Next, we introduce a functional formulation to self-attention which will allow us to analyze and generalize its equivariance properties.

**Definition.** Let  $\mathbf{X} \in \mathbb{R}^{N \times C_{\text{in}}}$  be an input matrix consisting of  $N$  tokens of  $C_{\text{in}}$  dimensions each.<sup>1</sup> A self-attention layer maps an input matrix  $\mathbf{X} \in \mathbb{R}^{N \times C_{\text{in}}}$  to an output matrix  $\mathbf{Y} \in \mathbb{R}^{N \times C_{\text{out}}}$  as:

$$\mathbf{Y} = \text{SA}(\mathbf{X}) := \text{softmax}_{[ , :]}(\mathbf{A})\mathbf{X}\mathbf{W}_{\text{val}}, \quad (9.1)$$

---

<sup>1</sup>We consequently consider an image as a set of  $N$  discrete objects  $i \in \{1, 2, \dots, N\}$ .

with  $\mathbf{W}_{\text{val}} \in \mathbb{R}^{C_{\text{in}} \times C_h}$  the *value matrix*,  $\mathbf{A} \in \mathbb{R}^{N \times N}$  the *attention scores matrix*, and  $\text{softmax}_{[:, :]}(\mathbf{A})$  the *attention probabilities*. The matrix  $\mathbf{A}$  is computed as:

$$\mathbf{A} := \mathbf{X}\mathbf{W}_{\text{qry}}(\mathbf{X}\mathbf{W}_{\text{key}})^{\top}, \quad (9.2)$$

parameterized by *query* and *key matrices*  $\mathbf{W}_{\text{qry}}, \mathbf{W}_{\text{key}} \in \mathbb{R}^{C_{\text{in}} \times C_h}$ . In practice, it has been found beneficial to apply multiple self-attention operations, also called *heads*, in parallel, such that different heads are able to attend to different parts of the input. In this *multi-head self-attention* formulation, the output of  $H$  heads of output dimension  $C_h$  are concatenated and projected to  $C_{\text{out}}$  as:

$$\text{MHSA}(\mathbf{X}) := \text{concat}_{h \in [H]} [\text{SA}^{(h)}(\mathbf{X})] \mathbf{W}_{\text{out}} + \mathbf{b}_{\text{out}}, \quad (9.3)$$

with a *projection matrix*  $\mathbf{W}_{\text{out}} \in \mathbb{R}^{HC_h \times C_{\text{out}}}$  and a bias term  $\mathbf{b}_{\text{out}} \in \mathbb{R}^{C_{\text{out}}}$ .

### 9.3.1 The role of the positional encoding

Note that the self-attention operation defined in Eq. 9.3 is equivariant to permutations of the input rows of  $\mathbf{X}$ . That is, a permutation of the rows of  $\mathbf{X}$  will produce the same output  $\mathbf{Y}$  up to this permutation. Hence, self-attention is blind to the order of its inputs, i.e., it is a set operation. Illustratively, an input image is processed as a bag of pixels and the structural content is not considered. To alleviate this limitation, the input representations in self-attention are often enriched with a *positional encoding* that provides positional information of the set elements.

**Absolute positional encoding.** Vaswani et al. [403] introduced a (learnable) positional encoding  $\mathbf{P} \in \mathbb{R}^{N \times C_{\text{in}}}$  for each input position which is added to the inputs when computing the attention scores:

$$\mathbf{A} := (\mathbf{X} + \mathbf{P})\mathbf{W}_{\text{qry}}((\mathbf{X} + \mathbf{P})\mathbf{W}_{\text{key}})^{\top}. \quad (9.4)$$

More generally,  $\mathbf{P}$  can be substituted by any function that returns a vector representation of the position and can be incorporated by means of addition or concatenation, e.g., Zhao et al. [471]. This positional encoding injects additional structural information about the tokens into the model, which makes it susceptible to changes in the token's positions. Unfortunately, the model must learn to recognize similar patterns at every position independently as absolute positional encodings are *unique* to each position. This undesired data inefficiency is addressed by *relative positional encodings*.

**Relative positional encoding.** Introduced by Shaw et al. [348], relative encodings consider the *relative distance* between the query token  $i$  – the token we compute the representation of –, and the key token  $j$  – the token we attend to –. The calculation of the attention scores (Eq. 9.2) then becomes:

$$\mathbf{A}_{i,j}^{\text{rel}} := \mathbf{X}_i \mathbf{W}_{\text{qry}} ((\mathbf{X}_j + \mathbf{P}_{x(j)-x(i)}) \mathbf{W}_{\text{key}})^{\top}, \quad (9.5)$$

where  $\mathbf{P}_{x(j)-x(i)} \in \mathbb{R}^{1 \times C_{\text{in}}}$  is a vector representation of the relative shift and  $x(i)$  is the position of the token  $i$  as defined in Sec. 9.3.2. Consequently, similar patterns can be

recognized at arbitrary positions, as relative query-key distances always remain equal.

### 9.3.2 A functional formulation to self-attention

**Notation.** We denote by  $[n]$  the set  $\{1, 2, \dots, n\}$ . Given a set  $\mathcal{S}$  and a vector space  $\mathcal{V}$ ,  $L_{\mathcal{V}}(\mathcal{S})$  will denote the space of functions  $\{f : \mathcal{S} \rightarrow \mathcal{V}\}$ . Square brackets are used when functions are arguments, e.g.,  $\alpha[f]$  receives a function  $f$  as argument.

Let  $\mathcal{S} = \{i\}_{i=1}^N$  be a set of  $N$  elements. A matrix  $\mathbf{X} \in \mathbb{R}^{N \times C_{\text{in}}}$  can be interpreted as a vector-valued function  $f : \mathcal{S} \rightarrow \mathbb{R}^{C_{\text{in}}}$  that maps element sets  $i \in \mathcal{S}$  to  $C_{\text{in}}$ -dimensional vectors:  $f : i \mapsto f(i)$ . Consequently, a matrix multiplication,  $\mathbf{XW}_y^\top$ , of matrices  $\mathbf{X} \in \mathbb{R}^{N \times C_{\text{in}}}$  and  $\mathbf{W}_y \in \mathbb{R}^{C_{\text{out}} \times C_{\text{in}}}$  can be represented as a function  $\varphi_y : L_{\mathcal{V}_{C_{\text{in}}}}(\mathcal{S}) \rightarrow L_{\mathcal{V}_{C_{\text{out}}}}(\mathcal{S})$ ,  $\varphi_y : f(i) \mapsto \varphi_y(f(i))$ , parameterized by  $\mathbf{W}_y$ , between functional spaces  $L_{\mathcal{V}_{C_{\text{in}}}}(\mathcal{S}) = \{f : \mathcal{S} \rightarrow \mathbb{R}^{C_{\text{in}}}\}$  and  $L_{\mathcal{V}_{C_{\text{out}}}}(\mathcal{S}) = \{f : \mathcal{S} \rightarrow \mathbb{R}^{C_{\text{out}}}\}$ . Following this notation, we can represent the position-less attention scores calculation (Eq. 9.2) as:

$$\mathbf{A}_{i,j} = \alpha[f](i, j) = \langle \varphi_{\text{qry}}(f(i)), \varphi_{\text{key}}(f(j)) \rangle. \quad (9.6)$$

The function  $\alpha[f] : \mathcal{S} \times \mathcal{S} \rightarrow \mathbb{R}$  maps pairs of set elements  $i, j \in \mathcal{S}$  to the *attention score* of  $j$  relative to  $i$ . Therefore, the self-attention (Eq. 9.1) can be written as:

$$\begin{aligned} \mathbf{Y}_{i,:} &= \zeta[f](i) = \sum_{j \in \mathcal{S}} \sigma_j(\alpha[f](i, j)) \varphi_{\text{val}}(f(j)) \\ &= \sum_{j \in \mathcal{S}} \sigma_j(\langle \varphi_{\text{qry}}(f(i)), \varphi_{\text{key}}(f(j)) \rangle) \varphi_{\text{val}}(f(j)), \end{aligned} \quad (9.7)$$

where  $\sigma_j = \text{softmax}_j$  and  $\zeta[f] : \mathcal{S} \rightarrow \mathbb{R}^{C_h}$ . Finally, multi-head self-attention (Eq. 9.3) can be written as:

$$\begin{aligned} \text{MHSA}(\mathbf{X})_{i,:} &= m[f](i) = \varphi_{\text{out}}\left(\bigcup_{h \in [H]} \zeta^{(h)}[f](i)\right) \\ &= \varphi_{\text{out}}\left(\bigcup_{h \in [H]} \sum_{j \in \mathcal{S}} \sigma_j(\langle \varphi_{\text{qry}}^{(h)}(f(i)), \varphi_{\text{key}}^{(h)}(f(j)) \rangle) \varphi_{\text{val}}^{(h)}(f(j))\right), \end{aligned} \quad (9.8)$$

where  $\cup$  is the functional equivalent of the concatenation operator, and  $m[f] : \mathcal{S} \rightarrow \mathbb{R}^{C_{\text{out}}}$ .

**Local self-attention.** Recall that  $\alpha[f]$  assigns an attention scores to every other set element  $j \in \mathcal{S}$  relative to the query element  $i$ . The computational cost of self-attention is often reduced by restricting its calculation to a local neighborhood  $\mathcal{N}(i)$  around the query token  $i$  analogous in nature to the local receptive field of CNNs (Fig. H.1a). Consequently, *local self-attention* can be written as:

$$m[f](i) = \varphi_{\text{out}}\left(\bigcup_{h \in [H]} \sum_{j \in \mathcal{N}(i)} \sigma_j(\langle \varphi_{\text{qry}}^{(h)}(f(i)), \varphi_{\text{key}}^{(h)}(f(j)) \rangle) \varphi_{\text{val}}^{(h)}(f(j))\right). \quad (9.9)$$

Note that Eq. 9.9 is equivalent to Eq. 9.8 for  $\mathcal{N}(i) = \mathcal{S}$ , i.e. for global neighborhoods.

**Absolute positional encoding.** The absolute positional encoding is a function  $\rho : \mathcal{S} \rightarrow \mathbb{R}^{C_{\text{in}}}$  that maps set elements  $i \in \mathcal{S}$  to a vector representation of its position:  $\rho : i \mapsto \rho(i)$ . Note that this encoding is not dependent on functions defined on the set but *only* on the

set itself.<sup>2</sup> Hence, absolute position-aware self-attention (Eq. 9.4) can be written as:

$$m[f, \rho](i) = \varphi_{\text{out}} \left( \bigcup_{h \in [H]} \sum_{j \in \mathcal{N}(i)} \sigma_j (\langle \varphi_{\text{qry}}^{(h)}(f(i) + \rho(i)), \varphi_{\text{key}}^{(h)}(f(j) + \rho(j)) \rangle) \varphi_{\text{val}}^{(h)}(f(j)) \right). \quad (9.10)$$

The function  $\rho$  can be decomposed as two functions  $\rho^P \circ x$ : (i) the *position function*  $x : \mathcal{S} \rightarrow \mathcal{X}$ , which provides the position of set elements in the underlying space  $\mathcal{X}$  (e.g., pixel positions), and, (ii) the *positional encoding*  $\rho^P : \mathcal{X} \rightarrow \mathbb{R}^{C_{\text{in}}}$ , which provides vector representations of elements in  $\mathcal{X}$ . This distinction will be of utmost importance when we pinpoint where exactly (group) equivariance must be imposed to the self-attention operation (Sec. 9.4.3, Sec. 9.5).

**Relative positional encoding.** Here, positional information is provided in a relative manner. That is, we now provide vector representations of relative positions  $\rho(i, j) := \rho^P(x(j) - x(i))$  among pairs  $(i, j)$ ,  $i \in \mathcal{S}, j \in \mathcal{N}(i)$ . Consequently, relative position-aware self-attention (Eq. 9.5) can be written as:

$$m^r[f, \rho](i) = \varphi_{\text{out}} \left( \bigcup_{h \in [H]} \sum_{j \in \mathcal{N}(i)} \sigma_j (\langle \varphi_{\text{qry}}^{(h)}(f(i)), \varphi_{\text{key}}^{(h)}(f(j) + \rho(i, j)) \rangle) \varphi_{\text{val}}^{(h)}(f(j)) \right). \quad (9.11)$$

## 9.4 Equivariance Analysis of Self-Attention

In this section we analyze the equivariance properties of self-attention. Since the analysis largely relies on group theory, we provide all concepts required for proper understanding in Appx. H.2.

### 9.4.1 Group equivariance and equivariance for functions defined on sets

First we provide the general definition of group equivariance and refine it to relevant groups next. Additionally, we define the property of *unique equivariance* to restrict equivariance to a given group.

**Definition 9.4.0.1 (Group equivariance).** Let  $\mathcal{G}$  be a group (Def. H.2.0.1),  $\mathcal{S}, \mathcal{S}'$  be sets,  $\mathcal{V}, \mathcal{V}'$  be vector spaces, and  $\mathcal{L}_g[\cdot], \mathcal{L}'_g[\cdot]$  be the induced (left regular) representation (Def. H.2.0.4) of  $\mathcal{G}$  on  $L_{\mathcal{V}}(\mathcal{S})$  and  $L_{\mathcal{V}'}(\mathcal{S}')$ , respectively. We say that a map  $\varphi : L_{\mathcal{V}}(\mathcal{S}) \rightarrow L_{\mathcal{V}'}(\mathcal{S}')$  is equivariant to the action of  $\mathcal{G}$  – or  $\mathcal{G}$ -equivariant –, if it commutes with the action of  $\mathcal{G}$ . That is, if:

$$\varphi[\mathcal{L}_g[f]] = \mathcal{L}'_g[\varphi[f]], \quad \forall f \in L_{\mathcal{V}}(\mathcal{S}), \quad \forall g \in \mathcal{G}.$$

**Example 9.4.0.1 (Permutation equivariance).** Let  $\mathcal{S} = \mathcal{S}' = \{i\}_{i=1}^N$  be a set of  $N$  elements, and  $\mathcal{G} = \mathbb{S}_N$  be the group of permutations on sets of  $N$  elements. A map  $\varphi : L_{\mathcal{V}}(\mathcal{S}) \rightarrow L_{\mathcal{V}'}(\mathcal{S})$  is said to be equivariant to the action of  $\mathbb{S}_N$  – or permutation equivariant –, if:

$$\varphi[\mathcal{L}_{\pi}[f]](i) = \mathcal{L}'_{\pi}[\varphi[f]](i), \quad \forall f \in L_{\mathcal{V}}(\mathcal{S}), \quad \forall \pi \in \mathbb{S}_N, \quad \forall i \in \mathcal{S},$$

---

<sup>2</sup>Illustratively, one can think of this as a function returning a vector representation of pixel positions in a grid. Regardless of any transformation performed to the image, the labeling of the grid remains equal.

where  $\mathcal{L}_\pi[f](i) := f(\pi^{-1}(i))$ , and  $\pi : \mathcal{S} \rightarrow \mathcal{S}$  is a bijection from the set to itself. The element  $\pi(i)$  indicates the index to which the  $i$ -th element of the set is moved to as an effect of the permutation  $\pi$ . I.o.w.,  $\varphi$  is said to be *permutation equivariant* if it commutes with permutations  $\pi \in \mathfrak{S}_N$ , i.e., if permutations of its input produce equivalent permutations on its output.

Several of the transformations of interest, e.g., rotations, translations, are not defined on sets. Luckily, as we consider sets gathered from homogeneous spaces  $\mathcal{X}$  where these transformations are well-defined, e.g.,  $\mathbb{R}^2$  for pixels, there exists an injective map  $x : \mathcal{S} \rightarrow \mathcal{X}$  that associates a position in  $\mathcal{X}$  to each set element, the *position function*. In Appx. H.3 we show that the action of  $\mathcal{G}$  on such a set is well-defined and induces a group representation to functions on it. With this in place, we are now able to define equivariance of set functions to groups whose actions are defined on homogeneous spaces.

**Definition 9.4.0.2 (Equivariance of set functions to groups acting on homogeneous spaces).** Let  $\mathcal{G}$  be a group acting on two homogeneous spaces  $\mathcal{X}$  and  $\mathcal{X}'$ , let  $\mathcal{S}, \mathcal{S}'$  be sets and  $\mathcal{V}, \mathcal{V}'$  be vector spaces. Let  $x : \mathcal{S} \rightarrow \mathcal{X}$  and  $x' : \mathcal{S}' \rightarrow \mathcal{X}'$  be injective maps. We say that a map  $\varphi : L_{\mathcal{V}}(\mathcal{S}) \rightarrow L_{\mathcal{V}'}(\mathcal{S}')$  is *equivariant* to the action of  $\mathcal{G}$  – or  $\mathcal{G}$ -equivariant –, if it commutes with the action of  $\mathcal{G}$ . That is, if:

$$\varphi[\mathcal{L}_g[f]] = \mathcal{L}'_g[\varphi[f]], \quad \forall f \in L_{\mathcal{V}}(\mathcal{S}), \quad \forall g \in \mathcal{G},$$

where  $\mathcal{L}_g[f](i) := f(x^{-1}(g^{-1}x(i)))$ ,  $\mathcal{L}'_g[f](i) := f(x'^{-1}(g^{-1}x'(i)))$  are the induced (left regular) representation of  $\mathcal{G}$  on  $L_{\mathcal{V}}(\mathcal{S})$  and  $L_{\mathcal{V}'}(\mathcal{S}')$ , respectively. I.o.w.,  $\varphi$  is said to be  $\mathcal{G}$ -equivariant if a transformation  $g \in \mathcal{G}$  on its input produces an equal transformation on its output.

**Example 9.4.0.2 (Translation equivariance).** Let  $\mathcal{S}, \mathcal{S}'$  be sets and let  $x : \mathcal{S} \rightarrow \mathcal{X}$  and  $x' : \mathcal{S}' \rightarrow \mathcal{X}'$  be injective maps from the sets  $\mathcal{S}, \mathcal{S}'$  to the corresponding homogeneous spaces  $\mathcal{X}, \mathcal{X}'$  on which they are defined, e.g.,  $\mathbb{R}^d$  and  $\mathcal{G}$ . With  $(\mathcal{X}, +)$  the translation group acting on  $\mathcal{X}$ , we say that a map  $\varphi : L_{\mathcal{V}}(\mathcal{S}) \rightarrow L_{\mathcal{V}'}(\mathcal{S}')$  is *equivariant* to the action of  $(\mathcal{X}, +)$  – or *translation equivariant* –, if:

$$\varphi[\mathcal{L}_y[f]](i) = \mathcal{L}'_y[\varphi[f]](i), \quad \forall f \in L_{\mathcal{V}}(\mathcal{S}), \quad \forall y \in \mathcal{X},$$

with  $\mathcal{L}_y[f](i) := f(x^{-1}(x(i) - y))$ ,  $\mathcal{L}'_y[f](i) := f(x'^{-1}(x'(i) - y))$ . I.o.w.,  $\varphi$  is said to be *translation equivariant* if a translation on its argument produces an equal translation on its output.

## 9.4.2 Equivariance properties of self-attention

In this section we analyze the equivariance properties of the self-attention. The proofs to all the propositions stated in the main text are provided in Appx. H.6.

**Proposition 9.4.1.** *The global self-attention formulation without positional encoding (Eqs. 9.3, 9.8) is permutation equivariant. That is, it holds that:  $m[\mathcal{L}_\pi[f]](i) = \mathcal{L}_\pi[m[f]](i)$ .*

Note that permutation equivariance only holds for global self-attention. The local variant proposed in Eq. 9.9 reduces permutation equivariance to a smaller set of permutations where neighborhoods are conserved under permutation, i.e.,  $\mathfrak{S}_n = \{\pi \in \mathfrak{S}_N \mid j \in n(i) \rightarrow \pi(j) \in n(i), \forall i \in \mathcal{S}\}$ .

**Permutation equivariance induces equivariance to important (sub)groups.** Consider the cyclic group of order 4,  $\mathcal{Z}_4 = \{e, r, r^2, r^3\}$  which induces planar rotations by  $90^\circ$ .<sup>3</sup> As every rotation in  $\mathcal{Z}_4$  effectively induces a permutation of the tokens positions, it can be shown that  $\mathcal{Z}_4$  is a subgroup of  $\mathbb{S}_N$ , i.e.,  $\mathbb{S}_N \geq \mathcal{Z}_4$ . Consequently, maps equivariant to permutations are automatically equivariant to  $\mathcal{Z}_4$ . However, as the permutation equivariance constraint is harder than that of  $\mathcal{Z}_4$ -equivariance, imposing  $\mathcal{Z}_4$ -equivariance as a result of permutation equivariance is undesirable in terms of expressivity. Consequently, Ravanbakhsh et al. [304] introduced the concept of *unique  $\mathcal{G}$ -equivariance* to express the family of functions equivariant to  $\mathcal{G}$  but not equivariant to other groups  $\mathcal{G}' \geq \mathcal{G}$ :

**Definition 9.4.0.3 (Unique  $\mathcal{G}$ -equivariance).** Let  $\mathcal{G}$  be a subgroup of  $\mathcal{G}'$  (Def. H.2.0.2). We say that a map  $\varphi$  is uniquely  $\mathcal{G}$ -equivariant if and only if it is  $\mathcal{G}$ -equivariant but not  $\mathcal{G}'$ -equivariant for any group  $\mathcal{G}'$  from which  $\mathcal{G}$  is a subgroup.

In the following sections, we show that we can enforce unique equivariance not only to subgroups of  $\mathbb{S}_N$ , e.g.,  $\mathcal{Z}_4$ , but also to other interesting groups not contained in  $\mathbb{S}_N$ , e.g., groups of rotations finer than 90 degrees. This is achieved by enriching set functions with a proper positional encoding.

**Proposition 9.4.2.** Absolute position-aware self-attention (Eqs. 9.4, 9.10) is neither permutation nor translation equivariant. i.e.,  $m[\mathcal{L}_\pi[f], \rho](i) \neq \mathcal{L}_\pi[m[f, \rho]](i)$  and  $m[\mathcal{L}_y[f], \rho](i) \neq \mathcal{L}_y[m[f, \rho]](i)$ .

Though absolute positional encodings do disrupt permutations equivariance, they are unable to provide translation equivariance. We show next that translation equivariance is obtained via relative encodings.

**Proposition 9.4.3.** Relative position-aware self-attention (Eq. 9.11) is translation equivariant. That is, it holds that:  $m^r[\mathcal{L}_y[f], \rho](i) = \mathcal{L}_y[m^r[f, \rho]](i)$ .

### 9.4.3 Where exactly is equivariance imposed in self-attention?

In the previous section we have seen two examples of successfully imposing group equivariance to self-attention. Specifically, we see that no positional encoding allows for permutation equivariance and that a relative positional encoding allows for translation equivariance. For the latter, as shown in the proof of Prop. 9.4.3 (Appx. H.6), this comes from the fact that for all shifts  $y \in \mathcal{X}$ ,

$$\rho(x^{-1}(x(i) + y), x^{-1}(x(j) + y)) = \rho^P(x(j) + y - (x(i) + y)) = \rho^P(x(j) - x(i)) = \rho(i, j). \quad (9.12)$$

That is, from the fact that the relative positional encoding is invariant to the action of the translation group, i.e.,  $\mathcal{L}_y[\rho](i, j) = \rho(i, j)$ ,  $\forall y \in \mathcal{X}$ . Similarly, the absence of positional encoding – more precisely, the use of a constant positional encoding –, is what allows for permutation equivariance (Prop. 9.4.1, Appx. H.6). Specifically, constant positional

---

<sup>3</sup> $e$  represents a  $0^\circ$  rotation, i.e., the identity. The remaining elements  $r^j$  represent rotations by  $(90 \cdot j)^\circ$ .

encodings  $\rho_c(i) = c$ ,  $\forall i \in \mathcal{S}$  are invariant to the action of the permutation group, i.e.,  $\mathcal{L}_\pi[\rho_c](i) = \rho_c(i)$ ,  $\forall \pi \in \mathfrak{S}_N$ .

From these observations, we conclude that  $\mathcal{G}$ -equivariance is obtained by providing positional encodings which are *invariant to the action of the group  $\mathcal{G}$* , i.e., s.t.,  $\mathcal{L}_g[\rho] = \rho$ ,  $\forall g \in \mathcal{G}$ . Furthermore, unique  $\mathcal{G}$ -equivariance is obtained by providing positional encodings which are invariant to the action of  $\mathcal{G}$  but *not invariant to the action of any other group  $\mathcal{G}' \geq \mathcal{G}$* . This is a key insight that allows us to provide (unique) equivariance to arbitrary symmetry groups, which we provide next.

## 9.5 Group Equivariant Stand-Alone Self-Attention

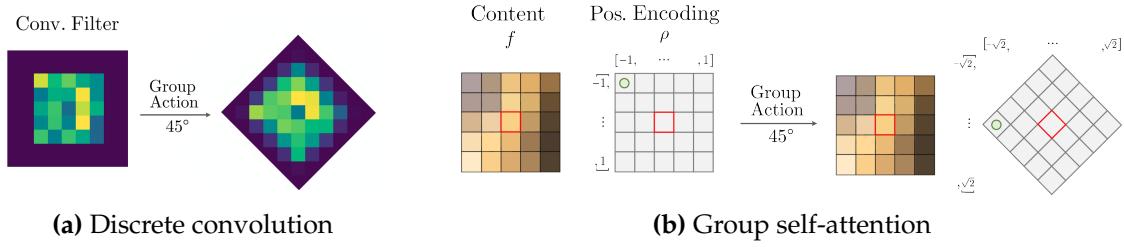
In Sec. 9.4.3 we concluded that unique  $\mathcal{G}$ -equivariance is induced in self-attention by introducing positional encodings which are invariant to the action of  $\mathcal{G}$  but not invariant to the action of other groups  $\mathcal{G}' \geq \mathcal{G}$ . However, this constraint does not provide any information about the expressivity of the mapping we have just made  $\mathcal{G}$ -equivariant. Let us first illustrate why this is important:

Consider the case of imposing rotation and translation equivariance to an encoding defined in  $\mathbb{R}^2$ . Since translation equivariance is desired, a relative positional encoding is required. For rotation equivariance, we must further impose the positional encoding to be equal for all rotations. That is  $\mathcal{L}_\theta[\rho](i, j) \stackrel{!}{=} \rho(i, j)$ ,  $\forall \theta \in [0, 2\pi]$ , where  $\mathcal{L}_\theta[\rho](i, j) := \rho^P(\theta^{-1}x(j) - \theta^{-1}x(i))$ , and  $\theta^{-1}$  depicts a rotation by  $-\theta$  degrees. This constraint leads to an isotropic positional encoding unable to discriminate among orientations, which in turn enforces rotation invariance instead of rotation equivariance.<sup>4</sup> This is alleviated by *lifting* the underlying function on  $\mathbb{R}^2$  to a space where rotations are explicitly encoded (Fig. 9.3). To this end, one performs self-attention operations for positional encodings  $\mathcal{L}_\theta[\rho]$  of varying values  $\theta$  and indexes their responses by the corresponding  $\theta$  value. Next, as rotations are now explicitly encoded, a positional encoding can be defined in this space which is able to discriminate among rotations (Fig. 9.4). This in turn allows for rotation equivariance instead of rotation invariance.

It has been shown both theoretically [303] and empirically [422] that the most expressive class of  $\mathcal{G}$ -equivariant functions is given by functions that follow the regular representation of  $\mathcal{G}$ . In order to obtain feature representations that behave that way, we introduce a lifting self-attention layer (Fig. 9.3, Eq. 9.14) that receives an input function on  $\mathbb{R}^d$  and produces a feature representation on  $\mathcal{G}$ . Subsequently, arbitrarily many group self-attention layers (Fig. 9.4, Eq. 9.16) interleaved with optional point-wise non-linearities can be applied. At the end of the network a feature representation on  $\mathbb{R}^d$  can be provided by pooling over  $\mathcal{H}$ . In short, we provide a pure self-attention analogous to [65].

---

<sup>4</sup>This phenomenon arises from the fact that  $\mathbb{R}^2$  is a quotient of the roto-translation group. Consequently, imposing group equivariance in the quotient space is equivalent to imposing an additional homomorphism of constant value over its cosets. Conclusively, the resulting map is of constant value over the rotation elements and, thus, unable to discriminate among them. See Ch. 3.1 [93] for an intuitive description.



**Figure 9.2:** Steerability analysis of discrete convolutions and group self-attention.

However, as the group acts directly on the positional encoding, our networks are *steerable* as well [424]. This allows us to go beyond group discretizations that live in the grid without introducing interpolation artifacts (Sec. 9.5.1).

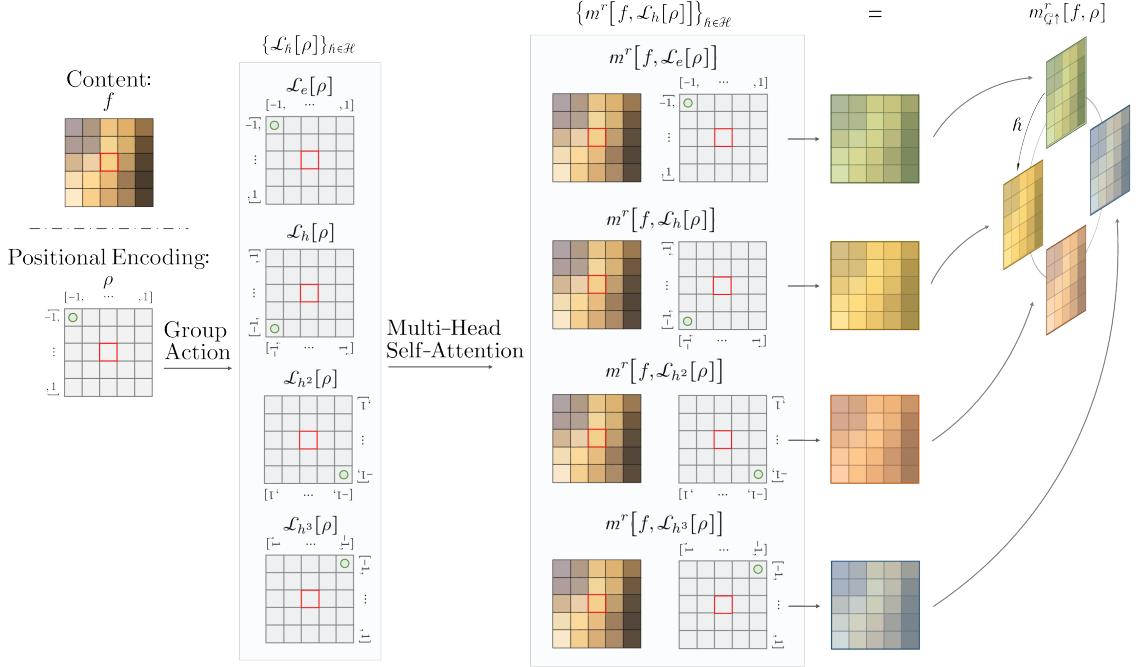
Though theoretically sound, neural architectures using regular representations are unable to handle continuous groups directly in practice. This is a result of the summation over elements  $\tilde{h} \in \mathcal{H}$  in Eq. 9.15, which becomes an integral for continuous groups. Interestingly, using discrete groups does not seem to be detrimental in practice. Our experiments indicate that performance saturates for fine discrete approximations of the underlying continuous group (Tab. 9.2). In fact, [422, Tab. 3] show via extensive experiments that networks using regular representations and fine enough discrete approximations consistently outperform networks handling continuous groups via irreducible representations. We conjecture this is a result of the networks receiving discrete signals as input. As the action of several group elements fall within the same pixel, no further improvement can be obtained.

### 9.5.1 Group self-attention is an steerable operation

Convolutional filters are commonly parameterized by weights on a discrete grid, which approximate the function implicitly described by the filter at the grid locations. Unfortunately, for groups whose action does not live in this grid, e.g.,  $45^\circ$  rotations, the filter must be interpolated. This is problematic as these filters are typically small and the resulting interpolation artifacts can be severe (Fig. 9.2a). *Steerable CNNs* tackle this problem by parameterizing convolutional filters on a continuous basis on which the action of the group is well-defined, e.g., *circular harmonics* [424], *B-splines* [17]. In group self-attention, the action of the group leaves the content of the image intact and only modifies the positional encoding (Figs. 9.3, 9.4). As the positional encoding lives on a continuous space, it can be transformed at an arbitrary grade of precision without interpolation (Fig. 9.2b).

### 9.5.2 Lifting and group self-attention

**Lifting self-attention (Fig. 9.3).** Let  $\mathcal{G} = \mathbb{R}^d \rtimes \mathcal{H}$  be an affine group (Def. H.2.0.3) acting on  $\mathbb{R}^d$ . The *lifting self-attention*  $m_{\mathcal{G}}^r[f, \rho] : L_V(\mathbb{R}^d) \rightarrow L_{V'}(\mathcal{G})$  is a map from functions on  $\mathbb{R}^d$  to functions on  $\mathcal{G}$  obtained by modifying the relative positional encoding  $\rho(i, j)$  by



**Figure 9.3:** Lifting self-attention on the roto-translation group for discrete rotations by 90 degrees (also called the  $\mathcal{Z}_4$  group). The  $\mathcal{Z}_4$  group is defined as  $\mathcal{H} = \{e, h, h^2, h^3\}$ , where  $h$  depicts a 90° rotation. The lifting self-attention corresponds to the concatenation of  $|\mathcal{H}| = 4$  self-attention operations between the input  $f$  and  $h$ -transformed versions of the positional encoding  $\mathcal{L}[\rho]$ ,  $\forall h \in \mathcal{H}$ . As a result, the model “sees” the input  $f$  at each of the rotations in the group at once. Since  $\mathcal{Z}_4$  is a cyclic group, i.e.,  $h^4 = e$ , functions on this group are often represented as responses on a ring (right side of the image).

the action of group elements  $h \in \mathcal{H}$ :  $\{\mathcal{L}_h[\rho](i, j)\}_{h \in \mathcal{H}}, \mathcal{L}_h[\rho](i, j) = \rho^P(h^{-1}x(j) - h^{-1}x(i))$ . It corresponds to the concatenation of multiple self-attention operations (Eq. 9.11) indexed by  $h$  with varying positional encodings  $\mathcal{L}_h[\rho]$ :

$$m_{\mathcal{G}}^r[f, \rho](i, h) = m^r[f, \mathcal{L}_h[\rho]](i) \quad (9.13)$$

$$= \varphi_{\text{out}} \left( \bigcup_{h \in [H]} \sum_{j \in \mathcal{N}(i)} \sigma_j \left( \langle \varphi_{\text{qry}}^{(h)}(f(i)), \varphi_{\text{key}}^{(h)}(f(i) + \mathcal{L}_h[\rho](i, j)) \rangle \right) \varphi_{\text{val}}^{(h)}(f(j)) \right). \quad (9.14)$$

**Proposition 9.5.1.** *Lifting self-attention is  $\mathcal{G}$ -equivariant. That is:  $m_{\mathcal{G}}^r[\mathcal{L}_g[f], \rho](i, h) = \mathcal{L}_g[m_{\mathcal{G}}^r[f, \rho]](i, h)$ .*

**Group self-attention (Fig. 9.4).** Let  $\mathcal{G} = \mathbb{R}^d \rtimes \mathcal{H}$  be an affine group acting on itself and  $f(i, \tilde{h}) \in L_V(\mathcal{G})$ ,  $i \in \mathcal{S}$ ,  $\tilde{h} \in \mathcal{H}$ , be a function defined on a set immersed with the structure of the group  $\mathcal{G}$ . That is, enriched with a positional encoding  $\rho((i, \tilde{h}), (j, \hat{h})) := \rho^P((x(j) - x(i), \tilde{h}^{-1}\hat{h}))$ ,  $i, j \in \mathcal{S}$ ,  $\tilde{h}, \hat{h} \in \mathcal{H}$ . The *group self-attention*  $m_{\mathcal{G}}^r[f, \rho] : L_V(\mathcal{G}) \rightarrow L_{V'}(\mathcal{G})$  is a map from functions on  $\mathcal{G}$  to functions on  $\mathcal{G}$  obtained by modifying the group positional encoding by the action of group elements  $h \in \mathcal{H}$ :  $\{\mathcal{L}_h[\rho]((i, \tilde{h}), (j, \hat{h}))\}_{h \in \mathcal{H}}$ ,

$\mathcal{L}_{\tilde{h}}[\rho]((i, \tilde{h}), (j, \hat{h})) = \rho^P(\tilde{h}^{-1}(x(j) - x(i)), \tilde{h}^{-1}(\tilde{h}^{-1}\hat{h}))$ . It corresponds to the concatenation of multiple self-attention operations (Eq. 9.11) indexed by  $h$  with varying positional encodings  $\mathcal{L}_h[\rho]$  and followed by a summation over the output domain along  $\tilde{h}$ :

$$m_G^r[f, \rho](i, h) = \sum_{\tilde{h} \in \mathcal{H}} m^r[\mathcal{L}_{\tilde{h}}[\rho]](i, \tilde{h}) \quad (9.15)$$

$$= \varphi_{\text{out}} \left( \bigcup_{h \in [H]} \sum_{\tilde{h} \in \mathcal{H}} \sum_{(j, \hat{h}) \in \mathcal{N}(i, \tilde{h})} \sigma_{j, \hat{h}} \left( \langle \varphi_{\text{qry}}^{(h)}(f(i, \tilde{h})), \varphi_{\text{key}}^{(h)}(f(j, \hat{h}) + \mathcal{L}_h[\rho]((i, \tilde{h}), (j, \hat{h}))) \rangle \right) \varphi_{\text{val}}^{(h)}(f(j, \hat{h})) \right). \quad (9.16)$$

In contrast to vanilla and lifting self-attention, the group self-attention neighborhood  $\mathcal{N}(i, \tilde{h})$  is now defined on the group. This allows distinguishing across group transformations, e.g., rotations.

**Proposition 9.5.2.** *Group self-attention is  $\mathcal{G}$ -equivariant. That is:  $m_G^r[\mathcal{L}_g[f], \rho](i, h) = \mathcal{L}_g[m_G^r[f, \rho]](i, h)$ .*

Non-unimodular groups, i.e., groups that modify the volume of the objects they act upon, e.g., the scale group, require a special treatment, which we provide in Appx. H.4.

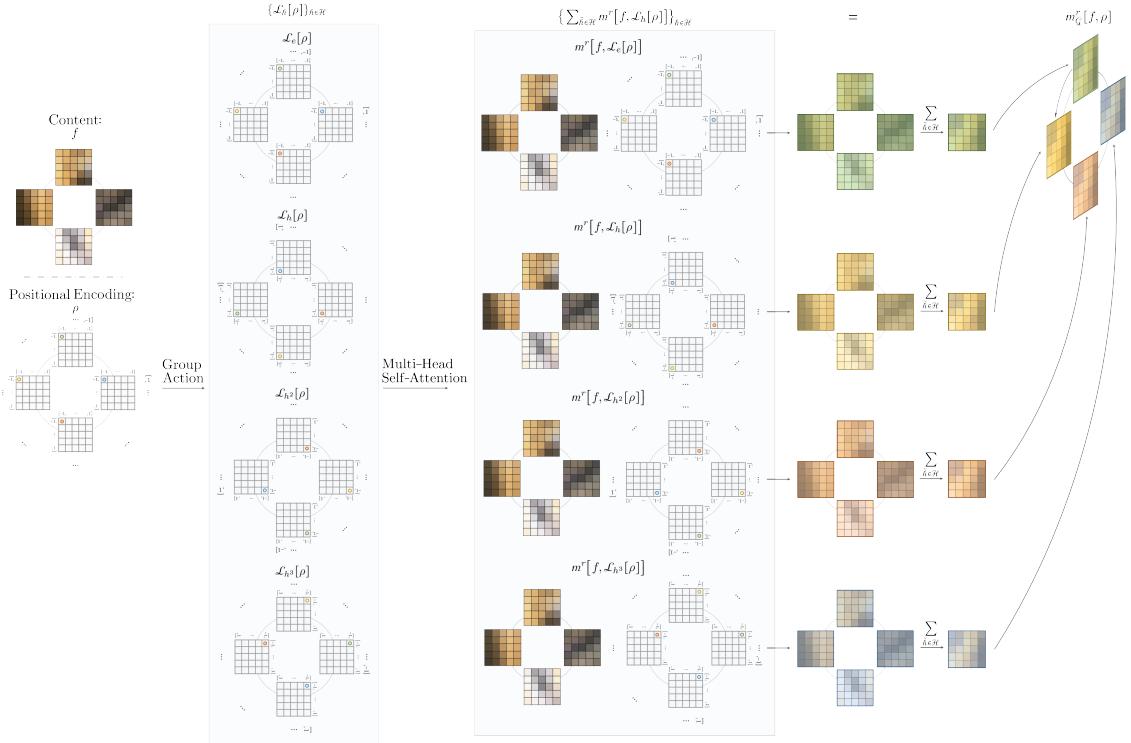
### 9.5.3 Group self-attention is a generalization of the group convolution

We have demonstrated that it is sufficient to define self-attention as a function on the group  $\mathcal{G}$  and ensure that  $\mathcal{L}_g[\rho] = \rho \forall g \in \mathcal{G}$  in order to enforce  $\mathcal{G}$ -equivariance. Interestingly, this observation is inline with the main statement of Kondor and Trivedi [194] for (group) convolutions: “*the group convolution on  $\mathcal{G}$  is the only (unique)  $\mathcal{G}$ -equivariant linear map*”. In fact, our finding can be formulated as a generalization of Kondor and Trivedi [194]’s statement as:

“*Linear mappings on  $\mathcal{G}$  whose positional encoding is  $\mathcal{G}$ -invariant are  $\mathcal{G}$ -equivariant.*”

This statement is more general than that of Kondor and Trivedi [194], as it holds for data structures where (group) convolutions are not well-defined, e.g., sets, and it is equivalent to Kondor and Trivedi [194]’s statement for structures where (group) convolutions are well-defined. It is also congruent with results complementary to [17, 68, 194] as well as several works on group equivariance handling set-like structures like point-clouds [82, 105, 111, 382] and symmetric sets [255].

In addition, we can characterize the expressivity of group self-attention. It holds that (i) group self-attention generalizes the group convolution and (ii) regular global group self-attention is an equivariant universal approximator. Statement (i) follows from the fact that any convolutional layer can be described as a multi-head self-attention layer provided enough heads [72], yet self-attention often uses larger receptive fields. As a result, self-attention is able to describe a larger set of functions than convolutions, e.g., Fig. H.1. Cordonnier et al. [72]’s statement can be easily extended to group self-attention by incorporating an additional dimension corresponding to  $\mathcal{H}$  in their derivations, and defining neighborhoods in this new space with a proportionally larger number of heads. Statement (ii) stems from the finding of Ravanbakhsh [303] that functions induced by regular



**Figure 9.4:** Lifting self-attention on the roto-translation group for discrete rotations by 90 degrees (also called the  $\mathcal{Z}_4$  group). The  $\mathcal{Z}_4$  group is defined as  $\mathcal{H} = \{e, h, h^2, h^3\}$ , where  $h$  depicts a  $90^\circ$  rotation. Analogous to lifting self-attention (Fig. 9.3), group self-attention corresponds to a concatenation of  $|\mathcal{H}| = 4$  self-attention operations between the input  $f$  and  $h$ -transformed versions of the positional encoding  $\mathcal{L}[\rho]$ ,  $\forall h \in \mathcal{H}$ . However, in contrast to lifting self-attention, both  $f$  and  $\rho$  are now defined on the group  $\mathcal{G}$ . Consequently, an additional sum over  $h$  is required during the operation (c.f., Eq. 9.16). Since  $\mathcal{Z}_4$  is a cyclic group, i.e.,  $h^4 = e$ , functions on  $\mathcal{Z}_4$  are often represented as responses on a ring (right side of the image).

group representations are equivariant universal approximators provided *full kernels*, i.e., global receptive fields. Global receptive fields are required to guarantee that the equivariant map is able to model any dependency among input components. Global receptive fields are readily utilized by our proposed regular global group self-attention and, provided enough heads, one can ensure that any such dependencies is properly modelled.

## 9.6 Experiments

We perform experiments on three image benchmark datasets for which particular forms of equivariance are desirable. We evaluate our approach by contrasting **GSA-Nets** equivariant to multiple symmetry groups. Additionally, we conduct an study on rotMNIST to evaluate the performance of **GSA-Nets** as a function of the neighborhood size. All our networks follow the structure shown in Fig. H.2 and vary only in the number of blocks and channels. We emphasize that both the architecture and the number of parameters in **GSA-Nets** is left unchanged regardless of the group used. Our results illustrate that **GSA-Nets** consistently outperform equivalent non-equivariant attention networks. We further compare **GSA-Nets** with convolutional architectures. Though our approach does not build upon these networks, this comparison provides a fair view to the yet present gap between self-attention and convolutional architectures in vision tasks, also present in their group equivariant counterparts.

**Efficient implementation of lifting and group self-attention.** Our self-attention implementation takes advantage of the fact that the group action only affects the positional encoding  $\mathbf{P}$  to reduce the total computational cost of the operation. Specifically, we calculate self-attention scores w.r.t. the content  $\mathbf{X}$  once, and reuse them for all transformed versions of the positional encoding  $\{\mathcal{L}_h[\rho]\}_{h \in \mathcal{H}}$ .

**Nomenclature.** We refer to translation equivariant self-attention models as `Z2_SA`. Reflection equivariant models receive the keyword `M`, e.g., `Z2M_SA`, and rotation equivariant models the keyword `Rn`, where `n` depicts the angle discretization. For example, `R8_SA` depicts a model equivariant to rotations by 45 degrees. Specific model architectures are provided in Appx. H.5.

**RotMNIST.** The rotated MNIST dataset [201] is a classification dataset often used as a standard benchmark for rotation equivariance. It consists of 62k gray-scale 28x28 uniformly rotated handwritten digits, divided into training, validation and test sets of 10k, 2k and 50k images. First, we study the effect of the neighborhood size on classification accuracy and convergence time. We train `R4_SA` networks for 300 epochs with vicinities  $N \times N$  of varying size (Tab. 9.1, Fig. 9.5). Since **GSA-Nets** optimize where to attend, the complexity of the optimization problem grows as a function of  $N$ . Consequently, models with big vicinities are expected to converge slower. However, as the family of functions describable by big vicinities contains those describable by small ones, models with big vicinities are expected to be at least as good upon convergence. Our results show that models with small vicinities do converge much faster (Fig. 9.5). However, though some

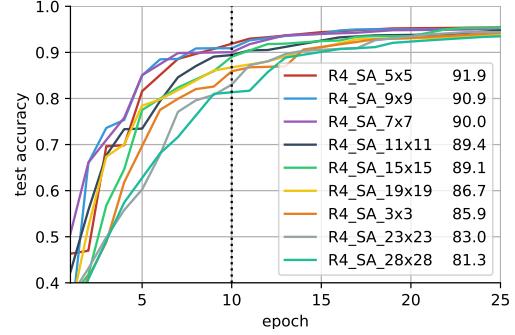
**Table 9.2:** Classification results. All convolutional architectures use 3x3 filters.

| ROT-MNIST                     |              |         | CIFAR10             |              |         | PATCH-CAMELYON                   |              |         |
|-------------------------------|--------------|---------|---------------------|--------------|---------|----------------------------------|--------------|---------|
| MODEL                         | ACC. (%)     | PARAMS. | MODEL               | ACC. (%)     | PARAMS. | MODEL                            | ACC. (%)     | PARAMS. |
| Z2_SA                         | 96.37        |         | Z2_SA               | 82.3         |         | Z2_SA                            | 83.04        |         |
| R4_SA                         | 97.46        |         | Z2M_SA              | <b>83.72</b> | 2.99M   | R4_SA                            | 83.44        |         |
| R8_SA                         | 97.90        | 44.67K  | Z2_CNN <sup>+</sup> | <b>90.56</b> | 1.37M   | R8_SA                            | 83.58        | 205.66K |
| R12_SA                        | <b>97.97</b> |         | +[65].              |              |         | R4M_SA                           | <b>84.76</b> |         |
| R16_SA                        | 97.66        |         |                     |              |         | Z2_CNN <sup>†</sup>              | 84.07        | 130.60K |
| Z2_CNN <sup>+</sup>           | 94.97        | 21.75K  |                     |              |         | R4_CNN <sup>†</sup>              | 87.55        | 129.65K |
| R4_CNN <sup>†</sup>           | 98.21        | 77.54K  |                     |              |         | R4M_CNN <sup>†</sup>             | 88.36        | 124.21K |
| $\alpha$ -R4_CNN <sup>†</sup> | <b>98.31</b> | 73.13K  |                     |              |         | $\alpha_F$ -R4_CNN <sup>†</sup>  | 88.66        | 140.45K |
| +[Cohen and Welling [65]]     |              |         |                     |              |         | $\alpha_F$ -R4M_CNN <sup>†</sup> | <b>89.12</b> | 141.22K |
| +[Romero et al. [318]]        |              |         |                     |              |         | +[Romero et al. [318]].          |              |         |

models with large vicinities do outperform models with small ones, e.g., 7x7 vs. 3x3, a trend of this behavior is not apparent. We conjecture that 300 epochs are insufficient for all models to converge equally well. Unfortunately, due to computational constraints, we were not able to perform this experiment for a larger number of epochs. We consider an in-depth study of this behavior an important direction for future work.

**Table 9.1:** Accuracy vs. neighborhood size.

| ROT-MNIST |            |              |                     |
|-----------|------------|--------------|---------------------|
| MODEL     | NBHD. SIZE | ACC. (%)     | TRAIN. TIME / EPOCH |
| R4_SA     | 3x3        | 96.56        | 04:53 - 1GPU        |
|           | 5x5        | <b>97.49</b> | 05:34 - 1GPU        |
|           | 7x7        | 97.33        | 09:03 - 1GPU        |
|           | 9x9        | 97.42        | 09:16 - 1GPU        |
|           | 11x11      | 97.17        | 12:09 - 1GPU        |
|           | 15x15      | 96.89        | 10:27 - 2GPU        |
|           | 19x19      | 96.86        | 14:27 - 2GPU        |
|           | 23x23      | 97.05        | 06:13 - 3GPU        |
|           | 28x28      | 96.81        | 12:12 - 4GPU        |

**Figure 9.5:** Test accuracy in early training.

Next, we compare GSA-Nets equivariant to translation and rotation at different angle discretizations (Tab. 9.2). Based on the results of the previous study, we select a 5x5 neighborhood, as it provides the best trade-off between accuracy and convergence time. Our results show that finer discretizations lead to better accuracy but saturates around R12. We conjecture that this is due to the discrete resolution of the images in the dataset, which leads finer angle discretizations to fall within the same pixel.

**CIFAR-10.** The CIFAR-10 dataset [195] consists of 60k real-world 32x32 RGB images uniformly drawn from 10 classes, divided into training, validation and test sets of 40k, 10k and 10k images. Since reflection is a symmetry that appears ubiquitously in natural images, we compare GSA-Nets equivariant to translation and reflection in this dataset (Tab. 9.2). Our results show that reflection equivariance indeed improves the classification performance of the model.

**PCam.** The PatchCamelyon dataset [404] consists of  $327k$   $96 \times 96$  RGB image patches of tumorous/non-tumorous breast tissues extracted from Camelyon16 [16]. Each patch is labeled as tumorous if the central region ( $32 \times 32$ ) contains at least one tumour pixel. As cells appear at arbitrary positions and poses, we compare **GSA-Nets** equivariant to translation, rotation and reflection (Tab. 9.2). Our results show that incorporating equivariance to reflection in addition to rotation, as well as providing finer group discretization, improve classification performance.

## 9.7 Discussion and Future Work

Though **GSA-Nets** perform competitively to **G-CNNs** for some tasks, **G-CNNs** still outperforms our approach in general. We conjecture that this is due to the harder nature of the optimization problem in **GSA-Nets** and the carefully crafted architecture design, initialization and optimization procedures developed for CNNs over the years. Though our theoretical results indicate that **GSA-Nets** can be more expressive than **G-CNNs** (Sec. 9.5.3), further research in terms of design, optimization, stability and generalization is required. These are in fact open questions for self-attention in general [233, 441, 471] and developments in this direction are of utmost importance.

The main drawback of our approach is the quadratic memory and time complexity typical of self-attention. This is an active area of research, e.g., Choromanski et al. [59], Kitaev et al. [189], Wang et al. [418], Zaheer et al. [458] and we believe that efficiency advances to vanilla self-attention can be seamlessly integrated in **GSA-Nets**. Our theoretical results indicate that **GSA-Nets** have the potential to become the standard solution for applications exhibiting symmetries, e.g., medical imagery. In addition, as self-attention is a set operation, **GSA-Nets** provide straightforward solutions to set-like data types, e.g., point-clouds, graphs, symmetric sets, which may benefit from additional geometrical information, e.g., Fuchs et al. [111], Maron et al. [255]. Finally, we hope our theoretical insights serve as a support point to further explore and understand the construction of equivariant maps for graphs and sets, which often come equipped with spatial coordinates: a type of positional encoding.

# 10

## Scale-Translation Equivariant Learning From Raw Time-Series

*Based on the paper:*

*Wavelet Networks: Scale-Translation Equivariant Learning From Raw Time-Series* [322]

### 10.1 Introduction

Leveraging the symmetries inherent to specific data domains for the construction of statistical models, such as neural networks, has proven highly advantageous, by restricting the model to the family of functions that accurately describes the data. A prime example of this principle are Convolutional Neural Networks (CNNs) [206]. CNNs embrace the translation symmetries in visual data by restricting their mappings to a *convolutional* structure. Convolutions possess a distinctive property called *translation equivariance*: if the input is translated, the output undergoes an equal translation. This property endows CNNs with better data efficiency and generalization than unconstrained models like multi-layered perceptrons.

Group equivariant convolutional neural networks (G-CNNs) [65] extend equivariance to more general symmetry groups through the use of *group convolutions*. Group convolutions are *group equivariant*: if the input is transformed by the symmetries described by the group, e.g., scaling, the output undergoes an equal transformation. Equivariance to larger symmetry groups endows G-CNNs with increased data efficiency and generalization on data exhibiting these symmetries. Existing group equivariance research primarily focuses on symmetries found in visual data, e.g., planar rotations, planar scaling,

etc [318, 424, 433], and more recently, on 3D symmetries, e.g., for spherical and molecular data [111, 336, 382]. Yet, an important category remains underexplored, which also exhibits symmetries: *time-series*. Notably, their translation symmetry is a cornerstone in signal processing and system analysis, e.g., Linear Time-Invariant (LTI) systems.

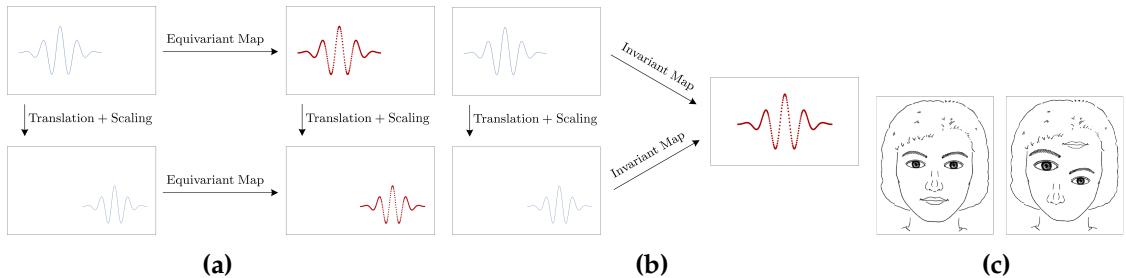
In this work, we bridge this gap by constructing neural networks that embrace the symmetries inherent to time-series. We begin by asking: “*What symmetries are inherently present in time-series?*” We identify two fundamental symmetries –*scale and translation*–, whose combination elucidate several phenomena observed in time-series, e.g., temporal translations, phase shifts, temporal scaling, resolution changes, pitch shifts, seasonal occurrences, etc. By leveraging group convolutions equivariant to the *scale-translation* group, we construct neural architectures such that when the input undergoes translation, scaling or a combination of the two, all intermediate layers will undergo an equal transformation in a hierarchical manner, akin to the methods proposed by Sosnovik et al. [362], Zhu et al. [478] for visual data. Interestingly, we observe that constructing convolutional layers equivariant to scale and translation results in layers that closely resemble the *wavelet transform*. However, we find that in order to preserve these symmetries consistently across the whole network, the output of each layer must be processed by a layer that also behaves like the wavelet transform. This approach substantially deviates from common approaches that rely on spectro-temporal representations, e.g., the wavelet transform, which compute spectro-temporal representations once and pass their response to a 2D CNN for further processing.

Inspired by the resemblance of scale-translation group equivariant convolutions with the wavelet transform, we term our scale-translation equivariant networks for time-series processing *Wavelet Networks*. Extensive empirical results show that Wavelet Networks consistently outperform conventional CNNs operating on raw waveforms, and match strongly engineered spectrogram-based approaches, e.g., on Mel-spectrograms, across several tasks and time-series types, e.g., audio, environmental sounds, electrical signals. To the best of our best knowledge, we are first to propose scale-translation equivariant neural networks for general time-series processing tasks.

## 10.2 Related Work

**Learning from raw time-series.** Several end-to-end learning approaches for time-series exist [75, 88, 89, 306, 368]. Given the considerable high-dimensionality of time-series, existing works focus on devising techniques with parameter- and compute-efficient large memory horizons [120, 321]. Due to small effective memory horizons and long training times, Recurrent Neural Networks (RNNs) [326] have gradually been overshadowed by CNN backbones [11].

While CNNs are equivariant to translations, they do not inherently incorporate a distinct notion of scale. Although methods involving layer-wise multi-scale representations have been proposed, e.g., Guizzo et al. [133], Lu et al. [240], von Platen et al. [408], Zhu



**Figure 10.1:** Equivariance, invariance and their impact on the hierarchical representations. In a group equivariant mapping, when the input is transformed by a group transformation, its output undergoes an equivalent transformation (Fig. 10.1a). In contrast, in group invariant maps, the output remains unchanged for all group transformations of the input (Fig. 10.1b). This distinction holds significant implications in the construction of hierarchical feature representations. For example, a face recognition system built upon invariant eye, nose and mouth detectors would be unable to set the portraits in Fig. 10.1c apart. However, by leveraging equivariant mappings, information about the input transformations can be used to distinguish these portraits effectively. In essence, in contrast to equivariant maps, invariant maps permit senseless pattern combinations resulting for overly restraining constraints in their design.

et al. [480], these layers are not scale equivariant. As a result, networks incorporating them struggle to maintain consistent scale information across layers.

**Group-invariant time-series learning.** Learning invariant representations from raw speech and sound has been extensively studied in past. Scattering operators [36, 249] construct group *invariant* feature representations that can be used to construct neural architectures invariant to scale and translation [6, 285, 331]. In contrast to the *invariant* feature representations developed by these works, Wavelet networks construct *equivariant* feature representations. Since group equivariance is a generalization of group invariance (Fig. 10.1b, Sec. 10.3.1), Wavelet Networks accommodate a broader functional family than previous works, while still upholding scale and translation preservation. Notably, equivariant methods shown superior performance compared to invariant methods across several tasks, even for intrinsically invariant tasks like classification [65, 254, 457]. This phenomenon stems from the hierarchical form in which neural networks extract features. Enforcing invariance early in the feature extraction process imposes an overly restrictive constraint in the resulting models (Fig. 10.1c).

**Group-equivariant time-series learning.** The concept of equivariant learning for time-series has mainly been explored in two contexts. Zhang et al. [463] propose to learn feature representations equivariant to vocal tract length changes –an inherent symmetry of speech. However, vocal tract length changes do not conform to the mathematical definition of a group, making their equivariance definition only approximate. Interestingly, vocal tract length changes can be characterized by specific (scale, translation) tuples. Consequently, considering scale-translation equivariance implicitly describes

vocal tract length changes as well as many other symmetries encountered in audio, speech and other time-series modalities. In addition, group equivariance has also been explored for the modeling and forecast of dynamical systems –an specific type of time-series [409, 412, 413]. Contrary to these methods, our work explores scale-translation equivariance for general time-series processing tasks, providing a framework that can enhance learning and generalization across diverse time-series applications.

## 10.3 Background

This work assumes a basic familiarity with the concepts of a group, a subgroup and a group action. For those not familiar with these terms, we introduce them in Appx. I.1.

### 10.3.1 Group equivariance, group invariance and symmetry preservation

**Group equivariance.** Group equivariance is the property of a mapping to respect the transformations in a group. We say that a map is equivariant to a group if a transformation of the input by elements of the group leads to an equivalent transformation of the output (Fig. 10.1a). Formally, for a group  $\mathcal{G}$  with elements  $g \in \mathcal{G}$  acting on a set  $\mathcal{X}$ , and a mapping  $\phi : \mathcal{X} \rightarrow \mathcal{X}$ , we say that  $\phi$  is equivariant to  $\mathcal{G}$  if:

$$\phi(gx) = g\phi(x), \quad \forall x \in \mathcal{X}, \forall g \in \mathcal{G}. \quad (10.1)$$

For example, the convolution of a signal  $f : \mathbb{R} \rightarrow \mathbb{R}$  and a kernel  $\psi : \mathbb{R} \rightarrow \mathbb{R}$  is *equivariant to the group of translations* –or *translation equivariant*– because when the input is translated, its convolutional descriptors are equivalently translated, i.e.,  $(\psi * \mathcal{L}_t f) = \mathcal{L}_t(\psi * f)$ , with  $\mathcal{L}_t$  a translation operator by  $t$ :  $\mathcal{L}_t f(x) = f(x-t)$ .

**Group invariance.** Group invariance is a special case of group equivariance in which the output of the map is equal for all transformations of the input (Fig. 10.1b). Formally, for a group  $\mathcal{G}$  with elements  $g \in \mathcal{G}$  acting on a set  $\mathcal{X}$ , and a mapping  $\phi : \mathcal{X} \rightarrow \mathcal{X}$ , we say that  $\phi$  is invariant to  $\mathcal{G}$  if:

$$\phi(gx) = \phi(x), \quad \forall x \in \mathcal{X}, \forall g \in \mathcal{G}. \quad (10.2)$$

*Relation to symmetry preservation.* A symmetry-preserving mapping preserves the symmetries of the input. That is, if the input has certain symmetries, e.g., translation, rotation, scale, these symmetries will also be present in the output. Since symmetries are mathematically described as groups, it follows that group equivariant mappings preserve the symmetries of the group to which the mapping is equivariant. In contrast, invariant mappings *do not* preserve symmetry, as they remove all symmetric information from the input.

### 10.3.2 Symmetry-preserving mappings: Group and lifting convolutions

When talking about (linear) symmetry-preserving mappings, we are obliged to talk about the group convolution. Previous work has shown that group convolutions are

the most general class of group equivariant linear maps [68]. Hence, it holds that any linear equivariant map is in fact a group convolution.

**Group convolution.** Let  $f : \mathcal{G} \rightarrow \mathbb{R}$  and  $\psi : \mathcal{G} \rightarrow \mathbb{R}$  be a scalar-valued signal and convolutional kernel defined on a group  $\mathcal{G}$ . Their group convolution ( $*_{\mathcal{G}}$ ) is given by:

$$(f *_{\mathcal{G}} \psi)(g) = \int_{\mathcal{G}} f(\gamma) \mathcal{L}_g \psi(\gamma) d\mu_{\mathcal{G}}(\gamma) = \int_{\mathcal{G}} f(\gamma) \psi(g^{-1}\gamma) d\mu_{\mathcal{G}}(\gamma). \quad (10.3)$$

where  $g, \gamma \in \mathcal{G}$ ,  $\mathcal{L}_g \psi(\gamma) = \psi(g^{-1}\gamma)$  is the action of the group  $\mathcal{G}$  on the kernel  $\psi$ , and  $\mu_{\mathcal{G}}(\gamma)$  is the (invariant) Haar measure of the group  $\mathcal{G}$  for  $\gamma$ . Notably, the group convolution generalizes the translation equivariance of convolutions to general groups. The group convolution is equivariant in the sense that for all  $\gamma, g \in \mathcal{G}$ ,

$$\mathcal{L}_g(f *_{\mathcal{G}} \psi)(\gamma) = (\mathcal{L}_g f *_{\mathcal{G}} \psi)(\gamma), \text{ with } \mathcal{L}_g f(\gamma) = f(g^{-1}\gamma). \quad (10.4)$$

**The lifting convolution.** In practice, the input signals  $f$  might not be readily defined on the group of interest  $\mathcal{G}$ , but on a sub-domain thereof  $\mathcal{X}$ , i.e.,  $f : \mathcal{X} \rightarrow \mathbb{R}$ . For example, time-series are defined on  $\mathbb{R}$  although we might want to consider larger groups such as the scale-translation group. Hence, we require a symmetry-preserving mapping from  $\mathcal{X}$  to  $\mathcal{G}$  that *lifts* the input signal to  $\mathcal{G}$  to use group convolutions. This operation is called a *lifting convolution*. Formally, with  $f : \mathcal{X} \rightarrow \mathbb{R}$  and  $\psi : \mathcal{X} \rightarrow \mathbb{R}$  a scalar-valued signal and convolutional kernel defined on  $\mathcal{X}$ , and  $\mathcal{X}$  a sub-group of  $\mathcal{G}$ , the lifting convolution ( $*_{\mathcal{G}\uparrow}$ ) is a mapping from functions on  $\mathcal{X}$  to functions on  $\mathcal{G}$  defined as:

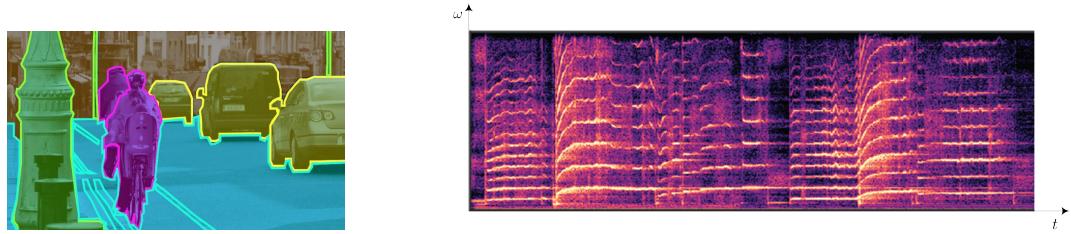
$$(f *_{\mathcal{G}\uparrow} \psi)(g) = \int_{\mathcal{X}} f(x) \mathcal{L}_g \psi(x) d\mu_{\mathcal{G}}(x) = \int_{\mathcal{X}} f(x) \psi(g^{-1}x) d\mu_{\mathcal{G}}(x) \quad (10.5)$$

The lifting convolution is also group equivariant. That is,  $\mathcal{L}_g(f *_{\mathcal{G}\uparrow} \psi) = (\mathcal{L}_g f *_{\mathcal{G}\uparrow} \psi)$ .

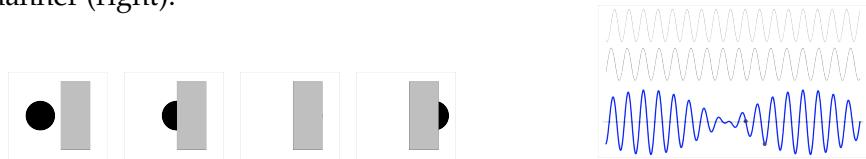
## 10.4 The problem of learning 2D convolutional kernels on the time-frequency plane

CNNs have been a major breakthrough in computer vision, yielding startling results in countless applications. Due to their success, several works have proposed to treat *spectro-temporal representations* –representations on the time-frequency plane– as 2D images and learn 2D CNNs on top. In this section, we delve into the differences between visual and spectro-temporal representations, and assess the suitability of training 2D CNNs directly on top of spectro-temporal representations. Our analysis suggest that treating spectro-temporal representations as images and learning 2D CNNs on top might not be adequate for effective time-series learning.

To enhance clarity, we define spectro-temporal representations in separate gray boxes throughout the section to avoid interrupting the reading flow. Those already familiar with these concepts may skip these boxes.



**Figure 10.2:** Locality of visual and auditory objects. Whereas visual objects are local (left), auditory objects are not. The latter often cover large parts of the frequency axis in a sparse manner (right).



**Figure 10.3:** Occlusion and superposition. Visual objects occlude each other when they appear simultaneously at a given position (left). Auditory objects, instead, superpose at all shared positions (right).

**Spectro-temporal representations.** Let  $f(t) \in L^2(\mathbb{R})$  be a square integrable function on  $\mathbb{R}$ . An spectro-temporal representation  $\Phi[f](t, \omega) : \mathbb{R}^2 \rightarrow \mathbb{C}$  of  $f$  is constructed by means of a *linear time-frequency transform*  $\Phi$  that correlates the signal  $f$  with a dictionary  $\mathcal{D}$  of localized *time-frequency atoms*  $\mathcal{D} = \{\phi_{t,\omega}\}_{t \in \mathbb{R}, \omega \in \mathbb{R}}$ ,  $\phi_{t,\omega} : \mathbb{R} \rightarrow \mathbb{C}$  of finite energy and unitary norm, i.e.,  $\phi_{t,\omega} \in L^2(\mathbb{R})$ ,  $\|\phi_{t,\omega}\|^2 = 1$ ,  $\forall t \in \mathbb{R}, \omega \in \mathbb{R}$ . The resulting spectro-temporal representation  $\Phi[f]$  is given by:

$$\Phi[f](t, \omega) = \langle f, \phi_{t,\omega} \rangle = \int_{\mathbb{R}} f(\tau) \phi_{t,\omega}^*(\tau) d\tau, \quad (10.6)$$

where  $\phi^*$  depicts the complex conjugate of  $\phi$ , and  $\langle \cdot, \cdot \rangle$  the dot product of its arguments. By selecting different parameterizations of the time-frequency components  $\phi_{t,\omega}$ , spectro-temporal representations with different properties can be obtained.

#### 10.4.1 Fundamental differences between visual representations and spectro-temporal representations

There exist two fundamental distinctions between visual data and spectro-temporal representations, which are universal to all spectro-temporal representations: (i) locality and (ii) transparency. Unlike visual data, auditory signals exhibit strong *non-local* characteristics. Auditory signals consist of auditory objects, e.g., spoken words, which contain components resonating at multiple non-local frequencies known as the *harmonics of the signal*. Consequently, the spectro-temporal representations of auditory objects often occupy a significant portion of the time-frequency plane –particularly along the frequency axis ( $\omega$ )– in a sparse manner (Fig. 10.2). Furthermore, when considering auditory sig-

nals comprising multiple auditory objects, these objects exhibit a phenomenon known as *superposition*. This property is notably different from visual data, where visual objects in the same location *occlude* one another, resulting in only the object closest to the camera being visible (Fig. 10.3). This inherent property of sound is referred to as *transparency*.

#### 10.4.2 The problem of learning 2D kernels on short-time Fourier spectro-temporal representations

The short-time Fourier transform constructs a representation in which a signal is decomposed in terms of its correlation with time-frequency atoms of constant time and frequency resolution. As a result, it is effective as long as the signal  $f$  does not exhibit *transient behavior* –components that evolve quickly over time– with some waveform structures being very localized in time and others very localized in frequency.

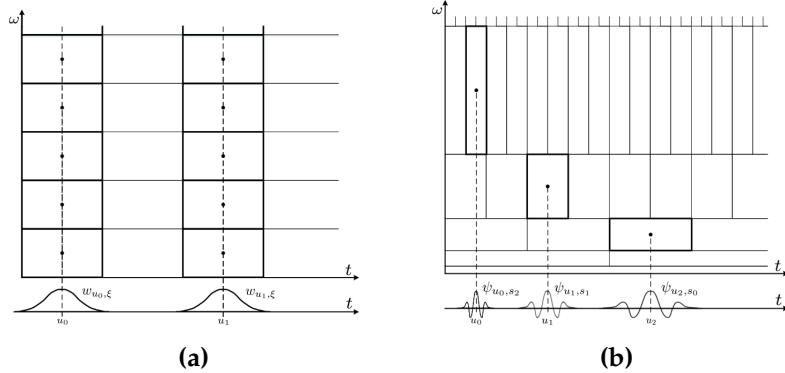
**The short-time Fourier transform.** The short-time Fourier transform  $\mathcal{S}$  –also called *windowed Fourier transform*– is a linear time-frequency transform that uses a dictionary of time-frequency atoms  $\phi_{t,\omega}(\tau) = w(\tau - t)e^{-i\omega\tau}$ ,  $t \in \mathbb{R}$ ,  $\omega \in \mathbb{R}$ , constructed with a symmetric window  $w(\tau)$  of local support shifted by  $t$  and modulated by the frequency  $\omega$ . The spectro-temporal representation  $\mathcal{S}[f]$  is given by:

$$\mathcal{S}[f](t, \omega) = \langle f, \phi_{t,\omega} \rangle = \int_{\mathbb{R}} f(\tau) \phi_{t,\omega}^*(\tau) d\tau = \int_{\mathbb{R}} f(\tau) w(\tau - t) e^{-i\omega\tau} d\tau. \quad (10.7)$$

Intuitively, the short-time Fourier transform divides the time-frequency plane in tiles of equal resolution, whose value is given by the correlation between  $f$  and the time-frequency atom  $\phi_{t,\omega}$  (Fig. 10.4a).

Nevertheless, decades of research in psychology and neuroscience have shown that humans largely rely in the transient behavior of auditory signals to distinguish auditory objects [54, 266, 400]. In addition, it has been shown that the human auditory system has high spectral resolution at low-frequencies and high temporal resolution at higher frequencies [27, 334, 367]. For example, a semitone at the bottom of the piano scale ( $\sim 30\text{Hz}$ ) is of about 1.5Hz, while at the top of the musical scale ( $\sim 5\text{kHz}$ ) it is of about 200Hz. These properties of the human auditory signal largely contrast both with (i) the inability of the short-time Fourier transform to detect transient signals, as well as with (ii) its constant spectro-temporal resolution.

To account for these differences, improved spectro-temporal representations on top of the short-time Fourier transform have been proposed such as log-Mel spectrograms [113, 367]. These developments revolve around transforming the frequency axis of the short-time Fourier transform in a logarithmic scale, thereby compressing the frequency axis and better aligning with the spectro-temporal resolution of the human auditory system. Consequently, this adjustment enables local structures, e.g., 2D convolutional kernels, to better capture non-local relationships [57, 389, 446]. However, despite improvements, these spectro-temporal representations remain *incomplete* due to their inability to modify



**Figure 10.4:** Tiling of the time-frequency plane for the short-time Fourier transform (Fig. 10.4a) and the wavelet transform (Fig. 10.4b). The short-time Fourier transform divides the time-frequency plane in tiles of equal resolution. This makes it adequate for signals without transient behaviour. The Wavelet transform, on the other hand, divides the time-frequency plane on tiles of changing spectro-temporal resolution. This allows it to represent detect highly localized events both on time and frequency.

the constant temporal resolution of the short-time Fourier transform.

#### 10.4.3 The problem of learning 2D kernels on Wavelet spectro-temporal representations

In contrast to the short-time Fourier transform, the Wavelet transform constructs a spectro-temporal representation in terms of correlations with time-frequency atoms, *whose time and frequency resolution change*. As a result, the resulting decomposition of the time-frequency plane allows the wavelet transform to *correctly describe signals with transient behaviour with localized components both on time and frequency* (Fig. 10.4b).

**The wavelet transform.** The wavelet transform  $\mathcal{W}$  is a linear time-frequency transform that uses a dictionary of time-frequency atoms  $\phi_{t,\omega}(\tau) = \frac{1}{\sqrt{\omega}}\psi\left(\frac{\tau-t}{\omega}\right)$ ,  $t \in \mathbb{R}$ ,  $\omega \in \mathbb{R}_{\geq 0}$ . The function  $\psi_{t,\omega}$  is called a *Wavelet* and satisfies the properties of having zero mean, i.e.,  $\int \psi_{t,\omega}(\tau)d\tau=0$ , and being unitary, i.e.,  $\|\psi_{t,\omega}\|^2=1$ , for any  $t \in \mathbb{R}$ ,  $\omega \in \mathbb{R}_{\geq 0}$ . The resulting spectro-temporal representation  $\mathcal{W}[f]$  is given by:

$$\mathcal{W}[f](t, \omega) = \langle f, \phi_{t,\omega} \rangle = \int_{\mathbb{R}} f(\tau) \phi_{t,\omega}^*(\tau) d\tau = \int_{\mathbb{R}} f(\tau) \frac{1}{\sqrt{\omega}} \psi\left(\frac{\tau-t}{\omega}\right) d\tau. \quad (10.8)$$

Intuitively, the Wavelet transform divides the time-frequency plane in tiles of different resolutions, with high frequency resolution and low spatial resolution at low frequencies, and low frequency resolution and high spatial resolution for high frequencies (Fig. 10.4b).<sup>a</sup>

<sup>a</sup>Importantly, it is not possible to have high frequency and spatial resolution at the same time due to the *uncertainty principle* [114]. It states that the joint time-frequency resolution of spectro-temporal

representations is limited by a minimum surface  $\sigma_{\phi,t}\sigma_{\phi,\omega} \geq \frac{1}{2}$ , with  $\sigma_{\phi,t}$ ,  $\sigma_{\phi,\omega}$  the spread of the time-frequency atom  $\phi$  on time and frequency.

Interestingly, the modus operandi of the wavelet transform *perfectly aligns* with the spectro-temporal resolution used by the human auditory system for the processing of auditory signals. Nevertheless, despite this resemblance, training local 2D structures, e.g., convolutional kernels, directly on the wavelet transform's output still falls short in addressing the non-local, transparent characteristics inherent in auditory signals. Consequently, researchers have devised many strategies to overcome these challenges, e.g., by defining separable kernels that span large memory horizons along frequency and time independently [293] or by prioritizing learning along the harmonics of a given frequency [470].

As shown in the next section, a better alternative arises from considering the symmetries appearing in time-series data. Starting from this perspective, we are led to scale-translation equivariant mappings and find striking relationships between these family of mappings and the wavelet transform. Nevertheless, our analysis indicates that *all layers within a neural network should be symmetry preserving* –a condition not met by the methods depicted in this section. By doing so, we devise neural architectures, whose convolutional layers process the output of previous layers in a manner akin to the wavelet transform. As a result, each convolutional layer performs spectro-temporal decompositions of the input in terms of localized time-frequency atoms able to process global and localized patterns both on time and frequency.

## 10.5 Wavelet networks: Scale-translation equivariant learning from raw waveforms

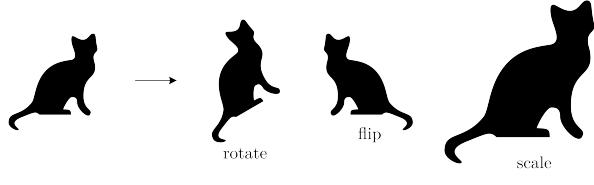
We are interested in mappings that preserve the scale and translation symmetries of time-series. In this section, we start by tailoring lifting and group convolutions to the scale-translation group. Next, we outline the general form of Wavelet Networks and make concrete practical considerations for their implementation. At the end of this section, we formalize the relationship between Wavelet Networks and the wavelet transform, and provide a thorough analysis on the equivariance properties of common spectro-temporal transforms.

### 10.5.1 Scale-translation preserving mappings: group convolutions on the scale-translation group

We are interested in mappings that preserve scale and translation. By imposing equivariance to the *scale-translation* group, we guarantee that if input patterns are scaled, translated, or both, their feature representations will transform accordingly.

**The scale-translation group.** From a mathematical perspective scale and translational symmetries are described by the affine *scale-translation group*  $G = \mathbb{R} \rtimes \mathbb{R}_{\geq 0}$ , which emerges from the semi-direct product of the translation group  $T = (\mathbb{R}, +)$  and the scale group

**Figure 10.5:** The action of unimodular and non-unimodular groups. Most unimodular groups, e.g., rotation, mirroring, keep the volume of the objects they act upon intact. In contrast, non-unimodular groups, e.g., scaling, change it through their action.



$\mathcal{S} = (\mathbb{R}_{\geq 0}, \times)$  acting on  $\mathbb{R}$ . As a result, we have that the resulting group product is given by  $g \cdot \gamma = (t, s) \cdot (\tau, \zeta) = (t + s\tau, s \cdot \zeta)$ , with  $t, \tau \in \mathbb{R}$  and  $s, \zeta \in \mathbb{R}_{\geq 0}$ . In addition, by solving  $g^{-1} \cdot g = e$ , we obtain that the inverse of a group element  $g = (t, s)$  is given by  $g^{-1} = s^{-1}(-t, 1)$ .

**Semi-direct product and affine groups.** When treating data defined on  $\mathbb{R}^d$ , one is mainly interested in the analysis of groups of the form  $\mathcal{G} = \mathbb{R}^d \rtimes \mathcal{H}$  resulting from the *semi-direct product* ( $\rtimes$ ) between the translation group  $(\mathbb{R}^d, +)$  and an arbitrary (Lie) group  $\mathcal{H}$  acting on  $\mathbb{R}^d$ , e.g., rotation, scale, etc. This kind of groups are called *affine groups* and their group product is defined as:

$$g_1 \cdot g_2 = (x_1, h_1) \cdot (x_2, h_2) = (x_1 + \mathcal{A}_{h_1}(x_2), h_1 \cdot h_2), \quad (10.9)$$

with  $g_1 = (x_1, h_1)$ ,  $g_2 = (x_2, h_2) \in \mathcal{G}$ ,  $x_1, x_2 \in \mathbb{R}^d$  and  $h_1, h_2 \in \mathcal{H}$ .  $\mathcal{A}$  denotes the action of  $\mathcal{H}$  on  $\mathbb{R}^d$ .

**Unimodular and non-unimodular groups.** Unimodular groups, such as rotation, translation and mirroring, are groups whose action keeps the volume of the objects on which they act intact (Fig. 10.5). Recall that a group convolution performs an integral over the whole group (Eq. 10.3). Hence, for its result to be invariant over different group actions, it is required for the Haar measure to be equal for all elements of the group –therefore the name *invariant* Haar measure. Since the action of (most) unimodular groups does not alter the size of the objects on which they act, their action on infinitesimal objects keeps their size unchanged. As a consequence, for (most) unimodular groups, the Haar measure is equal to the Lebesgue measure, i.e.,  $d\mu_{\mathcal{G}}(\gamma) = d\gamma$ ,  $\forall \gamma \in \mathcal{G}$ , and therefore, it is often omitted in literature, e.g., in Cohen and Welling [65].

In contrast, non-unimodular groups, such as the scale group and the scale-translation group, do modify the size of objects on which they act (Fig. 10.5 right). Consequently, their action on infinitesimal objects changes their size. As a result, the Haar measure must be treated carefully in order to obtain equivariance to non-unimodular groups [17]. The Haar measure guarantees that  $d\mu_{\mathcal{G}}(\gamma) = d\mu_{\mathcal{G}}(g\gamma)$ ,  $\forall g, \gamma \in \mathcal{G}$ . For the scale-translation group, it is obtained as:

$$d\mu_{\mathcal{G}}(\gamma) = d\mu_{\mathcal{G}}(g\gamma) = d\mu_{\mathcal{G}}(t + s\tau, s\zeta) = d\mu_{\mathcal{G}}(t + s\tau) d\mu_{\mathcal{G}}(s\zeta) = \frac{1}{|s|} d\tau \frac{1}{|s|} d\zeta, \quad (10.10)$$

where  $g = (t, s)$ ,  $\gamma = (\tau, \zeta) \in \mathcal{G}$ ,  $t, \tau \in \mathbb{R}$ ,  $s, \zeta \in \mathbb{R}_{>0}$ ;  $d\tau$ ,  $d\zeta$  are the Lebesgue measure of

the respective spaces; and  $|s|$  depicts the determinant of the matrix representation of the group element.<sup>1</sup> Intuitively, the Haar measure counteracts the growth of infinitesimal elements resulting from the action of  $s$  on  $\mathbb{R} \times \mathbb{R}_{>0}$ .

**Scale-translation group convolutions.** The general formulation of the group convolution is given in Eq. 10.3. Interestingly, the scale-translation group has additional properties with which this formulation can be simplified. In particular, by taking advantage of the fact that the scale-translation group is an *affine* group  $\mathcal{G} = \mathbb{R} \rtimes \mathcal{S}$ , with  $\mathcal{S} = (\mathbb{R}_{>0}, \times)$ , as well as of the definition of the Haar measure for the scale-translation group in Eq. 10.10 we can reformulate the group convolution for the scale-translation group as:

$$\begin{aligned}(f *_{\mathcal{G}} \psi)(g) &= \int_{\mathcal{G}} f(\gamma) \psi(g^{-1}\gamma) d\mu_{\mathcal{G}}(\gamma) \\(f *_{\mathcal{G}} \psi)(t, s) &= \int_{\mathcal{S}} \int_{\mathbb{R}} f(\tau, \zeta) \psi((t, s)^{-1}(\tau, \zeta)) \frac{1}{|s|} d\tau \frac{1}{|s|} d\zeta \\&= \int_{\mathcal{S}} \int_{\mathbb{R}} f(\tau, \zeta) \frac{1}{s^2} \psi(s^{-1}(\tau - t, \zeta)) d\tau d\zeta \\&= \int_{\mathcal{S}} \int_{\mathbb{R}} f(\tau, \zeta) \frac{1}{s^2} \mathcal{L}_s \psi(\tau - t, \zeta) d\tau d\zeta = \int_{\mathcal{S}} \left( f *_{\mathbb{R}} \frac{1}{s^2} \mathcal{L}_s \psi \right)(t, \zeta) d\zeta \quad (10.11)\end{aligned}$$

where  $g = (t, h)$ ,  $\gamma = (\tau, \zeta) \in \mathcal{G}$ ,  $t, \tau \in \mathbb{R}$ , and  $s, \zeta \in \mathbb{R}_{>0}$ ; and  $\mathcal{L}_s \psi(\tau, \zeta) = \psi(s^{-1}(\tau, \zeta))$  is the (left) action of the scale group  $\mathcal{S}$  on a convolutional kernel  $\psi : \mathbb{R} \times \mathbb{R}_{>0} \rightarrow \mathbb{R}$  defined on the scale-translation group. In other words, for the scale-translation group, *the group convolution can be seen as a set of 1D convolutions with a bank of scaled convolutional kernels  $\{\frac{1}{s^2} \mathcal{L}_s \psi\}_{s \in \mathcal{S}}$ , followed by an integral over scales  $\zeta \in \mathbb{R}$*  (Fig. 10.6, bottom).

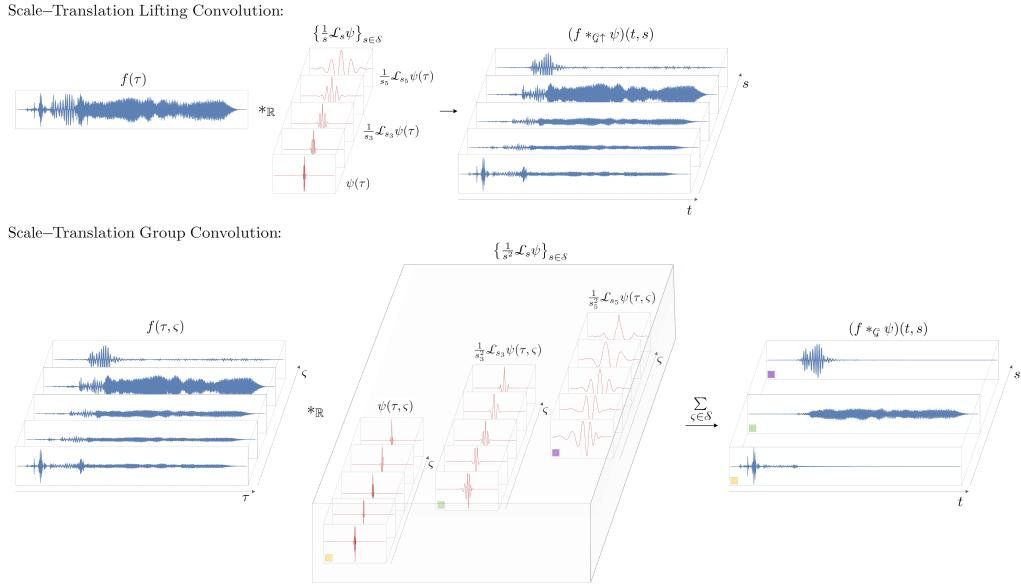
**Scale-translation lifting convolution.** Like the group convolution, the lifting convolution can also be simplified by considering the properties of the scale-translation group. In particular, we can rewrite it as:

$$\begin{aligned}(f *_{\mathcal{G}^\dagger} \psi)(g) &= \int_{\mathcal{X}} f(x) \psi(g^{-1}x) d\mu_{\mathcal{G}}(x) = \int_{\mathbb{R}} f(\tau) \psi(g^{-1}\tau) d\mu_{\mathcal{G}}(\tau) \\(f *_{\mathcal{G}^\dagger} \psi)(t, s) &= \int_{\mathbb{R}} f(\tau) \psi((t, s)^{-1}\tau) \frac{1}{|s|} d\tau = \int_{\mathbb{R}} f(\tau) \frac{1}{s} \psi(s^{-1}(\tau - t)) d\tau \\&= \left( f *_{\mathbb{R}} \frac{1}{s} \mathcal{L}_s \psi \right)(t) \quad (10.12)\end{aligned}$$

where  $g = (t, h)$ ,  $\gamma = (\tau, \zeta) \in \mathcal{G}$ ,  $t, \tau \in \mathbb{R}$ , and  $s, \zeta \in \mathbb{R}_{>0}$ ; and  $\mathcal{L}_s \psi(\tau, \zeta) = \psi(s^{-1}(\tau, \zeta))$  is the (left) action of the scale group  $\mathcal{S}$  on a 1D convolutional kernel  $\psi : \mathbb{R} \rightarrow \mathbb{R}$ . In other words, for the scale-translation group, *the lifting convolution can be seen as a set of 1D convolutions with a bank of scaled convolutional kernels  $\{\frac{1}{s} \mathcal{L}_s \psi\}_{s \in \mathcal{S}}$*  (Fig. 10.6, top). Note that the Haar measure imposes a normalization factor of  $\frac{1}{s^2}$  for group convolutions and of  $\frac{1}{s}$  for the lifting convolution. This is because space on which the group convolution is

---

<sup>1</sup>A member  $s$  of the scale group  $\mathbb{R}_{>0}$  acting on a  $N$ -dimensional space is represented a matrix  $\text{diag}(s, \dots, s)$ . Since, its determinant  $s^N$  depends on the value of the group element  $s$ , the factor  $\frac{1}{|s|} = \frac{1}{s^N}$  in Eq. 10.10 cannot be omitted.



**Figure 10.6:** Scale-translation lifting and group convolution. The lifting convolution can be seen as a set of 1D convolutions with a bank of scaled convolutional kernels  $\frac{1}{s}\mathcal{L}_s\psi$ , and the group convolution can be seen as a set of 1D convolutions with a bank of scaled convolutional kernels  $\frac{1}{s^2}\mathcal{L}_s\psi$ , followed by an integral over scales  $\xi \in \mathbb{R}$ . Their main difference is that, for group convolutions, the input  $f$  and the convolutional kernel  $\psi$  are functions on the scale-translation group whereas for lifting convolutions these are functions on  $\mathbb{R}$ . Lifting and group convolutions can be seen as spectro-temporal decompositions with large values of  $s$  relating to coarse features and small values to finer features.

performed ( $\mathbb{R} \times \mathbb{R}_{>0}$ ) has an additional dimension relative to the space on which the lifting convolution is performed ( $\mathbb{R}$ ).

### 10.5.2 Wavelet Networks: architecture and practical implementation

The general architecture of our proposed Wavelet networks is shown in Fig. 10.7. Wavelet networks consist of several stacked layers that respect scale and translation. They consist of a lifting group convolution layer that lifts input time-series to the scale-translation group, followed by arbitrarily many group convolutional layers. At the end of the network, a global pooling layer is used to produce scale-translation invariant representations. Due to their construction, Wavelet networks make sure that common neural operations, e.g., point-wise nonlinearities, do not disrupt scale and translation equivariance. This in turn, makes them broadly applicable and easily extendable to other existing neural architectures, e.g., ResNets [143], U-Nets [323].

#### Group convolutional kernels on continuous bases

Although our previous derivations build upon continuous functions, in practice, computations are performed on discretized versions of these functions. Continuous bases

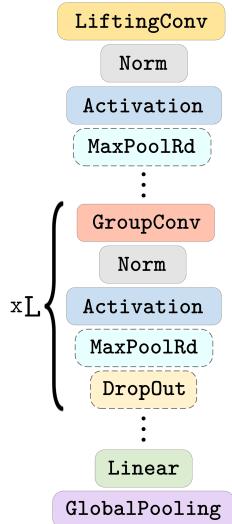
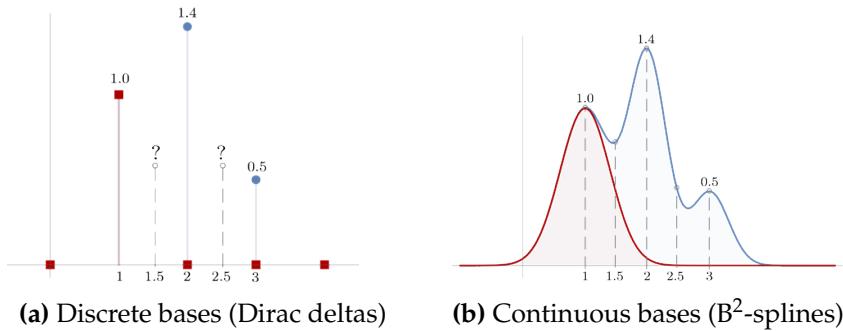
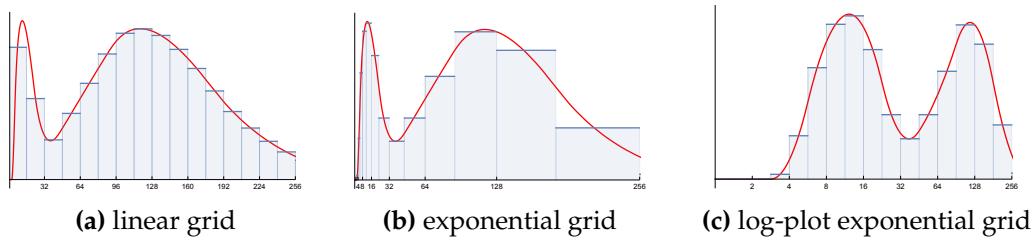


Figure 10.7: Architecture of wavelet networks.

Figure 10.8: Convolutional kernels on discrete and continuous bases. The canonical basis used for the construction of the convolutional kernel is shown in red: a delta Dirac for the discrete case, and a  $B^2$ -spline for the continuous case.

have proven advantageous for the construction of group convolutions as the action of relevant groups often impose transformations not well-defined for discrete bases [18, 422, 424]. For instance, in the context of scale-translations, scaling a kernel  $[w_1, w_2, w_3]$  by a factor of two results in a filter  $[w_1, w_{1.5}, w_2, w_{2.5}, w_3]$  wherein the introduced values  $[w_{1.5}, w_{2.5}]$  do not exist in the original basis (Fig. 10.8a).

The most adopted solution to address this problem is interpolation, i.e., deriving the value of  $[w_{1.5}, w_{2.5}]$  based on the neighbouring known pixels. However, interpolation introduces spurious artifacts which are particularly severe for small kernels. Instead, we adopt an alternative approach: we define convolutional kernels directly on a continuous basis (Fig. 10.8b). Drawing from the resemblance of gammatone filters –strongly motivated by the physiology of the human auditory system for the processing and recognition of auditory signals [146, 173, 226]– to  $B^2$ -splines, we parameterize our filters within



**Figure 10.9:** Riemann integration of functions on  $\mathbb{R}_{>0}$  using linear and exponential grids.

a  $B^2$ -spline basis as in Bekkers [17]. As a result, our convolutional filters are parameterized as a linear combination of shifted  $B^2$ -splines  $\psi(\tau) := \sum_{i=1}^N w_i B^2(\tau - \tau_i)$ , rather than the commonly used shifted Dirac delta's basis  $\psi(\tau) := \sum_{i=1}^N w_i \delta(\tau - \tau_i)$ .

### Constructing a discrete scale grid

From the response of the lifting layers onward, the feature representations of wavelet networks possess an additional axis  $s \in \mathbb{R}_{>0}$ . Just like the spatial axis, this axis must be discretized in order to perform computational operations. That is, we must approximate the scale axis  $\mathbb{R}_{>0}$  by a finite set of discrete scales  $\{s\}_{s=s_{\min}}^{s_{\max}}$ . Inspired by previous work, we approximate the scale axis with a *dyadic set*  $\{2^j\}_{j=j_{\min}}^{j_{\max}}$  [227, 248, 433]. Dyadic sets resemble the spectro-temporal resolution of the human auditory system, and are widely used for discrete versions of the wavelet transform.

**Integrating on exponential grids.** A subtlety arises with respect to integrating over the scale axis when implementing the continuous theory in a discrete setting that is suitable for numerical computations. The group convolutions include scale correction factors as part of the Haar measure, which makes the integration invariant to actions along the scale axis. That is, the integral of a signal  $f(s)$  over scale is the same as that of the same signal  $f(z^{-1}s)$ , whose scale is changed by a factor  $z \in \mathbb{R}_{>0}$ :

$$\int_{\mathbb{R}_{>0}} f(z^{-1}s) \frac{1}{s} ds \stackrel{s \rightarrow z s}{=} \int_{\mathbb{R}_{>0}} f(z^{-1}s) \frac{1}{zs} dz s = \int_{\mathbb{R}_{>0}} f(s) \frac{1}{s} ds. \quad (10.13)$$

We can translate the scale integration to the discrete setting via Riemann integrals, where we sample the function on a grid and take the weighted sum of these values with weights given by the bin-width:

$$\int_{\mathbb{R}_{>0}} f(s) \frac{1}{s} ds \approx \sum_i f(s_i) \frac{1}{s_i} \Delta_i. \quad (10.14)$$

When the scale grid is linear, the bin-widths  $\Delta_i$  are constant, as depicted in Fig. 10.9a. When the scale grid is exponential, e.g.,  $s_i = b^{i-1}$  with  $b$  some base factor, the bin widths are proportional to the scale values at the grid points, i.e.,  $\Delta_i \propto s_i$  (Fig. 10.9b). In this setting, the factor  $\frac{1}{s_i}$  cancels out (up to some constant) with the bin width  $\Delta_i$ , and integration is simply done by summing the values sampled on the scale grid. Consequently, when working with an exponential grid along the scale axis, the factor in the group

convolutions (Eq. 10.11) becomes  $\frac{1}{s}$  instead of  $\frac{1}{s^2}$ . It is worth mentioning that using an exponential grid is the natural thing to do when dealing with the scale group. The scale group is a multiplicative group with a natural distance between group elements  $z, s \in \mathbb{R}_{>0}$  defined by  $\|\log z^{-1}s\|$ . Consequently, on an exponential grid, the grid points are spaced uniformly with respect to this distance, as illustrated in Fig. 10.9c.

**Defining the discrete scale grid.** In practice, Wavelet networks must define the number of scales  $N_s$  to be considered in the dyadic set as well as its limits  $s_{\min}, s_{\max}$ . Fortunately, it turns out that these values are related to the spatial dimension of the input  $f$  itself, and thus, we can use it to determine these values.

Let us consider a signal  $f$  and a convolutional kernel  $\psi$  sampled on discrete grids  $[1, N_f]$ ,  $[1, N_\psi] \subset \mathbb{Z}$  of sizes  $N_f$ , and  $N_\psi$ , respectively. When we re-scale the convolutional kernel  $\psi$ , we are restricted (i) at the bottom of the scale axis by the Nyquist criterion, and (ii) at the top of the scale by the scale for which the filter becomes constant in an interval of  $N_f$  samples. The Nyquist criterion is required to avoid aliasing and intuitively restricts us to a compression factor on  $\psi$  such that it becomes as big as 2 grid samples. On the other hand, by having  $\psi$  re-scaled to an extreme to which it is constant in the support of the input signal  $f$ , the kernel will only be able to perform average operations.

*Considerations regarding computational complexity.* Note that the computational cost of Wavelet networks increases linearly with the number of scales considered. Hence, it is desirable to reduce the number of scales used as much as possible. To this end, we reason that using scales for which the sampled support of  $\psi$  is smaller than  $N_\psi$  is unnecessary as the functions that can be described at those scales can also be described –and learned– at the unscaled resolution of the kernel  $s=1$ . Therefore, we define the minimum scale as  $s_{\min}=1$ . Furthermore, we reason that using scales for which the support of the filter overpasses that of the input, i.e.,  $N_f \leq N_\psi$ , is also suboptimal, as the values outside of the region  $[1, N_f]$  are unknown. Therefore, we consider the set of sensible scales to be given by the interval  $[1, \frac{N_f}{N_\psi}]$ . In terms of a dyadic set  $\{2^j\}_{j=j_{\min}}^{j_{\max}}$ , this corresponds to the  $j$ -values given by the interval  $[0, 1, 2, \dots, j_{\max}]$  s.t.  $N_\psi 2^{j_{\max}} \leq N_f$ .

*Effect of downsampling on the scale grids used.* Neural architectures utilize pooling operations, e.g., max-pooling, to reduce the spatial dimension of the input as a function of depth. Following the rationale outlined in the previous paragraph, we take advantage of these reductions to reduce the number of scales that representations at a given depth should use. Specifically, we use the factor of downsampling as a proxy for the number of scales that can be disregarded. For example, if we use a pooling of 8 at a given layer, subsequent layers should reduce the number of scales considered by the same factor, i.e.,  $2^3$ . For a set of dyadic scales before a pooling layer given by  $\{2^j\}_{j=j_{\min}}^{j_{\max}}$  and a pooling layer of factor  $2^p$ , the set of dyadic scales considered after pooling will be given by  $\{2^j\}_{j=j_{\min}}^{j_{\max}-p}$ .

### Imposing wavelet structure to the learned convolutional kernels

In classical spectro-temporal analysis, wavelets are designed to have unit norm  $\|\psi\|^2=1$  and zero mean  $\int \psi(\tau) d\tau=0$ . These constraints are useful for both theoretical and practical reasons including energy preservation, numerical stability and the ability to act as band-pass filters [248]. Since Wavelet networks construct time-frequency representations of the input, we experiment with an additional regularization loss that encourages the learned convolutional kernels to behave like wavelets. First, we note that lifting and group convolutions inherently incorporate a normalization term  $-\frac{1}{s}, \frac{1}{s^2}$  in their definitions. Therefore, the normalization criterion is inherently satisfied. To encourage the learned kernels to have zero mean, we formulate a regularization term that promote this behaviour. Denoting  $\psi_d$  as the convolutional kernel at the  $d$ -th layer of a neural network with  $D$  convolutional layers, the regularization term  $\mathcal{L}_{\text{wavelet}}$  is defined as:

$$\mathcal{L}_{\text{wavelet}} = \sum_{d=1}^D \|\text{mean}(\psi_d)\|^2. \quad (10.15)$$

Interestingly, we observe that enforcing wavelet structure in the learned convolutional kernels consistently yields improved performance across all tasks considered (Sec. 10.6). This result underscores the potential value of integrating insights from classical signal processing, e.g., spectro-temporal analysis [78, 248, 338], in the design of deep models.

### 10.5.3 Wavelet networks perform nested non-linear time-frequency transforms

Interestingly, we can use spectro-temporal analysis to understand the modus operandi of wavelet networks. Our analysis reveals that wavelet networks perform nested time-frequency transforms interleaved with point-wise nonlinearities. In this process, each time-frequency transform emerges as a linear combination of parallel wavelet-like transformations of the input computed with learnable convolutional kernels  $\psi$ .

**The relation between scale-translation equivariant mappings and the wavelet transform.** The wavelet transform shows many similarities to the scale-translation group and lifting convolutions [125]. In fact, by analyzing the definition of the wavelet transform (Eq. 10.8), we obtain that the Wavelet transform is equivalent to a lifting group convolution (Eq. 10.12 with  $\omega=s$ ) up to a normalization factor  $\frac{1}{\sqrt{\omega}}$ :

$$\begin{aligned} \mathcal{W}[f](t, \omega) &= \int_{\mathbb{R}} f(\tau) \frac{1}{\sqrt{\omega}} \psi\left(\frac{\tau-t}{\omega}\right) d\tau \\ &= \int_{\mathbb{R}} f(\tau) \frac{1}{\sqrt{\omega}} \psi(\omega^{-1}(\tau-t)) d\tau = \int_{\mathbb{R}} f(\tau) \frac{1}{\sqrt{\omega}} \mathcal{L}_\omega \psi(\tau-t) d\tau \\ &= \left( f *_{\mathbb{R}} \frac{1}{\sqrt{\omega}} \mathcal{L}_\omega \psi \right)(t) = \frac{1}{\sqrt{\omega}} (f *_{\mathcal{G}} \psi)(t, \omega). \end{aligned} \quad (10.16)$$

Furthermore, if we let the input  $f$  be a function defined on the scale-translation group, and let  $\omega$  act on this group according to the group structure of the scale-translation

group, we have that the scale-translation group convolution is equivalent to a Wavelet transform whose input has been obtained by a previously applied Wavelet transform, , up to a normalization factor  $\frac{1}{\omega\sqrt{\omega}}$ :

$$\begin{aligned}\mathcal{W}[f](t, \omega) &= \int_{\mathbb{R}_{>0}} \int_{\mathbb{R}} f(\tau, \xi) \frac{1}{\sqrt{\omega}} \psi(\omega^{-1}(\tau - t), \xi) d\tau d\xi \\ &= \int_{\mathbb{R}_{>0}} \int_{\mathbb{R}} f(\tau, \xi) \frac{1}{\sqrt{\omega}} \mathcal{L}_\omega \psi(\tau - t, \xi) d\tau d\xi \\ &= \int_{\mathbb{R}_{>0}} \left( f *_{\mathbb{R}} \frac{1}{\sqrt{\omega}} \mathcal{L}_\omega \psi \right)(t, \xi) d\xi = \frac{1}{\omega\sqrt{\omega}} (f *_{\mathbb{G}} \psi)(t, \omega)\end{aligned}\quad (10.17)$$

In other words, lifting and group convolutions on the scale-translation group can be interpreted as linear time-frequency transforms that adopt time-frequency plane tiling akin wavelet transform (Fig. 10.4b), for which the group convolution accepts wavelet-like spectro-temporal representations as input.

**Equivariance properties of common time-frequency transforms.** For completeness, we also analyze the equivariance properties of common time-frequency transforms and their normalized representations, e.g., spectrogram. Careful interpretations and proofs are provided in Appx. I.2.

Let  $\mathcal{L}_{t_0}f = f(t - t_0)$  and  $\mathcal{L}_{s_0}f(t) = f(s_0^{-1}t)$ ,  $t_0 \in \mathbb{R}$ ,  $s_0 \in \mathbb{R}_{>0}$ , be translation and scaling operators. The Fourier, short-time Fourier and Wavelet transform of  $\mathcal{L}_{t_0}f$  and  $\mathcal{L}_{s_0}f$ ,  $f \in L^2(\mathbb{R})$ , are given by:

- **Fourier Transform:**

$$\mathcal{F}[\mathcal{L}_{t_0}f](\omega) = e^{-i\omega t_0} \mathcal{F}[f](\omega) \rightarrow |\mathcal{F}[\mathcal{L}_{t_0}f](\omega)|^2 = |\mathcal{F}[f](\omega)|^2 \quad (10.18)$$

$$\mathcal{F}[\mathcal{L}_{s_0}f](\omega) = s_0 \mathcal{L}_{s_0^{-1}} \mathcal{F}[f](\omega) \rightarrow |\mathcal{F}[\mathcal{L}_{s_0}f](\omega)|^2 = |s_0|^2 |\mathcal{L}_{s_0^{-1}} \mathcal{F}[f](\omega)|^2 \quad (10.19)$$

- **Short-Time Fourier Transform:**

$$\mathcal{S}[\mathcal{L}_{t_0}f](t, \omega) = e^{-i\omega t_0} \mathcal{L}_{t_0} \mathcal{S}[f](t, \omega) \rightarrow |\mathcal{S}[\mathcal{L}_{t_0}f](t, \omega)|^2 = |\mathcal{L}_{t_0} \mathcal{S}[f](t, \omega)|^2 \quad (10.20)$$

$$\mathcal{S}[\mathcal{L}_{s_0}f](t, \omega) \approx s_0 \mathcal{S}[f](s_0^{-1}t, s_0\omega) \rightarrow |\mathcal{S}[\mathcal{L}_{s_0}f](t, \omega)|^2 \approx |s_0|^2 |\mathcal{S}[f](s_0^{-1}t, s_0\omega)|^2 \quad (10.21)$$

- **Wavelet Transform:**

$$\mathcal{W}[\mathcal{L}_{t_0}f](t, \omega) = \mathcal{L}_{t_0} \mathcal{W}[f](t, \omega) \rightarrow |\mathcal{W}[\mathcal{L}_{t_0}f](t, \omega)|^2 = |\mathcal{L}_{t_0} \mathcal{W}[f](t, \omega)|^2 \quad (10.22)$$

$$\mathcal{W}[\mathcal{L}_{s_0}f](t, \omega) = \sqrt{s_0} \mathcal{L}_{s_0} \mathcal{W}[f](t, \omega) \rightarrow |\mathcal{W}[\mathcal{L}_{s_0}f](t, \omega)|^2 = |\mathcal{L}_{s_0} \mathcal{W}[f](t, \omega)|^2 \quad (10.23)$$

<sup>†</sup> Eq. 10.21 only holds approximately for large windows (see Appx. I.2.2). In other words, the Wavelet transform and the scalogram  $|\mathcal{W}[\cdot]|^2$  are the only time-frequency representations that exhibit translation and scaling equivariance in a practical way.

**Wavelet networks apply parallel time-frequency transforms with learned bases at every layer.** So far, our analysis has been defined for scalar-valued input and convolutional kernels. However, in practice, convolutional layers perform operations between inputs  $f : \mathbb{R} \rightarrow \mathbb{R}^{N_{in}}$  and convolutional kernels  $\psi : \mathbb{R} \rightarrow \mathbb{R}^{N_{out} \times N_{in}}$  to produce outputs  $(f * \psi) : \mathbb{R} \rightarrow \mathbb{R}^{N_{out}}$  as the linear combination along the  $N_{in}$  dimension of convolutions with several learned convolutional kernels computed in parallel:

$$(f * \psi)_o = \sum_{i=1}^{N_{in}} (f_i * \psi_i), \quad o \in [1, 2, \dots, N_{out}]. \quad (10.24)$$

In practice, both lifting and group convolutional layers adhere to the same structure. In a dilation-translation convolutional layer with  $N_{out}$  output channels,  $N_{out}$  independent convolutional kernels, each consisting of  $N_{in}$  channels, are learned. During the forward pass, the input is group-convolved with each of these kernels in parallel. The  $N_{out}$  output channels are then formed by linearly combining the outcomes of the  $N_{in}$  channels. In other words, lifting and group convolutional layers produce linear combinations of distinct time-frequency decompositions of the input computed in parallel at each layer.

**Wavelet networks are scale-translation equivariant nested non-linear time-frequency transforms.** Just like in conventional neural architectures, the outputs of lifting and group convolutional layers are interleaved with point-wise nonlinearities. Therefore, wavelet networks compute nonlinear scale-translation equivariant feature representations that resemble nested nonlinear time-frequency transforms of the input.

## 10.6 Experiments

In this section, we empirically evaluate wavelet networks. To this end, we take existing neural architectures designed to process raw signals and construct equivalent wavelet networks (W-Nets). We then compare the performance of W-Nets and the corresponding baselines on tasks defined on raw environmental sounds, raw audio and raw electric signals. We replicate as close as possible the training regime of the corresponding baselines and utilize their implementation as a baseline whenever possible. Detailed descriptions of the specific architectures as well as the hyperparameters used for each experiment are provided in Appx. I.3.

### 10.6.1 Classification of environmental sounds

First, we consider the task of classifying environmental sounds on the UrbanSound8K

(US8K) dataset [332]. The US8K dataset consists of 8732 audio clips uniformly drawn from 10 environmental sounds, e.g., siren, jackhammer, etc, of 4 seconds or less, with a total of 9.7 hours of audio.

**Experimental setup.** We compare the  $M_n$ -Nets of Dai et al. [75] and the 1DCNNs of Abdoli et al. [2] with equivalent W-Nets in terms of number of layers and parameters. Contrarily to Dai et al. [75] we sample the audio files at 22.05kHz as opposed to 8kHz. This results from preliminary studies of the data, which indicated that some classes become indistinguishable for the human ear at such low sampling rates.<sup>2</sup> For the comparison with the 1DCNN of Abdoli et al. [2], we select the 50999-1DCNN as baseline, as it is the network type that requires the less human engineering. We note, however, that we were unable to replicate the results reported in Abdoli et al. [2]. In contrast to the  $83 \pm 1.3\%$  reported, we were only able to obtain a final accuracy of  $62.0 \pm 6.791$ . This inconsistency is further detailed in Appx. I.3.1.

To compare to models other than  $M_n$ -nets and 1DCNNs, e.g., Pons et al. [294], Tokozume and Harada [383], we also provide 10-fold cross-validation results. This is done by taking 8 of the 10 official subsets for training, one for validation and one for test. We consistently select the  $(n-1) \bmod 10$  subset for validation when testing on the  $n$ -th subset. We note that this training regime might be different from those used in other works, as previous works often do not disclose which subsets are used for validation.

**Results.** Our results (Tab. 10.1) show that wavelet networks consistently outperform CNNs on raw waveforms. In addition, they are competitive to spectrogram-based approaches, while using significantly fewer parameters and bypassing the need for pre-processing. Furthermore, we observe that encouraging wavelet structure to the convolutional kernels –denoted by the WL suffix– consistently leads to improved accuracy.

### 10.6.2 Automatic music tagging

Next, we consider the task of automatic music tagging on the MagnaTagATune (MTAT) dataset [202]. The MTAT dataset consists of 25879 audio clips with a total of 170 hours of audio, along with several per-song tags. The goal of the task is to provide the right tags to each of the songs in the dataset.

**Experimental setup.** Following Lee et al. [211], we extract the most frequently used 50 tags and trim the audios to 29.1 seconds at a sample-rate of 22.05kHz. Following the convention in literature, we use ROC-curve (AUC) and mean average precision (MAP) as performance metrics. We compare the best performing model of Lee et al. [211], the  $3^9$ -Net with a corresponding wavelet network denoted  $W3^9$ -Net.

**Results.** Our results (Tab. 10.2) show that wavelet networks consistently outperform CNNs on raw waveforms and perform competitively to spectrogram-based approaches

---

<sup>2</sup>See [https://github.com/dwromero/wavelet\\_networks/blob/master/experiments/UrbanSound8K/data\\_analysis.ipynb](https://github.com/dwromero/wavelet_networks/blob/master/experiments/UrbanSound8K/data_analysis.ipynb).

**Table 10.1:** Experimental results on UrbanSound8K.

| URBANSOUND8K                     |                                |                                     |           |
|----------------------------------|--------------------------------|-------------------------------------|-----------|
| MODEL                            | 10 <sup>TH</sup> FOLD ACC. (%) | CROSS-VAL. ACC. (%)                 | # PARAMS. |
| M3-NET                           | 54.48                          | -                                   | 220.67K   |
| W3-NET                           | 61.05                          | -                                   |           |
| W3-NET-WL                        | <b>63.08</b>                   | -                                   | 219.45K   |
| M5-NET                           | 69.89                          | -                                   | 558.08K   |
| W5-NET                           | 72.28                          | -                                   |           |
| W5-NET-WL                        | <b>74.55</b>                   | -                                   | 558.03K   |
| M11-NET                          | 74.43                          | -                                   | 1.784M    |
| W11-NET                          | 79.33                          | $66.97 \pm 5.178$                   |           |
| W11-NET-WL                       | <b>80.41</b>                   | <b><math>68.47 \pm 4.914</math></b> | 1.806M    |
| M18-NET                          | 69.65                          | -                                   | 3.680M    |
| W18-NET                          | 75.87                          | $64.02 \pm 4.645$                   |           |
| W18-NET-WL                       | <b>78.26</b>                   | <b><math>65.01 \pm 5.431</math></b> | 3.759M    |
| M34-NET                          | 75.15                          | -                                   | 3.978M    |
| W34-NET                          | 76.22                          | $65.69 \pm 5.780$                   |           |
| W34-NET-WL                       | <b>78.38</b>                   | <b><math>66.77 \pm 4.771</math></b> | 4.021M    |
| 1DCNN                            | -                              | $62.00 \pm 6.791$                   | 453.42K   |
| W-1DCNN                          | -                              | $62.47 \pm 4.925$                   |           |
| W-1DCNN-WL                       | -                              | <b><math>62.64 \pm 4.979</math></b> | 458.61K   |
| COMPARISON WITH OTHER APPROACHES |                                |                                     |           |
| MODEL                            | TYPE                           | CROSS-VAL. ACC. (%)                 | # PARAMS. |
| W11-NET-WL                       | RAW                            | $68.47 \pm 4.914$                   | 1.806M    |
| PICZAKCNN [290]                  | MEL SPECTROGRAM                | 73.7                                | 26M       |
| VGG [293]                        |                                | 70.74                               | 77M       |
| ENVNET-V2 [383]                  | RAW (BAGGING)                  | <b>78</b>                           | 101M      |

in this dataset as well. In addition, we observe that encouraging the learning of wavelet-like kernels consistently results in increased accuracy as well.

### 10.6.3 Bearing fault detection

Finally, we also validate Wavelet networks for the task of condition monitoring in induction motors. To this end, we classify healthy and faulty bearings from raw data provided by [*Anonymized Company*]. The dataset consists of 246 clips of 15 seconds sampled at 20kHz. The dataset is slightly unbalanced containing 155 healthy and 91 faulty recordings [155, 91]. The dataset is previously split into a training set of [85, 52] and a test set of [70, 39] samples, respectively. These splits are provided ensuring that measurements from the same motor are not included both in the train and the test set. We utilize 20% of the training set for validation. Each clip is composed of 6 channels measuring both current and voltage on the 3 poles of the motor.

**Experimental setup.** We take the best performing networks on the US8K dataset: the M-11 and W-11 networks, and utilize variants of these architectures for this dataset.

**Table 10.2:** Experimental results on MTAT.

| MAGNATAGATUNE           |              |              |              |              |           |
|-------------------------|--------------|--------------|--------------|--------------|-----------|
| MODEL                   | AVERAGE AUC  |              | MAP          |              | # PARAMS. |
|                         | PER-CLASS    | PER-CLIP     | PER-CLASS    | PER-CLIP     |           |
| 3 <sup>9</sup> -NET     | 0.893        | 0.936        | 0.385        | 0.700        | 2.394M    |
| W3 <sup>9</sup> -NET    | 0.895        | 0.941        | 0.397        | 0.719        |           |
| W3 <sup>9</sup> -NET-WL | <b>0.899</b> | <b>0.943</b> | <b>0.404</b> | <b>0.723</b> | 2.404M    |

| COMPARISON WITH OTHER APPROACHES   |               |               |               |               |           |
|------------------------------------|---------------|---------------|---------------|---------------|-----------|
| MODEL                              | AVERAGE AUC   |               | MAP           |               | # PARAMS. |
|                                    | PER-CLASS     | PER-CLIP      | PER-CLASS     | PER-CLIP      |           |
| PCNN [232]                         | 0.9013        | <b>0.9365</b> | <b>0.4267</b> | <b>0.6902</b> | -         |
| CNN [294] <sup>*</sup><br>(RAW)    | 0.8905        | -             | 0.3492        | -             | 11.8M     |
| CNN [294] <sup>*</sup><br>(SPECT.) | <b>0.9040</b> | -             | 0.3811        | -             | 5M        |
| CNN [295]<br>(SPECT.)              | 0.893         | -             | -             | -             | 191K      |

\* Reported results are obtained in a more difficult version of this dataset.

**Table 10.3:** Experimental results on bearing fault detection.

| MODEL      | ACC. (%)       | # PARAMS. |
|------------|----------------|-----------|
| M11-NET    | 65.1376        | 1.806M    |
| W11-NET    | <b>68.8073</b> |           |
| W11-NET-WL | <b>70.207</b>  | 1.823M    |

**Results.** We observe that Wavelet networks outperform CNNs on raw waveforms and encouraging the learning of wavelet-like kernels consistently improves accuracy (Tab. 10.3).

#### 10.6.4 Discussion

Our empirical results firmly establish wavelet networks as a promising avenue for learning from raw time-series data. Notably, these results highlight that considering the symmetries inherent to time-series data –namely translation and scale– for the development of neural networks consistently leads to improved outcomes. Furthermore, we observe that the benefits of wavelet networks extend beyond sound and audio domains. This result advocates for the use of wavelet networks and scale-translation equivariance for learning on time-series data from different sources, e.g., financial data, sensory data. Finally, we also note that promoting the learning of wavelet-like convolutional kernels consistently leads to improved outcomes. We posit that this discovery may hold broader implications for group equivariant networks in general.

**Relation to scale-equivariant models of images and 2D signals.** In the past, multiple scale-equivariant models have been proposed for the processing of images and 2D signals [362, 364, 433]. Interestingly, we find that the difference in the lengths of the inputs received by image and time-series models leads to very different insights per modality. For comparison, Sosnovik et al. [362] considers images up to 96×96 pixels, whereas

audio files in the US8K dataset are 32.000 samples long. We find that this difference in input lengths has crucial implications for how scale interactions within scale-equivariant models function. Sosnovik et al. [362] mentions that using inter-scale interactions introduces additional equivariance errors due to the truncation of the set  $\mathcal{S}$ . Therefore, their networks are built with either no scale interaction or interactions of maximum 2 scales. This strongly contrasts with time-series where incorporating inter-scale interactions consistently leads to performance improvements. In our case, the number of scales and inter-scale interactions is rather constrained by the size and computational cost of convolutional kernels (Sec. 10.5.2) rather than their potential negative impact on the model’s accuracy.

## 10.7 Limitations and future work

**Memory and time complexity grows with the number of scales.** The main limitation of our approach is the increase memory and time demands as the number of scales grows. A potential solution could be adopting Monte-Carlo approximations for the computation of group convolutions [105]. This could not only make Wavelet networks equivariant to the continuous scale group –in expectation–, but also could dramatically reduce the number of scales used in each forward pass. Another promising direction involves extending the concept of partial equivariance [316] to the scale group. This would allow the model to learn the subset of scales to which it is equivariant, potentially improving execution and adaptability. Lastly, exploring separable group convolutions [190] could help reduce the computational and memory requirements of wavelet networks.

**Convolutions with large convolutional kernels: parameterization and efficiency.** The foundation of our approach hinges on computing convolutions with banks of dilated convolutional kernels (Eq. 10.12, 10.11). Consequently, considering how these kernels are parameterized as well as how these convolutions are computed can unveil avenues for future improvement. Recently, Romero et al. [321] introduced an expressive continuous parameterization for (large) convolutional kernels that has proven advantageous for complex tasks such as large language modelling [292] and processing DNA chains [273]. Exploring the use of this parameterization for wavelet networks could lead to valuable insights and improvements, potentially surpassing the current utilization of B<sup>2</sup>-spline bases. Furthermore, convolutional networks that rely on convolutions with very large convolutional kernels, e.g., [273, 292, 321], leverage the Fourier transform to compute convolutions in the frequency domain. In the context of wavelet networks, dynamically selecting between spatial and Fourier convolutions based on the size of convolutional kernels has the potential to significantly improve their efficiency.

## 10.8 Conclusion

In conclusion, this study introduces *Wavelet Networks*, a new class of neural networks for raw time-series processing that harness the symmetries inherent to time-series data

–scale and translation– for the construction of neural architectures that respect them. We observe a clear connection between the wavelet transform and scale-translation group convolutions, establishing a profound link between our approach and classical spectro-temporal analysis. In contrast to the usual approach, which uses spectro-temporal representations as a frontend for the subsequent use of 2D CNNs, wavelet networks consistently preserve these symmetries across the whole network through the use of convolutional layers that resemble the wavelet transform. Our analysis reveals that wavelet networks combine the benefits of wavelet-like time-frequency decompositions with the adaptability and non-linearity of neural networks.

Our empirical results demonstrate the superiority of Wavelet Networks over conventional CNNs on raw time-series data, achieving comparable performance to approaches that rely on engineered spectrogram-based methods, e.g., log-Mel spectrograms, with reduced parameters and no need for preprocessing. This work pioneers the concept of scale-translation equivariant neural networks for time-series analysis, opening new avenues for time-series processing.



# 11

## Learning Equivariances and Partial Equivariances from Data

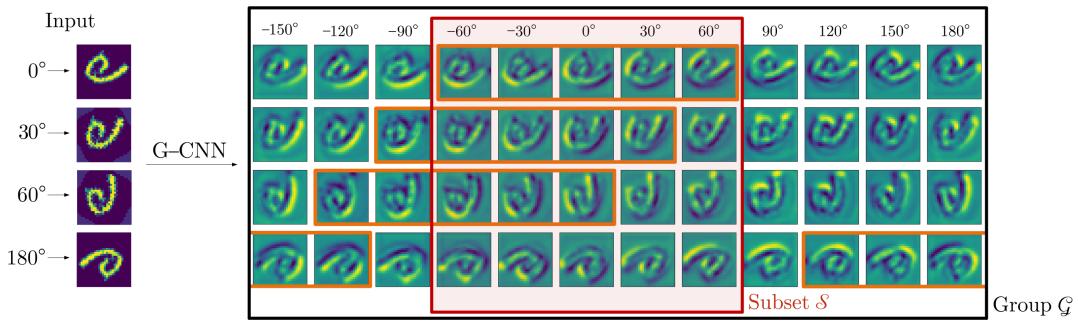
*Based on the paper:  
Learning Partial Equivariances from Data [316]*

### 11.1 Introduction

The translation equivariance of Convolutional Neural Networks (CNNs) [208] has proven an important inductive bias for good generalization on vision tasks. This is achieved by restricting learned features to respect the translation symmetry encountered in visual data, such that if an input is translated, its features are also translated, but not modified. Group equivariant CNNs (G-CNNs) [65] extend equivariance to other symmetry groups. Analogously, they restrict the learned features to respect the symmetries in the group considered such that if an input is transformed by an element in the group, e.g., a rotation, its features are also transformed, e.g., rotated, but not modified.

Nevertheless, the group to which G-CNNs are equivariant must be fixed prior to training, and imposing equivariance to symmetries not present in the data leads to overly constrained models and worse performance [48]. The latter comes from a difference in the data distribution, and the family of distributions the model can describe. Consequently, the group must be selected carefully, and it should correspond to the transformations that appear naturally in the data.

Frequently, transformations appearing in data can be better represented by a subset of



**Figure 11.1:** Partial group convolution. In a group convolution, the domain of the output is the group  $\mathcal{G}$ . Consequently, all output components are part of the output for any group transformation of the input. In a partial group convolution, however, the domain of the output is a learned subset  $\mathcal{S}$  and all values outside of  $\mathcal{S}$  are discarded. As a result, parts of the output in a partial group convolution can change for different group transformations of the input. In this figure, the output components within  $\mathcal{S}$  for a  $0^\circ$  rotation (outlined in orange) gradually leave  $\mathcal{S}$  for stronger transformations of the input. For strong transformations –here a  $180^\circ$  rotation–, the output components within  $\mathcal{S}$  are *entirely* different. This difference allows partial group convolutions to distinguish among input transformations. By controlling the size of  $\mathcal{S}$ , the level of equivariance of the operation can be adjusted.

a group than by a group as a whole, e.g., rotations in  $[-90^\circ, 90^\circ]$ . For instance, natural images much more likely show an elephant standing straight or slightly rotated than an elephant upside-down. In some cases, group transformations even change the desired model response, e.g., in the classification of the digits 6 and 9. In both examples, the data distribution is better represented by a model that respects rotation equivariance *partially*. That is, a model equivariant to some, but not all rotations.

Moreover, the optimal level of equivariance may change per layer. This results from changes in the likelihood of some transformations for low and high-level features. For instance, whereas the orientations of edges in an human face are properly described with full rotation equivariance, the poses of human faces relative to the camera are better represented by rotations in a subset of the circle.

The previous observations indicate that constructing a model with different levels of equivariance at each layer may be advantageous. Weiler and Cesa [422] empirically observed that manually tuning the level of equivariance at different layers leads to accuracy improvements for non-fully equivariant tasks. Nevertheless, manually tuning layer-wise levels of equivariance is not straightforward and requires iterations over several possible combinations of equivariance levels. Consequently, it is desirable to construct a model able to learn optimal levels of equivariance directly from data.

In this work, we introduce *Partial Group equivariant CNNs* (Partial G-CNNs): a family of equivariant models able to *learn* layer-wise levels of equivariance directly from data.

Instead of sampling group elements uniformly from the group during the group convolution –as in G-CNNs–, Partial G-CNNs learn a probability distribution over group elements at each group convolutional layer in the network, and sample group elements during group convolutions from the learned distributions. By tuning the learned distributions, Partial G-CNNs adjust their level of equivariance at each layer during training.

We evaluate Partial G-CNNs on illustrative toy tasks and vision benchmark datasets. We show that whenever full equivariance is beneficial, e.g., for rotated MNIST, Partial G-CNNs learn to remain fully equivariant. However, if equivariance becomes harmful, e.g., for classification of 6 / 9 digits and natural images, Partial G-CNNs learn to adjust equivariance to a subset of the group to improve accuracy. Partial G-CNNs improve upon conventional G-CNNs when equivariance reductions are advantageous, and match their performance whenever their design is optimal.

In summary, our **contributions** are:

- We present a novel design for the construction of equivariant neural networks able to learn layer-wise levels of partial or full equivariance from data.
- We empirically show that Partial G-CNNs perform better than conventional G-CNNs for tasks for which full equivariance is harmful, and match their performance if full equivariance is beneficial.

## 11.2 Background

This work expects the reader to have a basic understanding of concepts from group theory such as groups, subgroups and group actions. Please refer to Appx. J.1 if you are unfamiliar with these terms.

**Group equivariance.** Group equivariance is the property of a map to respect the transformations in a group. We say that a map is equivariant to a group if whenever the input is transformed by elements of the group, the output of the map is equally transformed but not modified. Formally, for a group  $\mathcal{G}$  with elements  $g \in \mathcal{G}$  acting on a set  $\mathcal{X}$ , and a map  $\phi : \mathcal{X} \rightarrow \mathcal{X}$ , we say that  $\phi$  is equivariant to  $\mathcal{G}$  if:

$$\phi(g \cdot x) = g \cdot \phi(x), \quad \forall x \in \mathcal{X}, \forall g \in \mathcal{G}. \quad (11.1)$$

For example, the convolution of a signal  $f : \mathbb{R} \rightarrow \mathbb{R}$  and a kernel  $\psi : \mathbb{R} \rightarrow \mathbb{R}$  is *translation equivariant* because  $\mathcal{L}_t(\psi * f) = \psi * \mathcal{L}_t f$ , where  $\mathcal{L}_t$  translates the function by  $t$ , i.e.,  $\mathcal{L}_t f(x) = f(x - t)$ . In other words, if the input is translated, its convolutional descriptors are also translated but not modified.

**The group convolution.** To construct neural networks equivariant to a group  $\mathcal{G}$ , we require an operation that respects the symmetries in the group. The *group convolution* is such a mapping. It generalizes the convolution for equivariance to general symmetry groups. Formally, for any  $u \in \mathcal{G}$ , the group convolution of a signal  $f : \mathcal{G} \rightarrow \mathbb{R}$  and a kernel

$\psi : \mathcal{G} \rightarrow \mathbb{R}$  is given by:

$$h(u) = (\psi * f)(u) = \int_{\mathcal{G}} \psi(v^{-1}u) f(v) d\mu_{\mathcal{G}}(v), \quad (11.2)$$

where  $\mu_{\mathcal{G}}(\cdot)$  is the (invariant) Haar measure of the group. The group convolution is  $\mathcal{G}$ -equivariant in the sense that for all  $u, v, w \in \mathcal{G}$ , it holds that:

$$(\psi * \mathcal{L}_w f)(u) = \mathcal{L}_w(\psi * f)(u), \text{ with } \mathcal{L}_w f(u) = f(w^{-1}u).$$

**The lifting convolution.** Regularly, the input of a neural network is not readily defined on the group of interest  $\mathcal{G}$ , but on a sub-domain thereof  $\mathcal{X}$ , i.e.,  $f : \mathcal{X} \rightarrow \mathbb{R}$ . For instance, medical images are functions on  $\mathbb{R}^2$  although equivariance to 2D-translations and planar rotations is desirable. In this case  $\mathcal{X} = \mathbb{R}^2$ , and the group of interest is  $\mathcal{G} = \text{SE}(2)$ . Consequently, we must first *lift* the input from  $\mathcal{X}$  to  $\mathcal{G}$  in order to use group convolutions. This is achieved via the *lifting convolution* defined as:

$$(\psi * \text{lift } f)(u) = \int_{\mathcal{X}} \psi(v^{-1}u) f(v) d\mu_{\mathcal{G}}(v); \quad u \in \mathcal{G}, v \in \mathcal{X}. \quad (11.3)$$

**Practical implementation of the group convolution.** The group convolution requires integration over a continuous domain and, in general, cannot be computed in finite time. As a result, it is generally approximated. Two main strategies exist to approximate group convolutions with regular group representations: group discretization [65] and Monte Carlo approximation [105]. The former approximates the group convolution with a fixed group discretization. Unfortunately, the approximation becomes *only* equivariant to the transformations in the discretization and not to the intrinsic continuous group.

A Monte Carlo approximation, on the other hand, ensures equivariance –in expectation– to the continuous group. This is done by uniformly sampling transformations  $\{v_j\}, \{u_i\}$  from the group during each forward pass, and using these transformations to approximate the group convolution as:

$$(\psi \hat{*} f)(u_i) = \sum_j \psi(v_j^{-1}u_i) f(v_j) \mu_{\mathcal{G}}(v_j). \quad (11.4)$$

Note that this Monte Carlo approximation requires the convolutional kernel  $\psi$  to be defined on the *continuous group*. As the domain cannot be enumerated, independent weights cannot be used to parameterize the convolutional kernel. Instead, Finzi et al. [105] parameterize it with a small neural network, i.e.,  $\psi(x) = \text{MLP}(x)$ . This allows them to map all elements  $v_j^{-1}u_i$  to a defined kernel value.

## 11.3 Partial Group Equivariant Networks

### 11.3.1 (Approximate) partial group equivariance

Before defining the partial group convolution, we first formalize what we mean by partial group equivariance. We say that a map  $\phi$  is *partially equivariant* to  $\mathcal{G}$ , if it is equivariant to transformations in a subset of the group  $\mathcal{S} \subset \mathcal{G}$ , but not necessarily to all transfor-

mations in the group  $\mathcal{G}$ . That is, if:

$$\phi(g \cdot x) = g \cdot \phi(x) \quad \forall x \in \mathcal{X}, \forall g \in \mathcal{S}. \quad (11.5)$$

Different from equivariance to a *subgroup* of  $\mathcal{G}$  – a subset of the group that also fulfills the group axioms –, we do not restrict the subset  $\mathcal{S}$  to be itself a group.

As explained in detail in Sec. 11.3.3, partial equivariance holds, in general, *only approximately*, and it is exact only if  $\mathcal{S}$  is a subgroup of  $\mathcal{G}$ . This results from the set  $\mathcal{S}$  not being necessarily closed under group actions. In other words, partial equivariance is a relaxation of group equivariance similar to *soft invariance* [398]: the property of a map to be approximately invariant. We opt for the word *partial* in the equivariance setting to emphasize that (approximate) partial group equivariance arises by restricting the domain of the signals in a group convolution to a subset, i.e., a part, of the group.

### 11.3.2 The partial group convolution

Let  $\mathcal{S}^{(1)}, \mathcal{S}^{(2)}$  be subsets of a group  $\mathcal{G}$  and  $p(u)$  be a probability distribution on the group, which is non-zero only on  $\mathcal{S}^{(2)}$ . The partial group convolution from a function  $f : \mathcal{S}^{(1)} \rightarrow \mathbb{R}$  to a function  $h : \mathcal{S}^{(2)} \rightarrow \mathbb{R}$  is given by:

$$h(u) = (\psi * f)(u) = \int_{\mathcal{S}^{(1)}} p(v) \psi(v^{-1}u) f(v) d\mu_{\mathcal{G}}(v); \quad u \in \mathcal{S}^{(2)}, v \in \mathcal{S}^{(1)}. \quad (11.6)$$

In contrast to group convolutions whose inputs and outputs are always defined on the entire group, i.e.,  $f, h : \mathcal{G} \rightarrow \mathbb{R}$ , the domain of the input and output of the partial group convolution can also be *subsets of the group*. By learning these subsets, the model can become (i) fully equivariant ( $\mathcal{S}^{(1)}, \mathcal{S}^{(2)} = \mathcal{G}$ ), (ii) partially equivariant ( $\mathcal{S}^{(1)}, \mathcal{S}^{(2)} \neq \mathcal{G}$ ), or (iii) forget some equivariances ( $\mathcal{S}^{(2)}$  a subgroup of  $\mathcal{G}$ ).

### 11.3.3 From group convolutions to partial group convolutions

In this section, we show how group convolutions can be extended to describe partial equivariances. Vital to our analysis is the equivariance proof of the group convolution [65, 68]. In addition, we must distinguish between the domains of the input and output of the group convolution, i.e., the domains of  $f$  and  $h$  in Eq. 11.2. This distinction is important because they may be different for partial group convolutions. From here on, we refer to these as the *input domain* and the *output domain*.

**Proposition 11.3.1.** *Let  $\mathcal{L}_w f(u) = f(w^{-1}u)$ . The group convolution is  $\mathcal{G}$ -equivariant in the sense that:*

$$(\psi * \mathcal{L}_w f)(u) = \mathcal{L}_w(\psi * f)(u), \text{ for all } u, v, w \in \mathcal{G}. \quad (11.7)$$

*Proof.* [68]

$$\begin{aligned} (\psi * \mathcal{L}_w f)(u) &= \int_{\mathcal{G}} \psi(v^{-1}u) f(w^{-1}v) d\mu_{\mathcal{G}}(v) = \int_{\mathcal{G}} \psi(\bar{v}^{-1}w^{-1}u) f(\bar{v}) d\mu_{\mathcal{G}}(\bar{v}) \\ &= (\psi * f)(w^{-1}u) = \mathcal{L}_w(\psi * f)(u). \end{aligned}$$

In the first line, the change of variables  $\bar{v} := w^{-1}v$  is used. This is possible because the group convolution is a map from the group to itself, and thus if  $w, v \in \mathcal{G}$ , so does  $w^{-1}v$ . Moreover, as the Haar measure is an invariant measure on the group, we have that  $\mu_{\mathcal{G}}(v) = \mu_{\mathcal{G}}(\bar{v})$ , for all  $v, \bar{v} \in \mathcal{G}$ .  $\square$

**Going from the group  $\mathcal{G}$  to a subset  $\mathcal{S}$ .** Crucial to the proof of Proposition 11.3.1 is the fact the group convolution is an operation from functions on the group to functions on the group. As a result,  $w^{-1}u$  is a member of the output domain for any  $w \in \mathcal{G}$  applied to the input domain. Consequently, a group transformation applied to the input can be reflected by an equivalent transformation on the output.

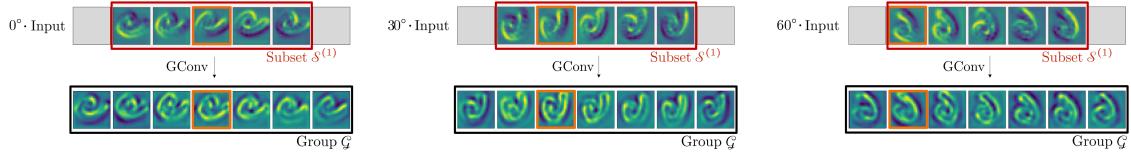
Now, consider the case in which the output domain is not the group  $\mathcal{G}$ , but instead an arbitrary subset  $\mathcal{S} \subset \mathcal{G}$ , e.g., rotations in  $[-\frac{\pi}{2}, \frac{\pi}{2}]$ . Following the proof of Proposition 11.3.1 with  $u \in \mathcal{S}$ , and  $v \in \mathcal{G}$ , we observe that the operation is equivariant to transformations  $w \in \mathcal{G}$  as long as  $w^{-1}u$  is a member of  $\mathcal{S}$ . However, if  $w^{-1}u$  does not belong to the output domain  $\mathcal{S}$ , the output of the operation cannot reflect an equivalent transformation to that of the input, and thus equivariance is not guaranteed (Fig. 11.1). By tuning the size of  $\mathcal{S}$ , partial group convolutions can adjust their level of equivariance.

Note that equivariance is *only* obtained if Eq. 11.7 holds for *all* elements in the output domain. That is, if  $w^{-1}u$  is a member of  $\mathcal{S}$ , for all elements  $u \in \mathcal{S}$ . For partial group convolutions, this is, in general, not the case as the output domain  $\mathcal{S}$  is not necessarily closed under group transformations. Nevertheless, we can precisely quantify how much the output response will change for any input transformation given an output domain  $\mathcal{S}$ . Intuitively, this difference is given by the difference in the parts of the output feature representation that go in and out of  $\mathcal{S}$  by the action of input group transformations. The stronger the transformation and the smaller the size of  $\mathcal{S}$ , the larger the equivariance difference in the output is (Fig. 11.1). The formal derivations are provided in Appx. J.2.1.

**Going from a subset  $\mathcal{S}^{(1)}$  to another subset  $\mathcal{S}^{(2)}$ .** Now, consider the case in which the domain of the input and the output are both subsets of the group, i.e.,  $v \in \mathcal{S}^{(1)}$  and  $u \in \mathcal{S}^{(2)}$ . Analogous to the previous case, equivariance to input transformations  $w \in \mathcal{G}$  holds at positions  $u \in \mathcal{S}^{(2)}$  for which  $w^{-1}u$  are also members of  $\mathcal{S}^{(2)}$ . Nevertheless, the input domain is not longer restricted to be closed, and thus the input can also change for different group transformations.

To see this, consider a partial group convolution from an input subset  $\mathcal{S}^{(1)}$  to the group  $\mathcal{G}$  (Fig. 11.2). Even if the output domain is the group, differences in the output feature map can be seen. This results from differences observed in the input feature map  $f$  for different group transformations of the input.

Similar to the previous case, we can precisely quantify how much the output response changes for an arbitrary subset  $\mathcal{S}^{(1)}$  in the input domain. Intuitively, the difference is given by the change in the parts of the input feature representation that go in and out of  $\mathcal{S}^{(2)}$  by the action of the input group transformation. The stronger the transformation



**Figure 11.2:** The effect of group subsets in the input domain. Partial group convolutions can receive input functions whose domain is not the group  $\mathcal{G}$  but a subset  $\mathcal{S}^{(1)}$ , e.g., in a mid layer of a Partial G-CNN. Consequently, even if the output domain is the group, i.e.,  $\mathcal{S}^{(2)} = \mathcal{G}$ , a partial group convolution does not produce exactly equivariant outputs by design. The difference from exact equivariance in the output response is proportional to the strength of the input transformation and the size of  $\mathcal{S}^{(1)}$ .

and the smaller the size of  $\mathcal{S}^{(1)}$ , the larger the difference in the output is (Fig. 11.2). The formal treatment and derivation of this quantity is provided in Appx. J.2.2.

**Special case: lifting convolutions.** To conclude, we note that the lifting convolution (Eq. 11.3) is a special case of a partial group convolution with  $\mathcal{S}^{(1)} = \mathcal{X}$  and  $\mathcal{S}^{(2)} = \mathcal{G}$ . Note however, that the lifting convolution is fully group equivariant. This is because the domain of the input  $-\mathcal{X}-$  is closed under group transformations. Hence no input component leaves  $\mathcal{X}$  for group transformations of the input.

### 11.3.4 Learning group subsets through probability distributions on the group

So far, we have discussed the properties of the partial group convolution in terms of group subsets without specifying how these subsets can be learned. Here, we describe how this can be done by learning a certain probability distribution on the group. We provide examples for discrete groups, continuous groups, and combinations thereof.

Vital to our approach is the Monte Carlo approximation to the group convolution presented in Sec. 11.2:

$$(\psi \hat{\ast} f)(u_i) = \sum_j \psi(v_j^{-1} u_i) f(v_j) \bar{\mu}_{\mathcal{G}}(v_j).$$

As shown in Appx. J.3, this approximation is equivariant to  $\mathcal{G}$  in expectation if the elements in the input and output domain are uniformly sampled from the Haar measure, i.e.,  $u_i, v_j \sim \mu_{\mathcal{G}}(\cdot)$ .<sup>1</sup>

**Approach.** Our main observation is that we can prioritize sampling specific group elements during the group convolution by learning a probability distribution  $p(u)$  over the elements of the group. When group convolutions draw elements uniformly from the group, each group element is drawn with equal probability and thus, the resulting approximation is fully equivariant in expectation. However, we can also define a different probability distribution that draws some samples with larger probability. For instance, we can sample from a certain region, e.g., rotations in  $[-\frac{\pi}{2}, \frac{\pi}{2}]$ , by defining a probability

<sup>1</sup>Finzi et al. [105] show a similar result where  $u_i$  and  $v_j$  are the same points and thus  $v_j \sim \mu_{\mathcal{G}}(\cdot)$  suffices.

distribution on the group  $p(u)$  which is uniform in this range, but zero otherwise. The same principle can be used to forget an equivariance by letting this distribution collapse to a single point, e.g., the identity, along the corresponding group dimension.

In other words, learning a probability distribution  $p(u)$  on the group that is non-zero *only* in a subset of the group can be used to effectively learn this subset. Specifically, we define a probability distribution  $p(u)$  on the output domain of the group convolution in order to learn a subset of the group  $\mathcal{S}^{(2)}$  upon which partial equivariance is defined. Note that we only need to define a distribution on the output domain of each layer. This is because neural networks apply layers sequentially, and thus the distribution on the output domain of the previous layer defines the input domain of the next layer.

**Distributions for one-dimensional continuous groups.** We take inspiration from Augerino [25], and use the reparameterization trick [186] to parameterize continuous distributions. In particular, we use the reparameterization trick on the Lie algebra of the group [102] to define a distribution which is uniform over a connected set of group elements  $[u^{-1}, \dots, e, \dots, u]$ , but zero otherwise. To this end, we define a uniform distribution  $\mathcal{U}(u \cdot [-1, 1])$  with learnable  $u$  on the Lie algebra  $\mathfrak{g}$ , and map it to the group via the push-forward of the exponential map  $\exp : \mathfrak{g} \rightarrow \mathcal{G}$ . This give us a distribution which is uniform over a connected set of elements  $[u^{-1}, \dots, e, \dots, u]$ , but zero otherwise.<sup>2</sup>

For instance, we can learn a distribution on the rotation group  $\text{SO}(2)$ , which is uniform between  $[-\theta, \theta]$  and zero otherwise by defining a uniform probability distribution  $\mathcal{U}(\theta \cdot [-1, 1])$  with learnable  $\theta$  on the Lie algebra, and mapping it to the group. If we parameterize group elements as scalars  $g \in [-\pi, \pi]$ , the exponential map is the identity, and thus  $p(g) = \mathcal{U}(\theta \cdot [-1, 1])$ . If we sample group elements from this distribution during the calculation of the group convolution, the output domain will only contain elements in  $[-\theta, \theta]$  and the output feature map will be partially equivariant.

**Distributions for one-dimensional discrete groups.** We can define a probability distribution on a discrete group as the probability of sampling from all possible element combinations. For instance, for the mirroring group  $\{1, -1\}$ , this distribution assigns a probability to each of the combinations  $\{0, 0\}, \{0, 1\}, \{1, 0\}, \{1, 1\}$  indicating whether the corresponding element is sampled (1) or not (0). For a group with elements  $\{e, g_1, \dots, g_n\}$ , however, this means sampling from  $2^{n+1}$  elements, which is computationally expensive and potentially difficult to train. To cope with this, we instead define element-wise Bernoulli distributions over each of the elements  $\{g_1, \dots, g_n\}$ , and learn the probability  $p_i$  of sampling each element  $g_i$ . The probability distribution on the group can then be formulated as the joint probability of the element-wise Bernoulli distributions  $p(e, g_1, \dots, g_n) = \prod_{i=1}^n p(g_i)$ .

To learn the element-wise Bernoulli distributions, we use the Gumbel-Softmax trick [169, 246], and use the Straight-Through Gumbel-Softmax to back-propagate through

---

<sup>2</sup>Note that an exp-push-forwarded local uniform distribution is locally equivalent to the Haar measure, and thus we can still use the Haar measurement for integration on group subsets.

**Algorithm 1** The Partial Group Convolution Layer

---

```

1: Inputs: position, function-value tuples on the group or a subset thereof  $\{v_j, f(v_j)\}$ .
2: Outputs: convolved position, function-value tuples on the output group subset
    $\{u_i, (f \hat{\ast} \psi)(u_i)\}$ .
3:  $\{u_i\} \sim p(u)$                                      ▷ Sample elements from  $p(u)$ 
4: for  $u_i \in \{u_i\}$  do
5:    $h(u_i) = \sum_j \psi(v_j^{-1} u_i) f(v_j) \bar{\mu}_G(v_j)$       ▷ Compute group convolution
6: end for
7: Return:  $\{u_i, h(u_i)\}$ 

```

---

sampling. If all the probabilities are equal to 1, i.e.,  $\{p_i=1\}_{i=1}^n$ , the group convolution will be fully equivariant. Whenever probabilities start declining, group equivariance becomes partial, and, in the limit, if all probabilities become zero, i.e.,  $\{p_i=0\}_{i=1}^n$ , then only the identity is sampled and this equivariance is effectively forgotten.

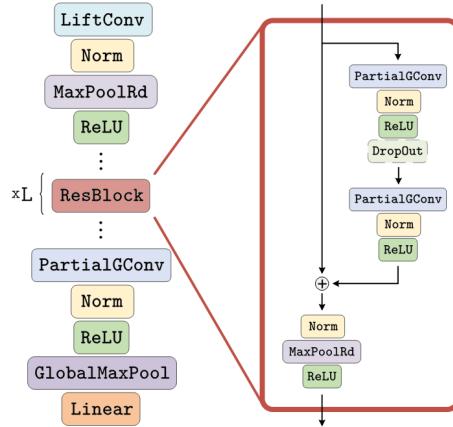
**Probability distributions for multi-dimensional groups.** There exist several multi-dimensional groups with important applications, such as the orthogonal group  $O(2)$  – parameterized by rotations and mirroring–, or the dilation-rotation group –parameterized by scaling and rotations–.

For multi-dimensional groups, we parameterize the probability distribution over the entire group as a combination of *independent probability distributions along each of the group axes*. For a group  $G$  with elements  $g$  decomposable along  $n$  dimensions  $g=(d_1, \dots, d_n)$ , we decompose the probability distribution as:  $p(g)=\prod_{i=1}^n p(d_i)$ , where the probability  $p(d_i)$  is defined given the type of space – continuous or discrete–. For instance, for the orthogonal group  $O(2)$  defined by rotations  $r$  and mirroring  $m$ , i.e.,  $g = (r, m)$ ,  $r \in SO(2)$ ,  $m \in \{\pm 1\}$ , we define the probability distribution on the group as  $p(g)=p(r) \cdot p(m)$ , where  $p(r)$  is a continuous distribution, and  $p(m)$  is a discrete one as defined above.

### 11.3.5 Partial Group Equivariant Networks

To conclude this section, we illustrate the structure of Partial G-CNNs. We build upon Finzi et al. [105] and extend their continuous G-CNNs to discrete groups. This is achieved by parameterizing the convolutional kernels on (continuous) Lie groups on their Lie algebra, and applying the action of discrete groups directly on the group representation of the kernels.

In addition, we replace the isotropic lifting of Finzi et al. [105] with lifting convolutions (Eq. 11.3). Inspired by Romero et al. [321], we parameterize convolutional kernels as implicit neural representations with SIRENs [358]. This parameterization leads to higher expressivity, faster convergence, and better accuracy than ReLU, LeakyReLU and Swish MLP parameterizations used so far for continuous G-CNNs, e.g., [105, 343] –see Tab. J.2, [190]–. The architecture of Partial G-CNNs is shown in Fig. 11.3.



**Figure 11.3:** The Partial G-CNN.

## 11.4 Related work

**Group equivariant neural networks.** The seminal work of G-CNNs [65] has inspired several methods equivariant to many different groups. Existing methods show equivariance to planar [89, 424, 434], spherical rotations [67, 98–100, 423], scaling [322, 362, 433], and other symmetry groups [31, 107, 318]. Group equivariant self-attention has also been proposed [111, 163, 314]. Common to all these methods is that they are fully equivariant and the group must be fixed prior to training. Contrarily, Partial G-CNNs learn their level of equivariance from data and can represent full, partial and no equivariance.

**Invariance learning.** Learning the right amount of global invariance from data has been explored by learning a probability distribution over continuous test-time augmentations [25], or by using the marginal likelihood of a Gaussian process [396, 398]. Contrary to these approaches, Partial G-CNNs aim to learn the right level of equivariance at every layer and do not require additional loss terms. Partial G-CNNs relate to Augerino [25] in the form that probability distributions are defined on continuous groups. However, Partial G-CNNs are intended to learn partial layer-wise equivariances and are able to learn probability distributions on discrete groups. There also exist learnable data augmentation strategies [45, 140, 219, 222] that can find transformations of the input that optimize the task loss. We can also view Augerino as learning a smart data augmentation technique which we compare with. In contrast to these methods, Partial G-CNNs find optimal partial equivariances at each layer.

**Equivariance learning.** Learning equivariant mappings from data has been explored by meta-learning of weight-tying matrices encoding symmetry equivariances [4, 476] and by learning the Lie algebra generators of the group jointly with the parameters of the network [83]. These approaches utilize the same learned symmetries across layers. MSR [476] is only applicable to (small) discrete groups, and requires long training times. L-Conv [83] is only applicable to continuous groups and is not fully compatible

**Table 11.2:** Augerino vs. Partial G-CNNs.**Table 11.1:** Results on MNIST6-180 and Terms in parentheses show accuracy of MNIST6-M.

| BASE GROUP         | DATASET    | G-CNN | PARTIAL G-CNN        | CLASSIFICATION ACCURACY (%) |               |          |
|--------------------|------------|-------|----------------------|-----------------------------|---------------|----------|
|                    |            |       |                      | ROTMNIST                    | CIFAR10       | CIFAR100 |
| SE(2)<br>Mirroring | MNIST6-180 | 50.0  | <b>100.0</b>         |                             |               |          |
|                    | MNIST6-M   | 50.0  | <b>100.0</b>         |                             |               |          |
| E(2)               | MNIST6-180 | 50.0  | <b>100.0</b>         |                             |               |          |
|                    | MNIST6-M   | 50.0  | <b>100.0</b>         |                             |               |          |
| SE(2)              |            | 8     | 99.17 (99.23)        | 82.38 (88.59)               | 52.98 (57.26) |          |
|                    |            | 16    | <b>99.25</b> (99.18) | 82.53 (88.59)               | 51.47 (57.31) |          |
| E(2)               |            | 8     | 99.12 (97.78)        | 84.29 (89.00)               | 52.59 (55.22) |          |
|                    |            | 16    | <b>99.18</b> (98.35) | 83.54 (90.12)               | 54.76 (61.46) |          |

with current deep learning components, e.g., pooling, normalization. Unlike these approaches, Partial G-CNNs can learn levels of equivariance at every layer, are fully compatible with current deep learning components, and are applicable for discrete groups, continuous groups and combinations thereof. We note, however, that Dehmamy et al. [83], Zhou et al. [476] learn the structure of the group from scratch. Contrarily, Partial G-CNNs start from a (very) large group and allows layers in the network to constrain their equivariance levels to better fit the data. Finzi et al. [106] incorporate soft equivariance constraints by combining outputs of equivariant and non-equivariant layers running in parallel, which incurs in large parameter and time costs. Differently, Partial G-CNNs learns from data the level of partial equivariance directly on the group manifold.

## 11.5 Experiments

**Experimental details.** We parameterize all our convolutional kernels as 3-layer SIRENs [358] with 32 hidden units. All our networks –except for the (partial) group equivariant 13-layer CNNs [200] used in Sec. 11.5.1– are constructed with 2 residual blocks of 32 channels each, batch normalization [166] following the structure shown in Fig. 11.3. Here, we intentionally select our networks to be simple as to better assess the effect of partial equivariance. We avoid learning probability distributions on the translation part of the considered groups, and assume all spatial positions to be sampled in order to use fast PyTorch convolution primitives in our implementation. Additional experimental details, e.g., the specific hyperparameters used, as well as complementary results can be found in Appx. J.4, J.5.

**Toy tasks: MNIST6-180 and MNIST6-M.** First, we validate whether Partial G-CNNs can learn partial equivariances. To this end, we construct two illustrative datasets: *MNIST6-180*, and *MNIST6-M*. *MNIST6-180* is constructed by extracting the digits of the class 6 from the MNIST dataset [208], and rotating them on the circle. The goal is to predict whether the number is a six, i.e., a rotation in  $[-90^\circ, 90^\circ]$  was applied, or a nine otherwise. Similarly, we construct *MNIST6-M* by mirroring digits over the y axis. The goal is to predict whether a digit was mirrored or not.

As shown in Tab. 11.1, G-CNNs are unable to solve these tasks as discrimination among group transformations is required. Specifically, SE(2)-CNNs are unable to solve MNIST6-

180, and Mirror-CNNs –G-CNNs equivariant to reflections– are unable to solve MNIST6-M. Furthermore, E(2)-CNNs cannot solve any of the two tasks, because E(2)-CNNs incorporate equivariance to both rotations and reflections. Partial G-CNNs, on the other hand, easily solve both tasks with corresponding base groups. This indicates that Partial G-CNNs learn to adjust the equivariance levels in order to solve the tasks.

In addition, we verify the learned levels of equivariance for a Partial SE(2)-CNN on MNIST6-180. To this end, we plot the probability of assigning the label 6 to test samples of MNIST6-180 rotated on the whole circle. Fig 11.4 shows that the network learns to predict “6” for rotated samples in  $[-90^\circ, 90^\circ]$ , and “9” otherwise. Note that Partial G-CNNs learn the expected levels of partial equivariance without any additional regularization loss terms to encourage them –as required in Benton et al. [25]–.

**Benchmark image datasets.** Next, we validate Partial G-CNNs on classification datasets: RotMNIST [201], CIFAR-10 and CIFAR-100 [195]. Additional results on the PatchCam dataset [404] can be found in Appx. J.5.

We construct Partial G-CNNs with base groups SE(2) and E(2), and varying number of elements used in the Monte Carlo approximation of the group convolution and compare them to equivalent G-CNNs and ResNets (equivalent to T(2)-CNNs). Our results (Tab. 11.3) show that Partial G-CNNs are competitive with G-CNNs when full-equivariance is advantageous (rotated MNIST and PatchCamelyon). However, for tasks in which the data does not naturally exhibit full rotation equivariance (CIFAR-10 and CIFAR-100), Partial G-CNNs consistently outperform fully equivariant G-CNNs.

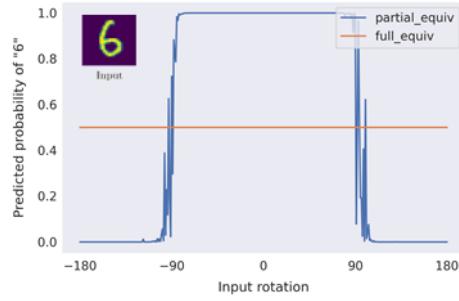
**The need for learning partial layer-wise equivariances.** Next, we evaluate (*i*) the effect of learning partial equivariances instead of soft invariances, and (*ii*) the effect of learning layer-wise levels of equivariances instead of a single level of partial equivariance for the entire network.

For the former, we compare Partial G-CNNs to equivalent ResNets with Augerino [25] (Tab. 11.2). We extend our strategy to learn distributions on discrete groups to the Augerino framework to allow it to handle groups with discrete components, e.g., E(2). For the latter, we construct regular G-CNNs and replace either the final group convolutional layer by a T(2) convolutional layer, or the global max pooling layer at the end by a learnable MLP (Tab. 11.4). If determining the level of equivariance at the end of the network is sufficient, these models should perform comparably to Partial G-CNNs.

Tab. 11.2 shows that Augerino is competitive to Partial G-CNNs on rotated MNIST, but falls behind by a large margin on CIFAR-10 and CIFAR-100. This result can be explained by how these datasets are constructed. RotMNIST is constructed by rotating MNIST digits globally, thus it is not surprising that a model able to encode global invariances can match Partial G-CNNs. The invariance and equivariance relationships in natural images, however, are more complex, as they can be local as well. Consequently, tackling different levels of equivariance at each layer using Partial G-CNNs leads to benefits over

**Table 11.3:** Test image classification results.

| BASE GROUP | NO. ELEMS | PARTIAL EQUIV. | CLASSIFICATION ACCURACY (%) |         |          |
|------------|-----------|----------------|-----------------------------|---------|----------|
|            |           |                | ROTMNIST                    | CIFAR10 | CIFAR100 |
| T(2)       | 1         | -              | 97.23                       | 83.11   | 47.99    |
|            | 4         | ✗              | 99.10                       | 83.73   | 52.35    |
| SE(2)      | 8         | ✗              | 99.17                       | 86.08   | 55.55    |
|            | 8         | ✓              | 99.23                       | 88.59   | 57.26    |
| E(2)       | 16        | ✗              | 99.24                       | 86.59   | 51.55    |
|            | 16        | ✓              | 99.18                       | 89.11   | 57.31    |
| SE(2)      | 8         | ✗              | 98.14                       | 85.55   | 54.29    |
|            | 8         | ✓              | 97.78                       | 89.00   | 55.22    |
| E(2)       | 16        | ✗              | 98.35                       | 88.95   | 57.78    |
|            | 16        | ✓              | 98.58                       | 90.12   | 61.46    |

**Figure 11.4:** Learned equivariances for a “6” on MNIST6-180. Partial G-CNNs become equivariant to rotations on the semi-circle, while G-CNNs are unable to solve the task.

using a single level of global invariance for the entire network.

Although the T(2) and MLP alternatives outlined before could solve the MNIST-180 and MNIST-M toy datasets, we observed that Partial G-CNNs perform consistently better on the visual benchmarks considered (Tab. 11.4). This indicates that learning layer-wise partial equivariances is beneficial over modifying the level of equivariance only at the end of the model. In addition, it is important to highlight that Partial G-CNNs can become fully-, partial-, and non-equivariant during training. Alternative models, on the other hand, are either unable to become fully equivariant (T(2) models) or very unlikely to do so in practice (MLP models).

**SIRENs as group convolution kernels.** Next, we validate SIRENs as parameterization for group convolutional kernels. Tab. J.2 shows that SE(2)-CNNs with SIREN kernels outperform SE(2)-CNNs with ReLU, LeakyReLU and Swish kernels by a large margin on all datasets considered. This result suggests that SIRENs are indeed better suited to represent continuous group convolutional kernels.

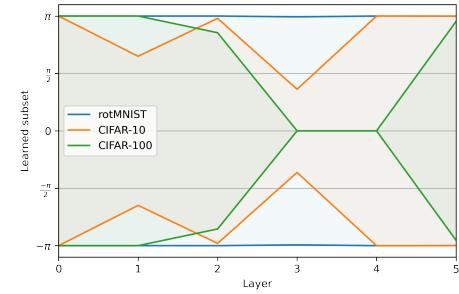
### 11.5.1 Experiments with deeper networks

In addition to the simple networks of the previous experiments, we also explore partial equivariance in a group equivariant version of the 13-layer CNN of Laine and Aila [200]. Specifically, we construct partial group equivariant 13-layer CNNs using SE(2) as base group, and vary the number of elements used in the Monte Carlo approximation of the group convolution. For each number of elements, we compare Partial 13-layer G-CNNs to their fully equivariant counterparts as well as equivalent 13-layer CNNs trained with Augerino. Our results are summarized in Tab. 11.5.

On partial equivariant settings (CIFAR10 and CIFAR100), partial equivariant networks consistently outperform fully equivariant and Augerino networks for all number of ele-

**Table 11.4:** Accuracy of G-CNNs with a MLP instead of global pooling, G-CNNs with a final T(2)-convolutional layer, and Partial G-CNNs.

| BASE GROUP | NO. ELEMS | NET TYPE | CLASSIFICATION ACCURACY (%) |              |              |
|------------|-----------|----------|-----------------------------|--------------|--------------|
|            |           |          | ROTMNIST                    | CIFAR10      | CIFAR100     |
| SE(2)      | 16        | T(2)     | 99.04                       | 82.76        | 52.51        |
|            |           | MLP      | 99.00                       | 86.25        | 56.29        |
|            |           | PARTIAL  | <b>99.18</b>                | <b>87.45</b> | <b>57.31</b> |
| E(2)       | 16        | T(2)     | 97.98                       | 86.68        | 57.61        |
|            |           | MLP      | <b>99.02</b>                | 87.43        | 58.87        |
|            |           | PARTIAL  | 98.58                       | <b>90.12</b> | <b>61.46</b> |



**Figure 11.5:** Example group subsets learned by Partial G-CNNs.

ments used. Interestingly, translation equivariant CNNs outperform CNNs equivariant to SE(2) on CIFAR10 and CIFAR100. This illustrates that overly restricting equivariance constraints can degrade accuracy. In addition, partial equivariant CNNs retain the accuracy of full equivariant networks in fully equivariant settings. By looking at the group subsets learned by partial equivariant networks (Fig 11.6), we corroborate that partial equivariant networks learn to preserve full equivariance if full equivariance is advantageous, and learn to disrupt it otherwise.

## 11.6 Discussion

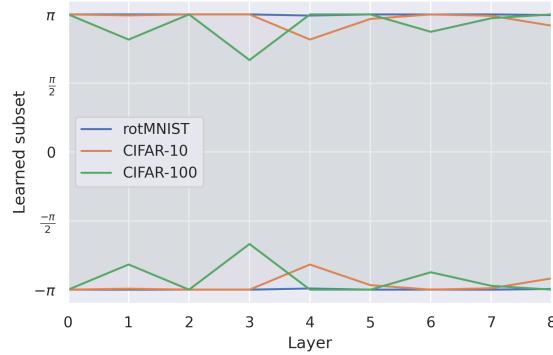
**Memory consumption in partial equivariant settings.** G-CNNs fix the number of samples used to approximate the group convolution prior to training. In Partial G-CNNs we fix a maximum number of samples and adjust the number used at every layer based on the group subset learned. Consequently, a partial group convolution with a learned distribution  $p(u)=\mathcal{U}(\frac{\pi}{2}[-1,1])$  uses half of the elements used in a corresponding group convolution. This reduction in memory and execution time leads to improvements in training and inference time for Partial G-CNNs on partial equivariant settings. We observe reductions up to  $2.5\times$  in execution time and memory on CIFAR-10 and CIFAR-100.

**Sampling per batch element.** In our experiments, we sample once from the learned distribution  $p(u)$  at every layer, and use this sample for all elements in the batch. A better estimation of  $p(u)$  can be obtained by drawing a sample per batch element. Though this method may lead to faster convergence and better estimations of the learned distributions, it comes at a prohibitive memory cost resulting from independent convolutional kernels that must be rendered for each batch element. Consequently, we use a single sample per batch at each layer in our experiments.

**Better kernels with implicit neural representations.** We replace LeakyReLU, ReLU and Swish kernels used so far for continuous group convolution kernels with a SIREN [358]. Our results show that SIRENs are better at modelling group convolutional kernels than existing alternatives.

**Table 11.5:** Classification accuracy with (partial) group equivariant 13-layer CNNs [200].

| BASE GROUP | NO. ELEMNS | PARTIAL EQUIV. | AUGERINO     | CLASSIFICATION ACCURACY (%) |              |              |
|------------|------------|----------------|--------------|-----------------------------|--------------|--------------|
|            |            |                |              | ROT-MNIST                   | CIFAR10      | CIFAR100     |
| T(2)       | 1          | -              | -            | 96.90                       | 91.21        | 67.14        |
|            | 2          | $\times$       | $\times$     | 98.70                       | 85.51        | 62.06        |
|            |            | $\checkmark$   | $\checkmark$ | <b>98.94</b>                | 87.78        | 65.79        |
| SE(2)      | 4          | $\times$       | $\times$     | 98.43                       | 89.73        | 65.97        |
|            |            | $\checkmark$   | $\checkmark$ | <b>98.94</b>                | 91.66        | 68.99        |
|            | 8          | $\times$       | $\times$     | 98.78                       | <b>92.28</b> | <b>69.83</b> |
|            |            | $\checkmark$   | $\checkmark$ | <b>98.54</b>                | 90.55        | 67.70        |

**Figure 11.6:** Group subsets learned by 13-layer Partial G-CNNs.

**Going from a small group subset to a larger one. What does it mean and why is it advantageous?** In Sec. 11.3.3 we described that a partial group convolution can go from a group subset  $\mathcal{S}^{(1)}$  to a larger group subset  $\mathcal{S}^{(2)}$ , e.g., the whole group  $\mathcal{G}$ . Nevertheless, once a layer becomes partially equivariant, subsequent layers cannot become fully equivariant even for  $\mathcal{S}^{(2)}=\mathcal{G}$ . Interestingly, we observe that Partial G-CNNs often learn to disrupt equivariance halfway in the network, and return to the whole group afterwards (Fig. 11.5). As explained below, this behavior is actually advantageous.

Full equivariance restricts group convolutions to apply the same mapping on the entire group. As a result, once the input is transformed, the output remains equal up to the same group transformations. In partial equivariance settings, Partial G-CNNs can output different feature representations for different input transformations. Consequently, Partial G-CNNs can use the group dimension to encode different feature mappings. Specifically, some kernel values are used for some input transformations and other ones are used for other input transformations. This means that when Partial G-CNNs go back to a larger group subset from a smaller one, they are able to use the group axis to encode transformation-dependent features, resulting in increased model expressivity.

In Appx. J.5, we evaluate the effect of enforcing monotonically decreasing group subsets as a function of depth. That is, Partial G-CNNs whose subsets at deeper layers are equal or smaller than those at previous ones. Our results show that this monotonicity leads to

slightly worse results compared to the unconstrained case, thus supporting the use of unconstrained learning of group subsets.

## 11.7 Limitations and future work

**Partial equivariances for other group representations.** The theory of learnable partial equivariances proposed here is only applicable to architectures using regular group representations, e.g., [65, 314]. Nevertheless, other type of representations exist with which exact equivariance to continuous groups can be obtained: irreducible representations [422, 424, 434]. We consider extending the learning of partial equivariances to irreducible representations a valuable extension of our work.

**Unstable training on discrete groups.** Although we can model partial equivariance on discrete groups with our proposed discrete probability distribution parameterization, we observed that these distributions can be unstable to train. To cope with this, we utilize a 10x lower learning rate for the parameters of the probability distributions (See Appx. J.4.3 for details). Nevertheless, finding good ways of learning discrete distributions is an active field of research [151, 152], and advances in this field could be used to further improve the learning of partial equivariances on discrete groups.

**Scaling partial equivariance to large groups.** Arguably the main limitation of G-CNNs with regular representations is their computational and memory complexity, which prevents the use of very large groups, e.g., simultaneous rotation, scaling, mirroring and translations. Partial equivariance is particularly promising for large groups as the network is initialized with a prior towards being equivariant to the entire group, but is able to focus on those relevant to the task at hand. We consider learning partial equivariances on large groups an interesting direction for further research which orthogonal to other advances to scale group convolutions to large groups, e.g., via separable group convolutional kernels [190, 214].

# 12

## Conclusion and Future Work

This dissertation aimed to study the role of continuous modeling and symmetry preservation in improving Deep Learning efficiency among its different efficiency aspects. Our novel contributions affirm that both continuous modeling and symmetry preservation can be used to improve Deep Learning efficiency widely amongst its diverse aspects, while simultaneously improving the state-of-the-art in many cases. Nevertheless, we find that the benefits of symmetry preservation –particularly on data and parameter efficiency– impose compromises in other efficiency aspects, specially on the computational efficiency side. However, as discussed below under *Research Question 2*, it is worth noting that the increased computational costs are tied to interpretation. For convenience, we replicate Tab 1.1 here summarizing the efficiency contribution of each chapter:

|                                      |            | Compute<br>Efficiency | Data<br>Efficiency | Parameter<br>Efficiency | Design<br>Efficiency |
|--------------------------------------|------------|-----------------------|--------------------|-------------------------|----------------------|
| PART I:<br>CONTINUOUS<br>MODELLING   | Chapter 2  | ✓                     | -                  | ✓                       | ✓                    |
|                                      | Chapter 3  | ✓                     | -                  | ✓                       | ✓                    |
|                                      | Chapter 4  | ✓                     | -                  | ✓                       | -                    |
|                                      | Chapter 5  | ✓                     | -                  | -                       | ✓                    |
|                                      | Chapter 6  | ✓                     | -                  | ✓                       | ✓                    |
| PART II:<br>SYMMETRY<br>PRESERVATION | Chapter 7  | -                     | ✓                  | -                       | -                    |
|                                      | Chapter 8  | -                     | ✓                  | -                       | -                    |
|                                      | Chapter 9  | -                     | ✓                  | ✓                       | -                    |
|                                      | Chapter 10 | -                     | ✓                  | ✓                       | -                    |
|                                      | Chapter 11 | ✓                     | ✓                  | ✓                       | ✓                    |

**Research Question 1.** *Can continuous modeling improve Deep Learning Efficiency?  
If so, which specific efficiency aspects does it improve?*

Our study indicates that continuous modeling is indeed a powerful inductive bias capable of bolstering multiple aspects of Deep Learning efficiency.

In Chapter 2, we showed how continuous modeling facilitates the creation of compact global convolutional kernels able to capture global long-term dependencies at a lower time complexity –and thus higher parameter and computational efficiency– than existing global models like Transformers ( $O(N \log N)$  vs.  $O(N^2)$ ). Due to the generality of their formulation, we show in Chapter 3 that the resulting models can be straightforwardly used to process multi-dimensional data as well as inputs of different length and resolution –therefore aiding design efficiency–. Existing models that share these properties are based on self-attention, e.g., Perceivers [168], and thus our proposed general-purpose convolutional architectures offer a more computationally efficient alternative. Furthermore, the use of a continuous parameterization in neural operations enables the treatment of irregularly-sampled data (Chapters 2-4) and the development of resolution-agnostic architectures able to generalize to unseen resolutions (Chapter 5). These properties lead to improved design and computational efficiency improvements.

In Chapter 4, we revealed that continuous modeling enables the creation of point-cloud processing pipelines that substantially improve the efficiency and scalability of native point-cloud methods such as Message Passing Networks [117, 336]. Leveraging the fact that point-cloud are in fact sparse representations of continuous functions, we can rely on continuous modeling to normalize the irregular nature of point-clouds by mapping them onto a regular grid. The result is a substantial gain in computational efficiency and scalability, both in terms of the size of the point-cloud and the size of the neighborhoods considered in their processing.

Finally, we showed that continuous modeling facilitates the automatic tuning of structural network parameters typically treated as hyperparameters: the size of convolutional kernels (Chapter 5) as well as many more structural components like depth, width and downsampling layers (Chapter 6). By conceptualizing neural architectures as continuous entities, we are able to explore a vast multitude of possible network configurations with little computational overhead. This approach sidesteps the inherent limitations of existing Differentiable Neural Architecture methods, e.g., DARTS [231], where the search space must be small, predefined and expensive to explore, leading to substantial improvements in terms of design and compute efficiency.

Collectively, these developments depict the transformative potential of continuous modeling in crafting more efficient deep learning algorithms, and our contributions to it.

**Note about generalization and data-efficiency.** As a final note, we emphasize that our proposed methods oftentimes surpassed the state-of-the-art at the time of publication. We highlight that there exists a correlation between data efficiency and generalization.

However, as this is rather a nuanced relationship, we refrain from categorizing the methods in this part as data efficient.

**Research Question 2.** *Can symmetry preservation improve Deep Learning Efficiency?  
If so, which specific efficiency aspects does it improve?*

Our study indicates that symmetry preservation is indeed a powerful inductive bias capable of bolstering multiple aspects of Deep Learning efficiency, specially in terms of data efficiency. Nevertheless, we find that these improvements impose computationally efficiency compromises.<sup>1</sup>

In Chapters 7 and 8, we proposed the use of neural components that explicitly encourage coherent symmetrical combinations along the stabilizer group and the whole group, respectively. These components result in improved data efficiency and generalization of the underlying symmetry preserving models. In addition, we find that the group equivariant attention maps generated by these neural components also serve other purposes. They can be used to explain decisions taken by the model, and to validate the prediction reliability of the model across symmetrical modifications of the input.

In Chapter 9, we broadened the applicability of symmetry preservation to the popular Transformer architecture. The resulting group equivariant Transformers show a noticeable increase in data and parameter efficiency over vanilla Transformer architectures, comparable to the gains seen in convolutional models. The versatility and generality of group equivariant self-attention has proven useful in diverse applications, notably in the processing and generation of 3D molecular structures [153, 163].

In Chapter 10, we investigated the role of symmetry preservation in the context of time-series processing. We identify two inherent symmetries of time-series: translation and scale, and build neural architectures that respect them. Our findings confirm that the benefits of symmetry preservation carry over to the time-series domain, leading to improved data and parameter efficiency. In addition, we delve into the relationship between scale-translation equivariance and the Wavelet transform, shedding light on the modus-operandi of these networks, and on ways to improve their parameterization.

Finally, we address a key limitation of symmetry-preserving models in Chapter 11: the risk of symmetry misspecification resulting from the need to specify data symmetries prior to training. Typically, symmetry-preserving models require a predefined set of symmetries to adhere to. However, this can lead to overly restrictive models if these symmetries are misspecified or only approximately present in practice. To overcome this issue, we introduce a data-driven approach to dynamically adjust the symmetry

---

<sup>1</sup>It is worth noting that the increase in computational cost is tied to on interpretation. Specifically, one can understand the additional group dimension as additional channels –as in Cohen and Welling [65]. Under this interpretation, the computational cost of group convolutions is lower than that of vanilla convolutions. However, a different interpretation refers to the number of channels as the number of features that are independently learned –analogously to the vanilla convolution case. Under this interpretation, a group convolution is quadratically more expensive than a vanilla convolution (Sec. 12.1.4).

specifications of a model based on the symmetries observed in data. This formulation leads to models with enhanced generalization, data efficiency and design efficiency than conventional symmetry-preserving models. Furthermore, our formulation makes it possible to adapt the number of group samples used during inference based on the level of symmetries observed in data, leading to improved computational efficiency in settings with partial symmetries.

Collectively, these developments depict the transformative potential of symmetry preservation in crafting more efficient deep learning algorithms, and our contributions to it.

## 12.1 Limitations and additional advances

Despite the multiple advances made by this research, there are several important limitations that remain. In this section, we outline these limitations and discuss potential solutions coming both from recent advancements and avenues for future research.

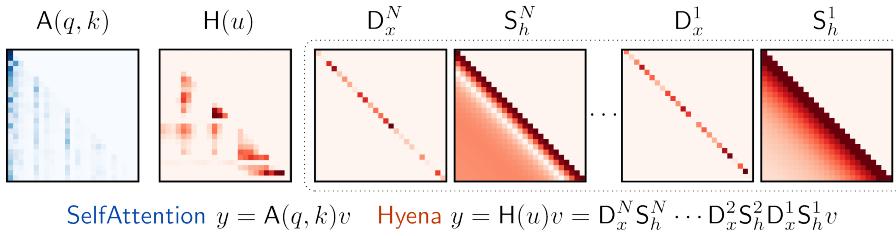
### CONTINUOUS MODELING

#### 12.1.1 Computational implications of global convolutions

Despite the advancements in Chapters 2-6, there is one important overarching limitation that remains: the computational cost of using (global) long convolutional kernels. While the computation efficiency these methods excel in comparison to other global models like Transformers, they have a steeper cost relative to discrete CNNs that rely on local convolutions, which –despite an emerging change of paradigm towards global operations, e.g., self-attention– have traditionally been the prime processing pipeline for multi-dimensional data like images and point-clouds.

It is worth noting that for 1D problems, this computational limitation is not pronounced, as long convolutions can be efficiently computed using the Fast Fourier Transform (FFT) in  $O(N \log N)$  time. Although the complexity of FFT convolutions on multiple dimensions should mirror that of 1D in theory, e.g.,  $O(N^2 \log N^2) = O(N^2 \log N)$  for images of size  $N \times N$ , we find that existing FFT convolution implementations in PyTorch and JAX lag much behind these expectations in practice. As such, computing convolutions on the spatial domain –which have  $O(N^4)$  complexity for an image of size  $N \times N$  and a global convolutional kernel– is typically *faster*. Given the growing interest in long convolutions, developing more efficient CUDA implementation of multidimensional FFT convolutions in Deep Learning frameworks would be very beneficial and impactful. Such implementations will likely result in significant speed-ups to existing methods.

From an algorithmic standpoint, FlexConv (Sec. 5) offers a partial remedy to this limitation by learning the size of convolutional kernels at each layer. However, it is likely –and desirable– for some kernels to remain global. An additional mitigation strategy comes from DNArch (Sec. 6) by learning the resolution on which long convolutions are computed based on data. However, even at lower resolutions, global convolutions can



**Figure 12.1:** Global, input-dependent mappings through the use of interleaved long convolutions and input-dependent gating. Taken from Poli et al. [292].

still be costly. Other potential structural alternatives may involve employing dilated convolutions with learnable spacings, as illustrated by Hassani et al. [139].

Other recent studies offer additional strategies to pare down the computational overhead of (global) long convolutions in ND by means of decomposition and sparsification. Nguyen et al. [272] decomposes ND convolutional kernels into  $N$  1D convolutional kernels along each dimension, leading to important efficiency improvements. Recently, Kirchmeyer and Deng [188] proposed the use of oriented 1D kernels for the processing of complex 2D problems, showing that oriented 1D kernels are able to match architectures that rely on 2D kernels at a much higher speed. Lastly, Liu et al. [234] proposes a low-rank decomposition of convolutional kernels into kernels of smaller sizes whose response can be computed in parallel, e.g.,  $5 \times 5$ ,  $5 \times 51$  and  $51 \times 5$  for a kernel of size  $51 \times 51$ .

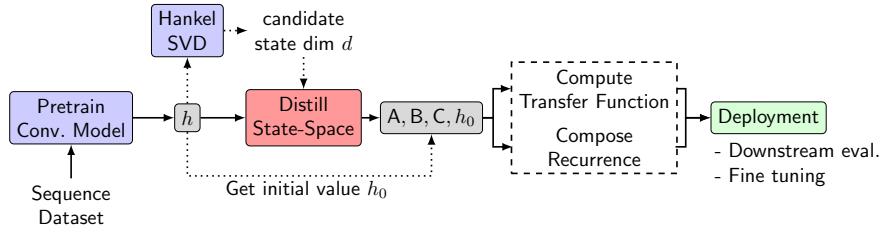
### 12.1.2 Input-dependence in long convolutional models

Although long convolutions are faster than self-attention, they lack a critical capability inherent to self-attention. Self-attention is not only able to capture global context, but it simultaneously is able to generate input-specific mappings –a feature absent in the long convolutional models discussed in this dissertation.

Interestingly, a recent study by Poli et al. [292] demonstrated that long convolutional models, e.g., those based on CKConvs (Ch. 2), can be extended to produce input-dependent mappings. This is achieved through a combination of  $D=2$  interleaved long convolutions and input-depending gating operations. The resulting models are both global and input-dependent with a computational complexity of  $O(DN \log N)$  (Fig. 12.1). Poli et al. [292] revealed that CKConv-based Hyenas match Transformers in intricate language processing tasks, marking them as the first attention-free model able to do so. Moreover, capitalizing on the computational benefits of long convolutions, Nguyen et al. [273] showed that Hyenas can process sequences with 1 million tokens at orders of magnitude faster speed than Transformers, while still considering global context and input-dependence.

### 12.1.3 Autoregressive inference with long convolutional models

When using convolutions, it is necessary to retain the entire history of past activations in memory to predict the next value in the sequence. This inherent characteristic of convo-



**Figure 12.2:** Distillation strategy to transform trained long convolutional networks into recurrent neural networks. Taken from Massaroli et al. [256].

lutional models presents a significant challenge for autoregressive tasks like text generation, as the history grows linearly during the generation process. While this property is not an issue during training –owing to the vectorization across full training sequences– it becomes a notable limitation during inference.

Addressing this challenge, our recent research presented in Massaroli et al. [256] provides a solution. Specifically, we introduce a method with which a trained long convolutional model can be distilled into a Recurrent Neural Network without loss in accuracy by converging the learned model into a State-Space Model representation [47] (Fig. 12.2). Since State-Space Models can be deployed both in convolutional and recurrent form, it is possible to conduct autoregressive inference post-training with constant memory complexity, without compromising on prediction accuracy.

## SYMMETRY PRESERVATION

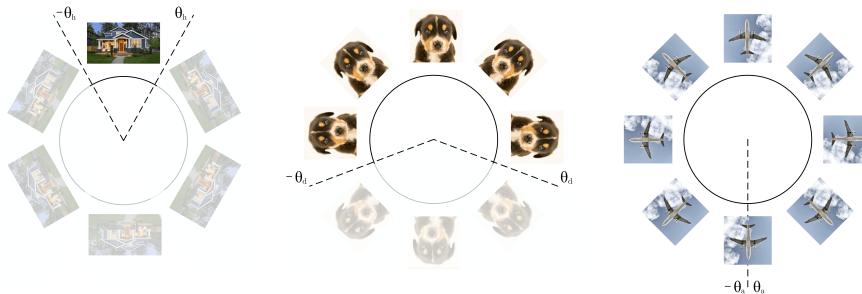
### 12.1.4 Computational implications of group convolutions and large groups

From a computational viewpoint, the main limitation of group convolutions stems from the additional complexity resulting from the use of additional axes to encode the symmetries of the group considered<sup>2</sup>. Specifically, relating back to the seminal G-CNNs of Cohen and Welling [65], given an image of size  $N \times N$ , a finite group with  $G$  elements, and a convolutional kernel of spatial dimensions  $K \times K$ , a group convolution has  $O(N^2 K^2 G^2)$  time complexity, which is  $G^2$  times higher than the complexity of the corresponding standard spatial convolution  $O(N^2 K^2)$ . This increased demand arises from the fact that both the input and the convolutional kernel are defined on the group, which introducing an extra dimension with  $G$  coordinates in this example.

A promising avenue to reduce the complexity of these operations comes from the use of separable group convolutions [190]. In Knigge et al. [190], we show that decomposing group convolutional kernels into spatial and group components substantially reduces the computational demands of group convolutional networks without sacrificing accuracy. This reduction, additionally allows for the creation of neural architectures that are

---

<sup>2</sup>This is the case when using regular group representations, as is done throughout this dissertation.



**Figure 12.3:** In real world scenarios, samples with different semantic meaning may have different symmetrical properties. Considering this input-dependency in the design of symmetry-preserving neural architectures facilitates the construction of more flexible neural architectures that accurately represent the symmetrical properties of data. Taken from Urbano and Romero [391].

equivariant to very large groups, such as the  $\text{Sim}(2)$  group, consisting of 2D translations, scaling and rotation. In addition, we utilize a similar strategy in Bekkers et al. [19] and rely on separability on the 3D space of positions and orientations  $\mathbb{R}^3 \times S^2$ , to device efficient neural architectures for the equivariant processing and generation of molecules.

Another proposing approach leverages Monte-Carlo approximations of group convolutions [105]. It entails approximating group convolutions by sampling different group elements during each forward pass of the network. This technique provides exact equivariance on expectation, and allows the construction of efficient equivariant neural architectures able to consider (infinite) continuous groups with regular group representations. Evidence suggests that using continuous parameterizations akin to CKConv (Ch. 2) for group convolutional kernels leads to enhanced expressivity [190, 316, 475].

It is important to mention that regular group representations are not the only possible alternative. In fact, several works have explored the use of irreducible group representations, which allow for the construction of neural architectures equivariant to continuous groups [41, 66, 422, 434, 475]. However, neural architectures based on irreducible group representations require a yet more intricate design than regular group representation. This imposes additional strong constraints over the kind of operations that can be used, e.g., the type of admissible nonlinearities, and the knowledge required for their construction. Furthermore, several studies indicate that regular representations lead to consistently better results than irreducible representations [41, 198, 422], specially if continuous groups are considered [105]. Based on these observations, we contend that pursuing efficiency improvements for neural architectures built on regular group representations may hold the highest potential for future progress.

### 12.1.5 Symmetry pre-specification in group CNNs

Typically, during the instantiation of group equivariant models, one must predefine the group of symmetries to which the model most be equivariant. This definition denotes

a limiting factor in terms of design efficiency and has the risk of producing overly constrained models the symmetry group is not properly specified [48, 316].

A recent line of work aims to release researchers from the need to define these symmetries prior to training, by inferring them directly from data during training. Existing approaches propose this by means of meta-learning [476], generative modeling [450], neural architecture search [247], probabilistic reasoning [165, 397] and the learning of infinitesimal group generators [83]. This line of work has the potential to be particularly impactful, as it may help uncover the symmetries that complex systems should respect, and release researchers from the need to specify them by hand.

An additional scenario in which the group of symmetries may be misspecified is in cases in which the symmetries encoded in the network are only partially observed in data (Ch. 11). To address this challenge, the concept of partial –or soft– equivariance has been introduced. Partially equivariant models aim to relax the symmetric constraints of a model during training based on the data observed. Existing approaches employ probabilistic methods to learn group subsets based on data [316, 395] or combine equivariant and non-equivariant components in a learnable fashion [106, 414].

Nevertheless, an important challenge persists in both cases: both the notions of equivariant and partial equivariance are inferred at a dataset level instead of at a sample level. This generalized approach is prone to overlooking nuanced symmetrical properties exhibited by individual samples within a single dataset (Fig. 12.3). As a result, such models can only predict the average level of symmetry seen in the dataset. In Urbano and Romero [391], we recently proposed a methodology that enables the extraction of symmetric characteristics –both exact and partial– on a per-sample basis. We achieve this behaviour through the learning of input dependent symmetry distributions that can be efficiently inferred from semantically similar objects in the dataset. We are optimistic about the potential of this approach, as it may empower the development of neural architectures able to consider a more flexible definition of symmetry preservation.

## 12.2 Future Work

### 12.2.1 Exploring other inductive biases for efficiency and beyond

While this dissertation focuses on the role of continuous modeling and symmetry preservation, these are by no means the only inductive biases that serve this purpose. There exist several other inductive biases that also hold substantial promise for improving the efficiency of deep learning algorithms. For example, sparsity and causality.

Sparsity pertains to the selective activation and utilization of neurons within a neural network, offering significant advantages for reducing the computational costs of neural networks during training and deployment [207]. It aims at constructing compressed representations where only the most critical features and connections are preserved, thereby reducing the model’s parameter, memory and computational complexity. Traditionally,

sparsity has been primarily used post-training via *pruning*, to distil a trained network into a smaller equally effective sub-network [137]. Recent studies have shown that sparsity can also be effectively introduced early in training, as illustrated in the PhD theses by Frankle [109] and Lee [212]. Sparsity holds great potential for the development of more compute- and parameter-efficient neural architectures.

Causality, on the other hand, pertains the structuring of models and algorithms around the concept of cause-and effect relationships between variables, thereby holding the potential to improve the data, compute and parameter efficiency of deep learning models [284, 288]. Unlike traditional correlation-based approaches, which merely identify correlating patterns in the data, causal approaches strive to capture and understand the underlying data-generating mechanisms. As a result, causality-informed models typically exhibit simpler structures that require less training data, and are less prone to overfitting [341, 351]. Beyond efficiency gains, causality may also improve the explainability of the resulting models by interpreting the data-generating process captured [309]. Causality holds considerable promise not only for the development of more efficient neural architectures, but also for increasing the reliability and applicability of neural networks.

We consider investigating the use of inductive biases for the development of Deep Learning architectures as a promising avenue for further research. Notably, inductive biases not only have implications for efficiency, but also hold value for the development of more robust, reliable, generalizable and explainable Deep Learning methods.

### 12.2.2 Future work on continuous modeling and symmetry preservation

Despite the progress in continuous modeling and symmetry preservation made in this dissertation, numerous open questions and avenues for future research remain. We conclude this dissertation highlighting research directions we consider most promising as a complement to addressing the limitations presented in Sec. 12.1.

#### Better parameterizations of continuous spatial functions

In this work, we have investigated several parameterization of spatial continuous functions such as continuous convolutional kernels. Specifically, we proposed parameterization based on multiple Neural Field families including SIRENs [358] (Ch. 2), RFNets [377] (Ch. 3, 4) and MFNs [104] (Ch. 5). However, numerous new families have emerged since the inception of this research, which may improve the parameterizations introduced herein, e.g., Müller et al. [267], Saragadam et al. [335], Wu et al. [437], Yang [451].

However, it is important to highlight that the way in which Neural Fields are used in this dissertation differs dramatically from the standard problem considered in Neural Field's literature. Specifically, we utilize Neural Fields to parameterize neural operations for which the ground truth is *unknown and is constantly changing*. This is in contrast to conventional Neural Field's literature, where Neural Fields map to *known and fixed*. Consequently, insights from Neural Field literature do not necessarily extrapolate to the

scenarios considered in this dissertation, and Neural Fields that perform well in Neural Field literature will not necessarily work well in this setting.

For instance, given that the “ground truth” of continuous neural operations are constantly changing during training, the speed at which a Neural Field comes to a rough approximation of this ground truth could be more valuable than its ability to fit this ground truth to perfection after several iterations. Additionally, certain properties deemed peripheral in Neural Fields literature may hold important advantages for the applications here considered. For example, the ability of MFNs [104] to provide analytic control over the frequency components of the approximation is of outmost importance for tasks like zero-shot predictions at higher resolutions (Ch. 5). Therefore, even though the fitting capabilities of MFNs have been surpassed by newer methods like InstantNGP [267], this unique feature makes MFNs an incredibly compelling candidate for future exploration.

In summary, we deem a systematic evaluation of recent Neural Field parameterizations in the context of this dissertation to be a critical avenue for future research. Given the increasing interest in long convolutions parameterized by Neural Fields [184, 273, 275, 292, 402, 481], such an investigation has the potential to have significant impact across multiple applications and disciplines.

### **From self-attention to (input-dependent) long convolutional models**

While self-attention excels in handling global dependencies, its computational overhead is a significant, increasingly prohibitive drawback. Recent research indicates that input-dependent long convolutional models can achieve comparable or even superior performance than Transformers, but with greater computational efficiency and scalability [273, 292, 481]. Further exploring the application of CKConvs [321] and Hyenas [292] across applications where Transformers are currently prevalent offers the prospect of improving both efficiency and versatility. In addition, the increased computational efficiency of long convolutional models could open new horizons in computationally-intensive domains such as those involving truly long contexts. For example, for the construction of foundational audio, time-series and video models.<sup>3,4</sup>

Unlike self-attention, convolutional operations are translation preserving. Since translation is a symmetry that is ubiquitously present in natural data, the use of translation preserving operations for the processing of natural data, e.g., time-series, audio, video, may lead –as observed in this dissertation– to a significant reduction in the model size and data requirements for training. In other words, this property may hold large potential for the scalability, generalizability and applicability of long convolutional models.

---

<sup>3</sup>In contrast to text, where a 32.000-token context encompasses a very long piece of text, a single second of audio easily contains 48.000 samples. This makes evident the scale of context foundational models on this kind of data should be able to handle.

<sup>4</sup>It is worth noting that long convolutional models can also be distilled into RNNs during inference to achieve constant memory costs in autoregressive generation (see Sec. 12.1.3, Massaroli et al. [256]). This is well below the linear cost of (cached) causal Transformers.

Lastly, the only method currently known for the implementation of input-dependency in long convolutional models is the one presented in Poli et al. [292], which involves an interleaved combination of input-dependent gating and long convolutions. Investigating alternative techniques could offer additional avenues for improvement. Since long convolutional models often rely on Neural Fields to parameterize their kernels, we consider the use of Neural Field conditioning techniques, e.g., Jiang et al. [172], Park et al. [280], Perez et al. [287], an interesting direction for future research.

### Extending group convolutions to very large groups

As discussed in Sec. 12.1.4, one of the main limitations of group convolutions lies in their computational cost, which scales quadratically with the size of the group considered. This is particularly challenging for large complex group such as the projective 2D group:  $\text{Proj}(2)$ , and several other groups that act in 3D, e.g.,  $\text{Sim}(3)$ : the group consisting of 3D rotations, 3D translations and homogeneous scaling. Nevertheless, considering equivariance to these larger groups offer rich structural prior information that could be invaluable for emerging applications, such as point-cloud processing and mesh generation, particularly for applications on which train data is scarce. Devising techniques that make the handling of such large groups manageable, e.g., through group decompositions [19, 190] and Monte-Carlo approximations [105, 316], holds great promise for the development of more data-efficient Deep Learning methods.



## Bibliography

- [1] Ossama Abdel-Hamid, Abdel-rahman Mohamed, Hui Jiang, Li Deng, Gerald Penn, and Dong Yu. Convolutional neural networks for speech recognition. *IEEE/ACM Transactions on audio, speech, and language processing*, 22(10):1533–1545, 2014.
- [2] Sajjad Abdoli, Patrick Cardinal, and Alessandro Lameiras Koerich. End-to-end environmental sound classification using a 1d convolutional neural network. *Expert Systems with Applications*, 136:252–263, 2019.
- [3] Krister Åhlander and Hans Munthe-Kaas. Applications of the generalized fourier transform in numerical linear algebra. *BIT Numerical Mathematics*, 45(4):819–850, 2005.
- [4] Ferran Alet, Dylan Doblar, Allan Zhou, Josh Tenenbaum, Kenji Kawaguchi, and Chelsea Finn. Noether networks: meta-learning useful conserved quantities. *Advances in Neural Information Processing Systems*, 34, 2021.
- [5] SH Alkarni. Statistical applications for equivariant matrices. *International Journal of Mathematics and Mathematical Sciences*, 25(1):53–61, 2001.
- [6] Joakim Andén and Stéphane Mallat. Deep scattering spectrum. *IEEE Transactions on Signal Processing*, 62(16):4114–4128, 2014.
- [7] Martin Arjovsky, Amar Shah, and Yoshua Bengio. Unitary evolution recurrent neural networks. In *International Conference on Machine Learning*, pages 1120–1128, 2016.
- [8] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.
- [9] Anthony Bagnall, Hoang Anh Dau, Jason Lines, Michael Flynn, James Large, Aaron Bostrom, Paul Southam, and Eamonn Keogh. The uea multivariate time series classification archive, 2018. *arXiv preprint arXiv:1811.00075*, 2018.
- [10] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In Yoshua Bengio and Yann Le-Cun, editors, *3rd International Conference on Learning Representations, ICLR 2015*,

- San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings, 2015.* URL <http://arxiv.org/abs/1409.0473>.
- [11] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018.
  - [12] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. Trellis networks for sequence modeling. *arXiv preprint arXiv:1810.06682*, 2018.
  - [13] Randall Balestriero and Richard Baraniuk. Mad max: Affine spline insights into deep learning. *arXiv preprint arXiv:1805.06576*, 2018.
  - [14] C Bradford Barber, David P Dobkin, and Hannu Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software (TOMS)*, 22(4):469–483, 1996.
  - [15] Ivana Bartoletti. Ai in healthcare: Ethical and privacy challenges. In *Artificial Intelligence in Medicine: 17th Conference on Artificial Intelligence in Medicine, AIME 2019, Poznan, Poland, June 26–29, 2019, Proceedings 17*, pages 7–10. Springer, 2019.
  - [16] Babak Ehteshami Bejnordi, Mitko Veta, Paul Johannes Van Diest, Bram Van Ginneken, Nico Karssemeijer, Geert Litjens, Jeroen AWM Van Der Laak, Meyke Hermsen, Quirine F Manson, Maschenka Balkenhol, et al. Diagnostic assessment of deep learning algorithms for detection of lymph node metastases in women with breast cancer. *Jama*, 318(22):2199–2210, 2017.
  - [17] Erik J Bekkers. B-spline {cnn}s on lie groups. In *International Conference on Learning Representations*, 2020. URL <https://openreview.net/forum?id=H1gBhkBFDH>.
  - [18] Erik J Bekkers, Maxime W Lafarge, Mitko Veta, Koen AJ Eppenhof, Josien PW Pluim, and Remco Duits. Roto-translation covariant convolutional networks for medical image analysis. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 440–448. Springer, 2018.
  - [19] Erik J Bekkers, Sharvaree Vadgama, Rob D Hesselink, Putri A van der Linden, and David W Romero. Fast, expressive se ( $n$ ) equivariant networks through weight-sharing in position-orientation space. *arXiv preprint arXiv:2310.02970*, 2023.
  - [20] Irwan Bello, Barret Zoph, Ashish Vaswani, Jonathon Shlens, and Quoc V Le. Attention augmented convolutional networks. *arXiv preprint arXiv:1904.09925*, 2019.
  - [21] Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. Understanding and simplifying one-shot architecture search. In *International conference on machine learning*, pages 550–559. PMLR, 2018.
  - [22] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term depen-

- dencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.
- [23] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.
  - [24] Hadjer Benmeziane, Kaoutar El Maghraoui, Hamza Ouarnoughi, Smail Niar, Martin Wistuba, and Naigang Wang. A comprehensive survey on hardware-aware neural architecture search. *arXiv preprint arXiv:2101.09336*, 2021.
  - [25] Gregory Benton, Marc Finzi, Pavel Izmailov, and Andrew G Wilson. Learning invariances in neural networks from training data. *Advances in Neural Information Processing Systems*, 33:17605–17616, 2020.
  - [26] Kaifeng Bi, Changping Hu, Lingxi Xie, Xin Chen, Longhui Wei, and Qi Tian. Stabilizing darts with amended gradient estimation on architectural parameters. *arXiv preprint arXiv:1910.11831*, 2019.
  - [27] Gavin M Bidelman and Ameenuddin Syed Khaja. Spectrotemporal resolution tradeoff in auditory processing as revealed by human auditory brainstem responses and psychophysical indices. *Neuroscience letters*, 572:53–57, 2014.
  - [28] Irving Biederman. Recognition-by-components: a theory of human image understanding. *Psychological review*, 94(2):115, 1987.
  - [29] Lukas Biewald. Experiment tracking with weights and biases, 2020. URL <https://www.wandb.com/>. Software available from wandb.com.
  - [30] Randolph Blake and Sang-Hun Lee. The role of temporal structure in human vision. *Behavioral and cognitive neuroscience reviews*, 4(1):21–42, 2005.
  - [31] Alexander Bogatskiy, Brandon Anderson, Jan Offermann, Marwah Roussi, David Miller, and Risi Kondor. Lorentz group equivariant neural network for particle physics. In *International Conference on Machine Learning*, pages 992–1002. PMLR, 2020.
  - [32] Andrew Brock, Theodore Lim, James M Ritchie, and Nick Weston. Smash: one-shot model architecture search through hypernetworks. *arXiv preprint arXiv:1708.05344*, 2017.
  - [33] Richard M Brooks and Alvin G Goldstein. Recognition by children of inverted photos of faces. *Child Development*, 1963.
  - [34] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

- [35] Vicki Bruce and Glyn W Humphreys. Recognizing objects and faces. *Visual cognition*, 1(2-3):141–180, 1994.
- [36] Joan Bruna and Stéphane Mallat. Invariant scattering convolution networks. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1872–1886, 2013.
- [37] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332*, 2018.
- [38] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment. *arXiv preprint arXiv:1908.09791*, 2019.
- [39] Yue Cao, Jiarui Xu, Stephen Lin, Fangyun Wei, and Han Hu. Gcnet: Non-local networks meet squeeze-excitation networks and beyond. *arXiv preprint arXiv:1904.11492*, 2019.
- [40] Ernst Cassirer. The concept of group and the theory of perception. *Philosophy and phenomenological research*, 5(1):1–36, 1944.
- [41] Gabriele Cesa, Leon Lang, and Maurice Weiler. A program to build e (n)-equivariant steerable cnns. In *International Conference on Learning Representations*, 2021.
- [42] Bo Chang, Minmin Chen, Eldad Haber, and Ed H Chi. Antisymmetri-crnn: A dynamical system view on recurrent neural networks. *arXiv preprint arXiv:1902.09689*, 2019.
- [43] Oscar Chang, Lampros Flokas, and Hod Lipson. Principled weight initialization for hypernetworks. In *International Conference on Learning Representations*, 2020. URL <https://openreview.net/forum?id=H11ma24tPB>.
- [44] Shiyu Chang, Yang Zhang, Wei Han, Mo Yu, Xiaoxiao Guo, Wei Tan, Xiaodong Cui, Michael Witbrock, Mark A Hasegawa-Johnson, and Thomas S Huang. Dilated recurrent neural networks. In *Advances in neural information processing systems*, pages 77–87, 2017.
- [45] Evangelos Chatzipantazis, Stefanos Pertigkiozoglou, Edgar Dobriban, and Kostas Daniilidis. Learning augmentation distributions using transformed risk minimization. *arXiv preprint arXiv:2111.08190*, 2021.
- [46] Zhengping Che, Sanjay Purushotham, Kyunghyun Cho, David Sontag, and Yan Liu. Recurrent neural networks for multivariate time series with missing values. *Scientific reports*, 8(1):1–12, 2018.
- [47] Chi-Tsong Chen. *Linear system theory and design*. Saunders college publishing, 1984.
- [48] Shuxiao Chen, Edgar Dobriban, and Jane H Lee. A group-theoretic framework for data augmentation. *Journal of Machine Learning Research*, 21(245):1–71, 2020.

- [49] Xiangning Chen and Cho-Jui Hsieh. Stabilizing differentiable architecture search via perturbation-based regularization. In *International conference on machine learning*, pages 1554–1565. PMLR, 2020.
- [50] Xin Chen, Lingxi Xie, Jun Wu, and Qi Tian. Progressive darts: Bridging the optimization gap for nas in the wild. *International Journal of Computer Vision*, 129:638–655, 2021.
- [51] Yunpeng Chen, Marcus Rohrbach, Zhicheng Yan, Yan Shuicheng, Jiashi Feng, and Yannis Kalantidis. Graph-based global reasoning networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 433–442, 2019.
- [52] Jianpeng Cheng, Li Dong, and Mirella Lapata. Long short-term memory-networks for machine reading. *arXiv preprint arXiv:1601.06733*, 2016.
- [53] Xiuyuan Cheng, Qiang Qiu, Robert Calderbank, and Guillermo Sapiro. Rotdcf: Decomposition of convolutional filters for rotation-equivariant deep networks. *arXiv preprint arXiv:1805.06846*, 2018.
- [54] E Colin Cherry. Some experiments on the recognition of speech, with one and with two ears. *The Journal of the acoustical society of America*, 25(5):975–979, 1953.
- [55] Narsimha Chilkuri and Chris Eliasmith. Parallelizing legendre memory unit training. *arXiv preprint arXiv:2102.11417*, 2021.
- [56] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.
- [57] Keunwoo Choi, George Fazekas, and Mark Sandler. Automatic tagging using deep convolutional neural networks. *arXiv preprint arXiv:1606.00298*, 2016.
- [58] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1251–1258, 2017.
- [59] Krzysztof Choromanski, Valerii Likhoshesterov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, et al. Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*, 2020.
- [60] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.
- [61] Patryk Chrabaszcz, Ilya Loshchilov, and Frank Hutter. A downsampled variant

- of imagenet as an alternative to the CIFAR datasets. *CoRR*, abs/1707.08819, 2017.  
URL <http://arxiv.org/abs/1707.08819>.
- [62] Xiangxiang Chu, Xiaoxing Wang, Bo Zhang, Shun Lu, Xiaolin Wei, and Junchi Yan. Darts-: robustly stepping out of performance collapse without indicators. *arXiv preprint arXiv:2009.01027*, 2020.
  - [63] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.
  - [64] Adam Coates, Andrew Ng, and Honglak Lee. An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 215–223. JMLR Workshop and Conference Proceedings, 2011.
  - [65] Taco Cohen and Max Welling. Group equivariant convolutional networks. In *International conference on machine learning*, pages 2990–2999. PMLR, 2016.
  - [66] Taco S Cohen and Max Welling. Steerable cnns. *arXiv preprint arXiv:1612.08498*, 2016.
  - [67] Taco S Cohen, Mario Geiger, Jonas Köhler, and Max Welling. Spherical cnns. In *International Conference on Learning Representations*, 2018.
  - [68] Taco S Cohen, Mario Geiger, and Maurice Weiler. A general theory of equivariant cnns on homogeneous spaces. In *Advances in Neural Information Processing Systems*, pages 9142–9153, 2019.
  - [69] Taco S Cohen, Maurice Weiler, Berkay Kicanaoglu, and Max Welling. Gauge equivariant convolutional networks and the icosahedral cnn. *arXiv preprint arXiv:1902.04615*, 2019.
  - [70] Alexis Conneau, Holger Schwenk, Loïc Barrault, and Yann Lecun. Very deep convolutional networks for text classification. *arXiv preprint arXiv:1606.01781*, 2016.
  - [71] Michael Connor and Piyush Kumar. Fast construction of k-nearest neighbor graphs for point clouds. *IEEE transactions on visualization and computer graphics*, 16(4):599–608, 2010.
  - [72] Jean-Baptiste Cordonnier, Andreas Loukas, and Martin Jaggi. On the relationship between self-attention and convolutional layers. In *International Conference on Learning Representations*, 2020. URL <https://openreview.net/forum?id=HJlnC1rKPB>.
  - [73] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 303–312, 1996.

- [74] Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. Deformable convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pages 764–773, 2017.
- [75] Wei Dai, Chia Dai, Shuhui Qu, Juncheng Li, and Samarjit Das. Very deep convolutional neural networks for raw waveforms. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 421–425. IEEE, 2017.
- [76] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019.
- [77] Kent Dallekett, Sandra G Wilcox, and Lester D'andrea. Picture memory experiments. *Journal of Experimental Psychology*, 76(2p1):312, 1968.
- [78] Ingrid Daubechies. *Fundamental papers in wavelet theory*. Princeton University Press, 2006.
- [79] J.G. Daugman. Complete discrete 2-d gabor transforms by neural networks for image analysis and compression. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 36(7):1169–1179, 1988. doi: 10.1109/29.1644.
- [80] Yann N Dauphin, Angela Fan, Michael Auli, and David Grangier. Language modeling with gated convolutional networks. In *International conference on machine learning*, pages 933–941, 2017.
- [81] Edward De Brouwer, Jaak Simm, Adam Arany, and Yves Moreau. Gru-ode-bayes: Continuous modeling of sporadically-observed time series. In *Advances in Neural Information Processing Systems*, pages 7379–7390, 2019.
- [82] Michaël Defferrard, Martino Milani, Frédéric Gusset, and Nathanaël Perraudin. Deepsphere: a graph-based spherical cnn. *arXiv preprint arXiv:2012.15000*, 2020.
- [83] Nima Dehmamy, Robin Walters, Yanchen Liu, Dashun Wang, and Rose Yu. Automatic symmetry discovery with lie algebra convolutional network. *Advances in Neural Information Processing Systems*, 34, 2021.
- [84] Charles B Delahunt and J Nathan Kutz. Insect cyborgs: Bio-mimetic feature generators improve ml accuracy on limited data. 2019.
- [85] Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. *Advances in Neural Information Processing Systems*, 34:8780–8794, 2021.
- [86] Nichita Diaconu and Daniel E Worrall. Affine self convolution. *arXiv preprint arXiv:1911.07704*, 2019.
- [87] Nichita Diaconu and Daniel E Worrall. Learning to convolve: A generalized weight-tying approach. 2019.

- [88] Sander Dieleman and Benjamin Schrauwen. End-to-end learning for music audio. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6964–6968. IEEE, 2014.
- [89] Sander Dieleman, Jeffrey De Fauw, and Koray Kavukcuoglu. Exploiting cyclic symmetry in convolutional neural networks. In *International conference on machine learning*, pages 1889–1898. PMLR, 2016.
- [90] Xuanyi Dong and Yi Yang. Searching for a robust neural architecture in four gpu hours. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1761–1770, 2019.
- [91] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.
- [92] Lukasz Dudziak, Thomas Chau, Mohamed Abdelfattah, Roysen Lee, Hyeji Kim, and Nicholas Lane. Brp-nas: Prediction-based nas using gcns. *Advances in Neural Information Processing Systems*, 33:10480–10490, 2020.
- [93] David Steven Dummit and Richard M Foote. *Abstract algebra*, volume 3. Wiley Hoboken, 2004.
- [94] Emilien Dupont, Yee Whye Teh, and Arnaud Doucet. Generative models as distributions of functions. *arXiv preprint arXiv:2102.04776*, 2021.
- [95] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Efficient multi-objective neural architecture search via lamarckian evolution. *arXiv preprint arXiv:1804.09081*, 2018.
- [96] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *The Journal of Machine Learning Research*, 20(1):1997–2017, 2019.
- [97] N Benjamin Erichson, Omri Azencot, Alejandro Queiruga, Liam Hodgkinson, and Michael W Mahoney. Lipschitz recurrent neural networks. *arXiv preprint arXiv:2006.12070*, 2020.
- [98] Carlos Esteves, Avneesh Sud, Zhengyi Luo, Kostas Daniilidis, and Ameesh Makadia. Cross-domain 3d equivariant image embeddings. In *International Conference on Machine Learning*, pages 1812–1822. PMLR, 2019.
- [99] Carlos Esteves, Yinshuang Xu, Christine Allen-Blanchette, and Kostas Daniilidis. Equivariant multi-view networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1568–1577, 2019.
- [100] Carlos Esteves, Ameesh Makadia, and Kostas Daniilidis. Spin-weighted spherical cnns. *Advances in Neural Information Processing Systems*, 33, 2020.

- [101] William Falcon et al. Pytorch lightning. *GitHub*. Note: <https://github.com/PyTorchLightning/pytorch-lightning>, 3, 2019.
- [102] Luca Falorsi, Pim de Haan, Tim R Davidson, and Patrick Forré. Reparameterizing distributions on lie groups. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 3244–3253. PMLR, 2019.
- [103] Jiemin Fang, Yuzhu Sun, Qian Zhang, Yuan Li, Wenyu Liu, and Xinggang Wang. Densely connected search space for more flexible neural architecture search. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10628–10637, 2020.
- [104] Rizal Fathony, Anit Kumar Sahu, Devin Willmott, and J Zico Kolter. Multiplicative filter networks. In *International Conference on Learning Representations*, 2021. URL <https://openreview.net/forum?id=OmtmcPkhhT>.
- [105] Marc Finzi, Samuel Stanton, Pavel Izmailov, and Andrew Gordon Wilson. Generalizing convolutional neural networks for equivariance to lie groups on arbitrary continuous data. *arXiv preprint arXiv:2002.12880*, 2020.
- [106] Marc Finzi, Gregory Benton, and Andrew G Wilson. Residual pathway priors for soft equivariance constraints. *Advances in Neural Information Processing Systems*, 34, 2021.
- [107] Marc Finzi, Max Welling, and Andrew Gordon Wilson. A practical method for constructing equivariant multilayer perceptrons for arbitrary matrix groups. In *International Conference on Machine Learning*, pages 3318–3328. PMLR, 2021.
- [108] J Fourier. Mémoire sur la propagation de la chaleur dans les corps solides, présenté le 21 décembre 1807 à l’institut national—nouveau bulletin des sciences par la société philomatique de paris. i. In *Paris: First European Conference on Signal Analysis and Prediction*, pages 17–21, 1807.
- [109] Jonathan Frankle. *The Lottery Ticket Hypothesis: On Sparse, Trainable Neural Networks*. PhD thesis, Massachusetts Institute of Technology, 2023.
- [110] Alejo Freire, Kang Lee, and Lawrence A Symons. The face-inversion effect as a deficit in the encoding of configural information: Direct evidence. *Perception*, 29(2):159–170, 2000.
- [111] Fabian B Fuchs, Daniel E Worrall, Volker Fischer, and Max Welling. Se (3)-transformers: 3d roto-translation equivariant attention networks. *arXiv preprint arXiv:2006.10503*, 2020.
- [112] Kunihiko Fukushima and Sei Miyake. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets*, pages 267–285. Springer, 1982.

- [113] Sadaoki Furui. Speaker-independent isolated word recognition based on emphasized spectral dynamics. In *ICASSP'86. IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 11, pages 1991–1994. IEEE, 1986.
- [114] Dennis Gabor. Theory of communication. part 1: The analysis of information. *Journal of the Institution of Electrical Engineers-Part III: Radio and Communication Engineering*, 93(26):429–441, 1946.
- [115] Robert Gens and Pedro M Domingos. Deep symmetry networks. In *Advances in neural information processing systems*, pages 2537–2545, 2014.
- [116] Amir Ghodrati, Babak Ehteshami Bejnordi, and Amirhossein Habibian. Frame-exit: Conditional early exiting for efficient video recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15608–15618, 2021.
- [117] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR, 2017.
- [118] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.
- [119] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.
- [120] Karan Goel, Albert Gu, Chris Donahue, and Christopher Ré. It’s raw! audio generation with state-space models. In *International Conference on Machine Learning*, pages 7616–7633. PMLR, 2022.
- [121] Ary L Goldberger, Luis AN Amaral, Leon Glass, Jeffrey M Hausdorff, Plamen Ch Ivanov, Roger G Mark, Joseph E Mietus, George B Moody, Chung-Kang Peng, and H Eugene Stanley. Physiobank, physiotoolkit, and physionet: components of a new research resource for complex physiologic signals. *circulation*, 101(23):e215–e220, 2000.
- [122] Zhangxiaowen Gong. *Exploiting and coping with sparsity to accelerate DNNs on CPUs*. PhD thesis, 2021.
- [123] Simon Graham, David Epstein, and Nasir Rajpoot. Dense steerable filter cnns for exploiting rotational symmetry in histology images. *arXiv preprint arXiv:2004.03037*, 2020.
- [124] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. Ieee, 2013.

- [125] Alex Grossmann, Jean Morlet, and T Paul. Transforms associated to square integrable group representations. i. general results. *Journal of Mathematical Physics*, 26(10):2473–2479, 1985.
- [126] Albert Gu, Tri Dao, Stefano Ermon, Atri Rudra, and Christopher Ré. Hippo: Recurrent memory with optimal polynomial projections. *arXiv preprint arXiv:2008.07669*, 2020.
- [127] Albert Gu, Caglar Gulcehre, Thomas Paine, Matt Hoffman, and Razvan Pascanu. Improving the gating mechanism of recurrent neural networks. In *International Conference on Machine Learning*, pages 3800–3809. PMLR, 2020.
- [128] Albert Gu, Isys Johnson, Karan Goel, Khaled Saab, Tri Dao, Atri Rudra, and Christopher Ré. Combining recurrent, convolutional, and continuous-time models with linear state space layers. *Advances in Neural Information Processing Systems*, 34, 2021.
- [129] Albert Gu, Karan Goel, and Christopher Re. Efficiently modeling long sequences with structured state spaces. In *International Conference on Learning Representations*, 2022. URL <https://openreview.net/forum?id=uYLFoz1vlAC>.
- [130] Albert Gu, Ankit Gupta, Karan Goel, and Christopher Ré. On the parameterization and initialization of diagonal state space models. *arXiv preprint arXiv:2206.11893*, 2022.
- [131] Julia Guerrero-Viu, Sven Hauns, Sergio Izquierdo, Guilherme Miotto, Simon Schrodi, Andre Biedenkapp, Thomas Elsken, Difan Deng, Marius Lindauer, and Frank Hutter. Bag of baselines for multi-objective joint neural architecture search and hyperparameter optimization. *arXiv preprint arXiv:2105.01015*, 2021.
- [132] Li Guolin, Zhang Xing, Wang Zitong, Li Zhenguo, and Zhang Tong. Stacnas: Towards stable and consistent optimization for differentiable neural architecture search. 2019.
- [133] Eric Guizzo, Tillman Weyde, and Jack Barnett Leveson. Multi-time-scale convolution for emotion recognition from speech audio signals. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6489–6493. IEEE, 2020.
- [134] David Ha, Andrew M. Dai, and Quoc V. Le. Hypernetworks. In *International Conference on Learning Representations*, 2017. URL <https://openreview.net/forum?id=rkpACe1lx>.
- [135] Anders Hald. De moivre’s normal approximation to the binomial, 1733, and its generalization. *A History of Parametric Statistical Inference from Bernoulli to Fisher, 1713–1935*, pages 17–24, 2007.
- [136] Shizhong Han, Zibo Meng, Zhiyuan Li, James O'Reilly, Jie Cai, Xiaofeng Wang, and Yan Tong. Optimizing filter size in convolutional neural networks for facial

- action unit recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.
- [137] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28, 2015.
  - [138] Boris Hanin and David Rolnick. Complexity of linear regions in deep networks. *arXiv preprint arXiv:1901.09021*, 2019.
  - [139] Ismail Khalfaoui Hassani, Thomas Pellegrini, and Timothée Masquelier. Dilated convolution with learnable spacings. In *The Eleventh International Conference on Learning Representations*, 2023. URL <https://openreview.net/forum?id=Q3-1vRh3HOA>.
  - [140] Ryuichiro Hataya, Jan Zdenek, Kazuki Yoshizoe, and Hideki Nakayama. Faster autoaugment: Learning augmentation strategies using backpropagation. In *European Conference on Computer Vision*, pages 1–16. Springer, 2020.
  - [141] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.
  - [142] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*, 37(9):1904–1916, 2015.
  - [143] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
  - [144] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.
  - [145] Mary Henle. An experimental investigation of past experience as a determinant of visual form perception. *Journal of Experimental Psychology*, 30(1):1, 1942.
  - [146] Michael J Hewitt and Ray Meddis. A computer model of amplitude-modulation sensitivity of single units in the inferior colliculus. *The Journal of the Acoustical Society of America*, 95(4):2145–2159, 1994.
  - [147] Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München*, 91(1), 1991.
  - [148] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
  - [149] Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. Sparsity in deep learning: Pruning and growth for efficient inference and training

- in neural networks. *The Journal of Machine Learning Research*, 22(1):10882–11005, 2021.
- [150] Emiel Hoogeboom, Jorn W.T. Peters, Taco S. Cohen, and Max Welling. Hexaconv. In *International Conference on Learning Representations*, 2018. URL <https://openreview.net/forum?id=r1vuQG-CW>.
- [151] Emiel Hoogeboom, Jorn Peters, Rianne Van Den Berg, and Max Welling. Integer discrete flows and lossless compression. *Advances in Neural Information Processing Systems*, 32, 2019.
- [152] Emiel Hoogeboom, Didrik Nielsen, Priyank Jaini, Patrick Forré, and Max Welling. Argmax flows and multinomial diffusion: Towards non-autoregressive language models. *arXiv preprint arXiv:2102.05379*, 2021.
- [153] Emiel Hoogeboom, Victor Garcia Satorras, Clément Vignac, and Max Welling. Equivariant diffusion for molecule generation in 3d. In *International Conference on Machine Learning*, pages 8867–8887. PMLR, 2022.
- [154] Mark Hoogendoorn. Waarom helpt kunstmatige intelligentie de arts en patiënt nog zo weinig? 2022.
- [155] Han Hu, Zheng Zhang, Zhenda Xie, and Stephen Lin. Local relation networks for image recognition. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3464–3473, 2019.
- [156] Hanzhang Hu, John Langford, Rich Caruana, Saurajit Mukherjee, Eric J Horvitz, and Debadeepa Dey. Efficient forward architecture search. *Advances in Neural Information Processing Systems*, 32, 2019.
- [157] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018.
- [158] Qingyong Hu, Bo Yang, Linhai Xie, Stefano Rosa, Yulan Guo, Zhihua Wang, Niki Trigoni, and Andrew Markham. Randla-net: Efficient semantic segmentation of large-scale point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11108–11117, 2020.
- [159] Shoukang Hu, Sirui Xie, Hehui Zheng, Chunxiao Liu, Jianping Shi, Xunying Liu, and Dahua Lin. Dsnas: Direct neural architecture search without parameter re-training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12084–12092, 2020.
- [160] Binh-Son Hua, Minh-Khoi Tran, and Sai-Kit Yeung. Pointwise convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 984–993, 2018.

- [161] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.
- [162] Andrew Hundt, Varun Jain, and Gregory D Hager. sharpdarts: Faster and more accurate differentiable architecture search. *arXiv preprint arXiv:1903.09900*, 2019.
- [163] Michael J Hutchinson, Charline Le Lan, Sheheryar Zaidi, Emilien Dupont, Yee Whye Teh, and Hyunjik Kim. Lietransformer: equivariant self-attention for lie groups. In *International Conference on Machine Learning*, pages 4533–4543. PMLR, 2021.
- [164] Maximilian Ilse, Jakub M Tomczak, and Max Welling. Attention-based deep multiple instance learning. *ICML*, 2018.
- [165] Alexander Immer, Tycho van der Ouderaa, Gunnar Rätsch, Vincent Fortuin, and Mark van der Wilk. Invariance learning in deep neural networks with differentiable laplace approximations. *Advances in Neural Information Processing Systems*, 35:12449–12463, 2022.
- [166] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.
- [167] Jorn-Henrik Jacobsen, Jan Van Gemert, Zhongyu Lou, and Arnold WM Smeulders. Structured receptive fields in cnns. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2610–2619, 2016.
- [168] Andrew Jaegle, Felix Gimeno, Andy Brock, Oriol Vinyals, Andrew Zisserman, and Joao Carreira. Perceiver: General perception with iterative attention. In *International conference on machine learning*, pages 4651–4664. PMLR, 2021.
- [169] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.
- [170] Yunho Jeon and Junmo Kim. Active convolution: Learning the shape of convolution for image classification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4201–4209, 2017.
- [171] Xu Jia, Bert De Brabandere, Tinne Tuytelaars, and Luc V Gool. Dynamic filter networks. In *Advances in neural information processing systems*, pages 667–675, 2016.
- [172] Wei Jiang, Eduard Trulls, Jan Hosang, Andrea Tagliasacchi, and Kwang Moo Yi. Cotr: Correspondence transformer for matching across images. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6207–6217, 2021.
- [173] PLM Johannesma. The pre-response stimulus ensemble of neurons in the cochlear nucleus. In *Symposium on Hearing Theory*, 1972. IPO, 1972.

- [174] Alistair EW Johnson, Tom J Pollard, Lu Shen, Li-wei H Lehman, Mengling Feng, Mohammad Ghassemi, Benjamin Moody, Peter Szolovits, Leo Anthony Celi, and Roger G Mark. Mimic-iii, a freely accessible critical care database. *Scientific data*, 3(1):1–9, 2016.
- [175] Michael I Jordan and Tom M Mitchell. Machine learning: Trends, perspectives, and prospects. *Science*, 349(6245):255–260, 2015.
- [176] HM Kabir, Moloud Abdar, Seyed Mohammad Jafar Jalali, Abbas Khosravi, Amir F Atiya, Saeid Nahavandi, and Dipti Srinivasan. Spinalnet: Deep neural network with gradual input. *arXiv preprint arXiv:2007.03347*, 2020.
- [177] Nal Kalchbrenner and Phil Blunsom. Recurrent continuous translation models. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1700–1709, 2013.
- [178] Kirthevasan Kandasamy, Willie Neiswanger, Jeff Schneider, Barnabas Poczos, and Eric P Xing. Neural architecture search with bayesian optimisation and optimal transport. *Advances in neural information processing systems*, 31, 2018.
- [179] Nilanjana Karmakar, Arindam Biswas, Partha Bhowmick, and Bhargab B Bhattacharya. Construction of 3d orthogonal cover of a digital object. In *Combinatorial Image Analysis: 14th International Workshop, IWCIA 2011, Madrid, Spain, May 23-25, 2011. Proceedings 14*, pages 70–83. Springer, 2011.
- [180] Tero Karras, Miika Aittala, Samuli Laine, Erik Häkkinen, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Alias-free generative adversarial networks. *arXiv preprint arXiv:2106.12423*, 2021.
- [181] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention, 2020.
- [182] Osman Semih Kayhan and Jan C. van Gemert. On translation invariance in cnns: Convolutional layers can exploit absolute spatial location. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.
- [183] Patrick Kidger, James Morrill, James Foster, and Terry Lyons. Neural controlled differential equations for irregular time series. *arXiv preprint arXiv:2005.08926*, 2020.
- [184] Sanghyeon Kim and Eunbyung Park. Smpconv: Self-moving point representations for continuous convolution. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10289–10299, 2023.
- [185] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

- [186] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.
- [187] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.
- [188] Alexandre Kirchmeyer and Jia Deng. Convolutional networks with oriented 1d kernels. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6222–6232, 2023.
- [189] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer, 2020.
- [190] David M Knigge, David W Romero, and Erik J Bekkers. Exploiting redundancy: Separable group convolutional networks on lie groups. In *International Conference on Machine Learning*, pages 11359–11386. PMLR, 2022.
- [191] David M Knigge, David W. Romero, Albert Gu, Efstratios Gavves, Erik J Bekkers, Jakub Mikolaj Tomczak, Mark Hoogendoorn, and Jan jakob Sonke. Modelling long range dependencies in \$n\\$d: From task-specific to a general purpose CNN. In *The Eleventh International Conference on Learning Representations*, 2023. URL <https://openreview.net/forum?id=ZW5aK4yCRqU>.
- [192] Will Knight. Openai’s ceo says the age of giant ai models is already over. *Wired, April*, 17:2023, 2023.
- [193] Kodak. Kodak dataset, 1991. URL <http://r0k.us/graphics/kodak/>.
- [194] Risi Kondor and Shubhendu Trivedi. On the generalization of equivariance and convolution in neural networks to the action of compact groups. *arXiv preprint arXiv:1802.03690*, 2018.
- [195] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.
- [196] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.
- [197] Anders Krogh and John Hertz. A simple weight decay can improve generalization. *Advances in neural information processing systems*, 4, 1991.
- [198] Thijs P Kuipers and Erik J Bekkers. Regular se (3) group convolutions for volumetric medical image analysis. *arXiv preprint arXiv:2306.13960*, 2023.
- [199] Maxime W. Lafarge, Erik J. Bekkers, Josien P. W. Pluim, Remco Duits, and Mitko Veta. Roto-translation equivariant convolutional networks: Application to histopathology image analysis. *arXiv preprint arXiv:2002.08725*, 2020.
- [200] Samuli Laine and Timo Aila. Temporal ensembling for semi-supervised learning. *arXiv preprint arXiv:1610.02242*, 2016.

- [201] Hugo Larochelle, Dumitru Erhan, Aaron Courville, James Bergstra, and Yoshua Bengio. An empirical evaluation of deep architectures on problems with many factors of variation. In *Proceedings of the 24th international conference on Machine learning*, pages 473–480. ACM, 2007.
- [202] Edith Law, Kris West, Michael I Mandel, Mert Bay, and J Stephen Downie. Evaluation of algorithms using games: The case of music tagging. In *ISMIR*, pages 387–392, 2009.
- [203] Quoc V Le, Navdeep Jaitly, and Geoffrey E Hinton. A simple way to initialize recurrent networks of rectified linear units. *arXiv preprint arXiv:1504.00941*, 2015.
- [204] Truc Le and Ye Duan. Pointgrid: A deep network for 3d shape understanding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 9204–9214, 2018.
- [205] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010. URL <http://yann.lecun.com/exdb/mnist/>.
- [206] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.
- [207] Yann LeCun, John Denker, and Sara Solla. Optimal brain damage. *Advances in neural information processing systems*, 2, 1989.
- [208] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
- [209] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.
- [210] Jongpil Lee, Taejun Kim, Jiyoung Park, and Juhan Nam. Raw waveform-based audio classification using sample-level cnn architectures. *arXiv preprint arXiv:1712.00866*, 2017.
- [211] Jongpil Lee, Jiyoung Park, Keunhyoung Luke Kim, and Juhan Nam. Sample-level deep convolutional neural networks for music auto-tagging using raw waveforms. *arXiv preprint arXiv:1703.01789*, 2017.
- [212] Namhoon Lee. *Toward efficient deep learning with sparse neural networks*. PhD thesis, University of Oxford, 2020.
- [213] James Lee-Thorp, Joshua Ainslie, Ilya Eckstein, and Santiago Ontanon. Fnet: Mixing tokens with fourier transforms. *arXiv preprint arXiv:2105.03824*, 2021.
- [214] Attila Lengyel and Jan van Gemert. Exploiting learned symmetries in group equivariant convolutions. In *2021 IEEE International Conference on Image Processing (ICIP)*, pages 759–763. IEEE, 2021.

- [215] Jan Eric Lenssen, Matthias Fey, and Pascal Libuschewski. Group equivariant capsule networks. In *Advances in Neural Information Processing Systems*, pages 8844–8853, 2018.
- [216] Junying Li, Zichen Yang, Haifeng Liu, and Deng Cai. Deep rotation equivariant network. *Neurocomputing*, 290:26–33, 2018.
- [217] Liam Li, Mikhail Khodak, Maria-Florina Balcan, and Ameet Talwalkar. Geometry-aware gradient algorithms for neural architecture search. *arXiv preprint arXiv:2004.07802*, 2020.
- [218] Shuai Li, Wanqing Li, Chris Cook, Ce Zhu, and Yanbo Gao. Independently recurrent neural network (indrnn): Building a longer and deeper rnn. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5457–5466, 2018.
- [219] Yonggang Li, Guosheng Hu, Yongtao Wang, Timothy M. Hospedales, Neil Martin Robertson, and Yongxin Yang. DADA: differentiable automatic data augmentation. 2020.
- [220] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020.
- [221] Hanwen Liang, Shifeng Zhang, Jiacheng Sun, Xingqiu He, Weiran Huang, Kechen Zhuang, and Zhenguo Li. Darts+: Improved differentiable architecture search with early stopping. *arXiv preprint arXiv:1909.06035*, 2019.
- [222] Sungbin Lim, Ildoo Kim, Taesup Kim, Chiheon Kim, and Sungwoong Kim. Fast autoaugment. *Advances in Neural Information Processing Systems*, 32, 2019.
- [223] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.
- [224] Xudong Lin, Lin Ma, Wei Liu, and Shih-Fu Chang. Context-gated convolution, 2019.
- [225] Tony Lindeberg. Scale-covariant and scale-invariant gaussian derivative networks. In *Scale Space and Variational Methods in Computer Vision :*, volume 12679 of *Springer Lecture Notes in Computer Science*, pages 3–14. Springer Nature, 2021. ISBN 978-3-030-75548-5. doi: 10.1007/978-3-030-75549-2\_1. URL <https://arxiv.org/abs/2011.14759>. Not duplicate with DiVA 1505585QC 20210317.
- [226] Tony Lindeberg and Anders Friberg. Idealized computational models for auditory receptive fields. *PLoS one*, 10(3), 2015.
- [227] Tony Lindeberg and Anders Friberg. Scale-space theory for auditory signals. In

- International Conference on Scale Space and Variational Methods in Computer Vision*, pages 3–15. Springer, 2015.
- [228] Drew Linsley, Junkyung Kim, Vijay Veerabadran, Charles Windolf, and Thomas Serre. Learning long-range spatial dependencies with horizontal gated recurrent units. *Advances in neural information processing systems*, 31, 2018.
  - [229] Chenxi Liu, Liang-Chieh Chen, Florian Schroff, Hartwig Adam, Wei Hua, Alan L Yuille, and Li Fei-Fei. Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 82–92, 2019.
  - [230] Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. Hierarchical representations for efficient architecture search. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL <https://openreview.net/forum?id=BJQRKzbA->.
  - [231] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018.
  - [232] Jen-Yu Liu, Shyh-Kang Jeng, and Yi-Hsuan Yang. Applying topological persistence in convolutional neural network for music audio signals. *arXiv preprint arXiv:1608.07373*, 2016.
  - [233] Liyuan Liu, Xiaodong Liu, Jianfeng Gao, Weizhu Chen, and Jiawei Han. Understanding the difficulty of training transformers. *arXiv preprint arXiv:2004.08249*, 2020.
  - [234] Shiwei Liu, Tianlong Chen, Xiaohan Chen, Xuxi Chen, Qiao Xiao, Boqian Wu, Tommi Kärkkäinen, Mykola Pechenizkiy, Decebal Constantin Mocanu, and Zhangyang Wang. More convnets in the 2020s: Scaling up kernels beyond 51x51 using sparsity. In *The Eleventh International Conference on Learning Representations*, 2023. URL <https://openreview.net/forum?id=bXNl-myZkJ1>.
  - [235] Zhijian Liu, Haotian Tang, Yujun Lin, and Song Han. Point-voxel cnn for efficient 3d deep learning. *Advances in Neural Information Processing Systems*, 32, 2019.
  - [236] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. *arXiv preprint arXiv:2201.03545*, 2022.
  - [237] Marco Loog and Francois Lauze. Supervised scale-regularized linear convolutionary filters. In Gabriel Brostow Tae-Kyun Kim, Stefanos Zafeiriou and Krystian Mikolajczyk, editors, *Proceedings of the British Machine Vision Conference (BMVC)*, pages 162.1–162.11. BMVA Press, September 2017. ISBN 1-901725-60-X. doi: 10.5244/C.31.162. URL <https://dx.doi.org/10.5244/C.31.162>.
  - [238] 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>.
- [239] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL <https://openreview.net/forum?id=Bkg6RiCqY7>.
- [240] Xugang Lu, Peng Shen, Sheng Li, Yu Tsao, and Hisashi Kawai. Deep progressive multi-scale attention for acoustic event classification. *arXiv preprint arXiv:1912.12011*, 2019.
- [241] Chunjie Luo, Jianfeng Zhan, Lei Wang, and Wanling Gao. Extended batch normalization. *arXiv preprint arXiv:2003.05569*, 2020.
- [242] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.
- [243] Xuezhe Ma, Xiang Kong, Sinong Wang, Chunting Zhou, Jonathan May, Hao Ma, and Luke Zettlemoyer. Luna: Linear unified nested attention. *Advances in Neural Information Processing Systems*, 34:2441–2453, 2021.
- [244] Xuezhe Ma, Chunting Zhou, Xiang Kong, Junxian He, Liangke Gui, Graham Neubig, Jonathan May, and Luke Zettlemoyer. Mega: moving average equipped gated attention. *arXiv preprint arXiv:2209.10655*, 2022.
- [245] Andrew Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies*, pages 142–150, 2011.
- [246] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.
- [247] Kaitlin Maile, Dennis George Wilson, and Patrick Forré. Equivariance-aware architectural optimization of neural networks. In *The Eleventh International Conference on Learning Representations*, 2023. URL <https://openreview.net/forum?id=a6rCdfABJXg>.
- [248] Stéphane Mallat. *A wavelet tour of signal processing*. Elsevier, 1999.
- [249] Stéphane Mallat. Group invariant scattering. *Communications on Pure and Applied Mathematics*, 65(10):1331–1398, 2012.
- [250] Ravi Manne and Sneha C Kantheti. Application of artificial intelligence in health-care: chances and challenges. *Current Journal of Applied Science and Technology*, 40(6):78–89, 2021.

- [251] Mary Ann Marcinkiewicz. Building a large annotated corpus of english: The penn treebank. *Using Large Corpora*, page 273, 1994.
- [252] Diego Marcos, Michele Volpi, Nikos Komodakis, and Devis Tuia. Rotation equivariant vector field networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5048–5057, 2017.
- [253] Diego Marcos, Benjamin Kellenberger, Sylvain Lobry, and Devis Tuia. Scale equivariance in cnns with vector fields. *arXiv preprint arXiv:1807.11783*, 2018.
- [254] Haggai Maron, Heli Ben-Hamu, Nadav Shamir, and Yaron Lipman. Invariant and equivariant graph networks. *arXiv preprint arXiv:1812.09902*, 2018.
- [255] Haggai Maron, Or Litany, Gal Chechik, and Ethan Fetaya. On learning sets of symmetric elements. *arXiv preprint arXiv:2002.08599*, 2020.
- [256] Stefano Massaroli, Michael Poli, Dan Fu, Hermann Kumbong, David W Romero, Rom Parnichkun, Aman Timalsina, Quinn McIntyre, Beidi Chen, Atri Rudra, Ce Zhang, Christopher Ré, Stefano Ermon, and Yoshua Bengio. Laughing hyena distillery: Extracting compact recurrences from convolutions. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.
- [257] Daniel Maturana and Sebastian Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *2015 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 922–928. IEEE, 2015.
- [258] Vittorio Mazzia, Francesco Salvetti, and Marcello Chiaberge. Efficient-capsnet: Capsule network with self-attention routing. *arXiv preprint arXiv:2101.12491*, 2021.
- [259] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5:115–133, 1943.
- [260] Gábor Melis, Tomáš Kočiský, and Phil Blunsom. Mogrifier lstm. *arXiv preprint arXiv:1909.01792*, 2019.
- [261] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4460–4470, 2019.
- [262] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *arXiv preprint arXiv:2003.08934*, 2020.
- [263] Geoffrey F Miller, Peter M Todd, and Shailesh U Hegde. Designing neural networks using genetic algorithms. In *ICGA*, volume 89, pages 379–384, 1989.
- [264] Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel. A simple neural attentive meta-learner. *arXiv preprint arXiv:1707.03141*, 2017.

- [265] Guido F Montufar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the number of linear regions of deep neural networks. In *Advances in neural information processing systems*, pages 2924–2932, 2014.
- [266] Brian CJ Moore and Hedwig E Gockel. Properties of auditory stream formation. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 367(1591):919–931, 2012.
- [267] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Transactions on Graphics (ToG)*, 41(4):1–15, 2022.
- [268] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Icml*, 2010.
- [269] Nikita Nangia and Samuel R Bowman. Listops: A diagnostic dataset for latent tree learning. *arXiv preprint arXiv:1804.06028*, 2018.
- [270] Renato Negrinho and Geoff Gordon. Deeparchitect: Automatically designing and training deep architectures. *arXiv preprint arXiv:1704.08792*, 2017.
- [271] Duc Nguyen. *Robust deep learning for computer vision to counteract data scarcity and label noise*. PhD thesis, 01 2020.
- [272] Eric Nguyen, Karan Goel, Albert Gu, Gordon W Downs, Preey Shah, Tri Dao, Stephen A Baccus, and Christopher Ré. S4nd: Modeling images and videos as multidimensional signals using state spaces. *arXiv preprint arXiv:2210.06583*, 2022.
- [273] Eric Nguyen, Michael Poli, Marjan Faizi, Armin Thomas, Callum Birch-Sykes, Michael Wornow, Aman Patel, Clayton Rabideau, Stefano Massaroli, Yoshua Bengio, et al. Hyenadna: Long-range genomic sequence modeling at single nucleotide resolution. *arXiv preprint arXiv:2306.15794*, 2023.
- [274] Vu Nguyen, Tam Le, Makoto Yamada, and Michael A Osborne. Optimal transport kernels for sequential and parallel neural architecture search. In *International Conference on Machine Learning*, pages 8084–8095. PMLR, 2021.
- [275] Gyutaek Oh, Baekgyu Choi, Inkyung Jung, and Jong Chul Ye. schyena: Foundation model for full-length single-cell rna-seq analysis in brain. *arXiv preprint arXiv:2310.02713*, 2023.
- [276] Aude Oliva and Antonio Torralba. The role of context in object recognition. *Trends in cognitive sciences*, 11(12):520–527, 2007.
- [277] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.
- [278] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape

- representation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 165–174, 2019.
- [279] Jongchan Park, Sanghyun Woo, Joon-Young Lee, and In So Kweon. Bam: Bottleneck attention module. *arXiv preprint arXiv:1807.06514*, 2018.
- [280] Keunhong Park, Utkarsh Sinha, Peter Hedman, Jonathan T Barron, Sofien Bouaziz, Dan B Goldman, Ricardo Martin-Brualla, and Steven M Seitz. Hypernerf: A higher-dimensional representation for topologically varying neural radiance fields. *arXiv preprint arXiv:2106.13228*, 2021.
- [281] Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. How to construct deep recurrent neural networks. *arXiv preprint arXiv:1312.6026*, 2013.
- [282] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318, 2013.
- [283] Harold Pashler. *Attention*. Psychology Press, 2016.
- [284] Judea Pearl. *Causality: Models, Reasoning, and Inference*. Cambridge University Press, Cambridge, UK, 2009. ISBN 978-0521895606.
- [285] Vijayaditya Peddinti, TaraN Sainath, Shay Maymon, Bhuvana Ramabhadran, David Nahamoo, and Vaibhava Goel. Deep scattering spectrum with deep neural networks. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 210–214. IEEE, 2014.
- [286] Chao Peng, Xiangyu Zhang, Gang Yu, Guiming Luo, and Jian Sun. Large kernel matters—improve semantic segmentation by global convolutional network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4353–4361, 2017.
- [287] Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. Film: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.
- [288] Jonas Peters, Dominik Janzing, and Bernhard Schölkopf. *Elements of causal inference: foundations and learning algorithms*. The MIT Press, 2017.
- [289] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In *International conference on machine learning*, pages 4095–4104. PMLR, 2018.
- [290] Karol J Piczak. Environmental sound classification with convolutional neural networks. In *2015 IEEE 25th International Workshop on Machine Learning for Signal Processing (MLSP)*, pages 1–6. IEEE, 2015.

- [291] Silvia L Pintea, Nergis Tomen, Stanley F Goes, Marco Loog, and Jan C van Gemert. Resolution learning in deep convolutional networks using scale-space theory. *arXiv preprint arXiv:2106.03412*, 2021.
- [292] Michael Poli, Stefano Massaroli, Eric Nguyen, Daniel Y Fu, Tri Dao, Stephen Baccus, Yoshua Bengio, Stefano Ermon, and Christopher Ré. Hyena hierarchy: Towards larger convolutional language models. *arXiv preprint arXiv:2302.10866*, 2023.
- [293] Jordi Pons and Xavier Serra. Randomly weighted cnns for (music) audio classification. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 336–340. IEEE, 2019.
- [294] Jordi Pons, Oriol Nieto, Matthew Prockup, Erik Schmidt, Andreas Ehmann, and Xavier Serra. End-to-end learning for music audio tagging at scale. *arXiv preprint arXiv:1711.02520*, 2017.
- [295] Jordi Pons, Olga Slizovskaia, Rong Gong, Emilia Gómez, and Xavier Serra. Timbre analysis of music audio signals with convolutional neural networks. In *2017 25th European Signal Processing Conference (EUSIPCO)*, pages 2744–2748. IEEE, 2017.
- [296] Charles R Qi, Hao Su, Matthias Nießner, Angela Dai, Mengyuan Yan, and Leonidas J Guibas. Volumetric and multi-view cnns for object classification on 3d data. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5648–5656, 2016.
- [297] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017.
- [298] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in neural information processing systems*, 30, 2017.
- [299] Dragomir R Radev, Pradeep Muthukrishnan, Vahed Qazvinian, and Amjad Abu-Jbara. The acl anthology network corpus. *Language Resources and Evaluation*, 47(4):919–944, 2013.
- [300] Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.
- [301] Prajit Ramachandran, Niki Parmar, Ashish Vaswani, Irwan Bello, Anselm Levskaya, and Jonathon Shlens. Stand-alone self-attention in vision models. *arXiv preprint arXiv:1906.05909*, 2019.
- [302] Raghunathan Ramakrishnan, Pavlo O Dral, Matthias Rupp, and O Anatole Von Lilienfeld. Quantum chemistry structures and properties of 134 kilo molecules. *Scientific data*, 1(1):1–7, 2014.

- [303] Siamak Ravanbakhsh. Universal equivariant multilayer perceptrons. *arXiv preprint arXiv:2002.02912*, 2020.
- [304] Siamak Ravanbakhsh, Jeff Schneider, and Barnabas Poczos. Equivariance through parameter-sharing. *arXiv preprint arXiv:1702.08389*, 2017.
- [305] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pages 4780–4789, 2019.
- [306] Dario Rethage, Jordi Pons, and Xavier Serra. A wavenet for speech denoising. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5069–5073. IEEE, 2018.
- [307] Matthew A Reyna, Chris Josef, Salman Seyedi, Russell Jeter, Supreeth P Shashikumar, M Brandon Westover, Ashish Sharma, Shamim Nemati, and Gari D Clifford. Early prediction of sepsis from clinical data: the physionet/computing in cardiology challenge 2019. In *2019 Computing in Cardiology (CinC)*, pages Page–1. IEEE, 2019.
- [308] Rachid Riad, Olivier Teboul, David Grangier, and Neil Zeghidour. Learning strides in convolutional neural networks. *arXiv preprint arXiv:2202.01653*, 2022.
- [309] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. " why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144, 2016.
- [310] Maximilian Riesenhuber, Izzat Jarudi, Sharon Gilad, and Pawan Sinha. Face processing in humans is compatible with a simple shape-based model of vision. *Proceedings of the Royal Society of London. Series B: Biological Sciences*, 271(suppl\_6):S448–S450, 2004.
- [311] Roberto Rigamonti, Amos Sironi, Vincent Lepetit, and Pascal Fua. Learning separable filters. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2754–2761, 2013.
- [312] Oren Rippel, Jasper Snoek, and Ryan P Adams. Spectral representations for convolutional neural networks. *Advances in neural information processing systems*, 28, 2015.
- [313] Nicholas Roberts, Mikhail Khodak, Tri Dao, Liam Li, Christopher Ré, and Ameet Talwalkar. Rethinking neural operations for diverse tasks. *Advances in Neural Information Processing Systems*, 34:15855–15869, 2021.
- [314] David W. Romero and Jean-Baptiste Cordonnier. Group equivariant stand-alone self-attention for vision. In *International Conference on Learning Representations*, 2021. URL <https://openreview.net/forum?id=JkfYjnOEo6M>.

- [315] David W. Romero and Mark Hoogendoorn. Co-attentive equivariant neural networks: Focusing equivariance on transformations co-occurring in data. In *International Conference on Learning Representations*, 2020. URL <https://openreview.net/forum?id=r1g6ogrtDr>.
- [316] David W Romero and Suhas Lohit. Learning partial equivariances from data. *Advances in Neural Information Processing Systems*, 35:36466–36478, 2022.
- [317] David W Romero and Neil Zeghidour. Dnarch: Learning convolutional neural architectures by backpropagation. *arXiv preprint arXiv:2302.05400*, 2023.
- [318] David W Romero, Erik J Bekkers, Jakub M Tomczak, and Mark Hoogendoorn. Attentive group equivariant convolutional networks. *arXiv preprint arXiv:2002.03830*, 2020.
- [319] David W. Romero, Robert-Jan Bruintjes, Jakub Mikolaj Tomczak, Erik J Bekkers, Mark Hoogendoorn, and Jan van Gemert. Flexconv: Continuous kernel convolutions with differentiable kernel sizes. In *International Conference on Learning Representations*, 2022. URL <https://openreview.net/forum?id=3jooF27-0Wy>.
- [320] David W Romero, David M Knigge, Albert Gu, Erik J Bekkers, Efstratios Gavves, Jakub M Tomczak, and Mark Hoogendoorn. Towards a general purpose cnn for long range dependencies in *nd*. *arXiv preprint arXiv:2206.03398*, 2022.
- [321] David W. Romero, Anna Kuzina, Erik J Bekkers, Jakub Mikolaj Tomczak, and Mark Hoogendoorn. CKConv: Continuous kernel convolution for sequential data. In *International Conference on Learning Representations*, 2022. URL <https://openreview.net/forum?id=8FhxBtXS10>.
- [322] David W Romero, Erik J Bekkers, Jakub M Tomczak, and Mark Hoogendoorn. Wavelet networks: Scale-translation equivariant learning from raw time-series. *Transactions on Machine Learning Research*, 2023. URL <https://openreview.net/forum?id=ga5SNulYet>.
- [323] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.
- [324] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.
- [325] Yulia Rubanova, Ricky TQ Chen, and David K Duvenaud. Latent ordinary differential equations for irregularly-sampled time series. In *Advances in Neural Information Processing Systems*, pages 5320–5330, 2019.
- [326] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

- [327] T Konstantin Rusch and Siddhartha Mishra. Coupled oscillatory recurrent neural network (cornn): An accurate and (gradient) stable architecture for learning long time dependencies. *arXiv preprint arXiv:2010.00951*, 2020.
- [328] T Konstantin Rusch and Siddhartha Mishra. Unicornnn: A recurrent model for learning very long time dependencies. *arXiv preprint arXiv:2103.05487*, 2021.
- [329] 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, 2015. doi: 10.1007/s11263-015-0816-y.
- [330] Lars Ruthotto and Eldad Haber. Deep neural networks motivated by partial differential equations. *Journal of Mathematical Imaging and Vision*, 62(3):352–364, 2020.
- [331] Justin Salamon and Juan Pablo Bello. Feature learning with deep scattering for urban sound analysis. In *2015 23rd European Signal Processing Conference (EUSIPCO)*, pages 724–728. IEEE, 2015.
- [332] Justin Salamon, Christopher Jacoby, and Juan Pablo Bello. A dataset and taxonomy for urban sound research. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 1041–1044, 2014.
- [333] Tim Salimans and Diederik P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *arXiv preprint arXiv:1602.07868*, 2016.
- [334] Roberta Santoro, Michelle Moerel, Federico De Martino, Rainer Goebel, Kamil Ugurbil, Essa Yacoub, and Elia Formisano. Encoding of natural sounds at multiple spectral and temporal resolutions in the human auditory cortex. *PLoS computational biology*, 10(1), 2014.
- [335] Vishwanath Saragadam, Daniel LeJeune, Jasper Tan, Guha Balakrishnan, Ashok Veeraraghavan, and Richard G Baraniuk. Wire: Wavelet implicit neural representations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 18507–18516, 2023.
- [336] Victor Garcia Satorras, Emiel Hoogeboom, Fabian B Fuchs, Ingmar Posner, and Max Welling. E (n) equivariant normalizing flows. *arXiv preprint arXiv:2105.09016*, 2021.
- [337] Shreyas Saxena and Jakob Verbeek. Convolutional neural fabrics. *Advances in neural information processing systems*, 29, 2016.
- [338] Louis L Scharf. *Statistical signal processing*, volume 98. Addison-Wesley Reading, MA, 1991.

- [339] Jürgen Schmidhuber. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 4(1):131–139, 1992.
- [340] Philipp Schmidt, Patrick Spröte, and Roland W Fleming. Perception of shape and space across rigid transformations. *Vision research*, 126:318–329, 2016.
- [341] Bernhard Schölkopf, Francesco Locatello, Stefan Bauer, Nan Rosemary Ke, Nal Kalchbrenner, Anirudh Goyal, and Yoshua Bengio. Toward causal representation learning. *Proceedings of the IEEE*, 109(5):612–634, 2021.
- [342] Tal Schuster, Adam Fisch, Jai Gupta, Mostafa Dehghani, Dara Bahri, Vinh Q Tran, Yi Tay, and Donald Metzler. Confident adaptive language modeling. *arXiv preprint arXiv:2207.07061*, 2022.
- [343] Kristof Schütt, Pieter-Jan Kindermans, Huziel Enoc Sauceda Felix, Stefan Chmiela, Alexandre Tkatchenko, and Klaus-Robert Müller. Schnet: A continuous-filter convolutional neural network for modeling quantum interactions. In *Advances in neural information processing systems*, pages 991–1001, 2017.
- [344] Katja Schwarz, Yiyi Liao, Michael Niemeyer, and Andreas Geiger. Graf: Generative radiance fields for 3d-aware image synthesis. *arXiv preprint arXiv:2007.02442*, 2020.
- [345] Gudrun Schwarzer. Development of face processing: The effect of face inversion. *Child development*, 71(2):391–401, 2000.
- [346] Tom Sercu, Christian Puhrsch, Brian Kingsbury, and Yann LeCun. Very deep multilingual convolutional neural networks for lvcsr. In *2016 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 4955–4959. IEEE, 2016.
- [347] Thiago Serra, Christian Tjandraatmadja, and Sri Kumar Ramalingam. Bounding and counting linear regions of deep neural networks. In *International Conference on Machine Learning*, pages 4558–4566. PMLR, 2018.
- [348] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations. In Marilyn A. Walker, Heng Ji, and Amanda Stent, editors, *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 2 (Short Papers)*, pages 464–468. Association for Computational Linguistics, 2018. doi: 10.18653/v1/n18-2074. URL <https://doi.org/10.18653/v1/n18-2074>.
- [349] Evan Shelhamer, Dequan Wang, and Trevor Darrell. Blurring the line between structure and learning to optimize and adapt receptive fields. *ArXiv*, abs/1904.11487, 2019.
- [350] Junhong Shen, Mikhail Khodak, and Ameet Talwalkar. Efficient architecture search for diverse tasks. *arXiv preprint arXiv:2204.07554*, 2022.

- [351] Xinwei Shen, Furui Liu, Hanze Dong, Qing Lian, Zhitang Chen, and Tong Zhang. Weakly supervised disentangled generative causal representation learning. *The Journal of Machine Learning Research*, 23(1):10994–11048, 2022.
- [352] Shaoshuai Shi, Zhe Wang, Jianping Shi, Xiaogang Wang, and Hongsheng Li. From points to parts: 3d object detection from point cloud with part-aware and part-aggregation network. *arXiv preprint arXiv:1907.03670*, 2019.
- [353] Weijing Shi and Raj Rajkumar. Point-gnn: Graph neural network for 3d object detection in a point cloud. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1711–1719, 2020.
- [354] Laurent Sifre and Stéphane Mallat. Rigid-motion scattering for texture classification. *arXiv preprint arXiv:1403.1687*, 2014.
- [355] Martin Simonovsky and Nikos Komodakis. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3693–3702, 2017.
- [356] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.
- [357] Pawan Sinha, Benjamin Balas, Yuri Ostrovsky, and Richard Russell. Face recognition by humans: Nineteen results all computer vision researchers should know about. *Proceedings of the IEEE*, 94(11):1948–1962, 2006.
- [358] Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. *Advances in Neural Information Processing Systems*, 33, 2020.
- [359] Bart Smets, Jim Portegies, Erik Bekkers, and Remco Duits. Pde-based group equivariant convolutional neural networks. *arXiv preprint arXiv:2001.09046*, 2020.
- [360] Jimmy TH Smith, Andrew Warrington, and Scott W Linderman. Simplified state space layers for sequence modeling. *arXiv preprint arXiv:2208.04933*, 2022.
- [361] Dehua Song, Chang Xu, Xu Jia, Yiyi Chen, Chunjing Xu, and Yunhe Wang. Efficient residual dense block search for image super-resolution. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 12007–12014, 2020.
- [362] Ivan Sosnovik, Michał Szmaja, and Arnold Smeulders. Scale-equivariant steerable networks. In *International Conference on Learning Representations*, 2020. URL <https://openreview.net/forum?id=HJgpugrKPS>.
- [363] Ivan Sosnovik, Artem Moskalev, and Arnold Smeulders. Disco: accurate discrete scale convolutions. *arXiv preprint arXiv:2106.02733*, 2021.
- [364] Ivan Sosnovik, Artem Moskalev, and Arnold WM Smeulders. Scale equivariance improves siamese tracking. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 2765–2774, 2021.

- [365] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.
- [366] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. URL <http://jmlr.org/papers/v15/srivastava14a.html>.
- [367] Stanley Smith Stevens, John Volkman, and Edwin 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.
- [368] Daniel Stoller, Sebastian Ewert, and Simon Dixon. Wave-u-net: A multi-scale neural network for end-to-end audio source separation. *arXiv preprint arXiv:1806.03185*, 2018.
- [369] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in nlp. *arXiv preprint arXiv:1906.02243*, 2019.
- [370] Bing Su and Ji-Rong Wen. Log-polar space convolution for convolutional neural networks. *arXiv preprint arXiv:2107.11943*, 2021.
- [371] Kevin Swersky, David Duvenaud, Jasper Snoek, Frank Hutter, and Michael A Osborne. Raiders of the lost architecture: Kernels for bayesian optimization in conditional parameter spaces. *arXiv preprint arXiv:1409.4011*, 2014.
- [372] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.
- [373] Domen Tabernik, Matej Kristan, and Ales Leonardis. Spatially-adaptive filter units for compact and efficient deep neural networks. *International Journal of Computer Vision*, 128, 09 2020. doi: 10.1007/s11263-019-01282-1.
- [374] Kai Sheng Tai, Peter Bailis, and Gregory Valiant. Equivariant transformer networks. *arXiv preprint arXiv:1901.11399*, 2019.
- [375] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pages 6105–6114. PMLR, 2019.
- [376] 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 *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2820–2828, 2019.

- [377] Matthew Tancik, Pratul P Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. *arXiv preprint arXiv:2006.10739*, 2020.
- [378] Michael J Tarr and Steven Pinker. Mental rotation and orientation-dependence in shape recognition. *Cognitive psychology*, 21(2):233–282, 1989.
- [379] Yi Tay, Mostafa Dehghani, Samira Abnar, Yikang Shen, Dara Bahri, Philip Pham, Jinfeng Rao, Liu Yang, Sebastian Ruder, and Donald Metzler. Long range arena : A benchmark for efficient transformers. In *International Conference on Learning Representations*, 2021. URL <https://openreview.net/forum?id=qVyeW-grC2k>.
- [380] Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. Branchynet: Fast inference via early exiting from deep neural networks. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 2464–2469. IEEE, 2016.
- [381] Hugues Thomas, Charles R Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, François Goulette, and Leonidas J Guibas. Kpconv: Flexible and deformable convolution for point clouds. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 6411–6420, 2019.
- [382] Nathaniel Thomas, Tess Smidt, Steven Kearnes, Lusann Yang, Li Li, Kai Kohlhoff, and Patrick Riley. Tensor field networks: Rotation-and translation-equivariant neural networks for 3d point clouds. *arXiv preprint arXiv:1802.08219*, 2018.
- [383] Yuji Tokozume and Tatsuya Harada. Learning environmental sounds with end-to-end convolutional neural network. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2721–2725. IEEE, 2017.
- [384] Nergis Tomen, Silvia-Laura Pintea, and Jan Van Gemert. Deep continuous networks. 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 10324–10335. PMLR, 18–24 Jul 2021. URL <https://proceedings.mlr.press/v139/tomen21a.html>.
- [385] Hugo Touvron, A. Vedaldi, M. Douze, and H. Jégou. Fixing the train-test resolution discrepancy. In *NeurIPS*, 2019.
- [386] Trieu H Trinh, Andrew M Dai, Minh-Thang Luong, and Quoc V Le. Learning longer-term dependencies in rnns with auxiliary losses. *arXiv preprint arXiv:1803.00144*, 2018.
- [387] Renbo Tu, Nicholas Roberts, Mikhail Khodak, Junhong Shen, Frederic Sala, and Ameet Talwalkar. Nas-bench-360: Benchmarking neural architecture search on diverse tasks. In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022.

- [388] Greg Turk and Marc Levoy. Zippered polygon meshes from range images. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 311–318, 1994.
- [389] Karen Ullrich, Jan Schlüter, and Thomas Grill. Boundary detection in music structure analysis using convolutional neural networks. In *ISMIR*, pages 417–422, 2014.
- [390] Devrim Unay, Ahmet Ekin, Mujdat Cetin, Radu Jasinschi, and Aytul Ercil. Robustness of local binary patterns in brain mr image analysis. In *2007 29th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pages 2098–2101. IEEE, 2007.
- [391] Alonso Urbano and David W Romero. Self-supervised detection of perfect and partial input-dependent symmetries. *Under review*, 2023.
- [392] Aäron Van Den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *SSW*, 125:2, 2016.
- [393] Aaron Van den Oord, Nal Kalchbrenner, Lasse Espeholt, Oriol Vinyals, Alex Graves, et al. Conditional image generation with pixelcnn decoders. *Advances in neural information processing systems*, 29, 2016.
- [394] Putri A Van der Linden, David W Romero, and Erik J Bekkers. Learned gridification for efficient point cloud processing. *Proceedings of the Second Annual Workshop on Topology, Algebra and Geometry in Machine Learning (TAG-ML) at the fortieth International Conference on Machine Learning*, 2023.
- [395] Tycho van der Ouderaa, David W Romero, and Mark van der Wilk. Relaxing equivariance constraints with non-stationary continuous filters. *Advances in Neural Information Processing Systems*, 35:33818–33830, 2022.
- [396] Tycho F.A van der Ouderaa and Mark van der Wilk. Learning invariant weights in neural networks. *Workshop in Uncertainty & Robustness in Deep Learning, ICML*, 2021.
- [397] Tycho F.A. van der Ouderaa, Alexander Immer, and Mark van der Wilk. Learning layer-wise equivariances automatically using gradients. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.
- [398] Mark van der Wilk, Matthias Bauer, ST John, and James Hensman. Learning invariances using the marginal likelihood. *Advances in Neural Information Processing Systems*, 31, 2018.
- [399] Jan Van Dijk and Kenneth Hacker. The digital divide as a complex and dynamic phenomenon. *The information society*, 19(4):315–326, 2003.
- [400] Leo Paulus Antonie Servatius van Noorden et al. *Temporal coherence in the percep-*

- tion of tone sequences*, volume 3. Institute for Perceptual Research Eindhoven, the Netherlands, 1975.
- [401] Cristina Vasconcelos, Hugo Larochelle, Vincent Dumoulin, Rob Romijnders, Nicolas Le Roux, and Ross Goroshin. Impact of aliasing on generalization in deep convolutional networks. *arXiv preprint arXiv:2108.03489*, 2021.
  - [402] Cristina N Vasconcelos, Cengiz Oztireli, Mark Matthews, Milad Hashemi, Kevin Swersky, and Andrea Tagliasacchi. Cuf: Continuous upsampling filters. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9999–10008, 2023.
  - [403] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.
  - [404] Bastiaan S Veeling, Jasper Linmans, Jim Winkens, Taco Cohen, and Max Welling. Rotation equivariant cnns for digital pathology. In *International Conference on Medical image computing and computer-assisted intervention*, pages 210–218. Springer, 2018.
  - [405] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.
  - [406] Sai Raam Venkataraman, S. Balasubramanian, and R. Raghunatha Sarma. Building deep equivariant capsule networks. In *International Conference on Learning Representations*, 2020. URL <https://openreview.net/forum?id=BJgNJgSFPS>.
  - [407] Hermann von Helmholtz. Über die Tatsachen, die der Geometrie zugrunde liegen. 1868.
  - [408] Patrick von Platen, Chao Zhang, and Philip Woodland. Multi-span acoustic modelling using raw waveform signals. *arXiv preprint arXiv:1906.11047*, 2019.
  - [409] Robin Walters, Jinxi Li, and Rose Yu. Trajectory prediction using equivariant continuous convolution. *arXiv preprint arXiv:2010.11344*, 2020.
  - [410] Linnan Wang, Yiyang Zhao, Yuu Jinnai, Yuandong Tian, and Rodrigo Fonseca. Alphax: exploring neural architectures with deep neural networks and monte carlo tree search. *arXiv preprint arXiv:1903.11059*, 2019.
  - [411] Linnan Wang, Yiyang Zhao, Yuu Jinnai, Yuandong Tian, and Rodrigo Fonseca. Neural architecture search using deep neural networks and monte carlo tree search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 9983–9991, 2020.
  - [412] Rui Wang, Karthik Kashinath, Mustafa Mustafa, Adrian Albert, and Rose Yu. Towards physics-informed deep learning for turbulent flow prediction. In *Proceed-*

- ings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 1457–1466, 2020.
- [413] Rui Wang, Robin Walters, and Rose Yu. Incorporating symmetry into deep dynamics models for improved generalization. *arXiv preprint arXiv:2002.03061*, 2020.
  - [414] Rui Wang, Robin Walters, and Rose Yu. Approximately equivariant networks for imperfectly symmetric dynamics. In *International Conference on Machine Learning*, pages 23078–23091. PMLR, 2022.
  - [415] Rui Wang, Robin Walters, and Rose Yu. Data augmentation vs. equivariant networks: A theory of generalization on dynamics forecasting. *arXiv preprint arXiv:2206.09450*, 2022.
  - [416] Ruochen Wang, Minhao Cheng, Xiangning Chen, Xiaocheng Tang, and Cho-Jui Hsieh. Rethinking architecture selection in differentiable nas. *arXiv preprint arXiv:2108.04392*, 2021.
  - [417] Shenlong Wang, Simon Suo, Wei-Chiu Ma, Andrei Pokrovsky, and Raquel Urtasun. Deep parametric continuous convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2589–2597, 2018.
  - [418] Sinong Wang, Belinda Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020.
  - [419] 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*, pages 7794–7803, 2018.
  - [420] Zongji Wang and Feng Lu. Voxsegnet: Volumetric cnns for semantic part segmentation of 3d shapes. *IEEE transactions on visualization and computer graphics*, 26(9):2919–2930, 2019.
  - [421] Pete Warden. Speech commands: A dataset for limited-vocabulary speech recognition. *arXiv preprint arXiv:1804.03209*, 2018.
  - [422] Maurice Weiler and Gabriele Cesa. General e (2)-equivariant steerable cnns. *Advances in Neural Information Processing Systems*, 32, 2019.
  - [423] Maurice Weiler, Mario Geiger, Max Welling, Wouter Boomsma, and Taco S Cohen. 3d steerable cnns: Learning rotationally equivariant features in volumetric data. *Advances in Neural Information Processing Systems*, 31, 2018.
  - [424] Maurice Weiler, Fred A Hamprecht, and Martin Storath. Learning steerable filters for rotation equivariant cnns. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 849–858, 2018.

- [425] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. *Advances in neural information processing systems*, 29, 2016.
- [426] Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.
- [427] Max Wertheimer. Gestalt theory. 1938.
- [428] Colin White, Mahmoud Safari, Rhea Sukthanker, Binxin Ru, Thomas Elsken, Arber Zela, Debadatta Dey, and Frank Hutter. Neural architecture search: Insights from 1000 papers. *arXiv preprint arXiv:2301.08727*, 2023.
- [429] Marysia Winkels and Taco S Cohen. 3d g-cnns for pulmonary nodule detection. *arXiv preprint arXiv:1804.04656*, 2018.
- [430] Martin Wistuba, Ambrish Rawat, and Tejaswini Pedapati. A survey on neural architecture search. *arXiv preprint arXiv:1905.01392*, 2019.
- [431] Sanghyun Woo, Jongchan Park, Joon-Young Lee, and In So Kweon. Cbam: Convolutional block attention module. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 3–19, 2018.
- [432] Daniel Worrall and Gabriel Brostow. Cubenet: Equivariance to 3d rotation and translation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 567–584, 2018.
- [433] Daniel E Worrall and Max Welling. Deep scale-spaces: Equivariance over scale. *arXiv preprint arXiv:1905.11697*, 2019.
- [434] Daniel E Worrall, Stephan J Garbin, Daniyar Turmukhambetov, and Gabriel J Brostow. Harmonic networks: Deep translation and rotation equivariance. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5028–5037, 2017.
- [435] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10734–10742, 2019.
- [436] Wenxuan Wu, Zhongang Qi, and Li Fuxin. Pointconv: Deep convolutional networks on 3d point clouds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9621–9630, 2019.
- [437] Zhijie Wu, Yuhe Jin, and Kwang Moo Yi. Neural fourier filter bank. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14153–14163, 2023.

- [438] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1912–1920, 2015.
- [439] Lingxi Xie, Xin Chen, Kaifeng Bi, Longhui Wei, Yuhui Xu, Lanfei Wang, Zhengsu Chen, An Xiao, Jianlong Chang, Xiaopeng Zhang, et al. Weight-sharing neural architecture search: A battle to shrink the optimization gap. *ACM Computing Surveys (CSUR)*, 54(9):1–37, 2021.
- [440] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1492–1500, 2017.
- [441] Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tieyan Liu. On layer normalization in the transformer architecture. In *International Conference on Machine Learning*, pages 10524–10533. PMLR, 2020.
- [442] Zhitong Xiong, Yuan Yuan, Nianhui Guo, and Qi Wang. Variational context-deformable convnets for indoor scene parsing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.
- [443] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.
- [444] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, pages 2048–2057, 2015.
- [445] Yifan Xu, Tianqi Fan, Mingye Xu, Long Zeng, and Yu Qiao. Spidercnn: Deep learning on point sets with parameterized convolutional filters. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 87–102, 2018.
- [446] Yong Xu, Qiuqiang Kong, Wenwu Wang, and Mark D Plumbley. Large-scale weakly supervised audio classification using gated convolutional neural network. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 121–125. IEEE, 2018.
- [447] Youjun Xu, Kangjie Lin, Shiwei Wang, Lei Wang, Chenjing Cai, Chen Song, Luhua Lai, and Jianfeng Pei. Deep learning for molecular generation. *Future medicinal chemistry*, 11(6):567–597, 2019.
- [448] Yuhui Xu, Lingxi Xie, Xiaopeng Zhang, Xin Chen, Guo-Jun Qi, Qi Tian, and Hongkai Xiong. Pc-darts: Partial channel connections for memory-efficient architecture search. In *International Conference on Learning Representations*, 2020. URL <https://openreview.net/forum?id=BJ1S634tPr>.

- [449] Omry Yadan. Hydra - a framework for elegantly configuring complex applications. Github, 2019. URL <https://github.com/facebookresearch/hydra>.
- [450] Jianke Yang, Robin Walters, Nima Dehmamy, and Rose Yu. Generative adversarial symmetry discovery. *arXiv preprint arXiv:2302.00236*, 2023.
- [451] Runzhao Yang. Tinc: Tree-structured implicit neural compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 18517–18526, 2023.
- [452] Li Yi, Vladimir G. Kim, Duygu Ceylan, I-Chao Shen, Mengyan Yan, Hao Su, Cewu Lu, Qixing Huang, Alla Sheffer, and Leonidas Guibas. A scalable active framework for region annotation in 3d shape collections. *ACM Trans. Graph.*, 35(6), dec 2016. ISSN 0730-0301. doi: 10.1145/2980179.2980238. URL <https://doi.org/10.1145/2980179.2980238>.
- [453] Robert K Yin. Looking at upside-down faces. *Journal of experimental psychology*, 81(1):141, 1969.
- [454] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015.
- [455] Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. Slimmable neural networks. *arXiv preprint arXiv:1812.08928*, 2018.
- [456] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.
- [457] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. *Advances in neural information processing systems*, 30, 2017.
- [458] Manzil Zaheer, Guru Guruganesh, Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. Big bird: Transformers for longer sequences. *arXiv preprint arXiv:2007.14062*, 2020.
- [459] Neil Zeghidour, Qiantong Xu, Vitaliy Liptchinsky, Nicolas Usunier, Gabriel Synnaeve, and Ronan Collobert. Fully convolutional speech recognition. *arXiv preprint arXiv:1812.06864*, 2018.
- [460] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.
- [461] Arber Zela, Thomas Elsken, Tommoy Saikia, Yassine Marrakchi, Thomas Brox, and Frank Hutter. Understanding and robustifying differentiable architecture search. *arXiv preprint arXiv:1909.09656*, 2019.
- [462] Cheng Zhang, Haocheng Wan, Xinyi Shen, and Zizhao Wu. Pvt: Point-voxel transformer for point cloud learning. *International Journal of Intelligent Systems*, 37(12):11985–12008, 2022.

- [463] Chiyuan Zhang, Stephen Voinea, Georgios Evangelopoulos, Lorenzo Rosasco, and Tomaso Poggio. Discriminative template learning in group-convolutional networks for invariant speech representations. In *Sixteenth Annual Conference of the International Speech Communication Association*, 2015.
- [464] Chris Zhang, Mengye Ren, and Raquel Urtasun. Graph hypernetworks for neural architecture search. *arXiv preprint arXiv:1810.05749*, 2018.
- [465] Han Zhang, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena. Self-attention generative adversarial networks. *arXiv preprint arXiv:1805.08318*, 2018.
- [466] Keming Zhang and Joshua S Bloom. deepcrr: cosmic ray rejection with deep learning. *The Astrophysical Journal*, 889(1):24, 2020.
- [467] Miao Zhang, Steven W. Su, Shirui Pan, Xiaojun Chang, Ehsan M Abbasnejad, and Reza Haffari. idarts: Differentiable architecture search with stochastic implicit gradients. 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 12557–12566. PMLR, 18–24 Jul 2021. URL <https://proceedings.mlr.press/v139/zhang21s.html>.
- [468] Richard Zhang. Making convolutional networks shift-invariant again. In *International conference on machine learning*, pages 7324–7334. PMLR, 2019.
- [469] Ruochi Zhang, Yuesong Zou, and Jian Ma. Hyper-{sagnn}: a self-attention based graph neural network for hypergraphs. In *International Conference on Learning Representations*, 2020. URL <https://openreview.net/forum?id=ryeHuJBtPH>.
- [470] Zhoutong Zhang, Yunyun Wang, Chuang Gan, Jiajun Wu, Joshua B. Tenenbaum, Antonio Torralba, and William T. Freeman. Deep audio priors emerge from harmonic convolutional networks. In *International Conference on Learning Representations*, 2020. URL <https://openreview.net/forum?id=rygjHxrYDB>.
- [471] Hengshuang Zhao, Jiaya Jia, and Vladlen Koltun. Exploring self-attention for image recognition. In *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.
- [472] Hengshuang Zhao, Li Jiang, Jiaya Jia, Philip HS Torr, and Vladlen Koltun. Point transformer. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 16259–16268, 2021.
- [473] Jingyu Zhao, Feiqing Huang, Jia Lv, Yanjie Duan, Zhen Qin, Guodong Li, and Guangjian Tian. Do rnn and lstm have long memory? In *International Conference on Machine Learning*, pages 11365–11375. PMLR, 2020.
- [474] Li Zhaoping. *The V1 hypothesis—creating a bottom-up saliency map for preattentive selection and segmentation*, pages 189–314. 05 2014. ISBN 9780199564668. doi: 10.1093/acprof:oso/9780199564668.003.0005.

- [475] Maksim Zhdanov, Nico Hoffmann, and Gabriele Cesa. Implicit neural convolutional kernels for steerable cnns. *arXiv preprint arXiv:2212.06096*, 2022.
- [476] Allan Zhou, Tom Knowles, and Chelsea Finn. Meta-learning symmetries by reparameterization. In *International Conference on Learning Representations*, 2020.
- [477] Hongpeng Zhou, Minghao Yang, Jun Wang, and Wei Pan. Bayesnas: A bayesian approach for neural architecture search. In *International conference on machine learning*, pages 7603–7613. PMLR, 2019.
- [478] Wei Zhu, Qiang Qiu, Robert Calderbank, Guillermo Sapiro, and Xiuyuan Cheng. Scaling-translation-equivariant networks with decomposed convolutional filters. *The Journal of Machine Learning Research*, 23(1):2958–3002, 2022.
- [479] Xizhou Zhu, Dazhi Cheng, Zheng Zhang, Stephen Lin, and Jifeng Dai. An empirical study of spatial attention mechanisms in deep networks. *arXiv preprint arXiv:1904.05873*, 2019.
- [480] Zhenyao Zhu, Jesse H Engel, and Awni Hannun. Learning multiscale features directly from waveforms. *arXiv preprint arXiv:1603.09509*, 2016.
- [481] Itamar Zimerman and Lior Wolf. Multi-dimensional hyena for spatial inductive bias. *arXiv preprint arXiv:2309.13600*, 2023.
- [482] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL <https://openreview.net/forum?id=r1Ue8Hcxg>.
- [483] 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*, pages 8697–8710, 2018.



# **APPENDIX**



# A

## Continuous Kernel Convolutions for Sequential Data

### A.1 Properties of CKConvs

#### A.1.1 Very Irregularly Sampled Data

CKConvs can readily handle irregularly-sampled, and partially observed data. This is a result of the convolutional kernel  $\text{MLP}^\psi$  being able to be sampled at arbitrary positions. For very non-uniformly sampled inputs, however, the corresponding sampling of the convolutional kernel can provide a biased estimation of the operation. To overcome this, one can follow the strategy proposed by Wu et al. [436], which we summarize here for completeness.

For very non-uniformly sampled inputs, the continuous convolution  $(x * \psi)(t) = \int_{\mathbb{R}} x(\tau)\psi(t - \tau) d\tau$ , must account for the distribution of samples in the input. Specifically, it is reformulated as:

$$(x * \psi)(t) = \int_{\mathbb{R}} s(\tau)x(\tau)\psi(t - \tau) d\tau, \quad (\text{A.1})$$

where  $s(\tau)$  depicts the inverse sample density of the input at point  $\tau$ . Intuitively,  $s(\tau)$  controls the contribution of points  $x(\tau)$  to the output response. If multiple points are close to one another, their contribution should be smaller than the contribution of points in regions where the sample distribution is much sparser. This provides a Monte Carlo

estimate of  $(x * \psi)$  from biased samples. In particular, one has that:

$$\int f(\tau) d\tau = \int \frac{f(\tau)}{p(\tau)} p(\tau) d\tau \approx \sum_i \frac{f(\tau_i)}{p(\tau_i)}, \text{ for } \tau_i \sim p(\tau).$$

With  $s(\tau) = \frac{1}{p(\tau)}$ , Eq. A.1 provides an unbiased estimation of  $(x * \psi)$ .

### A.1.2 Data Sampled at Different Sampling Rates

In addition, CKConvs are readily able to handle data at different resolutions. In particular, the continuous kernel convolution between an input signal  $x$  and a continuous convolutional kernel  $\psi$  calculated at sampling rates  $sr_1$ :  $(x * \psi)_{sr_1}$ , and  $sr_2$ :  $(x * \psi)_{sr_2}$ , are approximately equal up to a normalization factor given by  $\frac{sr_2}{sr_1}$ :

$$(x * \psi)_{sr_2}(t) \approx \frac{sr_2}{sr_1} (x * \psi)_{sr_1}(t).$$

Consequently, CKCNNs (i) can be deployed at sampling rates different than those seen during training, and (ii) can be trained on data with varying spatial resolutions. The later is important for tasks in which data can be given at different resolutions such as super-resolution and segmentation.

*Proof.* To prove the previous statement, we start with the continuous definition of the convolution:

$$(x * \psi)(t) = \int_{\mathbb{R}} x(\tau) \psi(t - \tau) d\tau,$$

where we assume for simplicity and without loss of generality that the functions  $x, \psi$  are scalar-valued.

In practice, an integral on a continuous function  $f : \mathbb{R} \rightarrow \mathbb{R}$  cannot be computed on finite time. Consequently, it is approximated via a Riemann integral defined on a finite grid  $\{\tau_{sr,i}\}_{i=1}^{N_{sr}}$  obtained by sampling  $\tau$  at a sampling rate  $sr$ :

$$\int f(\tau) d\tau \approx \sum_{i=1}^{N_{sr}} f(\tau_{sr,i}) \Delta_{sr},$$

where  $\Delta_{sr} = \frac{1}{sr}$  depicts the distance between sampled points. For two sampling rates  $sr_1, sr_2$ , the convolution can be approximated through the corresponding Riemann integrals:

$$\begin{aligned} \int_{\mathbb{R}} x(\tau) \psi(t - \tau) d\tau &\approx \sum_{i=1}^{N_{sr_1}} x(\tau_{sr_1,i}) \psi(t - \tau_{sr_1,i}) \Delta_{sr_1} \\ &\approx \sum_{i=1}^{N_{sr_2}} x(\tau_{sr_2,i}) \psi(t - \tau_{sr_2,i}) \Delta_{sr_2} \end{aligned}$$

As a result, we have that both approximations are approximately equal to the continuous integral at positions  $t$  defined on both discrete grids. By equating both approximations,

we obtain that:

$$\begin{aligned} \sum_{i=1}^{N_{sr_2}} x(\tau_{sr_2,i}) \psi(t - \tau_{sr_2,i}) \Delta_{sr_2} &\approx \sum_{i=1}^{N_{sr_1}} x(\tau_{sr_1,i}) \psi(t - \tau_{sr_1,i}) \Delta_{sr_1} \\ \underbrace{\sum_{i=1}^{N_{sr_2}} x(\tau_{sr_2,i}) \psi(t - \tau_{sr_2,i})}_{(x * \psi)_{sr_2}(t)} \frac{1}{sr_2} &\approx \underbrace{\sum_{i=1}^{N_{sr_1}} x(\tau_{sr_1,i}) \psi(t - \tau_{sr_1,i})}_{(x * \psi)_{sr_1}(t)} \frac{1}{sr_1} \\ (x * \psi)_{sr_2}(t) &\approx \frac{sr_2}{sr_1} (x * \psi)_{sr_1}(t) \end{aligned}$$

which concludes the proof.

### A.1.3 Linear Recurrent Units Are CKConvs

Interesting insights can be obtained by drawing connections between convolutions and recurrent units. In particular, we can show that linear recurrent units are equal to a CKConv with a particular family of convolutional kernels: exponential functions. Besides providing a generalization to recurrent units, this equality provides a fresh and intuitive view to the analysis of vanishing and exploding gradients.

**Recurrent unit.** Given an input sequence  $\mathcal{X} = \{\mathbf{x}(\tau)\}_{\tau=0}^{N_X}$ , a recurrent unit is defined as:

$$\mathbf{h}(\tau) = \sigma(\mathbf{W}\mathbf{h}(\tau-1) + \mathbf{U}\mathbf{x}(\tau)) \quad (\text{A.2})$$

$$\tilde{\mathbf{y}}(\tau) = \text{softmax}(\mathbf{V}\mathbf{h}(\tau)), \quad (\text{A.3})$$

where  $\mathbf{U}, \mathbf{W}, \mathbf{V}$  parameterize the *input-to-hidden*, *hidden-to-hidden* and *hidden-to-output* connections of the unit.  $\mathbf{h}(\tau)$ ,  $\tilde{\mathbf{y}}(\tau)$  depict the hidden representation and the output at time-step  $\tau$ , and  $\sigma$  represents a point-wise non-linearity.

The hidden representation  $\mathbf{h}$  of a *linear* recurrent unit, i.e., with  $\sigma=\text{Id}$ , can be written as a convolution. To see this, consider the hidden representation of the unit unrolled for  $t$  steps, with  $\mathbf{h}(-1)$  the initial state of the hidden representation:

$$\mathbf{h}(t) = \mathbf{W}^{t+1}\mathbf{h}(-1) + \sum_{\tau=0}^t \mathbf{W}^\tau \mathbf{U} \mathbf{x}(t-\tau). \quad (\text{A.4})$$

We see that in fact it corresponds to a convolution between an input signal  $\mathbf{x}$  and a convolutional kernel  $\psi$  given by:<sup>1</sup>

$$\mathbf{x} = [\mathbf{x}(0), \mathbf{x}(1), \dots, \mathbf{x}(t-1), \mathbf{x}(t)] \quad (\text{A.5})$$

$$\psi = [\mathbf{U}, \mathbf{W}\mathbf{U}, \dots, \mathbf{W}^{t-1}\mathbf{U}, \mathbf{W}^t\mathbf{U}] \quad (\text{A.6})$$

$$\mathbf{h}(t) = \sum_{\tau=0}^t \mathbf{x}(\tau) \psi(t-\tau) = \sum_{\tau=0}^t \mathbf{x}(t-\tau) \psi(\tau). \quad (\text{A.7})$$

Drawing this equality yields some important insights:

**The cause of the exploding and vanishing gradients.** Eqs. A.5-A.7 intuitively depict the

---

<sup>1</sup>We discard  $\mathbf{h}(-1)$  as it only describes the initialization of  $\mathbf{h}$ .

root of the exploding and vanishing gradient problem. It stems from sequence elements  $x(t - \tau)$   $\tau$  steps back in the past being multiplied with an effective convolutional weight  $\psi(\tau) = W^\tau U$ . For eigenvalues of  $W$ ,  $\lambda$ , other than one, the resulting convolutional kernel  $\psi$  can only represent functions that either grow ( $\lambda \geq 1$ ) or decrease ( $\lambda \leq 1$ ) exponentially as a function of the sequence length (Figs. 2.3a, 2.3b). As a result, the contribution of input values in the past either rapidly fades away or governs the updates of the model parameters. As exponentially growing gradients lead to divergence, the eigenvalues of  $W$  for converging architectures are often smaller than 1. This explains the effective small effective memory horizon of recurrent networks.

**Linear recurrent units are a subclass of CKConvs.** Linear recurrent units can be described as a convolution between the input and a very specific class of convolutional kernels: exponential functions (Eq. A.6). In general, however, convolutional kernels are not restricted to this functional class. This can be seen in conventional (discrete) convolutions, whose kernels are able to model complex functions within their memory horizon. Unfortunately, discrete convolutions use a predefined, small kernel size, and thus possess a restricted memory horizon. This is equivalent to imposing an effective magnitude of zero to all input values outside the memory horizon (Fig. 2.3c). CKConvs, on the other hand, are able to define arbitrary large memory horizons. For memory horizons of size equal to the input length, CKConvs are able to model complex functions upon the entire input (Fig. 2.3d).

In conclusion, we illustrate that CKConvs are also a generalization of (linear) recurrent architectures which allows for parallel training and enhanced expressivity.

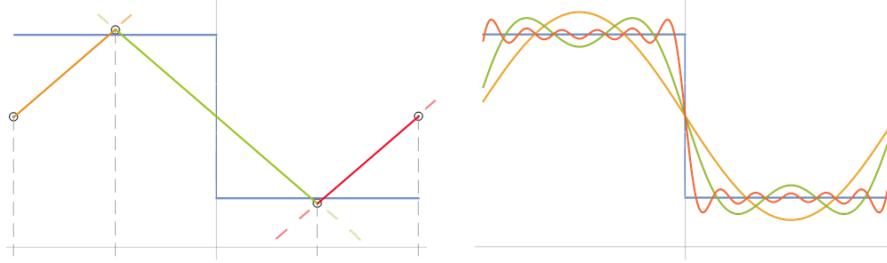
## A.2 An Spline Interpretation Of ReLU and Sine Networks

Sitzmann et al. [358] motivates the usage of Sine nonlinearities for implicit neural representations. However, there is no clear understanding as of *why* Sine nonlinearities are better suited for this task than (smooth) piece-wise nonlinearities. Here, we provide an interpretation to this phenomenon from a spline function approximation perspective.

### A.2.1 Kernel Parameterization via ReLU Networks

**The importance of initialization.** There is an important distinction between implicit neural representations and conventional neural applications regarding the assumed distribution of the input. Conventional applications assume the distribution of the input features to be centered around the origin. This is orthogonal to implicit neural representations, where the spatial distribution of the output, i.e., the value of the function being implicitly represented, is *uniformly distributed*.

For ReLU networks, function approximation is equivalent to an approximation via a max-spline basis [13], and its expressiveness is determined by the number of *knots* the basis provides, i.e., places where a non-linearity bends the space. Naturally, the better



**Figure A.1:** A step function approximated via a spline basis (left) and a periodic basis (right). As the target function is defined uniformly on a given interval, uniformly initializing the knots of the spline basis provides faster and better approximations. Periodic bases, on the other hand, periodically bend space, and thus can be tuned easier to approximate the target function at arbitrary points in space.

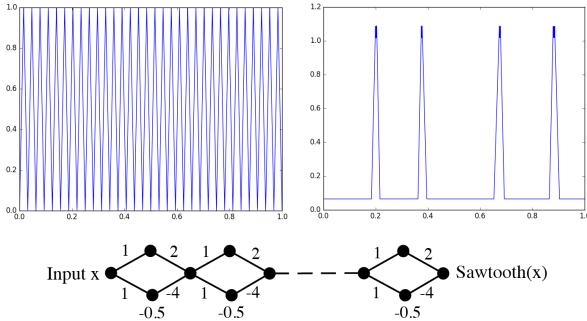
the placing of these knots at initialization, the faster the approximation may converge. For applications in which the data is centered around zero, initializing the knots around zero is a good inductive bias.<sup>2</sup> However, for spatially uniform distributed inputs, the knots should be uniformly distributed (Fig. A.1). As a result, conventional initializations lead to very poor reconstructions (ReLU 0-Init, Fig. 2.4), and explicitly aggregating positional encodings to the mappings leads to important improvements, e.g., [262].

For ReLU layers  $y = \max\{0, \mathbf{W}\mathbf{x} + \mathbf{b}\}$  knots appear at the point where  $0 = \mathbf{W}\mathbf{x} + \mathbf{b}$ . To place the knots at  $\mathbf{x}=0$ , it is sufficient to set the bias to zero:  $\mathbf{b}=0$ . For uniformly distributed knots in a range  $[\mathbf{x}_{\min}, \mathbf{x}_{\max}]$ , however, one must solve the ReLU equation for uniformly distributed points in that range:  $0 = \mathbf{W}\mathbf{x}_{\text{unif}} + \mathbf{b}$ . It results that  $\mathbf{b} = -\mathbf{W}\mathbf{x}_{\text{unif}}$ , for arbitrary values of  $\mathbf{W}$ .

In multilayered networks, the approximation problem can be understood as reconstructing the target function in terms of a basis  $\mathbf{h}^{(L-1)}$ . Consequently, the expressivity of the network is determined by the number of knots in  $\mathbf{h}^{(L-1)}$ . In theory, each ReLU layer is able to divide the linear regions of the previous layer in exponentially many sub-regions [265, 347], or equivalently, to induce an exponential layer-wise increase in the number of knots. For the first layer, the positions of the knots are described by the bias term, and for subsequent layers, these positions also depend on  $\mathbf{W}^{(l)}$ . Unfortunately, as depicted by Hanin and Rolnick [138], slight modifications of  $\{\mathbf{W}^{(l)}, \mathbf{b}^{(l)}\}$  can strongly simplify the landscape of the linear regions, and thus the knots (Fig. A.2). More importantly, Hanin and Rolnick [138] showed that the number of linear regions at initialization is actually equal to a constant times the number of neurons in the network (with a constant very close to one in their experiments). In addition, they show that this behavior barely changes throughout training.

**An improved initialization scheme.** Following the previous reasoning, we explore in-

<sup>2</sup>This is why  $\mathbf{b}=0$  is common in regular initialization schemes.



**Figure A.2:** The sensitivity of networks with layer-wise exponential growing to slight changes. Taken from Hanin and Rolnick [138]. The sawtooth function with  $2^n$  teeth (left) can be easily expressed via a ReLU network with  $3n + 4$  neurons (bottom). However, a slight perturbation of the network parameters –Gaussian noise with standard deviation 0.1– greatly simplifies the linear regions captured by the network, and thus the distribution of the knots in the basis (right).

ducing a uniformly distributed initialization of the knots. However, we observe that finding an initialization with an exponential number of knots is a cumbersome and unstable procedure. In fact, it is not always possible, and, whenever possible, it strongly restricts the values the weights  $\mathbf{W}^{(l)}$  can assume.

Following the findings of Hanin and Rolnick [138], we instead employ an initialization procedure with which the total number of knots is equal to the number of neurons of the network. This is obtained by replicating the initialization procedure of the first layer throughout the network: For randomly initialized weights  $\mathbf{W}^{(l)}$ , the bias term  $\mathbf{b}^{(l)}$  is given by the equality  $\mathbf{b}^{(l)} = -\mathbf{W}^{(l)}\mathbf{h}^{(l)}(\mathbf{x}_{\text{unif}})$ , where  $\mathbf{x}_{\text{unif}}$  is a vector of uniformly distributed points in  $[\mathbf{x}_{\text{min}}, \mathbf{x}_{\text{max}}]$ . Interestingly, we observe that this initialization strategy consistently outperforms the standard initialization for a large range of target functions (ReLU Unif-Init, Fig. A.3). However, we note that ReLU networks still show large difficulties in representing very nonlinear and non-smooth functions. In Fig. A.3, we illustrate that other popular nonlinearities: LeakyReLU, Swish, exhibit the same behavior.

### A.2.2 Kernel Parameterization via Sine Networks

Recently, Sitzmann et al. [358] proposed to replace ReLU nonlinearities by Sine for the task of implicit neural representation learning. Based on their relation with implicit neural representations, we explore using Sine networks to parameterize our continuous kernels. Intriguingly, we observe that this slight modification allows our kernels to approximate *any* provided function to near perfection, and leads to a consistent improvement for all tasks considered in this paper (Appx. A.4.1, Fig. A.3).

A possible explanation for these impressive results can be given via our prior analysis:

**Periodic bending of the space.** A Sine layer is given by:  $\mathbf{y} = \text{Sin}(\omega_0[\mathbf{W}\mathbf{x} + \mathbf{b}])$ , where  $\omega_0$  works as a prior on the variability of the target function. Orthogonal to ReLU layers,

Sine layers periodically bend the space. As a result, the same  $\mathbf{y}$  value is obtained for all bias values  $\mathbf{b}'_i = \mathbf{b}_i + n2\pi\|\mathbf{W}_{i,:}\|^{-1}$ ,  $\forall n \in \mathbb{Z}$ . This is important from a spline approximation perspective. While for ReLU layers a unique value of  $\mathbf{b}$  exists that bends the space at a desired position, infinitely many values of  $\mathbf{b}$  do so for Sine ones. Resultantly, Sine layers are much more robust to parameter selection, and can be tuned to benefit pattern approximation at arbitrary –or even multiple– positions in space (Fig. A.1, right). We conjecture that this leads to more reliable approximations and faster convergence.

**An exponentially big Fourier basis.** It is not surprising for a (large) basis of phase-shifted sinusoidal functions to be able to approximate arbitrary functions with high fidelity. This result was first observed over two centuries ago by Fourier [108] and lies at the core of the well-known *Fourier transform*: any integrable function can be described as a linear combination of a (possibly) infinite basis of phase-shifted sinusoidal functions. Sitzmann et al. [358] proposed an initialization of  $\{\mathbf{W}^{(l)}\}$  that allows for the construction of deep Sine networks able to periodically divide the space into exponentially many regions as a function of depth. Intuitively, approximations via Sine networks can be seen in terms of an exponentially large Fourier-like basis. We conjecture that this exponential growth combined with the periodicity of sine is what allows for excellent approximations: the more terms in a Fourier transform, the better the approximation becomes.

Interestingly, we find that a uniformly distributed initialization of the bias term  $\mathbf{b}_i \sim \mathcal{U}(-\pi\|\mathbf{W}_{i,:}\|^{-1}, \pi\|\mathbf{W}_{i,:}\|^{-1})$  also leads to better and faster convergence for Sine networks.

### A.3 Dataset Description

**Copy Memory Problem.** The copy memory task consists of sequences of length  $T+20$ , for which the first 10 values are chosen randomly among the digits  $\{1, \dots, 8\}$ , the subsequent  $T-1$  digits are set to zero, and the last 11 entries are filled with the digit 9. The goal is to generate an output of the same size of the input filled with zeros everywhere except for the last 10 values, for which the model is expected to predict the first 10 elements of the input sequence.

**The Adding Problem.** The adding problem consists of input sequences of length  $T$  and depth 2. The first dimension is filled with random values in  $[0, 1]$ , whereas the second dimension is set to zeros except for two elements marked by 1. The objective is to sum the random values for which the second dimension is equal to 1. Simply predicting the sum to be 1 results in a MSE of about 0.1767.

**Sequential and Permuted MNIST.** The MNIST dataset [208] consists of 70K gray-scale  $28 \times 28$  handwritten digits divided into training and test sets of 60K and 10K samples, respectively. The sequential MNIST dataset (sMNIST) presents MNIST images as a sequence of 784 pixels for digit classification. Consequently, good predictions require preserving long-term dependencies up to 784 steps in the past: much longer than most language modelling tasks [12].

The permuted MNIST dataset (pMNIST) additionally permutes the order or the sMNIST sequences at random. Consequently, models can no longer rely on local features to perform classification. As a result, the classification problem becomes more difficult and the importance of long-term dependencies more pronounced.

**Sequential CIFAR10.** The CIFAR10 dataset [195] consists of 60K real-world  $32 \times 32$  RGB images uniformly drawn from 10 classes divided into training and test sets of 50K and 10K samples, respectively. Analogously to the sMNIST dataset, the sequential CIFAR10 (sCIFAR10) dataset presents CIFAR10 images as a sequence of 1024 pixels for image classification. This dataset is more difficult than sMNIST, as (*i*) even larger memory horizons are required to solve the task, and (*ii*) more complex structures and intra-class variations are present in the images [386].

**CharacterTrajectories.** The CharacterTrajectories dataset is part of the UEA time series classification archive [9]. It consists of 2858 time series of different lengths and 3 channels representing the  $x, y$  positions and the pen tip force while writing a Latin alphabet character in a single stroke. The goal is to classify which of the different 20 characters was written using the time series data. The maximum length of the time-series is 182.

**Speech Commands.** The Speech Commands dataset [421] consists of 105809 one-second audio recordings of 35 spoken words sampled at 16kHz. Following Kidger et al. [183], we extract 34975 recordings from ten spoken words to construct a balanced classification problem. We refer to this dataset as **SC\_raw**. In addition, we utilize the preprocessing steps of Kidger et al. [183] and extract mel-frequency cepstrum coefficients from the raw data. The resulting dataset, named **SC**, consists of time series of length 161 and 20 channels.

**PhysioNet.** The PhysioNet 2019 challenge on sepsis prediction [121, 307] is a irregularly sampled, partially observed dataset consisting of 40335 time series of variable length describing the stay of patients within an ICU. Time-series are made out of 5 static features, e.g., age, and 34 time-dependent features, e.g., respiration rate, creatinine blood concentration, and 10.3% of the values are observed. We follow Kidger et al. [183] and consider the first 72 hours of a patient’s stay to predict whether sepsis is developed over the course of their entire stay –which can extend for a month for some patients–.

**PennTreeBank.** The PennTreeBank (PTB) [251] is a language corpus which consists of 5,095K characters for training, 396K for validation and 446K for testing. On a char lever that we use in our experiment the vocabulary size is 50 characters (or the size of the alphabet, including end-of-string char). We follow [11] in performing character-level language modeling task on this dataset.

## A.4 Ablation Studies

In this section, we perform an ablative study of our approach. Specifically, we analyze the effect of multiple components of our network, and provide additional comparisons

with alternative architectures. Specifications on the architectures and hyperparameters used are given in Appx. A.5.

#### A.4.1 Using Sine Non-Linearities Over Popular Alternatives

As shown in Sec. 2.4.2, Sine nonlinearities provide astonishing improvements over equivalent networks with ReLU nonlinearities for function reconstruction. In this section, we provide additional experiments to highlight the suitability of Sine nonlinearities over other popular alternatives both for function approximation and the rest of the tasks considered in this work. The same architectures are used across all experiments and vary only in the nonlinearity used in  $\text{MLP}^\psi$ . We find that nonlinearities other than Sine benefit from layer normalization and thus we incorporate it in these variants.

**Case I: Function Approximation via  $\text{MLP}^\psi$ .** First, we evaluate the problem of function approximation in Sec. 2.4.2, Fig. 2.4, for nonlinearities other than ReLU and Sine. In particular, we approximate several functions with a  $\text{MLP}^\psi$  network which varies only in the type of nonlinearity used: ReLU [268], LeakyReLU [443], Swish [300], and Sine [358].

Our results (Fig. A.3), illustrate that Sine provides astonishing approximation capabilities over all other nonlinearities considered. In particular, we observe that Sine is the only nonlinearity able to reconstruct very nonlinear and very non-smooth functions, while all other alternatives fail poorly.

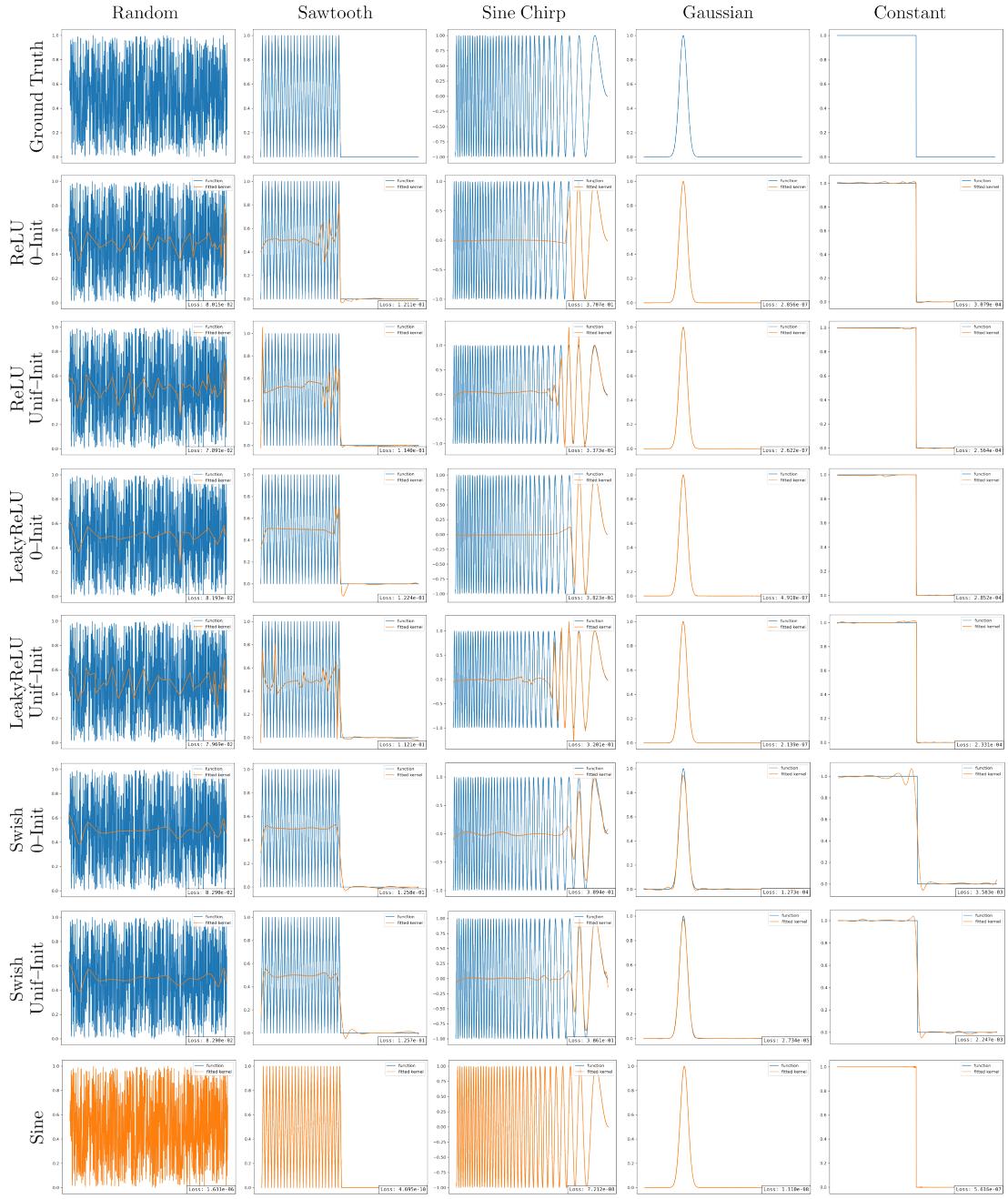
**Case II: CKCNNs with nonlinearities other than Sine.** Next, we consider the case in which CKCNNs with nonlinearities other than Sine are used to solve the tasks considered in Sec. 2.5. In particular, we train CKCNNs on sMNIST, pMNIST, SC and SC\_raw for four different nonlinearities: ReLU, LeakyReLU, Swish, Sine. We utilize the same backbone architecture used in the main text for the corresponding dataset.

Our results (Tab. A.1) indicate that CKCNNs relying on Sine nonlinearities outperform CKCNNs relying on all other nonlinearities.

**Analysis of the results.** Our findings indicate Sine is much better suited to describe continuous spatial functions via neural networks than all other nonlinearities considered. This result motivates replacing popular nonlinearities by Sine for applications in which neural networks are used to describe continuous positional functions. This family of models encompass –but is not restricted to– continuous types of convolutions, e.g., Finzi et al. [105], Fuchs et al. [111], Schütt et al. [343], Thomas et al. [382], as well as positional encodings in transformers, e.g., Dai et al. [76], Ramachandran et al. [301], Romero and Cordonnier [314], and graph neural networks, e.g., Defferrard et al. [82]. We consider this result to be of large relevance to the deep learning community.

#### A.4.2 Going Deeper with CKCNNs

The experimental results shown in Sec. 2.5 are obtained by shallow CKCNNs with 2 residual blocks. An interesting question is whether going deeper can be used to improve



**Figure A.3:** Function approximation via ReLU, LeakyReLU, Swish and Sine networks. All network variants perform a decent job in approximating simple functions. However, for non-linear, non-smooth functions, all networks using nonlinearities other than Sine provide very poor approximations. Interestingly, the uniform knot initialization proposed in Sec. 2.4.2 provides consistent improvements for all network variants. However, despite this improvement, the approximation results remain insufficient. Contrarily, Sine networks quickly and seamlessly approximate all functions. All network configurations are equal up to the non-linearities used.

**Table A.1:** Test accuracies of CKCNNs with different nonlinearities. Model size = 100K.

| NON-LINEARITY | DATASET      |              |              |              |
|---------------|--------------|--------------|--------------|--------------|
|               | SMNIST       | PMNIST       | SC           | SC_RAW       |
| RELU          | 81.21        | 59.15        | 94.97        | 49.15        |
| LEAKYRELU     | 80.57        | 55.85        | 95.03        | 38.67        |
| SWISH         | 85.20        | 61.77        | 93.43        | 62.23        |
| SINE          | <b>99.31</b> | <b>98.00</b> | <b>95.27</b> | <b>71.66</b> |

**Table A.2:** Test accuracy of CKCNNs for various depths and widths.

| PMNIST    |            |               |            |              |
|-----------|------------|---------------|------------|--------------|
| DEPTH     | FIXED SIZE | WIDTH ACC.(%) | FIXED SIZE | ACC.(%)      |
| 2 Blocks  | 98k        | 99.21         | 98k        | <b>99.21</b> |
| 4 Blocks  | 225k       | <b>99.26</b>  | 95k        | <b>99.19</b> |
| 8 Blocks  | 480k       | <b>99.29</b>  | 105k       | 99.12        |
| 16 Blocks | 990k       | 99.19         | 107k       | 99.02        |

the performance of CKCNNs. To analyze this, we compare deep and shallow CKCNNs with the same architecture for equal width, and equal number of parameters.

Our results (Tab. A.2) indicate that constructing deeper CKCNNs do not lead to accuracy improvements. In fact, we observe that deep CKCNNs underperform shallow CKCNNs of equal size. This is an interesting results as CKCNNs do not strongly rely on deep-wise compositionality of features: largely considered indispensable in deep learning.

**Analysis of the results.** The dynamics governing these results are not fully understood. Our findings may lead to two different conclusions, both of which we consider important for the development and understanding of CKCNNs and deep learning in general:

**Outcome I: Deep CKCNNs.** The first possible outcome is that our current parameterization does not correctly leverage depth. In this case, efforts to construct proper *deep* CKCNNs will likely lead to performance improvements over the current architectures, and thus have the potential to advance the state-of-the-art further.

**Outcome II: Depth is not needed when global memory horizons are provided with shallow networks.** The second possible outcome is that depth is used mainly as a means to construct global memory horizons. Consequently, neural networks do not have to be very deep at all provided that global memory horizons are defined by shallow neural networks. Interestingly, this conclusion is in line with the predominant design of recurrent architectures, for which a moderate number of layers are used, e.g., [124, 126, 127, 281]. This possible outcome is very exciting as depth is largely considered indispensable in the deep learning community.

## A.5 Experimental Details

In this section, we provide extended details over our implementation as well as the exact architectures and optimization schemes used in our experiments.

### A.5.1 General Remarks

Our models follow the structure shown in Fig. A.4 and vary only in the number of channels. We use layer normalization [8] in our backbone network, and use the Adam optimizer [185] across all our experiments. Our code is implemented in PyTorch and is

publicly available at *link removed for the sake of the double-blind review*. We utilize `wandb` [29] to log our results, and use NVIDIA TITAN RTX GPUs throughout our experiments.

**Continuous Convolutional Kernels**  $\text{MLP}^\psi$ . All our convolutional kernels are parameterized by a 3-layer neural network with 32 hidden units and Sine nonlinearities:

$$1 \rightarrow 32 \rightarrow 32 \rightarrow N_{\text{out}} \times N_{\text{in}}.$$

Here,  $N_{\text{in}}$ ,  $N_{\text{Cout}}$  are the number of input and output channels of the convolutional layer. We utilize weight normalization [333] in our  $\text{MLP}^\psi$  networks, and select a hidden size of 32 based on empirical evidence and findings from previous work [105].

**Normalized relative positions.** The MLPs parameterizing our convolutional kernels receive relative positions as input. However, considering unitary step-wise relative positions, i.e., 0, 1, 2, ...,  $N$ , can be problematic from a numerical stability perspective as  $N$  may grow very large, e.g.,  $N=16000$  for the SC\_raw dataset. Consequently, we follow good practices from works modelling continuous functions with neural networks, and map the largest unitary step-wise relative positions seen during training  $[0, N]$  to a uniform linear space in  $[-1, 1]$ .

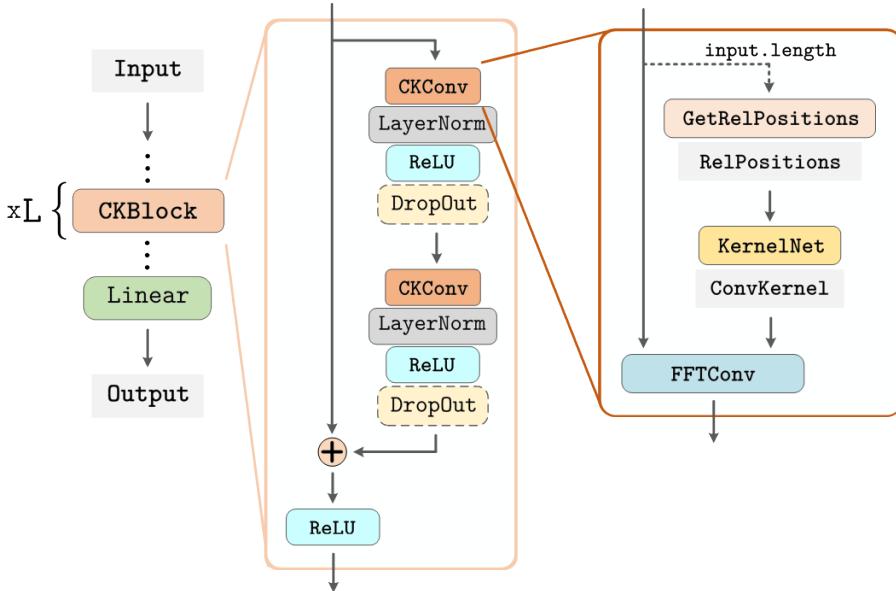
**Hyperparameter tuning.** We tune the hyperparameters of our models via the `bayes` method given in `wandb` Sweeps, which selects hyperparameter values via a Gaussian process over the results obtained so far. We perform tuning on a validation dataset until a predefined maximum number of runs of 100 is exhausted. Further improvements upon our results may be obtained by leveraging more sophisticated tuning methods as well as additional runs.

**Selecting  $\omega_0$ .** CKCNNs are very susceptible to the value of  $\omega_0$ . In order to obtain a reasonable  $\omega_0$ , we first perform a random search on a large interval  $\omega_0 \in [0, 3000]$ . After a few runs, we stop the random search and select the subinterval in which the validation accuracy is most promising. Next, we restart the random search on this sub-interval and repeat the process until a  $\omega_0$  value is obtained, for which the validation accuracy is sufficiently high. Surprisingly, we found optimal values of  $\omega_0$  to be always enclosed in the interval  $[1, 70]$  even for very long sequences as in SC\_raw.

### A.5.2 Accounting for Spatial Displacements of the Sampled Convolutional Kernels

We follow the sampling procedure of Gu et al. [126] throughout our test sampling rate discrepancy experiments. Specifically, for a sequence `seq` of length  $N$ , subsampling by a factor  $n$  is performed by running `seq[::n]`. That is, by taking the  $n$ -th element of the sequence starting from its first element. For example, for a sequence of length  $N=182$ , different values of  $n$  would yield the following sequences:

$$\begin{aligned} (n = 1) &\rightarrow [1, 2, 3, \dots, 180, 181, 182] \\ (n = 2) &\rightarrow [1, 3, 5, \dots, 177, 179, 181] \end{aligned}$$



**Figure A.4:** Graphical description of continuous kernel convolutional networks. Dotted-line blocks depict optional blocks, and blocks without borders depict variables. KernelNet blocks use Sine nonlinearities. We replace spatial convolutions by Fourier Convolutions (FFTConv), which leverages the convolution theorem to speed up computations.

$$\begin{aligned} (n = 4) &\rightarrow [1, 5, 9, \dots, 173, 177, 181] \\ (n = 8) &\rightarrow [1, 9, 17, \dots, 161, 169, 177] \end{aligned}$$

Recall that  $\text{MLP}^\psi$  takes normalized relative positions in  $[-1, 1]$  as input, which are computed based on the max input length seen during training. However, some of these subsampling transitions *change* the max value of the sequence, e.g., for  $(n = 8)$  the maximum is given by 177, whereas for  $(n = 1)$  this value corresponds to 182. Consequently, a naive approach would consider the last position in each subsampled sequence to correspond to the maximum normalized relative position 1. This effectively induces displacement and re-scaling of the sampled convolutional kernel used during training.

This misalignment is automatically handled under the hood in our CKConv implementation, but we highlight this subtlety to prevent it in future applications.

### A.5.3 Dealing with High-Frequency Components

Interestingly, our experiments revealed that our continuous kernels often contain frequency components of frequency higher than the resolution of the sampling grid used during training (Fig. A.5). As these high-frequency components are not observed during training, we observe that they hurt performance when evaluated at higher resolutions.

In order to neutralize their influence, we filter these components before doing the convolution by means of blurring. This is done by applying a convolution upon the convo-

**Table A.3:** Hyperparameter specifications of the best performing CKCNN models.

| PARAMS.           | COPY MEMORY     | ADDING PROBLEM  | S <small>MNIST</small><br>Small / Big | P <small>MNIST</small><br>Small / Big | S <small>CIFAR10</small><br>Small / Big | CT <sup>†</sup> | SC      | SC_RAW <sup>†</sup> | PTB     |
|-------------------|-----------------|-----------------|---------------------------------------|---------------------------------------|---|-----------------|---------|---------------------|---------|
| Epochs            | See Appx. A.5.4 | See Appx. A.5.4 | 200                                   | 200                                   | 200                                     | 200             | 200     | 300                 | 200     |
| Batch Size        | 32              | 32              | 64                                    | 64                                    | 64                                      | 32              | 64      | 32                  | 24      |
| Optimizer         | Adam            | Adam            | Adam                                  | Adam                                  | Adam                                    | Adam            | Adam    | Adam                | Adam    |
| Learning Rate     | 5e-4            | 0.001           | 0.001                                 | 0.001                                 | 0.001                                   | 0.001           | 0.001   | 0.001               | 0.002   |
| # Blocks          | 2               | 2               | 2                                     | 2                                     | 2                                       | 2               | 2       | 2                   | 2       |
| Hidden Size       | 10              | 25              | 30 / 100                              | 30 / 100                              | 30 / 100                                | 30              | 30      | 30                  | 128     |
| $\omega_0$        | See Appx. A.5.4 | See Appx. A.5.4 | 31.09 / 30.5                          | 43.46 / 42.16                         | 25.67                                   | 21.45           | 30.90   | 39.45               | 25.78   |
| Dropout           | -               | -               | 0.1 / 0.2                             | -                                     | 0.2 / 0.3                               | 0.1             | 0.2     | -                   | -       |
| Input Dropout     | -               | -               | 0.1 / 0.2                             | 0.1 / 0.2                             | 0.0 / 0.0                               | -               | -       | -                   | 0.1     |
| Embedding Dropout | -               | -               | -                                     | -                                     | -                                       | -               | -       | -                   | 0.1     |
| Weight Dropout    | -               | -               | -                                     | -                                     | - / 0.1                                 | -               | -       | -                   | -       |
| Weight Decay      | -               | -               | -                                     | -                                     | - / 1e-4                                | -               | -       | 1e-4                | 1e-6    |
| Scheduler         | -               | -               | Plateau                               | Plateau                               | Plateau                                 | Plateau         | Plateau | Plateau             | Plateau |
| Patience          | -               | -               | 20                                    | 20                                    | 20                                      | 20              | 15      | 20                  | 5       |
| Scheduler Decay   | -               | -               | 5                                     | 5                                     | 5                                       | 5               | 5       | 5                   | 5       |
| Model Size        | 15.52K          | 70.59K          | 98.29K / 1.03M                        | 98.29K / 1.03M                        | 100.04K / 1.04M                         | 100.67K         | 118.24K | 98.29K              | 1.8M    |

<sup>†</sup>Hyperparameter values for the classification and varying sampling rate tasks. For hyperparameters w.r.t. irregularly-sampled data please see Tab. A.4.

lutional kernel with a Gaussian filter  $G$  of length  $2 \frac{\text{sr}_{\text{test}}}{\text{sr}_{\text{train}}} + 1$  and parameters  $\mu=0$ ,  $\sigma=0.5$ :

$$\left[ G\left(-\frac{\text{sr}_{\text{test}}}{\text{sr}_{\text{train}}}\right), G\left(-\frac{\text{sr}_{\text{test}}}{\text{sr}_{\text{train}}} + 1\right), \dots, G(0), \dots, G\left(\frac{\text{sr}_{\text{test}}}{\text{sr}_{\text{train}}} - 1\right), G\left(\frac{\text{sr}_{\text{test}}}{\text{sr}_{\text{train}}}\right) \right]$$

Note that blurring is only used when the test sampling rate is higher than the train sampling rate, as opposed to the normalization factor  $\frac{\text{sr}_{\text{test}}}{\text{sr}_{\text{train}}}$  discussed in Eq. 2.5, Appx. A.1.2, which is applied whenever the sampling rates differ.

#### A.5.4 Hyperparameters and Experimental Details

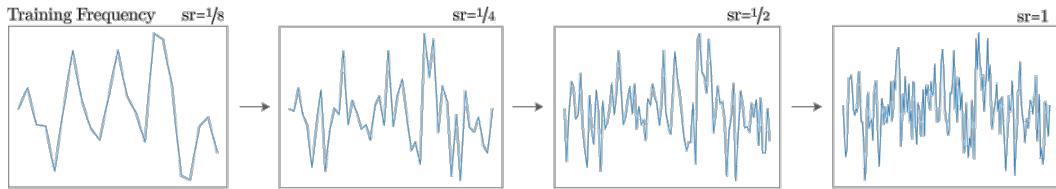
In this section, we specify the hyperparameter configurations with which our models are trained. An overview of these hyperparameters is provided in Tab. A.3.

**Copy Memory.** We set the number of channels of our CKCNN as to roughly match the number of parameters of the GRU and TCN networks of Bai et al. [11]. This is obtained with 10 hidden channels at every layer. We observe that the time to convergence grew proportional to the length of the sequence considered. Whereas for sequences of length 100 convergence was shown after as few as 10 epochs, for sequences of length 6000 approximately 250 epochs were required. The maximum number of epochs is set to 50, 50, 100, 200 and 300 for sequences of size 100, 200, 1000, 3000 and 6000. We observe that different values of  $\omega_0$  are optimal for different sequence lengths. The optimal  $\omega_0$  values found are 19.20, 34.71, 68.69, 43.65 and 69.97 for the corresponding sequence lengths.

**Adding Problem.** We set the number of channels of our CKCNN as to roughly match the number of parameters of the GRU and TCN networks of Bai et al. [11]. This is obtained with 25 hidden channels at every layer. Similarly to the Copy Memory task, we observe that the time to convergence grew proportional to the length of the sequence considered. Interestingly, this task was much easier to solve for our models, with convergence for sequences of length 6000 observed after 38 epochs. The maximum number of epochs is set to 20, 20, 30, 50 and 50 for sequences of size 100, 200, 1000, 3000 and 6000. We observe that different values of  $\omega_0$  are optimal for different sequence lengths. The optimal  $\omega_0$

**Table A.4:** Hyperparameter values for experiments on irregularly sampled data. Non-listed parameters correspond to those in Tab. A.3.

| PARAMS.      | PHYSIONET | CT      |       |       | SC_RAW |       |       |
|--------------|-----------|---------|-------|-------|--------|-------|-------|
|              |           | (30%)   | (50%) | (70%) | (30%)  | (50%) | (70%) |
| $\omega_0$   | 4.38      | 17.24   | 12.00 | 4.24  | 35.66  | 31.70 | 25.29 |
| Dropout      | 0.0       | 0.2     | 0.2   | 0.0   | 0.1    | 0     | 0     |
| Weight Decay | 0.0       | 0.0     | 1e-4  | 0.0   | 1e-4   | 1e-4  | 1e-4  |
| Batch Size   | 1024      |         |       |       |        |       |       |
| Model Size   | 175.71K   | 101.75K |       |       | 99.34K |       |       |



**Figure A.5:** High-frequency components in Sine continuous kernels. We observe that continuous kernels parameterized by Sine networks often contain frequency components of frequency higher than the resolution of the grid used during training. Here, for instance, the kernel looks smooth on the training grid. However, several high-frequency components appear when sampled on a finer grid. Although this may be problematic, we believe that, if used properly, these high-frequency components can prove advantageous to model fine details in tasks such as super-resolution and compression.

values found are 14.55, 18.19, 2.03, 2.23 and 4.3 for the corresponding sequence lengths.

**sMNIST, pMNIST and sCIFAR10.** We construct two models of different sizes for these datasets: CKCNN and CKCNN-Big. The first is constructed to obtain a parameter count close to 100K. The second model, is constructed to obtain a parameter count close to 1M. The parameters utilized for these datasets are summarized in Tab. A.3. Despite our efforts, we observed that our models heavily overfitted sCIFAR10. Combinations of weight decay, dropout and weight dropout were not enough to counteract overfitting.

**CT, SC and SC\_raw.** The parameters utilized for classification on these datasets are summarized in Tab. A.3. For hyperparameters regarding experiments with irregularly-sampled data please refer to Tab. A.4. Any non-specified parameter value in Tab. A.4 can be safely consider to be the one listed for corresponding dataset in Tab. A.3.

**PennTreeBank** For a character-level language modeling on PTB dataset we use hyperparameters specified in Tab. A.3. We use embedding of size 100 following the TCN model from [11].



*Nulla facilisi. In vel sem. Morbi id urna in diam dignissim feugiat. Proin molestie tortor eu velit. Aliquam erat volutpat. Nullam ultrices, diam tempus vulputate egestas, eros pede varius leo.*

Quoteauthor Lastname



## Modelling Long Context in ND: From Task-Specific to a General Purpose CNN

### B.1 Extended Related Work

In this section, we provide a more extensive treatment of similar methods related to continuous reparameterizations of convolutional kernels and their limitations in use for general purpose architectures.

**Rethinking convolutional kernels.** Several previous works investigate a reformulation of CNNs based on modifications to the classical convolutional layer. These works provide formulations of continuous convolutional kernels pursuing several different motivations. Several works apply specifically to point-cloud data –for which discrete kernels are not appropriate– by interpolating a set of weights to obtain a definition over the continuous input space [160, 381] or expressing the kernels in a polynomial [445] or neural network basis [171, 343, 417, 436]. These architectures often rely on sophisticated data augmentation and feature engineering schemes, perform pooling or downsampling operations, and have not been shown to perform well on regular data, e.g. images [436].

Other work focuses on continuous kernel formulations to address specific architectural considerations, for example to increase receptive fields efficiently [370], or to learn more appropriate kernel geometries [74, 170]. In these examples, a fixed set of weights is interpolated over the kernel domain, which implicitly defines a low-resolution kernel that impedes modeling long term dependencies in a dense manner [319].

A different line of work incorporating continuous kernel definitions focuses on exploit-

ing desirable properties of spatially structured kernels for implementing convolutions equivariant to roto-translations [17, 105, 354, 422, 423, 434], or dilation-translation transformations [362, 363, 433]. Although these formulations also in principle give us handles for a continuous definition of convolutional kernels over the spatial domain, we choose not to restrict the kernel functions learned in our models to be equivariant, as this limits applicability to settings in which such priors are not warranted.

Note that graph convolutions [187] may similarly be viewed as providing a convolution operation invariant to arbitrary permutation symmetries on the input. Since this operation disregards positional information, it essentially shares the kernel function over the domain. In practice, the bias to permutation invariance needlessly impedes network expressivity for data where such assumptions do not hold, such as sequences, images or point-clouds.

Continuous Kernel Convolutions [321] may be seen as a special case of neural message passing [117] on a fully connected graph over the input, with messages conditioned on relative positions of nodes. In the case of FlexConv [319], graph connectivity is dynamic, conditioned on distance.

## B.2 Dataset description

**Sequential and Permuted MNIST.** The MNIST dataset [208] consists of 70K gray-scale  $28 \times 28$  handwritten digits divided into training validation and test sets of 60K and 10K samples, respectively. For validation purposes, the training dataset is further divided into training and validation sets of 55K and 5K samples, respectively.

The sequential MNIST dataset (sMNIST) presents MNIST images as a sequence of 784 pixels for digit classification. Consequently, good predictions require the model to preserve long-term dependencies up to 784 steps in the past. The permuted MNIST dataset (pMNIST) incorporates an additional level of difficulty by permuting the order of all sMNIST sequences with a random permutation. Resultantly, models can no longer rely on local information for the construction of their features and the importance of modelling long-term dependencies becomes more pronounced.

**CIFAR10, CIFAR100 and Sequential CIFAR10.** The CIFAR10 dataset [195] consists of 60K real-world  $32 \times 32$  RGB images uniformly drawn from 10 classes divided into training and test sets of 50K and 10K samples, respectively. The CIFAR100 dataset [195] is similar to the CIFAR10 dataset, with the difference that the images are now uniformly drawn from 100 different classes. For validation purposes, the training dataset of both CIFAR10 and CIFAR100 are further divided into training and validation sets of 45K and 5K samples, respectively.

Analogously to the sMNIST dataset, the sequential CIFAR10 (sCIFAR10) dataset presents CIFAR10 images as a sequence of 1024 pixels for image classification. This dataset is

more difficult than sMNIST, as (*i*) larger memory horizons are required to solve the task, and (*ii*) more complex structures and intra-class variations are present.

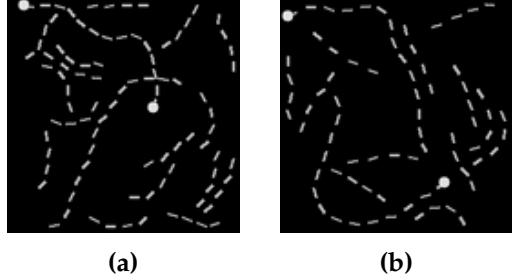
**Speech Commands.** The Speech Commands dataset [421] consists of 105809 one-second audio recordings of 35 spoken words sampled at 16kHz. Following Kidger et al. [183], we extract 34975 recordings from ten spoken words to construct a balanced classification problem. We refer to this dataset as *Raw Speech Commands*. In addition, we use the pre-processing steps of Kidger et al. [183] and extract mel-frequency cepstrum coefficients from the raw data. The resulting dataset, referred to as *MFCC Speech Commands*, consists of time series of length 161 and 20 channels.

**Long Range Arena.** The Long Range Arena benchmark [379] consists of 6 tasks with lengths 1K-16K steps encompassing modalities and objectives that require similarity, structural, and visuospatial reasoning. The Pathfinder, Path-X and Image tasks are similar in nature to the sMNIST and sCIFAR10 tasks. These tasks consists of classification tasks performed on images that are treated as sequences.

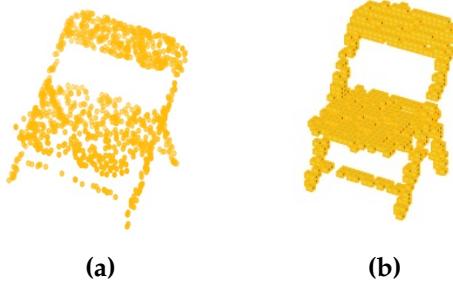
The Image task corresponds to the sequential CIFAR10 dataset with the only difference that the CIFAR10 images are treated as gray-scale images. The Pathfinder and Path-X tasks are binary tasks in which binary images are provided and the model must predict whether the two points in the images are connected with a line or not – see Fig. B.1 for an example–. The difference between both datasets is their resolution. Whereas Pathfinder has images of size  $32 \times 32$ , Path-X has images of size  $128 \times 128$ . It is important to mention that these tasks are so difficult that even if treated as 2D signals, CNNs without global receptive fields cant solve them [129].

**STL-10.** The STL-10 dataset [64] is a subset of the ImageNet dataset [196] consisting of 13,000  $96 \times 96$  real-world RGB images uniformly drawn from 10 classes divided into training and test sets of 5K and 8K images, respectively. For validation purposes, the training dataset is further divided into training and validation sets of 4,500 and 500 samples, respectively.

**ModelNet40.** The ModelNet40 dataset [438] contains 12,311 3D meshes of objects belonging to 40 classes. We use the official split with 9,843 training samples and 2,468 validation samples. In contrast with previous works which sample 1,024 points from these meshes, (e.g. [436]), we sample 512 points uniformly from the faces of each mesh, along with the normal vectors at each of these positions. These position and normal vectors serve as input to the model.



**Figure B.1:** Positive and negative samples from the Path-X dataset.



**Figure B.2:** An example of a point-cloud sampled from ModelNet10 (a), and a corresponding voxel representation of the same sample (b).

**ModelNet10 Voxelized.** ModelNet10 is a subset of ModelNet40 containing orientation-aligned samples from 10 classes of objects. We voxelize ModelNet10 based on a sub-sampling of 4096 points from the meshes. First, we normalize these points to fall within the interval  $[-1, 1]$ . Next, we bin these points into a grid of  $40 \times 40 \times 40$  voxels. All nonzero voxels get assigned their location as feature value. Afterwards, to align with our point-cloud configuration, we mask out the all but 512 nonzero voxels. Throughout the network, we only apply convolutions at the locations of each nonzero voxel (effectively masking out activations at all other voxel locations). As for point-clouds, we limit the convolution to integrate over the closest 256 points to limit its computational cost.

## B.3 Experimental details

### B.3.1 General remarks

**Code repository and logging.** Our code is written in PyTorch. We utilize wandb [29] hydra [449] and pytorch-lightning [101] for logging and code structuring. Our experiments are performed on NVIDIA TITAN RTX, A6000 and A100 GPUs, depending on the size of the datasets and inputs considered. Our code is publicly available at *url hidden for the sake of the double blind review process*.

**Normalized relative positions.** The kernel network  $\varphi_{\text{Kernel}}$  can, in principle, receive arbitrary coordinates as input. However, considering unitary step-wise relative positions, i.e.,  $0, 1, 2, \dots, N$ , can be problematic from a numerical stability perspective as  $N$  may grow very large, e.g.,  $N=16000$  for the Speech Commands dataset. Consequently, based on insights from the Implicit Neural Representations, e.g., Fathony et al. [104], Sitzmann et al. [358], we normalize the coordinates such that they lie in the space  $[-1, 1]^D$  for  $D$ -dimensional kernels. To this end, we map largest unitary positions seen during training  $[0, N]$  to a uniform linear space in  $[-1, 1]$ . Any relative kernel positions outside of the trained kernel domain that may appear during inference are masked out.

**Table B.1:** Best hyperparameters found for  $\text{CCNN}_{4,140}$  on all tasks considered.

|                        | $w_0$   | Dropout | Learning Rate | Weight Decay | Batch Size | Epochs |
|------------------------|---------|---------|---------------|--------------|------------|--------|
| sMNIST                 | 2976.49 | 0.1     | 0.01          | 1e-6         | 100        | 210    |
| pMNIST                 | 2985.63 | 0.2     | 0.02          | 0            | 100        | 210    |
| sCIFAR10               | 2386.49 | 0.0     | 0.02          | 0            | 50         | 210    |
| SPEECH COMMANDS (RAW)  | 1295.61 | 0.2     | 0.02          | 1e-6         | 20         | 160    |
| SPEECH COMMANDS (MFCC) | 750.18  | 0.2     | 0.02          | 1e-6         | 100        | 110    |
| LISTOPS                | 784.66  | 0.1     | 0.001         | 1e-6         | 50         | 60     |
| TEXT                   | 2966.60 | 0.2     | 0.001         | 1e-5         | 50         | 60     |
| IMAGE                  | 4005.15 | 0.2     | 0.01          | 0            | 50         | 210    |
| PATHFINDER             | 2272.56 | 0.2     | 0.01          | 0            | 100        | 210    |
| CIFAR10                | 1435.77 | 0.1     | 0.02          | 0.0001       | 50         | 210    |
| CIFAR100               | 3521.55 | 0.1     | 0.02          | 0.0001       | 50         | 210    |
| STL10                  | 954.28  | 0.1     | 0.02          | 0            | 64         | 210    |
| 2DIMAGE                | 2085.43 | 0.2     | 0.02          | 1e-6         | 50         | 210    |
| 2DPATHFINDER           | 1239.14 | 0.1     | 0.01          | 0            | 100        | 210    |
| MODELNET40             | 50      | 0.0     | 0.02          | 1e-8         | 16         | 200    |

**Table B.2:** Best hyperparameters found for  $\text{CCNN}_{6,380}$  on all tasks considered.

|                        | $w_0$   | Dropout | Learning Rate | Weight Decay | Batch Size | Epochs |
|------------------------|---------|---------|---------------|--------------|------------|--------|
| sMNIST                 | 2976.49 | 0.1     | 0.01          | 0            | 100        | 210    |
| pMNIST                 | 2985.63 | 0.2     | 0.02          | 0            | 100        | 210    |
| sCIFAR10               | 4005.15 | 0.25    | 0.01          | 0            | 50         | 210    |
| SPEECH COMMANDS (RAW)  | 1295.61 | 0.2     | 0.02          | 1e-6         | 20         | 160    |
| SPEECH COMMANDS (MFCC) | 750.18  | 0.2     | 0.02          | 1e-6         | 100        | 110    |
| LISTOPS                | 784.66  | 0.25    | 0.001         | 0            | 50         | 60     |
| TEXT                   | 2966.60 | 0.3     | 0.02          | 0            | 50         | 60     |
| IMAGE                  | 4005.15 | 0.1     | 0.01          | 0            | 50         | 210    |
| PATHFINDER             | 2272.56 | 0.1     | 0.01          | 1e-6         | 100        | 210    |
| CIFAR10                | 1435.77 | 0.15    | 0.02          | 0            | 50         | 210    |
| CIFAR100               | 679.14  | 0.2     | 0.02          | 0            | 50         | 210    |
| STL10                  | 954.28  | 0.1     | 0.01          | 1e-6         | 64         | 210    |
| 2DIMAGE                | 2306.08 | 0.2     | 0.02          | 0            | 50         | 210    |
| 2DPATHFINDER           | 3908.32 | 0.2     | 0.01          | 0            | 100        | 210    |
| MODELNET40             | 50      | 0.0     | 0.02          | 1e-7         | 32         | 200    |

### B.3.2 Hyperparameters and training details

**Optimizer and learning rate scheduler.** All our models are optimized with AdamW [239] in combination with a cosine annealing learning rate scheduler [238] and a linear learning rate warm-up stage of 10 epochs.

**Best hyperparameters found.** We perform hyperparameter search on the learning rate, dropout rate, weight decay, and  $\omega_0$  of our CNNs for each task considered.<sup>1</sup> The best hyperparameters found are reported in Tables B.1 and B.2.

**Parameter efficiency experiments on ModelNet40.** To further assess parameter efficiency of the CNN on ModelNet40, we run a number of experiments with smaller model sizes. Results for the following models are shown in Fig. 3.8:  $\text{CCNN}_{4,16}$  with 4

<sup>1</sup> $\omega_0$  serves as a prior on the variance of the data that is fitted with several types of implicit neural representations, e.g., SIRENs [358], MFNs [104], etc.

residual blocks, a channel size of 16 and a kernel network hidden size of 8, with 6,545 total parameters. CCNN<sub>4,32</sub> with 4 residual blocks, a channel size of 32 and a kernel network hidden size of 8, with 14,401 total parameters. CCNN<sub>4,48</sub> with 4 residual blocks, a channel size of 48 and a kernel network hidden size of 8, with 26,353 total parameters. We use a learning rate of  $2e^{-2}$ , no weight decay, and an  $\omega_0$  of 50.

## B.4 Additional Experiments and Details

### B.4.1 Computational Efficiency

#### Efficiency of continuous convolutional kernels

**Experimental setup.** We experimentally asses the computational complexity of our model. We measure the time of a single forward/backward pass for the two architectures used in our experiments: CCNN<sub>4,140</sub> (4 blocks, 140 channels) and CCNN<sub>6,380</sub> (6 blocks, 380 channels). It is important to note that we assume the “worst-case” scenario for CNNs in which the learned kernel size equals the input length, i.e., the model performs convolutions with global kernels at each layer. FlexConvs [319] learn the kernel size during training and –as shown exemplarily in Fig B.3– having global kernels at each layer is not something that we observe in practice.

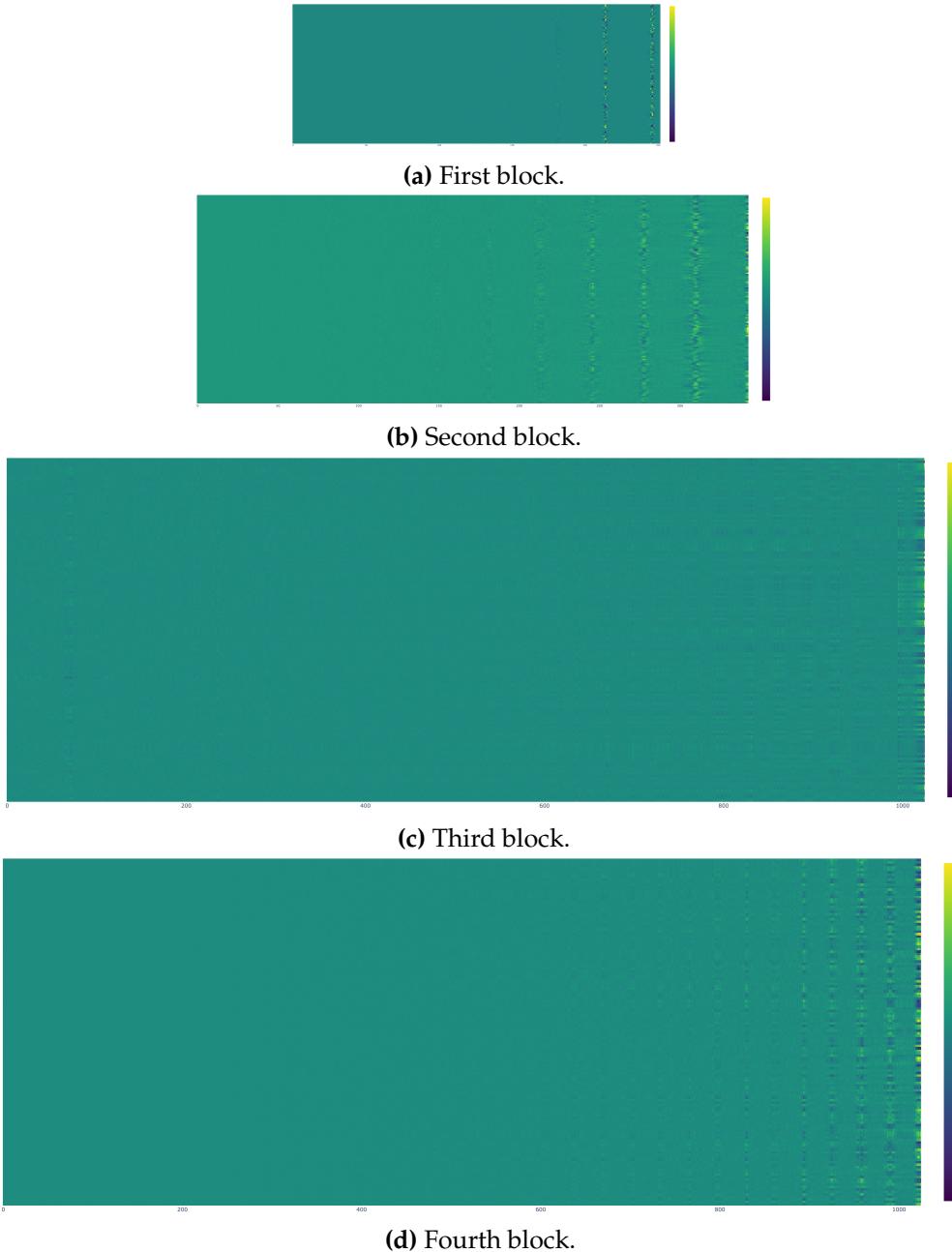
We compare to (i) discrete convolutional networks which match in architecture to ours, with global depthwise separable convolutional kernels; Global CNN<sub>4,140</sub>, and Global CNN<sub>6,380</sub>, (ii) more traditional CNN architectures with downsampling after each block (pooling of size 2), local depthwise separable kernels (kernels of size 5), and a depth tuned to the input size such that the receptive field of the final residual block covers the entire input. We take depth = [d] with d given by:

$$\frac{\text{input length}}{\text{pooling size}^d} \geq \text{kernel size}.$$

These architectures are denoted Local CNN<sub>4,140</sub> and Local CNN<sub>6,380</sub>. For input lengths of 1024, 4096, 8192 and 16000 this results in depths of 8, 10, 11 and 12 blocks respectively.

We measure training time (forward and backward pass) for sequence data of different lengths between 1024 –the length of sCIFAR10–, and 16000 –the length of Speech-Raw–, the largest sequence length in our experiments. We measure the time of training over 1000 batches of size 8 on a RTX 3090.

**Results.** Results are summarized in Tab B.3. Note that for the Global CNN<sub>6,380</sub>, we were unable to complete the experiment on data of lengths exceeding 8192 because of insufficient memory during the backward pass. This highlights another benefit of the continuous convolutional layer: its relative memory efficiency in computing weight updates, which results from the fact that gradients for individual kernel values can be discarded. As only the kernel network weights need to be updated, gradient values for the kernel weights do not need to be stored during the backward pass.



**Figure B.3:** Learned (causal) convolutional kernels in a  $\text{CCNN}_{4,140}$  trained on sequential CIFAR10. The vertical axis indexes the channels, and the horizontal axis the length of the kernels in a single layer. Interestingly, we observe a clear periodic pattern of 32 steps along the spatial dimension across all layers –a period that corresponds to the width of the underlying  $32 \times 32$  CIFAR10 images–. This illustrates that CNNs in fact learn to represent 2D structures on the flattened 1D space on which sequential CIFAR10 is defined. Despite these important capabilities, it is important to note that modelling ND signals as flattened 1D signals poses a unnecessary burden to the model, and modelling the signal in the original 2D space leads to faster convergence and better accuracy (Fig. 3.9).

**Table B.3:** Training time (ms).  $\times$  - unavailable due to memory issues.  $n$  - # blocks tuned per input length.

| SIZE                                      | 1024 | 4096 | 8192 | 16000    |
|---|------|------|------|----------|
| Local CNN <sub><math>n, 140</math></sub>  | 24   | 32   | 43   | 65       |
| Global CNN <sub><math>4, 140</math></sub> | 13   | 25   | 53   | 122      |
| CCNN <sub><math>4, 140</math></sub>       | 52   | 56   | 77   | 145      |
| Local CNN <sub><math>n, 380</math></sub>  | 24   | 38   | 93   | 178      |
| Global CNN <sub><math>6, 380</math></sub> | 31   | 102  | 220  | $\times$ |
| CCNN <sub><math>6, 380</math></sub>       | 71   | 136  | 255  | 537      |

Our results show that our approach requires additional computation compared to more traditional discrete CNNs. We note that although the use of a kernel network to infer kernel values brings computational overhead, this overhead is not a limiting factor in practice. The global convolutional kernels used in our network architectures have an impact on computational efficiency, but the use of convolutions in the Fourier domain help address this issue.

#### Efficiency of CCNNs with regard to other global methods

**Experimental Setup.** Next, we benchmark the computational efficiency of CCNNs compared to other input-global methods: S4 [129] and LSSL [128].

We reproduce the efficiency benchmark in [129]. That is, we provide the runtime for a single layer of forward and backward pass for different hidden dimensionalities, measured for different input lengths. The kernel network used in these experiments is a MAGNet with 3 layers and a hidden dimensionality of 32. Again, we assume a "worst-case" setting in which the learned kernel size equals the input length, i.e. the layer performs convolutions with global kernels. Note that in this setting, we perform convolutions in the spatial domain. We average runtime over 10000 steps. Experiments are performed on a RTX 3090.

**Results.** Results are summarized in Tab. B.4. We show that, up to a sequence length of 16000, runtimes in comparison with S4 are marginally faster, but remain in the same order of magnitude. Differences in memory utilization become more pronounced for increases in hidden dimensionality and sequence length, but remain a constant factor. Next, note that in both memory and speed, we show that a CCNN layer is asymptotically more efficient than LSSL. For LSSL, only results present in [129] are shown, as we were unable to successfully reproduce the LSSL layer performance.

#### B.4.2 Ablation: Performance of FlexCNN, S4 and CCNN Residual Blocks

**Experimental setup.** We provide an ablation over the impact on performance of the proposed residual CCNNBlock compared to the FlexCNNBlock used in the architectures in [319] and the S4Block used in [129] (Fig. 3.7).

To this end, we run experiments on a select number of datasets covering sequence and image data with a version of our architecture that includes the FlexCNNBlock (F-

**Table B.4:** Forward and backward for a single layer of CCNN in comparison with S4 [129] and LSSL [128] in speed (ms).  $\times$  - result not available.

| INPUT LENGTH |  | 128         |             |             | 1024        |             |             | 4096        |             |              | 16000        |              |              |
|--------------|--|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|--------------|--------------|--------------|--------------|
| HIDDEN SIZE  |  | 256         | 512         | 1024        | 256         | 512         | 1024        | 256         | 512         | 1024         | 256          | 512          | 1024         |
| LSSL         |  | 9.32        | 20.60       | 140.70      | $\times$    | $\times$    | $\times$    | $\times$    | $\times$    | $\times$     | $\times$     | $\times$     | $\times$     |
| S4           |  | 4.16        | <b>3.41</b> | 4.16        | 4.11        | 6.39        | 9.06        | 6.86        | 10.57       | 25.20        | <b>18.10</b> | <b>35.72</b> | <b>96.64</b> |
| CCNN         |  | <b>3.99</b> | 3.71        | <b>4.12</b> | <b>3.90</b> | <b>5.07</b> | <b>7.25</b> | <b>5.91</b> | <b>9.67</b> | <b>22.39</b> | 20.64        | 42.49        | 102.57       |

**Table B.5:** Forward and backward for a single layer of CCNN in comparison with S4 [129] and LSSL [128] memory usage (Mb).  $\times$  - result not available.

| INPUT LENGTH |  | 128        |              |              | 1024         |              |               | 4096          |               |               | 16000         |                |                |
|--------------|--|------------|--------------|--------------|--------------|--------------|---------------|---------------|---------------|---------------|---------------|----------------|----------------|
| HIDDEN SIZE  |  | 256        | 512          | 1024         | 256          | 512          | 1024          | 256           | 512           | 1024          | 256           | 512            | 1024           |
| LSSL         |  | 222.1      | 1685         | 13140        | $\times$     | $\times$     | $\times$      | $\times$      | $\times$      | $\times$      | $\times$      | $\times$       | $\times$       |
| S4           |  | <b>5.6</b> | <b>14.46</b> | <b>42.64</b> | <b>33.65</b> | <b>70.47</b> | <b>153.69</b> | <b>129.69</b> | <b>262.51</b> | <b>537.72</b> | <b>508.93</b> | <b>1014.08</b> | <b>2033.82</b> |
| CCNN         |  | 14.46      | 28.94        | 60.38        | 113.13       | 222.97       | 444.17        | 451.39        | 888.24        | 1763.43       | 1764.46       | 3467.34        | 6878.26        |

CCNN<sub>4,140</sub> with 4 blocks, 140 channels, 233K params and F-CCNN<sub>6,380</sub> with 6 blocks, 380 channels, 2.24M params), and a version of our architecture that includes the S4Block (S4-CCNN<sub>4,140</sub> with 4 blocks, 140 channels, 200K params and S4-CCNN<sub>6,380</sub> with 6 blocks, 380 channels, 2M params). We compare performance against the two architectures used throughout the experiments in Sec. 3.5, which uses the CCNNBlock (CCNN<sub>4,140</sub> with 4 blocks, 140 channels, 200K params and CCNN<sub>6,380</sub> with 6 blocks, 380 channels, 2M params). To isolate the impact of the residual block architecture, we replace the FlexConv layers in the original formulation of the FlexCNNBlock with our proposed depthwise separable implementation SepFlexConv. Note that we do not parameter-match these models, which results in the architectures with FlexCNNBlocks having more parameters compared to ones with the CCNNBlock, due to the FlexCNNBlock having two convolutional layers instead of the one convolutional layer and one pointwise linear layer of the CCNNBlock. Training regimes and hyperparameters for all architectures are as per Sec. B.3.

**Results.** Results are summarized in table B.6. First, note that without the nonlinearity added at the end of the block, the S4Block performs poorly over all tasks. Next, as shown, the CCNNBlock formulation improves performance of the architecture compared to FlexCNNBlock, in some cases by a significant margin. Note that architectures with the S4Block and CCNNBlock contain fewer convolutional layers and parameters compared to architecturally matched ones with FlexCNNBlocks, indicating higher parameter-efficiency as well as computational efficiency of CCNNBlocks (e.g. 8h 55m runtime for F-CCNN<sub>6,380</sub> vs. 8h 12m runtime for CCNN<sub>6,380</sub> on sCIFAR10). Because of the shown performance benefits, we use the CCNNBlock throughout our experiments.

**Table B.6:** Test accuracy of models with FlexCNNBlock ( $F\text{-CCNN}_{4,140}$ ,  $F\text{-CCNN}_{6,380}$ ), models with S4Block ( $S4\text{-CCNN}_{4,140}$ ,  $S4\text{-CCNN}_{6,380}$ ) and models with CCNNBlock ( $CCNN_{4,140}$ ,  $CCNN_{6,380}$ ).

|                          | SPEECHCOMMANDS-MFCC | sCIFAR10     | CIFAR10      |
|--------------------------|---------------------|--------------|--------------|
| $F\text{-CCNN}_{4,140}$  | 94.38               | 85.54        | 86.34        |
| $S4\text{-CCNN}_{4,140}$ | 73.60               | 53.25        | 60.06        |
| $CCNN_{4,140}$           | <b>95.01</b>        | <b>90.30</b> | <b>92.78</b> |
| $F\text{-CCNN}_{6,380}$  | 94.92               | 84.68        | 92.24        |
| $S4\text{-CCNN}_{6,380}$ | 60.82               | 51.50        | 67.01        |
| $CCNN_{6,380}$           | <b>97.98</b>        | <b>93.08</b> | <b>95.20</b> |

# C

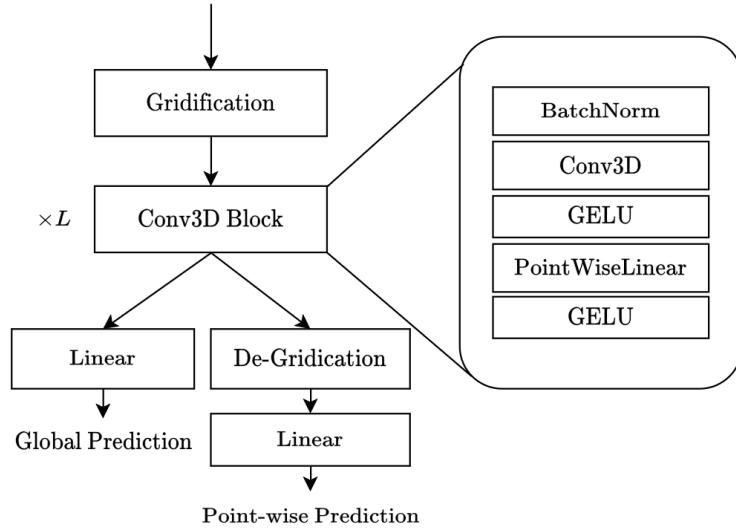
## Learnable Gridification for Efficient Point-Cloud Processing

### C.1 Network structure and convolution blocks

Fig. C.1 shows how we instantiated the grid network as described in section 4.2.6 in practice. The Conv3D blocks are CCNN blocks as in Knigge et al. [191].

### C.2 Hyperparameters

Table C.1 contains the best hyperparameters for the specific datasets found through hyperparameter sweeps.



**Figure C.1:** Practical pipeline and convolution blocks.

**Table C.1:** Hyperparameter settings for the different datasets.

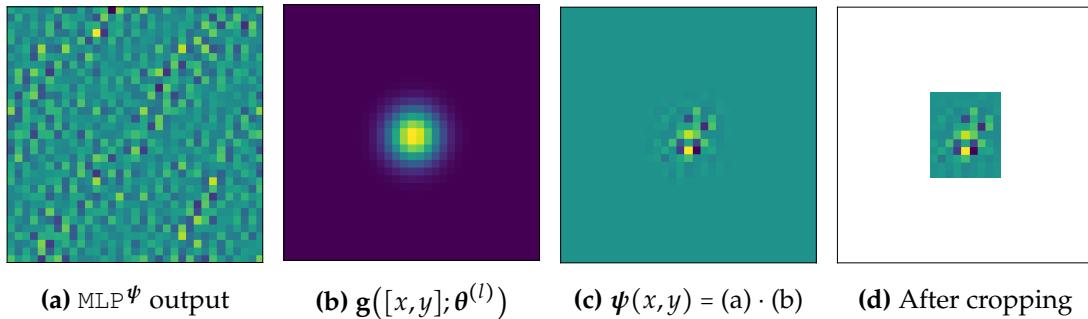
|                             | ModelNet40       | ShapeNet         |
|-----------------------------|------------------|------------------|
| batch size                  | 32               | 16               |
| nr. conv blocks             | 3                | 6                |
| hidden channels             | 128              | 256              |
| nr. epochs                  | 60               | 50               |
| nr. input points            | 1000             | 2047             |
| $\Omega$ position embedding | 0.1              | 1.0              |
| optimizer                   | AdamW            | AdamW            |
| learning rate               | 0.005            | 0.001            |
| learning rate scheduler     | Cosine Annealing | Cosine Annealing |
| learning rate warmup        | 10               | 10               |
| nr. neighbors               | 9                | 9                |
| grid resolution             | 9                | 13               |
| conv. kernel size           | 9                | 9                |
| dropout                     | 0.1              | 0.3              |
| weight decay                | 0                | 0.001            |
| aggregation                 | mean             | max              |

# D

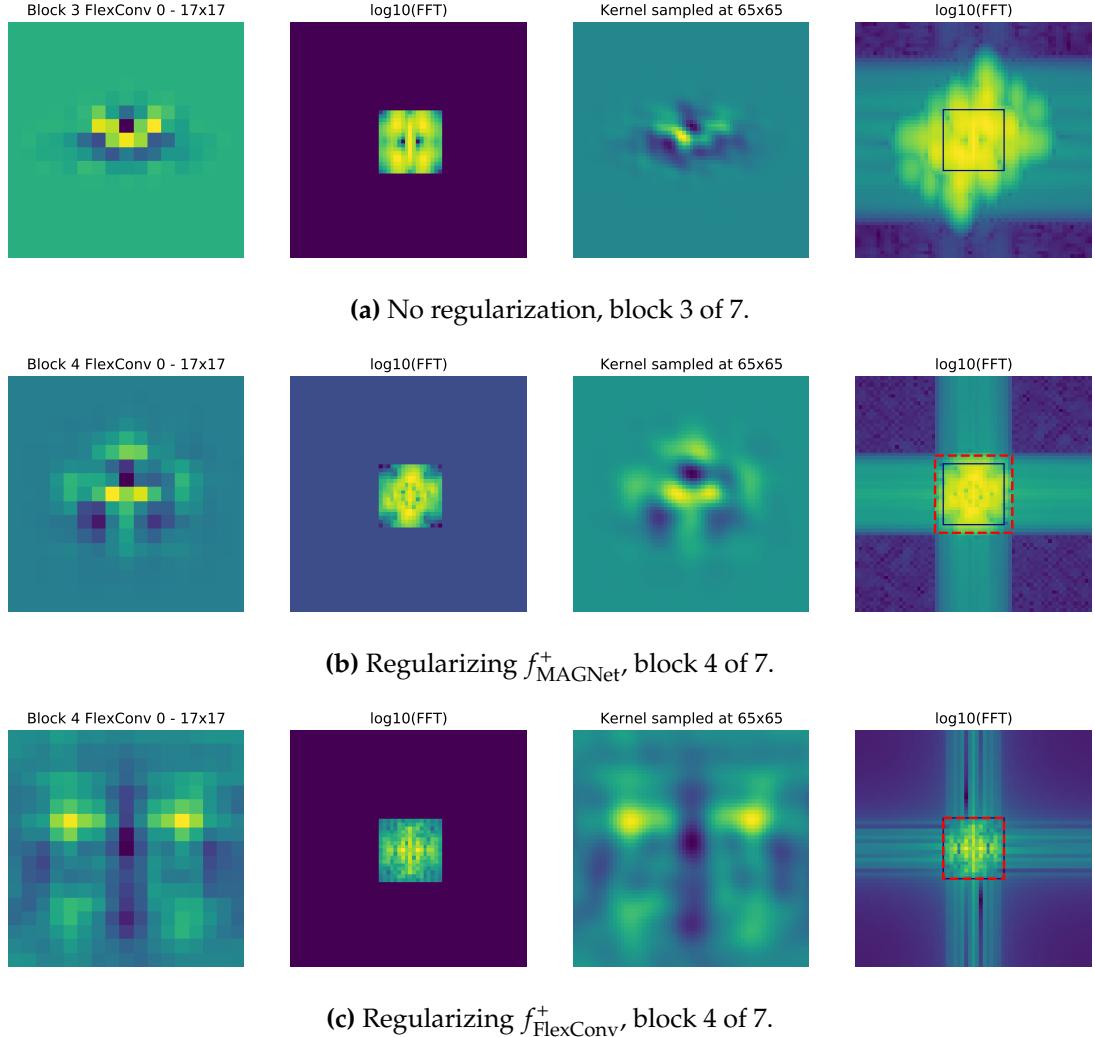
## Continuous Kernel Convolutions with Differentiable Kernel Sizes

### D.1 Alias-free FlexConv regularization

In this section we provide the complete derivation and analysis for our FlexConv regularization against aliasing. First, we derive the analytic maximum frequency component of a FlexConv kernel. Next, we compute the Nyquist frequency of a FlexConv kernel, and subsequently show how to combine the previous results into a regularization term to train alias-free FlexConvs.



**Figure D.1:** Example kernels, generated step by step. FlexConv samples a kernel from  $\text{MLP}^\psi$  (a), which is attenuated by an anisotropic Gaussian envelope with learned parameters  $\theta^{(l)}$  (b), creating (c) which is cropped to contain only values of  $> 0.1$  (d).



**Figure D.2:** Example kernels from FlexNet-16 models trained (i) without regularization, (ii) with aliasing regularization of  $f_{\text{MAGNet}}^+$ , (iii) with aliasing regularization of  $f_{\text{FlexConv}}^+$ . In the columns, from left to right: (i) original kernel at  $33 \times 33$ , (ii) FFT of the original kernel, (iii) kernel inferred at  $65 \times 65$ , to find aliasing effects, (iv) FFT of the  $65 \times 65$  kernel, with the solid line showing the Nyquist frequency of the  $33 \times 33$  kernel, and the red dotted line showing the maximum frequency component as computed by our analysis. For  $f_{\text{FlexConv}}^+$  the maximum frequency matches almost exactly with the Nyquist frequency, showing that our aliasing regularization works. For  $f_{\text{MAGNet}}^+$ , the maximum frequency is slightly higher than the Nyquist frequency, as the FlexConv mask is not included in the frequency term derivation. This is reflected in the slightly worse resolution generalization results reported in Sec. 5.4.3. Furthermore, some aliasing effects are still apparent for the aliasing regularized models, as discussed in Sec. 5.6.

### D.1.1 Analyzing the frequency spectrum of FlexConv

In order to make FlexConv alias-free (Sec. 5.3.3), we need to compute the maximum frequency component of the kernels generated by a MAGNet, so that we can regularize it during training. In this section we analytically derive this maximum frequency component from the parameters of the MAGNet.

Recall that MAGNets generate a kernel  $\psi(x, y)$  through of a succession of anisotropic Gabor filters and linear layers (Sec. 5.3.2, Eqs. 5.2–5.7):

$$\begin{aligned} \mathbf{h}^{(1)} &= \mathbf{g}([x, y]; \theta^{(1)}) & \mathbf{g} : \mathbb{R}^2 \rightarrow \mathbb{R}^{N_{\text{hid}}} \\ \mathbf{h}^{(l)} &= (\mathbf{W}^{(l)} \mathbf{h}^{(l-1)} + \mathbf{b}^{(l)}) \cdot \mathbf{g}([x, y]; \theta^{(l)}) & \mathbf{W}^{(l)} \in \mathbb{R}^{N_{\text{hid}} \times N_{\text{hid}}}, \mathbf{b}^{(l)} \in \mathbb{R}^{N_{\text{hid}}} \\ \psi(x, y) &= \mathbf{W}^{(L)} \mathbf{h}^{(L-1)} + \mathbf{b}^{(L)} & \mathbf{W}^{(L)} \in \mathbb{R}^{(N_{\text{out}} \times N_{\text{in}}) \times N_{\text{hid}}}, \mathbf{b}^{(L)} \in \mathbb{R}^{(N_{\text{out}} \times N_{\text{in}})} \end{aligned}$$

$$\begin{aligned} \mathbf{g}([x, y]; \theta^{(l)}) &= \exp \left( -\frac{1}{2} \left[ \left( \gamma_X^{(l)} (x - \mu_X^{(l)}) \right)^2 + \left( \gamma_Y^{(l)} (y - \mu_Y^{(l)}) \right)^2 \right] \right) \sin(\mathbf{W}_g^{(l)} [x, y] + \mathbf{b}_g^{(l)}) \\ \theta^{(l)} &= \left\{ \gamma_X^{(l)} \in \mathbb{R}^{N_{\text{hid}}}, \gamma_Y^{(l)} \in \mathbb{R}^{N_{\text{hid}}}, \mu_X^{(l)} \in \mathbb{R}^{N_{\text{hid}}}, \mu_Y^{(l)} \in \mathbb{R}^{N_{\text{hid}}}, \mathbf{W}_g^{(l)} \in \mathbb{R}^{N_{\text{hid}} \times 2}, \mathbf{b}_g^{(l)} \in \mathbb{R}^{N_{\text{hid}}} \right\} \end{aligned}$$

To analyse the maximum frequency component  $f_{\text{MAGNet}}^+$ , we analyse the frequency components of the Gabor filters used in MAGNet, and retain their maximum. We then plug the found frequency component into the analysis of Fathony et al. [104] to show how the frequency responses of Gabor filters and linear layers interact in MFNs. Finally, we add the effect of the FlexConv Gaussian mask to our analysis to obtain the maximum frequency component of the final FlexConv kernel  $f_{\text{FlexConv}}^+$ .

**Sine term in Gabor filters.** In a Gabor filter, the sine term is multiplied with a Gaussian envelope. The frequency (in radians) of a sine function of the form  $\sin(w^T [x, y] + b)$  is given by  $w$ . We divide by  $2\pi$  to convert the frequency units to Hertz, for compatibility with the rest of the analysis. For 2D inputs, the maximum frequency component of the sine function correspond to the largest frequency in the two input dimensions:

$$f_{\text{Sin}}^+ = \max_j \frac{w_j}{2\pi}. \quad (\text{D.1})$$

The sine terms in MAGNets have multiple output channels:  $\sin(\mathbf{W}_g \cdot [x, y] + \mathbf{b}_g^{(l)})$ . Effectively, we compute the sine term independently for each channel:

$$f_{\text{Sin}, i}^+ = \max_j \frac{\mathbf{W}_{g,i,j}}{2\pi}. \quad (\text{D.2})$$

**Gaussian term in Gabor filters.** In a Gabor filter, a Gaussian envelope modulates a sine term. Let us assume for now that the Gaussian envelope is isotropic, rather than anisotropic as in MAGNets, and has single-channel output. By applying the convolution theorem, the sine term is equivalently convolved with the Fourier transform of the Gaussian envelope in the frequency domain. Since the Fourier transform of a Gaussian

envelope is another Gaussian envelope, the application of a Gaussian envelope amounts to blurring with a Gaussian kernel in the frequency domain. The size of the envelope in the Fourier domain  $\sigma_F$  can be derived from the standard deviation of the Gaussian envelope in the spatial domain  $\sigma_T$  as follows:

$$\sigma_T \sigma_F = \frac{1}{2\pi} \Rightarrow \sigma_F = \frac{1}{2\pi \sigma_T}. \quad (\text{D.3})$$

Gaussian blurs induce impulse signals to have a long tail. Consequently, we must define a cutoff point for this tail in terms of standard deviations to derive the maximum added frequency induced by the blur. We describe the cutoff point as  $\sigma_{\text{cut}} \in \mathbb{N}$ . Typical choices for  $\sigma_{\text{cut}}$  are known as the *empirical*, or the "68-95-99.7" rule [135]. We choose two standard deviations, i.e.,  $\sigma_{\text{cut}}=2$ , which covers 95% of the mass of the Gaussian envelope.

For an isotropic Gabor filter with  $\gamma=\sigma_T^{-1}$ , the maximum frequency of its Gaussian envelope  $f_{\text{env}}^+$  is given by:

$$f_{\text{env}}^+ = \frac{\sigma_{\text{cut}}}{2\pi(\sigma_T)^{-1}} = \frac{\sigma_{\text{cut}}\gamma}{2\pi}. \quad (\text{D.4})$$

**Anisotropic envelopes.** Our analysis so far assumes an isotropic Gaussian envelope in the Gabor filter. However, we need to account for the anisotropic Gaussian envelopes in MAGNets. Anisotropic filters have not one but two  $\gamma$  parameters:  $\{\gamma_X, \gamma_Y\}$ . The smallest of these will contribute most to  $f_{\text{env}}^+$ , as it will blur the most, so it is sufficient to compute  $f_{\text{env}}^+$  only using the smallest of the two  $\gamma$  terms:

$$f_{\text{env}}^+(\gamma_X, \gamma_Y) = f_{\text{env}}^+(\min\{\gamma_X, \gamma_Y\}). \quad (\text{D.5})$$

The other assumption we made before was to work with single-channel outputs. MAGNets however use multi-channel outputs with independent Gaussian terms. The maximum frequency of multi-channel Gaussian envelopes is given by:

$$f_{\text{env},i}^+(\gamma_X, \gamma_Y) = f_{\text{env}}^+(\min\{\gamma_{X,i}, \gamma_{Y,i}\}) = \frac{\sigma_{\text{cut}} \min\{\gamma_{X,i}, \gamma_{Y,i}\}}{2\pi}, \quad (\text{D.6})$$

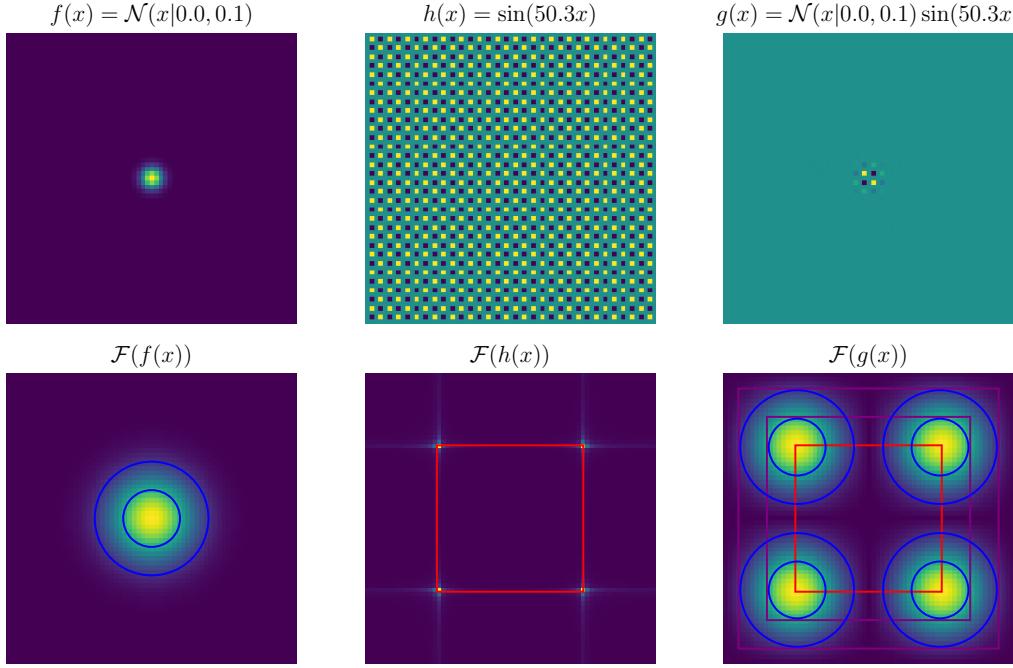
where the subscript  $i$  indexes the channels of the multi-channel Gaussian envelopes.

**Maximum frequency component of anisotropic Gabor filters.** Finally, the maximum frequency component of the  $i$ -th channel of an anisotropic Gabor filter  $\mathbf{g}$  is given by:

$$\begin{aligned} f_{\text{Gabor},i}^+ &= f_{\text{Sin},i}^+(\mathbf{W}_g) + f_{\text{env},i}^+(\gamma_X, \gamma_Y) \\ &= \left( \max_j \frac{\mathbf{W}_{g,i,j}}{2\pi} \right) + \frac{\sigma_{\text{cut}} \min\{\gamma_{X,i}, \gamma_{Y,i}\}}{2\pi}. \end{aligned} \quad (\text{D.7})$$

Figure D.3 illustrates the frequency spectrum of an example Gabor filter.

**Maximum frequency component of a MAGNet.** Fathony et al. [104] characterize the expansion of each term of the isotropic Gabor layers in MFNs in the final MFN output. In Eq. 25, Fathony et al. [104] demonstrate that the MFN representation contains a set of



**Figure D.3:** Decomposition of a Gabor filter and its frequency spectrum. Top row: a decomposition of a Gabor filter (right) into its Gaussian term (left) and its sine term (center). Bottom row: frequency responses for each respective filter. The Fourier transform of a Gaussian envelope is a Gaussian envelope (blue circles show  $\sigma_{\mathcal{F}}$  for  $h = \{1, 2\}$ ). The Fourier transform of a sine pattern is a collection of symmetrical impulse signals (red box shows the Nyquist frequency). The Gaussian envelope blurs the frequency response of the sine term (purple boxes show the frequency response for  $h = \{1, 2, 3\}$ ).

sine frequencies  $\bar{\omega}$  given by:

$$\bar{\omega} = \left\{ s_L \omega_{i_L}^{(L)} + s_{L-1} \omega_{i_{L-1}}^{(L-1)} + \dots + s_1 \omega_{i_1}^{(2)} + \omega_{i_1}^{(1)} \right\}. \quad (\text{D.8})$$

Here, the indexes  $i_1, i_2, \dots, i_{L-1}$  range over all possible indices of each hidden unit of each layer of an MFN, and  $s_2, \dots, s_L \in \{-1, +1\}$  range over all  $2^{L-1}$  possible binary signs. In other words, Fathony et al. [104] demonstrate that the representation of an MFN at a particular layer contains an exponential combination of all possible positive and negative combinations of the frequencies of the sine terms in each hidden unit at each layer in the MFN up to the current layer.

The original analysis uses these terms to argue that MFNs model exponentially many terms through a linear amount of layers. For our purpose of computing the frequency response of the MAGNet kernel, we can plug our derivation of the frequencies of the Gabor filter  $f_{\text{Gabor}}$  into  $\bar{\omega}$  to compute its frequency spectrum:

$$f_{\text{MAGNet}}^+ = \left\{ s_L f_{\text{Gabor}, i_L}^{(L)} + s_{L-1} f_{\text{Gabor}, i_{L-1}}^{(L-1)} + \dots + s_2 f_{\text{Gabor}, i_2}^{(2)} + f_{\text{Gabor}, i_1}^{(1)} \right\} \quad (\text{D.9})$$

As stated before, we are only interested in the maximum frequency in the frequency spectrum. We can therefore simplify Eq. D.9 in two ways. First, we simplify over MAGNet layers by taking the maximum value of the spectrum, which is the sum over all layers using only the positive binary signs in  $s_L$  (Eq. D.10). Next, we simplify over channel indices by retaining only the channel index that results in the highest frequency (Eq. D.11). The maximum frequency of a MAGNet is shown in Eq. D.12:

$$\begin{aligned} f_{\text{MAGNet}}^+ &= \left\{ (+1)f_{\text{Gabor},i_L}^{+(L)} + (+1)f_{\text{Gabor},i_{L-1}}^{+(L-1)} + \dots + (+1)f_{\text{Gabor},i_2}^{+(2)} + f_{\text{Gabor},i_1}^{+(1)} \right\} \\ &= \left\{ f_{\text{Gabor},i_L}^{+(L)} + f_{\text{Gabor},i_{L-1}}^{+(L-1)} + \dots + f_{\text{Gabor},i_2}^{+(2)} + f_{\text{Gabor},i_1}^{+(1)} \right\} \end{aligned} \quad (\text{D.10})$$

$$\begin{aligned} f_{\text{MAGNet}}^+ &= \max_{i_L} \left( f_{\text{Gabor},i_L}^{+(L)} \right) + \max_{i_{L-1}} \left( f_{\text{Gabor},i_{L-1}}^{+(L-1)} \right) + \dots + \max_{i_2} \left( f_{\text{Gabor},i_2}^{+(2)} \right) + \max_{i_1} \left( f_{\text{Gabor},i_1}^{+(1)} \right) \\ &= \sum_{l=1}^L \max_{i_l} \left( f_{\text{Gabor},i_l}^{+(l)} \right) \\ &= \sum_{l=1}^L \max_{i_l} \left( \left( \max_j \frac{\mathbf{W}_{g,i_l,j}^{(l)}}{2\pi} \right) + \frac{\sigma_{\text{cut}} \min\{\gamma_{X,i_l}^{(l)}, \gamma_{Y,i_l}^{(l)}\}}{2\pi} \right). \end{aligned} \quad (\text{D.11})$$

**Effect of the Gaussian mask in the frequency components of a FlexConv.** FlexConvs attenuate the MAGNet output with a Gaussian mask. The Gaussian mask (Eq. 5.1) works analogously to the Gaussian envelope term in the Gabor filter: it blurs the frequency components of the generated kernel with standard deviation  $\sigma_F$ . Therefore, we can reuse our derivation for the Gaussian envelope of the Gabor filter (Eq. D.6). The maximum frequency component of a FlexConv kernel is given by:

$$\begin{aligned} f_{\text{FlexConv}}^+ &= f_{\text{MAGNet}}^+ + f_{\text{env}}^+ \\ &= f_{\text{MAGNet}}^+ + \frac{\sigma_{\text{cut}} \min\{\sigma_X^{-1}, \sigma_Y^{-1}\}}{2\pi} = f_{\text{MAGNet}}^+ + \frac{\sigma_{\text{cut}}}{\max\{\sigma_X, \sigma_Y\} 2\pi} \\ &= \sum_{l=1}^L \max_{i_l} \left( \left( \max_j \frac{\mathbf{W}_{g,i_l,j}^{(l)}}{2\pi} \right) + \frac{\sigma_{\text{cut}} \min\{\gamma_{X,i_l}^{(l)}, \gamma_{Y,i_l}^{(l)}\}}{2\pi} \right) + \frac{\sigma_{\text{cut}}}{\max\{\sigma_X, \sigma_Y\} 2\pi}. \end{aligned} \quad (\text{D.13})$$

**Visualization of regularized kernels.** Fig. D.2 shows example kernels from FlexNets trained with aliasing regularization. The frequency domain plots confirm the accuracy of our frequency component regularization.

### D.1.2 Regularizing the frequency response of FlexConv

**Nyquist frequency of a FlexConv kernel.** Given the sampling rate  $f_s$  of the kernel, we can compute its Nyquist frequency  $f_{\text{Nyq}}$  as:

$$f_{\text{Nyq}} = \frac{1}{2} f_s \quad (\text{D.14})$$

To compute the sampling rate, we note that the kernel coordinates input to our MAGNet stretch over a  $[-1, 1]^D$  domain. For a kernel of length  $k$ , we therefore sample one point

in every  $f_s = \frac{k-1}{2}$  units. Knowing the sampling rate in terms of the kernel size allows us to express the Nyquist frequency in terms of the (pre-masked) kernel size:

$$f_{\text{Nyq}}(k) = \frac{1}{2} \frac{k-1}{2} = \frac{k-1}{4}. \quad (\text{D.15})$$

Note that the kernel size in a FlexConv is initialized to be equal to the resolution of the input, if it is odd. For even resolutions, it is equal to the input resolution plus one.

**Constructing the regularization term.** We train FlexConv with a regularization term on the frequency response of the generated kernel to ensure that aliasing effects do not distort the performance of the model when it is inferred at a higher resolution. This section details the implementation of the regularization function.

From the parameters of each FlexConv module, we compute  $f_{\text{FlexConv}}^+$  according to Eq. D.13. For the amount of standard deviations to use in determining  $f_{\text{env}}^+$  (Eq. D.6) we use  $h = 2$ . From the kernel size  $k$  of the FlexConv module we compute  $f_{\text{Nyq}}(k)$  according to Eq. D.15. We then apply an L2 regularizer over the amount that  $f_{\text{FlexConv}}^+$  exceeds  $f_{\text{Nyq}}(k)$ :

$$\mathcal{L}_{\text{HF}} = \|\max\{f_{\text{FlexConv}}^+, f_{\text{Nyq}}(k)\} - f_{\text{Nyq}}(k)\|^2. \quad (\text{D.16})$$

We weight  $\mathcal{L}_{\text{HF}}$  by  $\lambda = 0.1$  when adding it to our loss function.

**Improved implementation.** Eq. D.16 contains a sum over the  $L$  layers of the MAGNet. In practice, we prefer to regularize each layer  $l \in L$  separately, so that the gradients of the regularization of different layers are not dependent on each other. We therefore implement the anti-aliasing regularization by regularizing each MAGNet layer independently, and spreading the  $f_{\text{env}}^+$  term from the gaussian mask uniformly over all MAGNet layers:

$$\mathcal{L}_{\text{HF},l} = \left\| \max \left\{ f_{\text{MAGNet},l}^+ + \frac{f_{\text{env}}^+}{L}, \frac{f_{\text{Nyq}}(k)}{L} \right\} - \frac{f_{\text{Nyq}}(k)}{L} \right\|^2 \quad (\text{D.17})$$

$$= \left\| \max_{i_l} \left( f_{\text{Gabor},i_l}^{+(l)} + \frac{f_{\text{env}}^+}{L}, \frac{f_{\text{Nyq}}(k)}{L} \right) - \frac{f_{\text{Nyq}}(k)}{L} \right\|^2. \quad (\text{D.18})$$

In the code, we refer to this method as the *together* method, versus the *summed* method of Eq. D.16. In preliminary experiments, we observed improved performance of anti-aliasing training when using the *together* method. All of our experiments anti-aliasing experiments therefore use the *together* setting.

## D.2 Dataset Description

### D.2.1 Image Fitting Datasets

**Kodak dataset.** The Kodak dataset [193] consists of 24 natural images of size  $768 \times 512$ . This dataset is a popular benchmark used for compression and image fitting methods.

### D.2.2 Sequential Datasets

**Sequential and Permuted MNIST.** The sequential MNIST dataset (sMNIST) [203] takes the  $28 \times 28$  images from the original MNIST dataset [208], and presents them as a sequence of 784 pixels. The goal of this task is to perform digit classification given the representation of the last sequence element of a sequential model. Consequently, good predictions require the model to preserve dependencies up to 784 steps in the past.

The permuted MNIST dataset (pMNIST) additionally changes the order of all the sMNIST sequences by a random permutation. Consequently, models can no longer rely on local features to construct good feature representations. As a result, the classification problem becomes more difficult, and the importance of long-term dependencies more pronounced.

**Sequential and Noise-Padded CIFAR10.** The sequential CIFAR10 dataset (sCIFAR10) [44] takes the  $32 \times 32$  images from the original CIFAR10 dataset [195] and presents them as a sequence of 1,024 pixels. The goal of this task is to perform image classification given the representation of the last sequence element of a sequential model. This task is more difficult than sMNIST, as a larger memory horizon is required to solve the task and more complex structures and intra-class variations are present in the data [12].

The noise-padded CIFAR10 dataset (npCIFAR10) [42] flattens the images from the original CIFAR10 dataset [195] along their rows to create a sequence of length 32, and 96 channels (32 rows  $\times$  3 channels). Next, these sequences are concatenated with 968 entries of noise to form the final sequences of length 1000. As for sCIFAR10, the goal of the task is to perform image classification given the representation of the last sequence element of a sequential model.

**CharacterTrajectories.** The CharacterTrajectories dataset is part of the UEA time series classification archive [9]. It consists of 2858 time series of length 182 and 3 channels representing the  $x, y$  positions, and the tip force of a pen while writing Latin alphabet characters in a single stroke. The goal is to classify out of 20 classes the written character using the time series data.

**Speech Commands.** The Speech Commands dataset [421] consists of 105,809 one-second audio recordings of 35 spoken words sampled at 16kHz. Following Kidger et al. [183], we extract 34975 recordings from ten spoken words to construct a balanced classification problem. We refer to this dataset as **SpeechCommands.raw**, or **SC\_raw** for short. Furthermore, we utilize the preprocessing steps of Kidger et al. [183] and extract mel-frequency cepstrum coefficients from the raw data. The resulting dataset, abbreviated **SC**, consists of time series of length 101, and 20 channels.

**Table D.1:** Average PSNR for fitting of images in the Kodak dataset. Both our improved initialization scheme, as well as the inclusion of anisotropic Gabor functions lead to better reconstructions.

| MODEL                  | # PARAMS | IMPROVED INIT | PSNR          |
|------------------------|----------|---------------|---------------|
| SIREN                  | 7.14K    | -             | 25.665        |
| MFN <sub>Fourier</sub> | 7.40K    | -             | 23.276        |
| MFN <sub>Gabor</sub>   | 7.11K    | ✗             | 25.361        |
|                        |          | ✓             | <b>25.606</b> |
| MAGNet                 | 7.36K    | ✗             | 25.791        |
|                        |          | ✓             | <b>25.893</b> |

### D.2.3 Image Benchmark Datasets

**MNIST.** The MNIST handwritten digits dataset [205] consists of 70,000 gray-scale handwritten digits of size  $28 \times 28$ , divided into a training and test sets of 60,000 and 10,000 images, respectively. The goal of the task is to classify these digits as one of the ten possible digits (0, 1, .., 8, 9).

**CIFAR-10** The CIFAR-10 dataset [195] consists of 60,000 natural images from 10 classes of size  $32 \times 32$ , divided into training and test sets of 50,000 and 10,000 images.

**STL-10.** The STL-10 dataset [64] is a subset of the ImageNet dataset [196] consisting of 5,000 natural images from 10 classes of size  $96 \times 96$ , divided into trainint and test sets of 4,500 and 500 images, respectively.

**ImageNet-k.** The Imagenet-k [61] dataset is derived from the ImageNet dataset [329] by downsampling all samples to a resolution  $k \in [64, 32, 16, 8]$ . The dataset contains 1000 classes with 1,281,167 training samples and 50,000 validation samples.

## D.3 Additional Experiments

### D.3.1 Image Classification

**CIFAR-10.** Tab. D.2 shows our CIFAR-10 results including more ablations.

**ImageNet-32.** Results for the ImageNet-32 experiment are shown in Table D.3. FlexNets are slightly worse than CIFARResNet-32 [143] with slightly less parameters. However, the results reported by Chrabaszcz et al. [61] for Wide ResNets [456] outperform FlexNets by a significant margin.

**Alias-free ImageNet-32.** We report results for alias-free FlexNets on ImageNet-k [61] in Table D.4, to verify the results of alias-free training at a larger scale. We find that FlexConv and N-Jet both mostly retain classification accuracy between source and target resolution, while CIFARResNet-32 degrades drastically.

**MNIST and STL-10.** We additionally report results on MNIST (Tab. D.5) and STL-10 (Tab. D.6. We choose these dataset for the difference in image sizes of the training data.

**Table D.2:** Full results on CIFAR-10. We report results over three runs per setting. CIFARResNet-44 w/ CKConv is a CIFARResNet-44 where all convolutional layers are replaced with CKConvs with  $k = 3$ . CIFARResNet-44 w/ FlexConv is a CIFARResNet-44 where all convolutional layers are replaced with FlexConv with learned kernel size, except for the shortcut connections of the strided convolutional layers, which are pointwise convolutions. \*Results are taken from the respective original works instead of reproduced. †Results are from single run.

| MODEL                                   | SIZE  | CIFAR-10<br>ACC. |
|---|-------|------------------|
| DCN- $\sigma^{ji}$ [384]                | 0.47M | 89.7 ± 0.3*      |
| N-Jet-CIFARResNet32 [291]               | 0.52M | 92.3 ± 0.3*      |
| N-Jet-ALLCNN [291]                      | 1.07M | 92.5 ± 0.1*      |
| CIFARResNet-44 [143]                    | 0.66M | 92.9*†           |
| CIFARResNet-44 [143] (our reproduction) | 0.66M | 90.9 ± 0.2       |
| CIFARResNet-44 w/ CKConv ( $k = 3$ )    | 2.58M | 86.1 ± 0.9       |
| CIFARResNet-44 w/ FlexConv              | 2.58M | 81.6 ± 0.8       |
| FlexNet-7 w/ conv. ( $k = 3$ )          | 0.17M | 89.5 ± 0.3       |
| FlexNet-7 w/ conv. ( $k = 33$ )         | 20.0M | 78.0 ± 0.3       |
| FlexNet-7 w/ N-Jet [291]                | 0.70M | 91.7 ± 0.1       |
| CKCNN <sub>SIREN</sub> -3               | 0.26M | 72.4*            |
| CKCNN <sub>Fourier</sub> -3             | 0.27M | 83.8*            |
| CKCNN <sub>Gabor</sub> -3               | 0.28M | 85.6*            |
| CKCNN <sub>MAGNet</sub> -3              | 0.28M | 86.2*            |
| CKCNN-7                                 | 0.63M | 71.7*            |
| CKCNN <sub>Fourier</sub> -7             | 0.63M | 84.6*            |
| CKCNN <sub>Gabor</sub> -7               | 0.67M | 87.7*            |
| CKCNN <sub>MAGNet</sub> -7              | 0.67M | 85.9*            |
| FlexNet <sub>SIREN</sub> -7             | 0.63M | 88.9*            |
| FlexNet <sub>Fourier</sub> -7           | 0.66M | 91.6*            |
| FlexNet <sub>Gabor</sub> -7             | 0.67M | 92.0*            |
| FlexNet-3                               | 0.27M | 90.4 ± 0.2       |
| FlexNet-5                               | 0.44M | 91.0 ± 0.5       |
| FlexNet-7                               | 0.67M | 92.2 ± 0.1       |

**Table D.3:** Results on ImageNet-32. \*Results are taken from the respective original works instead of reproduced. †Results are from a single run.

| MODEL          | SIZE  | IMAGENET-32  |              |
|----------------|-------|--------------|--------------|
|                |       | TOP-1        | TOP-5        |
| CIFARResNet-32 | 0.53M | 26.41 ± 0.13 | 49.37 ± 0.15 |
| WRN-28-1       | 0.44M | 32.03*†      | 57.51*†      |
| FlexNet-5      | 0.44M | 24.9 ± 0.4   | 47.7 ± 0.6   |

**Table D.4:** Results for alias-free FlexNets on CIFAR-10 and ImageNet-k.  $\Delta$  denotes difference in accuracy.

| MODEL               | SIZE  | IMAGENET-K TOP-1 |                        |
|---------------------|-------|------------------|------------------------|
|                     |       | $k = 16$         | $\Delta_{k=16} k = 32$ |
| CIFARResNet-32      | 0.52M | 16.1 ± 0.0       | -11.6 ± 0.4            |
| FlexNet-5 w/ N-Jets | 0.46M | 15.7 ± 0.1       | -1.9 ± 0.4             |
| FlexNet-5           | 0.44M | 14.9 ± 0.1       | -1.9 ± 1.7             |

**Table D.5:** Results on MNIST. We train each model with three different seeds and report mean and standard deviation. \*Results are taken from the respective original works instead of reproduced. †Results are from single run.

| MODEL                                   | SIZE  | MNIST ACC. |
|---|-------|------------|
| Efficient-CapsNet [258]                 | 0.16M | 99.8*†     |
| Network in Network [223]                | N/A   | 99.6*†     |
| VGG-5 (results from Kabir et al. [176]) | 3.65M | 99.7*†     |
| FlexNet-16                              | 0.67M | 99.7 ± 0.0 |

**Table D.6:** Results on STL-10. We train each model with three different seeds and report mean and standard deviation. \*Results are taken from Luo et al. [241]. †Results are from single run.

| MODEL          | SIZE  | STL-10 ACC. |
|----------------|-------|-------------|
| CIFARResNet-18 | 11.2M | 81.0*†      |
| FlexNet-16     | 0.67M | 68.6 ± 0.7  |

On MNIST, though performance on MNIST is quite saturated, we are competitive with state of the art methods. On STL-10 we are significantly worse than the baseline CIFARResNet from [241], though with significantly less parameters. We were not able to prepare a more relevant baseline for this experiment.

## D.4 Experimental Details

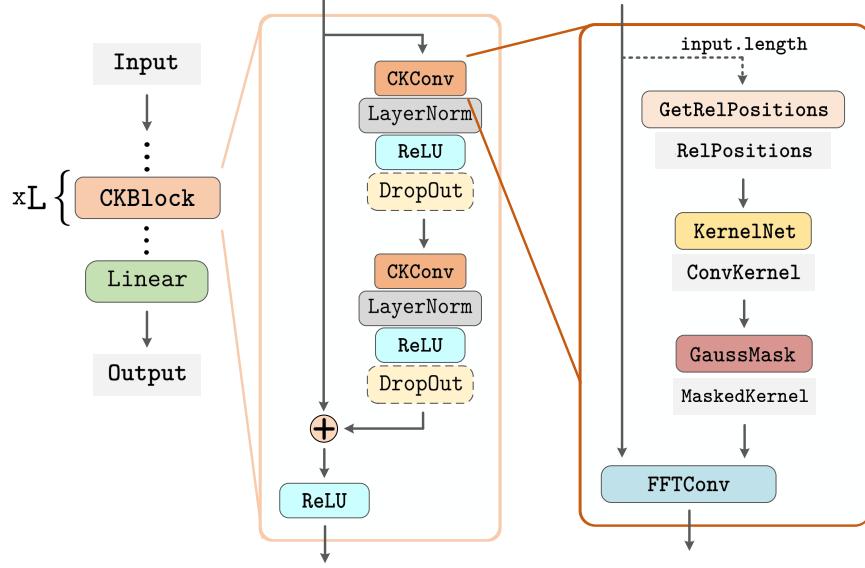
### D.4.1 FlexNet

We propose an image classification architecture named *FlexNet* (Fig. D.4), consisting of a stack of FlexConv blocks followed by a global average pooling layer and a linear layer. FlexNets are named “FlexNet-L” where L indicates the number of layers in the network.

**FlexBlock.** Each FlexBlock consists of two FlexConvs with BatchNorm [166] and dropout [366] ( $d = 0.2$ ) as well as a residual connection. The width of a block  $i$  is determined by scaling a base amount  $c$  by progressively increasing factors:  $c_i = [c, c \times 1.5, c \times 1.5, c \times 2.0, c \times 2.0](i)$ . The default configuration of FlexNet uses  $c = 22$ . In FlexNet-N-Jet models, we scale  $c$  to match the amount of parameters of the FlexNet in the comparison.

**FlexConv initialization.** We initialize the FlexConv mask variances small, at  $\sigma_X^2, \sigma_Y^2 = 0.125$ . For initializing MAGNet, we initialize the Gaussian envelopes as discussed in Sec. 5.3.2. We initialize the linear layer weights by the same Gamma distribution as used for the enveloped, modulated by a scaling factor of 25.6. We found that this value of the scaling factor, rather than a higher one, helped in reducing the performance of alias-free models. We initialize the bias of the linear layers by  $\mathcal{U}(-\pi, \pi)$ .

**CIFAR-10.** In FlexNet-16 models for CIFAR-10 we use  $c = 24$  to approximate the param-



**Figure D.4:** FlexNet architecture. FlexNet-L consists of  $\frac{L}{2}$  FlexBlocks, where each FlexBlock is a residual block of FlexConvs.

eter count of CIFARResNets in the experiment.

#### D.4.2 Optimization

We use Adam [185] to optimize FlexNet. Unless otherwise specified, we use a learning rate of 0.01 with a cosine annealing scheme [238] with five warmup epochs. We use a different learning rate of  $0.1 \times$  the regular learning rate for the FlexConv Gaussian mask parameters. We do not use weight decay, unless otherwise specified.

**Kodak.** We overfit on each image of the dataset for 20,000 iterations. We use a learning rate of 0.01 without any learning rate scheme. We observe that SIRENs diverge with this learning rate and thus, reduce the learning rate to 0.001 for these models.

**CIFAR-10.** We train for 350 epochs with a batch size of 64. We use the data augmentation from He et al. [143] when training CIFAR-10: a four pixel padding, followed by a random 32 pixel crop and a random horizontal flip.

**ImageNet-32.** We train for 350 epochs with a batch size of 2048. We use the same data augmentation as used for CIFAR-10, and a weight decay of  $1e-5$ .

**Sequential and Permuted MNIST.** We train for 200 epochs with a batch size of 64 and a learning rate of 0.01. We use a weight decay of  $1e-5$ .

**Sequential and Noise-Padded CIFAR-10.** For sequential CIFAR-10, we train for 200 epochs with a batch size of 64, a learning rate of 0.001 and a weight decay of  $1e-5$ . For

noise-padded CIFAR-10, we train for 300 epochs with a batch size of 32, a learning rate of 0.01 and no weight decay.

**Speech Commands and CharTrajectories.** We train for 300 epochs with a batch size of 32 and a learning rate of 0.001. For CharTrajectories, we use a weight decay of 1e-5.

### D.4.3 Rotated Gaussian masks

MAGNets use anisotropic Gaussian terms in the Gabor filters, which yields improvements in descriptive power and convergence speed (Sec. 5.3.2). For the same reason, we explore making the anisotropic FlexConv Gaussian mask steerable, by including an additional vector of learnable angle parameters  $\phi^{(l)} \in \mathbb{R}^{N_{\text{hid}}}$  that rotates the Gaussian masks. Although preliminary experiments show rotated masks lead to slight additional improvements, the computational overhead required to rotate the masks is large. Consequently, we do not consider rotated Gaussian masks in our final experiments.





# Learning Convolutional Neural Architectures by Backpropagation

## E.1 Extended Related Work

In this section, we position DNArch among the broad existing NAS literature, and provide a thorough comparison of DNArch with existing differentiable NAS techniques.<sup>1</sup>

### E.1.1 Positioning of DNArch on the NAS literature Taxonomy

Neural Architecture Search (NAS) techniques can be categorized based on three aspects [96, 428, 430]: (*i*) their search space, (*ii*) their search strategy, and (*iii*) their performance evaluation strategy:

- The *search space* refers to the set of all architectures the NAS technique is allowed to select from. While the search space can be extremely general, incorporating domain knowledge into the design of the search spaces simplifies the search. However, this infusion of human expertise introduces human biases, which can reduce the chances of NAS techniques finding truly novel architectures.
- The *search strategy* refers to the optimization technique used to find high-performing architectures in the search space.
- The *performance evaluation strategy* refers to the method used to predict the performance of neural architectures in order to avoid fully training each candidate architecture. While it is possible to run a discrete search strategy by fully training and

---

<sup>1</sup>This section is deeply based on the excellent review of White et al. [428].

evaluating architectures chosen throughout the search, using a performance estimation can greatly increase the speed of the search.

It is important to note, though, that in this trichotomy, *one-shot methods* –from which DNArch is part of– (see below), the search strategy and the performance evaluation strategy are coupled [439].

Within this trichotomy, DNArch can be understood as a novel search strategy. However, as we will see in the rest of this section, DNArch also has important implications for the design and flexibility of the search space, as well as the efficiency of the performance evaluation strategy.

**DNArch as a search strategy.** DNArch proposes a novel way to search among possible candidate architectures by viewing neural architectures as multi-dimensional continuous entities –with dimensions corresponding to depth, width, etc– and using learnable parametric differentiable masks to learn their size during training. Existing search strategies can be divided into two classes: (i) black-box optimization techniques and (ii) one-shot techniques. Black-box optimization techniques, such as reinforcement learning [289, 482, 483], evolutionary / genetic algorithms [263, 305, 361], bayesian optimization [178, 274, 371] and Monte-Carlo tree search [270, 410, 411], have been widely used due to their strong performance and ease of use. However, despite their numerous advantages –simple optimization of non-differentiable objectives, simple parallelism, joint optimization with other hyperparameters, etc–, these techniques require training a (vast) population of candidate architectures and often incur in immense computational costs, sometimes on the order of thousands of GPU days [305, 482]. *One-shot* techniques avoid training each architecture from scratch by training all architectures in the search space via a single (“one-shot”) training of an over-parameterized network [32, 231, 337]. Due to the extreme weight-sharing used, these methods are more scalable and efficient.

DNArch falls within the family of one-shot techniques, as it trains a single neural architecture from which several possible sub-networks can be extracted by changing the size of the differentiable masks.

**DNArch as a one-shot model technique.** One-shot models can be divided in two main classes based on the type of over-parameterized network used to contain the space of all possible networks in the search space: (i) a *hypernetwork* [32, 464] or (ii) a *supernetwork* [231, 337]. A *hypernetwork* is a neural network that generates the weights of other neural networks [134, 321, 339]. A *supernetwork*, on the other hand, is an over-parameterized architecture that contains all possible architectures in the search space as subnetworks. Currently, methods relying on supernetworks are more popular, with notorious examples being DARTS [231], once-for-all networks [38], and slimable networks [455].

Although DNArch relies on Continuous Kernel Convolution [321] –a type of hypernetwork– for the generation of global kernels regardless of the resolution, length and dimensionality of the input, DNArch falls within the family of supernetworks. This is due to the fact

that the differentiable masks are applied on a shared architecture to define their size. This is different from NAS techniques based on hypernetworks, where whole subnetworks are generated by the hypernetwork [32, 464].

**DNArch as a supernet technique.** The idea of a supernet was introduced by Saxena and Verbeek [337] and was popularized by works such as ENAS [289] and DARTS [231]. Supernet methods have become popular due to their scalability and efficiency. The reason for these properties being that *a linear increase in the number of candidate operations only causes a linear increase in computational costs for training, but the number of subnetworks in the supernet increases exponentially* [428]. Hence, an exponential number of architectures can be trained at linear cost.

While the supernet allows quick evaluation of all architectures, one must still define a search strategy for the selection of the final architecture. Supernet-based techniques can be divided in terms of the type of search strategy used: (i) non-differentiable, and (ii) differentiable techniques. While the first family uses black-box optimization algorithms to search for the best architecture [21, 289], differentiable supernet-based methods, e.g., DARTS [231], use of a continuous relaxation of the discrete architecture search space and rely on gradient descent to search for the best architecture in tandem with the training of the supernet. Through the use of gradient descent makes these methods are *significantly faster* than black-box optimization methods [38, 231].

DNArch falls within the family of differentiable supernet-based methods. However, as discussed below, by treating the search space as a continuous space, DNArc does not experience several of the limitations present in existing differentiable supernet-based methods.

### E.1.2 DNArc as a differentiable supernet method: DARTS, follow-up works and DNArc

**DARTS and follow-up works.** To understand the differences between DNArc and other existing differentiable supernet-based methods, it is important to start with a comprehensive description of DARTS [231]: the seminal work of differentiable NAS.

At the start, each edge  $(i, j)$  in the DARTS search space consists of multiple possible candidate operations  $o$ , each of which are associated with a continuous hyperparameter  $\alpha_o^{(i,j)} \in [0, 1]$ . While the supernet is training, the edge  $(i, j)$  consists of a *mix of all candidate operations* weighted by each  $\alpha_o^{(i,j)} \in [0, 1]$ . The architecture hyperparameters  $\alpha$  are optimized jointly with the supernet weights  $w$  via alternating gradient descent. After the training phase, DARTS uses a finetuning step to obtain the final discrete architecture. This is done by selecting the operation with the maximum value of  $\alpha$  on each edge (the discretization step) and then re-trains it from scratch. In other words, DARTS consists of the following steps:

- (i) *Training.* The supernet is trained as a mixture of all possible components.

- (ii) *Discretization.* Once the training of supernet has finished, the operation with the maximum  $\alpha$  value on each edge is selected to form the final architecture.
- (iii) *Fine-tuning.* The final architecture is retrained from scratch to get the final model.

Due to its simplicity, novelty and speed, DARTS gained significant attention in the AutoML community. However, DARTS comes with several limitations, which have been addressed in several follow-up works. In the scope of DNArc, the most relevant limitations are:

- *Poor test generalization.* Zela et al. [461] and Chen and Hsieh [49] have shown that DARTS often converges to sharp local minima in the loss landscape with high validation loss curvature for the architecture hyperparameter space. As a result, running the discretization step can cause the algorithm to return an architecture with poor generalization. To alleviate this, existing works propose to make training more robust either through data augmentation and regularization, e.g., RobustDARTS [461], by optimizing the expected or worst-case training loss over a local neighborhood of the architecture hyperparameters, e.g., Smooth-DARTS [49], or by deriving more accurate approximations of the gradients of  $\alpha$ , e.g., Amended-DARTS [26], iDARTS [467].

On a different vein, GAEA [217], XD [313] and StacNAS [132] propose a single-level optimization rather than the typical bi-level optimization by *treating architecture hyperparameters as normal weights*, showing that this leads to better generalization.

- *High memory consumption.* The memory required to train a supernet is much higher than that of a normal neural network. This is due to the fact that the size of the supernet scales linearly with the size of the set of candidate operations used. To alleviate this cost, subsequent works propose to mask out all operations except for the ones corresponding to one of a few subnetworks in each training step. This has been proposed either by randomly sampling random operations with probabilities proportional to their current value of  $\alpha$ , e.g., ProxylessNAS [37], by using a Gumbel-Softmax distribution over a one-hot encoding of the operation choices to sample single operations in a differentiable way, e.g., GDAS [90], DSNAS [159, 439], or by sampling a subset of the channels to be sent through the possible operations [448].
- *Operation biases.* Several works show that DARTS tend to favor skip connections over other operations [221, 416, 461]. To avoid this, subsequent works propose to use an early stopping method based on the stability of the raking of the architecture weights, e.g., DARTS+ [221], separating the skip connection weights from other operation weights via auxiliary edges, e.g., DARTS- [62], or by selecting the operations whose removal has the largest decrease of accuracy in the supernet, e.g., DARTS-PT [416]. Alternatively, DASH [350] learns the size and dilation of convolutional kernels by modelling them as a mixture-of-operations.
- *Discrete search space.* Although DARTS and follow-up works utilize a continuous relaxation of the discrete search space to utilize gradient descent, the nature of the

search space is still *discrete*. As a consequence, these methods require defining a small subset of possible operations to select from a priori, e.g., convolutions by kernels of size 3x3, 5x5, 7x7. Despite numerous advances in terms of memory consumption [37, 159, 448], the combinatorial nature the discrete space restrict the number of possible operations to be small for DARTS-like methods to work well. As a result, a good initial guess must be provided based on the problem at hand. However, this results in two problems: (i) this pre-selection introduces important human biases in the search process –which prevents NAS methods from finding truly novel architectures–, and (ii) the need to select a priori a (small) set of options poses challenges in terms of adaptation to new problems and datasets –one of the primary goals of the field of NAS–. It is important to note that in addition to the pre-selection of possible operations, well-performing methods *also require a different backbone architecture depending on the problem at hand*. For instance, DASH [350] uses a TCN backbone [11] for time series classification, and a WRN [456] for image classification.

### Comparison of DNArc with existing differentiable supernet-based methods

DNArc exhibits several differences to DARTS and follow-up works. To see this, we briefly recap the definition of DNArc and its use in combination with CCNNs [191].

DNArc proposes to view neural architectures as continuous multidimensional entities multi-dimensional continuous entities –with dimensions corresponding to depth, width, etc– and using learnable parametric differentiable masks to learn their size during training. By combining DNArc with a CCNN backbone [191] –a general purpose convolutional architecture–, DNArc can be used across a broad range of tasks on data of varying lengths and dimensionalities, e.g., classification on 32×32 images, segmentation on 125×125 images, and classification on sequences of length 16000, without the need for structural changes (Sec. 6.3).

**DNArc operates on a continuous search space.** First of all, by viewing the neural architectures as continuous entities, DNArc *operates on a continuous search space*. As a result, it does not need to consider a small subset of possible operations as in existing works, e.g., convolutional kernels of size 3x3, 5x5 and 7x7, e.g., DASH [350], or an small set of width multipliers [0.25, 0.5, 0.75, 1.0], e.g., OFA [38]. In contrast, DNArc is able to consider *all possible values* within an admissible range determined by the maximum size of the supernet.<sup>2</sup> For example, for a layer with N channels acting on a dataset of images of size H×W, DNArc consider all convolutional sizes between 1×1 and H×W as well as all possible width values between 1 and N channels. Consequently, for a single layer on CIFAR-10 with 140 channels, this corresponds to 32×32×140=143.360 possibilities –a much larger number of possibilities than the 3 × 4=12 possibilities given in the previous example–. Naturally, this number grows much larger when considering the

---

<sup>2</sup>This supernet can be made as big as desired as long as the network itself fits in memory. For clarity, we emphasize that this refers to the size of the network, i.e., the number of parameters used, and not the cost of performing a forward pass.

depth as well as downsampling layers, which are also learned by DNArch in the same manner. For a CCNN<sub>(8,280)</sub> and a CCNN<sub>(12,760)</sub> backbone on CIFAR-10, DNArch is able to select from approximately  $2.3 \times 10^{12}$  and  $5.5 \times 10^{15}$ , respectively.

In addition to the much larger search space, using a continuous search space allows DNArch to (*i*) strongly reduce the human biases included in the search process of existing methods and (*ii*) allows for the straightforward application of DNArch to (new) diverse problems and datasets. When looking at existing works, it is notorious that most of the existing works focus only on image classification problems and heavily rely on the design of widely-used neural architectures for the definition of the discrete search space, e.g., [50, 132, 162, 221, 231, 448]. For example, most works either take an image classification ResNet backbone, and either learn the size of convolutional kernels among predefined choices known to work well, e.g., 3x3, 5x5, 7x7, or the width of the network based on a few width multiplier values, e.g., [0.25, 0.5, 0.75, 1]. Even most of the works that learn the depth of the network do so by learning the number of layers *within each downsampling block* –using a very small set of options, e.g., 2, 3, 4 in OFA [38]–. As a result, all searchable architectures are *restricted* to architectures that *follow the resolution bottleneck structure of ResNets used for classification*. We emphasize that DNArch *does not have such constraints*. For example, we observe that DNArch finds neural architectures that resemble *concatenated U-Net-like networks for classification problems* (Tabs. E.1, E.2, E.3). We note that this kind of architectures are *dramatically different* from standard neural designs used for classification tasks. This highlights the lack of human biases in DNArch, and its ability to find truly novel architectures.

Furthermore, it is important to note that defining a good set of options for the search space prior to the start of the NAS procedure on under-explored tasks, e.g., PDE prediction, long sequences, etc, can be very difficult. Even recent works that tackle the problem of NAS on diverse tasks, e.g., DASH [350], only use a very restricted number of operations dictated by informed guesses, and *rely on backbone architectures that already encompass much prior information about the problem at hand*, therefore preventing finding truly novel architectures. In contrast, by relying on a continuous search space and a general-purpose backbone, DNArch is able to search thoroughly through a vast space of possible options without the need to use a previously tailored backbone architecture for the task at hand.

**DNArch does not suffer from operation biases.** DNArch operates on a continuous space, and utilizes differentiable parametric masks to learn the size of the operations. hence, similarly to DASH [350], DNArch learns operations as a mixture-of-operations of all operations of size smaller or equal to the current size. To visualize this, note that a kernel of size 10×10 encompasses all kernels of smaller size. Similarly, a layer of width 6 encompasses all channels within that size, i.e., 1, 2, ..., 6. As a result, DNArch *does not suffer from operation biases by design*.

**DNArch does not require discretization / fine-tuning.** Recall that the differentiable

masks define the size of the different components of the neural architecture. Since both the differentiable masks as well as the weights of the network are *trained simultaneously*, the search and the tuning of the network weights *happens simultaneously*. Consequently, once the training is finished, the final architecture is already ready to use, i.e., DNArch does not require the discretization and fine-tuning steps required in DARTS. As a consequence, DNArch *does not suffer from poor test generalization issues by design*.

**DNArch is memory efficient.** Although the supernet must fit in memory, DNArch is not required to compute the response of all candidate subnetworks at every forward pass. Hence, DNArch the cost of the forward pass is constant regardless of the number of candidate operations used. As a result, DNArch *does not suffer from the high-memory consumption issues of DARTS by design*.

In conclusion, DNArch encompasses several of the advantages that differentiable network-based methods have. However, thanks to its novel view of the problem of differentiable NAS, it provides several advantages and alleviates several disadvantages observed by existing works by design.

### E.1.3 Constrained and Multi-Objective NAS

In many settings, optimizing for a single objective, e.g., classification accuracy, is not sufficient. For instance, if a neural network must be deployed on an edge device, we may need to constrain its latency, memory usage or computational requirements. To achieve one or more objectives in addition to accuracy, the standard NAS objective is typically modified to (i) a *constrained optimization problem* [21, 37, 376] or (ii) a *multi-objective optimization problem* [95, 131, 156]. Most relevant to DNArch is the constrained optimization setting, in which one solves the equation:

$$\min_{a \in \mathcal{A}} f(a) \text{ subject to } h_i(a) \leq c_i \text{ for } i \in \{1, \dots, k\}, \quad (\text{E.1})$$

where  $f(a)$  denotes the original objective function, e.g., classification accuracy, of a network  $a$ , and  $\{h_i\}_{i \in \{1, \dots, k\}}$  represents constraints as a function of the architecture. In practice, this problem is solved by transforming it into an additive unconstrained problem of the form:

$$\min_{a \in \mathcal{A}} f(a) + \sum_i \lambda_i g_i(a); g_i(a) = \max(0, h_i(a) - c_i), \quad (\text{E.2})$$

with penalty functions  $g_i$  penalizing architectures not satisfying the constraints, and hyperparameters  $\lambda_i$  trading off the objective and constraints. This problem is subsequently solved using black-box optimization methods or one-shot methods. In the latter case, the penalty functions  $g_i$  need to be differentiable, which is *often not the case* for DARTS-like methods. Consequently, such metrics are often relaxed to continuous variables through various techniques, e.g., Gumbel softmax [435].

One of the most widely-studied constrained NAS problems is *hardware-aware NAS* [24]: a field that concerns itself with hardware efficiency constraints such as memory and

latency. While simple metrics such as the number of parameters are easily computed, these often do not correlate enough with other metrics of interest. As such, several works try to approximate hardware metrics of interest either by computing hardware costs as the sum of the cost of each operation [37], or by using a surrogate model to predict the total cost [92].

**Constrained DNArc.** As shown in the main text (Sec. 6.2.5), DNArc can also be used under constrained settings by following the method outlined in Eq. E.2. Importantly, by using a continuous search space in combination with continuous differentiable masks, DNArc is able to cast constraints on a continuous space based on the size of the masks on a continuum. As a result, DNArc permits the definition of penalty functions  $g_i$  which are differentiable with regard to the parameters of the differentiable masks, without the tricks needed in other works, e.g., gumbel softmax [435].

Although we focus on computational constraints, we note that DNArc can be easily extended to other types of constraints, as long as these constraints can be defined in terms of the mask sizes.

## E.2 Architectures Found by DNArc

The architectures found by DNArc in the experiments shown in Sec. 6.3, are shown in Tabs. E.1, E.2, and E.3.

## E.3 Learning downsampling in the spatial Domain

Differentiable Masking learns downsampling by multiplying the spectrum  $\tilde{f} = \mathcal{F}[f]$  of a signal  $f$  with a differentiable mask  $m(\cdot; \theta)$ , and cropping the output above the cutoff frequency of the mask  $\omega_{\max}$  next. However, it is not strictly necessary to perform this operation in the Fourier domain. An equivalent downsampling can also be learned directly in the spatial domain.

The *Fourier convolution theorem* states that the spatial convolution is equivalent to a pointwise multiplication in the Fourier domain. However, this equivalence works in both directions. That is, we can equivalently say that the pointwise multiplication on the Fourier domain is equal to a convolution on the spatial domain. Consequently, we can represent the pointwise multiplication of the spectrum of the input  $\mathcal{F}[f]$  and the differentiable mask  $m(\cdot; \theta)$  as the convolution of their inverse Fourier transforms. Formally:

$$\begin{aligned}\tilde{f} \cdot m(\cdot; \theta) &= \mathcal{F}[\mathcal{F}^{-1}[\tilde{f}] * \mathcal{F}^{-1}[m(\cdot; \theta)]] \\ &= \mathcal{F}[\mathcal{F}^{-1}[\mathcal{F}[f]] * \mathcal{F}^{-1}[m(\cdot; \theta)]] \\ &= \mathcal{F}[f * \mathcal{F}^{-1}[m(\cdot; \theta)]]\end{aligned}\tag{E.3}$$

In other words, we can perform the same operation in the spatial domain by convolution the original input signal  $f$  with the inverse Fourier transform of the mask  $m(\cdot; \theta)$ .

**Table E.1:** Architectures found by DNArch on LRA with the target complexity of a CCNN<sub>4</sub>, 140.

| TASK       | DEPTH | KERNEL SIZE | RESOLUTION | WIDTH<br>[N <sub>in</sub> , N <sub>mid</sub> , N <sub>out</sub> ] |
|------------|-------|-------------|------------|---|
| LISTOPS    | 8     | 266         | 2048       | [150 189 145]   |
|            |       | 569         | 632        | [150 168 168]   |
|            |       | 1401        | 1416       | [176 186 162]   |
|            |       | 310         | 310        | [166 175 153]   |
|            |       | 213         | 213        | [154 159 163]   |
|            |       | 12          | 301        | [168 128 162]   |
|            |       | 5           | 250        | [170 158 153]   |
|            |       | 24          | 502        | [153 171 165]   |
|            |       | 445         | 2284       | [180 217 205]   |
| TEXT       | 8     | 691         | 2939       | [208 176 153]   |
|            |       | 1420        | 1420       | [152 152 120]   |
|            |       | 415         | 1313       | [120 120 147]   |
|            |       | 1467        | 1467       | [147 118 135]   |
|            |       | 52          | 594        | [134 173 153]   |
|            |       | 101         | 932        | [150 156 183]   |
|            |       | 149         | 1036       | [180 92 192]  |
|            |       | 2           | 1913       | [29 33 172]   |
|            |       | 136         | 2058       | [184 174 183]   |
| RETRIEVAL  | 8     | 1013        | 2363       | [205 171 161]   |
|            |       | 1446        | 2724       | [188 164 115]   |
|            |       | 7           | 2604       | [29 29 163]   |
|            |       | 1           | 2756       | [29 35 154]   |
|            |       | 6           | 3545       | [71 110 147]  |
|            |       | 1           | 3899       | [71 88 137]   |
|            |       | 203         | 1024       | [118 155 147]   |
|            |       | 279         | 1024       | [146 172 164]   |
|            |       | 219         | 486        | [173 166 196]   |
| IMAGE      | 8     | 308         | 308        | [199 197 196]   |
|            |       | 144         | 144        | [207 197 92]  |
|            |       | 8           | 125        | [106 29 75]   |
|            |       | 30          | 96         | [78 28 110]   |
|            |       | 40          | 126        | [104 51 104]  |
|            |       | 195         | 1024       | [109 140 171]   |
|            |       | 493         | 770        | [171 168 158]   |
|            |       | 418         | 507        | [144 183 170]   |
|            |       | 318         | 318        | [173 187 178]   |
| PATHFINDER | 8     | 236         | 236        | [182 162 160]   |
|            |       | 231         | 231        | [161 121 103]   |
|            |       | 8           | 251        | [105 47 210]  |
|            |       | 4           | 253        | [116 29 188]  |
|            |       | 2484        | 15331      | [280 174 157]   |
|            |       | 7204        | 7204       | [177 280 159]   |
|            |       | 3669        | 3772       | [167 280 98]  |
|            |       | 2323        | 5496       | [123 164 164]   |
|            |       | 513         | 4768       | [136 128 195]   |
| PATH-X     | 5     | 2484        | 15331      | [280 174 157]   |
|            |       | 7204        | 7204       | [177 280 159]   |
|            |       | 3669        | 3772       | [167 280 98]  |
|            |       | 2323        | 5496       | [123 164 164]   |
|            |       | 513         | 4768       | [136 128 195]   |

**Table E.2:** Architectures found by DNArch on 2D datasets with the target complexity of a  $\text{CCNN}_{4,140}$ .

| TASK                       | DEPTH | KERNEL SIZE<br>[Y X] | RESOLUTION<br>[Y X] | WIDTH<br>[N <sub>in</sub> , N <sub>mid</sub> , N <sub>out</sub> ] |
|----------------------------|-------|----------------------|---------------------|---|
| IMAGE CLASSIFICATION TASKS |       |                      |                     |   |
| CIFAR10                    | 8     | [9 7]                | [32 32]             | [142 139 145]   |
|                            |       | [12 8]               | [32 32]             | [145 160 157]   |
|                            |       | [25 7]               | [32 20]             | [158 186 182]   |
|                            |       | [9 10]               | [9 15]              | [186 208 168]   |
|                            |       | [1 13]               | [5 15]              | [169 177 150]   |
|                            |       | [1 10]               | [6 11]              | [151 139 156]   |
|                            |       | [5 1]                | [15 4]              | [154 115 110]   |
|                            |       | [6 5]                | [11 7]              | [108 41 166]  |
|                            |       | [13 7]               | [32 32]             | [104 107 116]   |
| CIFAR100                   | 8     | [6 10]               | [32 32]             | [114 134 134]   |
|                            |       | [11 8]               | [22 22]             | [139 192 166]   |
|                            |       | [13 7]               | [16 18]             | [173 201 197]   |
|                            |       | [8 12]               | [10 12]             | [205 251 51]  |
|                            |       | [1 1]                | [8 9]               | [62 56 157]   |
|                            |       | [5 9]                | [8 10]              | [162 175 254]   |
|                            |       | [8 7]                | [9 9]               | [280 280 280]   |
|                            |       | [43 38]              | [80 72]             | [156 280 107]   |
|                            |       | [22 22]              | [22 22]             | [180 204 78]  |
| DARCY<br>FLOW              | 3     | [76 76]              | [85 85]             | [280 280 50]  |
|                            |       | [94 111]             | [128 128]           | [18 110 23]   |
|                            |       | [2 13]               | [20 45]             | [186 207 139]   |
|                            |       | [129 129]            | [129 129]           | [126 265 100]   |
|                            |       | [129 121]            | [129 129]           | [78 105 59]   |
|                            |       | [90 89]              | [129 129]           | [57 201 197]  |
|                            |       | [76 74]              | [76 74]             | [202 145 216]   |
|                            |       | [43 38]              | [80 72]             | [156 280 107]   |
|                            |       | [22 22]              | [22 22]             | [180 204 78]  |
| COSMIC                     | 6     | [76 76]              | [85 85]             | [280 280 50]  |
|                            |       | [94 111]             | [128 128]           | [18 110 23]   |
|                            |       | [2 13]               | [20 45]             | [186 207 139]   |
|                            |       | [129 129]            | [129 129]           | [126 265 100]   |
|                            |       | [129 121]            | [129 129]           | [78 105 59]   |
|                            |       | [90 89]              | [129 129]           | [57 201 197]  |
| DARCY<br>FLOW              | 3     | [76 74]              | [76 74]             | [202 145 216]   |

**Table E.3:** Architectures found by DNArch on 2D datasets with the target complexity of a CCNN<sub>6,380</sub>.

| TASK                       | DEPTH | KERNEL SIZE<br>[Y X] | RESOLUTION<br>[Y X] | WIDTH<br>[N <sub>in</sub> , N <sub>mid</sub> , N <sub>out</sub> ] |
|----------------------------|-------|----------------------|---------------------|---|
| IMAGE CLASSIFICATION TASKS |       |                      |                     |   |
| CIFAR10                    | 12    | [4 7]                | [32 32]             | [380 328 384]   |
|                            |       | [9 10]               | [32 32]             | [384 371 393]   |
|                            |       | [12 6]               | [32 32]             | [392 361 391]   |
|                            |       | [20 6]               | [32 32]             | [388 370 421]   |
|                            |       | [10 11]              | [23 26]             | [421 417 486]   |
|                            |       | [11 11]              | [12 22]             | [496 444 479]   |
|                            |       | [1 11]               | [6 11]              | [493 482 304]   |
|                            |       | [1 6]                | [5 21]              | [211 78 384]  |
|                            |       | [29 4]               | [32 4]              | [363 459 280]   |
|                            |       | [18 15]              | [18 15]             | [277 394 67]  |
|                            |       | [1 1]                | [4 4]               | [111 109 361]   |
|                            |       | [4 3]                | [21 15]             | [121 374 449]   |
| CIFAR100                   | 12    | [8 9]                | [32 32]             | [343 275 354]   |
|                            |       | [12 10]              | [32 32]             | [351 316 397]   |
|                            |       | [11 10]              | [32 32]             | [495 355 420]   |
|                            |       | [18 12]              | [29 21]             | [421 498 419]   |
|                            |       | [11 15]              | [27 24]             | [432 449 407]   |
|                            |       | [19 8]               | [25 20]             | [412 419 413]   |
|                            |       | [11 10]              | [12 23]             | [423 454 600]   |
|                            |       | [8 8]                | [8 9]               | [709 685 416]   |
|                            |       | [5 7]                | [5 8]               | [419 311 446]   |
|                            |       | [8 4]                | [8 4]               | [446 433 389]   |
|                            |       | [6 4]                | [6 4]               | [386 501 570]   |
|                            |       | [8 9]                | [8 9]               | [568 453 655]   |
| DENSE TASKS                |       |                      |                     |   |
| DARCY FLOW                 | 7     | [54 49]              | [54 49]             | [435 428 289]   |
|                            |       | [43 47]              | [70 72]             | [499 393 284]   |
|                            |       | [65 69]              | [85 85]             | [496 434 281]   |
|                            |       | [67 66]              | [85 85]             | [323 412 275]   |
|                            |       | [85 85]              | [85 85]             | [319 369 271]   |
|                            |       | [85 85]              | [85 85]             | [306 379 258]   |
|                            |       | [68 68]              | [85 85]             | [521 435 271]   |
|                            |       | [35 32]              | [35 33]             | [146 236 272]   |
| COSMIC                     | 12    | [11 21]              | [95 72]             | [170 284 319]   |
|                            |       | [44 24]              | [128 128]           | [141 339 388]   |
|                            |       | [23 41]              | [128 128]           | [385 407 361]   |
|                            |       | [28 27]              | [128 128]           | [351 279 356]   |
|                            |       | [21 19]              | [128 128]           | [354 362 310]   |
|                            |       | [29 24]              | [128 128]           | [310 351 466]   |
|                            |       | [18 25]              | [128 128]           | [396 292 183]   |
|                            |       | [57 16]              | [128 128]           | [179 210 580]   |
|                            |       | [50 11]              | [127 77]            | [273 250 63]  |
|                            |       | [18 12]              | [89 67]             | [347 400 77]  |
|                            |       | [22 23]              | [97 79]             | [171 241 79]  |

**Defining the output resolution.** Eq. E.3 defines how low-pass filtering can be performed on the spatial domain, but it does not provide information regarding the final resolution of the operation. To derive the resolution of the output, we can simply use Eqs. 6.9, 6.10 to analytically derive the size of the mask. Once the size of the mask is derived, we can simply take the downsampled signal –after using Eq. E.3–, and downsample it to match the size of the mask.

## E.4 Computational complexity of masked network components

Here, we derive the computational complexity of all layers used in the CCNN architecture with and without the use of masks. The calculation of these complexities follows the same reasoning as the pointwise linear layer provided as example in the main text.

With  $L$ ,  $N_{\text{in}}$  and  $N_{\text{out}}$  the length, number of input channels and number of output channels of a given layer, and  $\text{size}(m_{\text{res}})$ ,  $\text{size}(m_{N_{\text{in}}})$ ,  $\text{size}(m_{N_{\text{out}}})$  the size of the masks along the corresponding dimensions, the complexity of the layers used in the CCNN architectures are given by:

**Pointwise linear layer:**

$$\begin{aligned} C_{\text{lin}}(f) &= L \cdot N_{\text{in}} \cdot N_{\text{out}} \\ C_{\text{lin},\text{masked}} &= \text{size}(m_{\text{res}}) \cdot \text{size}(m_{N_{\text{in}}}) \cdot \text{size}(m_{N_{\text{out}}}) \end{aligned}$$

**Fourier convolution:**

$$\begin{aligned} C_{\mathcal{F}\text{conv}} &= L \log(L) \\ C_{\mathcal{F}\text{conv},\text{masked}} &= \text{size}(m_{\text{res}}) \log(\text{size}(m_{\text{res}})) \end{aligned}$$

**Pointwise operations –GELU, DropOut, etc.–:**

$$\begin{aligned} C_{\text{pointwise}} &= L \cdot N_{\text{in}} \\ C_{\text{pointwise},\text{masked}} &= \text{size}(m_{\text{res}}) \cdot \text{size}(m_{N_{\text{in}}}) \end{aligned}$$

## E.5 Dataset descriptions

### E.5.1 The Long Range Arena benchmark

The Long Range Arena [379] consists of six sequence modelling tasks with sequence lengths ranging from 1024 to over 16000. It encompasses modalities and objectives that require similarity, structural, and visuospatial reasoning. We follow the data preprocessing steps of Gu et al. [129], which we also include here for completeness.

**ListOps.** An extended version of the dataset presented by Nangia and Bowman [269]. The task involves computing the integer result in the range zero to nine of a mathematical expression represented in prefix notation with brackets, e.g., [MAX29[MIN47]0] → 9. Characters are encoded as one-hot vectors, with 17 unique values possible (opening

brackets and operators are grouped into a single token). The sequences are of unequal length. Hence, the end of shorter sequences is padded with a fixed indicator value to a maximum length of 2048. The task has 10 different classes representing the possible integer results of the expression. It consists of 96K training sequences, 2K validation sequences, and 2K test sequences. No data normalization is applied.

**Text.** Based on the IMDB sentiment analysis dataset presented by Maas et al. [245], the task is to classify movie reviews as having a positive or negative sentiment. The reviews are presented as a sequence of 129 unique integer tokens padded to a maximum length of 4096. The dataset contains 25K training sequences and 25K test sequences. No validation set is provided. No data normalization is applied.

**Retrieval.** Based on the ACL Anthology network corpus presented by Radev et al. [299], the task is to classify whether two given textual citations are equivalent. To accomplish this, each citation is separately passed through an encoder, and passed to a final classifier layer. Denoting  $X_1$  as the encoding for the first document and  $X_2$  as the encoding for the second document, four features are created and concatenated together as:

$$X = [X_1, X_2, X_1 \times X_2, X_1 - X_2],$$

which are subsequently passed to a two layered MLP. The goal of the task is to evaluate how well the network can represent the text by evaluating if the two citations are equivalent or not. Characters are encoded into a one-hot vector with 97 unique values and sequences are padded to a maximum length of 4000. The dataset includes 147.086 training pairs, 18.090 validation pairs, and 17.437 test pairs. No normalization is applied.

**Image.** The Image task uses  $32 \times 32$  images of the CIFAR10 dataset [195]. It views the images as sequences of length 1024 that correspond to a one-dimensional raster scan of the image. There are a total of 10 classes, 45K training examples, 5K validation examples and 10K test examples. The RGB pixel values are converted to grayscale intensities and then normalized to have zero mean and unit variance across the entire dataset.

**PathFinder.** Based on the PathFinder challenge introduced by Linsley et al. [228], the task presents a  $32 \times 32$  grayscale image with a start and an end point depicted as small circles. The task is to classify whether there is a dashed line (or path) joining the start and end points while presenting the input as a one-dimension raster scan of the image, alike the Image task. The dataset includes 160K training examples, 20K validation examples and 20K test examples. The input data is normalized to be in the range [-1, 1].

**Path-X.** Path-X is an “extreme” version of the PathFinder dataset, in which the input images are of size  $128 \times 128$ . As a result, the input sequences are sixteen times longer with a total length of 16384. Aside from this difference, the task is identical to the PathFinder dataset.

### E.5.2 Image classification datasets

**CIFAR10 and CIFAR100.** The CIFAR10 dataset [195] consists of 60K real-world  $32 \times 32$  RGB images uniformly drawn from 10 classes divided into training and test sets of 50K and 10K samples, respectively. The CIFAR100 dataset [195] is similar to the CIFAR10 dataset, with the difference that the images are uniformly drawn from 100 different classes. For validation purposes, we divide the training dataset of both CIFAR10 and CIFAR100 into training and validation sets of 45K and 5K samples, respectively. Both datasets are normalized to have zero mean and unit variance across the entire dataset.

### E.5.3 NAS-Bench-360

NAS-Bench-360 [387] is a benchmark suite to evaluate Neural Architecture Search methods beyond image classification. The benchmark is composed of ten tasks spanning a diverse array of application domains, dataset sizes, problem dimensionalities, and learning objectives. In this work, we consider two tasks from the NAS-Bench-360 suite which require dense predictions: The DarcyFlow [220] and Cosmic [466] datasets.

**DarcyFlow: Solving Partial Differential Equations.** DarcyFlow aims to solve Partial Differential Equation (PDE) by using neural networks as a replacement for traditional solvers. The input for this task is a  $85 \times 85$  grid specifying the initial conditions and coordinates of a fluid, and the output is a 2D grid of the same dimensions representing the fluid state at a later time. The ground truth for this task is the result computed by a traditional solver, and the objective is to minimize the Mean Squared Error (MSE) between the predicted fluid state and the ground truth.

**Cosmic: Identifying Cosmic Ray Contamination.** The Cosmic task involves identifying and masking corruption caused by charged particles collectively referred to as “cosmic rays” on images taken from space-based facilities. It uses imaging data of local resolved galaxies collected from the Gubble Space Telescope. The input is an  $128 \times 128$  image corresponding to the artifact of cosmic rays, and the output is a 2D grid of the same dimensions predicting whether each pixel in the input is an artifact of cosmic rays. We report the false-negative rate of identification results.

## E.6 Experimental details

### E.6.1 General remarks

**Code.** Our code is written in JAX and our experiments are conducted on TPUs and GPUs. As outlined in the Limitations (Sec. 6.4), JAX and TPU training prevent DNAArch from performing operations that change the dimensions of arrays during training. In addition to our JAX implementation, we release a PyTorch implementation of DNAArch that supports these operations. This implementation makes DNAArch more flexible and accessible, especially in scenarios where it is crucial to keep candidate networks close to the target complexity during the course of training.

**The Continuous CNN and the CCNN residual block.** The CCNN architecture is shown in Fig. 6.4. It is composed by an Encoder, a Decoder, and a number of CCNN residual blocks ResBlock [191]. The Encoder is defined as a sequence of [PWLinear, BatchNorm [166], GELU [144]] layers. For tasks dealing with text, we additionally utilize an Embedding layer mapping each token in the vocabulary to a vector representation of length equal to that used by Gu et al. [129] (see Appx. E.5.1). For dense prediction tasks, the Decoder is a PWLinear layer, which is preceded by GlobalAvgPooling for global prediction tasks.

**Batch Normalization in DNArc.** As the architecture is constantly changing during the search process, we use batch-specific statistics for batch normalization instead of the global moving average. This approach was adopted after early experiments showed that using the global moving average leads to a significant discrepancy in the behavior of the validation and training curves. Specifically, we observed that while the training curves were converging to a good solution, the validation curves resembled random predictions. This issue was resolved by deactivating the global moving average in Batch Normalization layers.

**Continuous convolutional kernels  $\text{MLP}_\psi$ .** We parameterize our convolutional kernels as a 4-layer MLP with 128 hidden units and a Fourier Encoding [377] of the form  $\gamma(\mathbf{x}) = [\cos(2\pi\omega_0 \mathbf{W}\mathbf{x}), \sin(2\pi\omega_0 \mathbf{W}\mathbf{x})]$ , with  $\mathbf{W} \in \mathbb{R}^{D \times 128}$  and  $\omega_0$  a hyperparameter that acts as a prior on the frequency content of the kernels [321, 358]. In contrast to Romero et al. [321], we utilize a single larger  $\text{MLP}_\psi$  to generate the kernels of the entire network. This allows the network  $\text{MLP}_\psi$  to administrate its capacity across all layers. Using different  $\text{MLP}_\psi$ 's for each layer as Romero et al. [321] is inadequate in the learnable architectures setting as some layers can be entirely erased. With our proposed solution, the capacity of the otherwise zeroed-out  $\text{MLP}_\psi$  is used to generate the kernels of the remaining layers.

**Normalized relative positions.** Following Romero et al. [319, 321], we normalize the coordinates going into  $\text{MLP}_\psi$  to lie in the space  $[-1, 1]^D$  for D-dimensional kernels.

**Parameters and hyperparameters of the differentiable masks.** We learn some of the parameters of the masks, and leave the others constant or treat them as a hyperparameter. Specifically, for Gaussian masks, we only learn their width, i.e.,  $\sigma$ , and fix its mean to zero. For Sigmoid masks, we learn their offset  $\mu$  and treat their temperature  $\tau$  as a hyperparameter. For more information regarding the values of  $\tau$  used in our parameter tuning step, please refer to Appx. E.6.2.

**Maximum and minimum allowable sizes for the learnable differentiable masks.** We define some minimum and maximum allowable sizes for the mask parameters, and reset them to these values after each training iteration if the updated parameter values lie outside that range. For the Gaussian mask, we constraint the minimum admissible value of  $\sigma$  such that the length of the corresponding dimension never collapses to a value of 1. This is to prevent the corresponding dimension to collapse such that it can grow afterwards if required. The minimum value depends on the resolution of the cor-

**Table E.4:** Hyperparameters used for the experiments with the target complexity of a  $\text{CCNN}_{4,140}$ .

| DATASET    | EPOCHS | BATCH SIZE | LEARNING RATE | DROPOUT | WEIGHT DECAY | $\omega_0$ | $\lambda$ | $\tau_{\text{resolution}}$ | $\tau_{\text{channel}}$ | $\tau_{\text{depth}}$ |
|------------|--------|------------|---------------|---------|--------------|------------|-----------|----------------------------|-------------------------|-----------------------|
| LISTOPS    | 50     | 50         | 0.005         | 0.0     | 0.01         | 27.5K      | 5.0       | 50                         | 25                      | 8                     |
| TEXT       | 100    | 50         | 0.02          | 0.2     | 0.01         | 19.5K      | 0.1       | 50                         | 25                      | 8                     |
| RETRIEVAL  | 50     | 50         | 0.001         | 0.1     | 0.01         | 21.5K      | 0.1       | 50                         | 25                      | 8                     |
| IMAGE      | 210    | 50         | 0.02          | 0.1     | 0.001        | 12.5K      | 0.1       | 25                         | 25                      | 8                     |
| PATHFINDER | 210    | 50         | 0.005         | 0.0     | 0.001        | 21.5K      | 0.1       | 50                         | 25                      | 8                     |
| PATH-X     | 80     | 32         | 0.001         | 0.0     | 0.0          | 30K        | 0.1       | 100                        | 25                      | 8                     |
| CIFAR10    | 210    | 50         | 0.01          | 0.1     | 0.01         | 21.5K      | 0.1       | 25                         | 25                      | 8                     |
| CIFAR100   | 210    | 50         | 0.01          | 0.0     | 0.01         | 6.5K       | 5.0       | 25                         | 25                      | 8                     |
| DARCYFLOW  | 310    | 8          | 0.02          | 0.0     | 0.0001       | 24.5K      | 0.1       | 50                         | 25                      | 8                     |
| COSMIC     | 310    | 8          | 0.02          | 0.3     | 0.0001       | 5.5K       | 0.1       | 100                        | 25                      | 8                     |

**Table E.5:** Hyperparameters used for the experiments with the target complexity of a  $\text{CCNN}_{6,380}$ .

| DATASET   | EPOCHS | BATCH SIZE | LEARNING RATE | DROPOUT | WEIGHT DECAY | $\omega_0$ | $\lambda$ | $\tau_{\text{resolution}}$ | $\tau_{\text{channel}}$ | $\tau_{\text{depth}}$ |
|-----------|--------|------------|---------------|---------|--------------|------------|-----------|----------------------------|-------------------------|-----------------------|
| CIFAR10   | 210    | 50         | 0.005         | 0.0     | 0.01         | 21.5K      | 0.1       | 50                         | 50                      | 16                    |
| CIFAR100  | 210    | 50         | 0.01          | 0.0     | 0.01         | 6.5K       | 0.1       | 25                         | 25                      | 8                     |
| DARCYFLOW | 310    | 12         | 0.01          | 0.2     | 0.0          | 7.5K       | 0.1       | 50                         | 25                      | 8                     |
| COSMIC    | 310    | 4          | 0.01          | 0.2     | 0.01         | 5.5K       | 0.1       | 50                         | 50                      | 8                     |

responding dimension, e.g., the maximum size of the convolutional kernel, and can be easily calculated with Eq. 6.4.

Note that the offset value of the Sigmoid mask  $\mu$  could in principle assume any value in  $\mathbb{R}$ . However, if not controlled,  $\mu$  could become too small and mask all values along a particular dimension to zero. Similarly, if  $\mu$  is too large, the gradient of the mask at all positions would become very small and it would difficult to update the mask. To avoid these situations, we define minimum and maximum values of  $\mu$  such that for the lowest value, the mask at the lowest position is equal to 0.95, and for the largest value, the mask at the highest position is equal 0.85. These values are dependent on the value of the mask temperature  $\tau$ , and can be easily calculated with Eq. 6.5.

**Limiting the size of the mask to the maximum allowable ranges.** As outlined in the Limitations (Sec. 6.4), we must set a maximum allowable size for the width and depth of the network on JAX. However, the maximum allowed value for the parameters of the masks (see previous paragraph) allows both masks to grow beyond the point on which the theoretical size of the masks is equal to the maximum allowable network size. For instance, for the maximum allowed parameter values of a Sigmoid mask, the last channel, i.e., the 280-th channel, would be weighted by a factor of 0.85. Consequently, the theoretical size of the mask as calculated by Eq. 6.10 will be well beyond 280. This value would lead to an unrealistic theoretical computational complexity that surpasses the real computational complexity the CCNN used.

To overcome this issue, we limit the maximum size of the mask calculated by Eq. 6.10 to be less or equal than the maximum allowable size, e.g.,  $\text{size}(m) = \min(\text{size}(m), 280)$ . It is important to note, however, that clipping the value of size directly would stop the gradient flow for parameter values leading to sizes larger 280. As a result, once the maximum size is reached, the mask would not be able to contract anymore. We avoid gradient flow stop by using clipping in combination with a straight-through estimator [23]. As a result, we are able to propagate the gradient across the clipping operation, and the resulting mask can still be modified even in the cropping operation is used.

## E.6.2 Hyperparameters and training configurations

In this section, we include more information about the found hyperparameters, the values that were considered during hyperparameter tuning, and other training settings. The final hyperparameters used are listed in Table E.5.

**Optimizer, learning rates and learning rate schedule.** All our models are optimized with AdamW [239] in combination with a cosine annealing learning rate scheduler [238], and a linear learning rate warm-up stage of 10 epochs, except for `ListOps`, `Retrieval` and `Path-X` for which we have a warm-up stage of 5 epochs.

**Regularization.** We utilize dropout [366] –as shown in Fig. 6.4– as well as weight decay during training.

### Hyperparameter tuning

**Frequency prior  $\omega_0$ .** The possible  $\omega_0$  values explored in this work are:  $[1, 500, 1500, 2500, \dots, 28500, 29500, 30000]$ .

**Tuning the value of  $\lambda$ .**  $\lambda$  plays the role of controlling the weight of the computational loss  $\mathcal{L}_{\text{comp}}$  relative to the task objective loss  $\mathcal{L}_{\text{obj}}$ . In this work, we find two settings which require different values of  $\lambda$ . One, given by the tasks that converge to a low prediction values relative to perfection, i.e., `ListOps` and `CIFAR100`, and for which the loss  $\mathcal{L}_{\text{obj}}$  remains relatively high at the end of training. The other group is given by all the other tasks, which converge to high prediction values –many even obtaining a perfect accuracy on the train set–, and for which  $\mathcal{L}_{\text{obj}}$  converges to values close to zero. For the first group, we require a higher value of  $\lambda$  such that the computational complexity loss  $\mathcal{L}_{\text{comp}}$  remains relevant to the optimization objective. The final values of  $\lambda$  used are 5.0 and 0.1, respectively.

**Tuning the temperature of the Sigmoid masks  $\tau$ .** For the resolution mask, we consider three values of  $\tau$ ,  $[25, 50, 100]$  which correspond to a minimum size of 10%, 5% and 2.5% of the corresponding dimension. For the channel mask, we consider two values  $\tau \in [25, 50]$  which correspond to a minimum size of 10% and 5% of the corresponding dimension, but observe early during tuning that models prefer  $\tau=25$ . For the depth dimension, which is much more sparse than the channel and resolution dimensions, we

consider two values  $\tau \in [8, 16]$ , which result on a minimum depth of 2 and 1 layers, respectively. We observe early during tuning that models prefer  $\tau=8$ .

**Learning rate.** The possible learning rate values explored in this work are: [0.0001, 0.0005, 0.001, 0.005, 0.01, 0.02].

**Dropout.** The possible dropout values explored in this work are: [0.0, 0.1, 0.2, 0.3].

**Weight decay.** The possible weight decay values considered in this work are: [0.0, 0.0001, 0.001, 0.01, 0.05].

# F

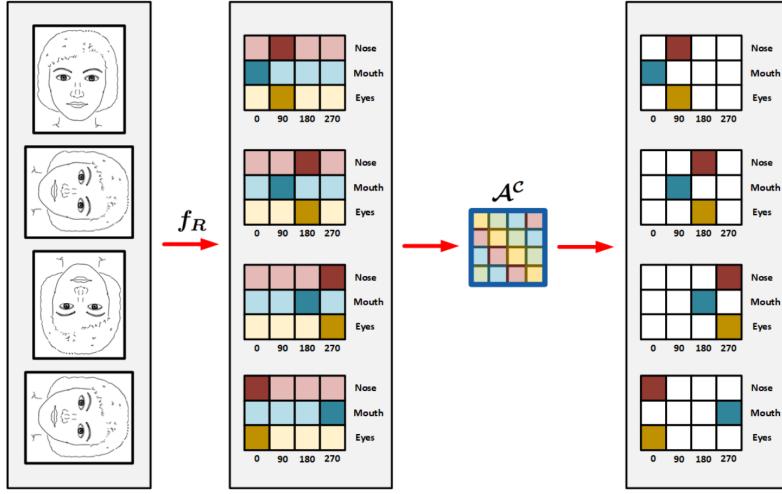
## Focusing Equivariance on Transformations Co-Occurring in Data

### F.1 Obtaining Co-Occurrent Attention via Equation 7.5

In this section, we provide a meticulous description on how co-occurrent attention is obtained via the method presented in the paper. Intuitively, a direct approach to address the problem illustrated in the introduction (Section 7.1) and Figure 7.2 requires an attention mechanism that acts simultaneously on  $r$  and  $\lambda$  (see Eq. 7.3). However, we illustrate how the depicted problem can be simplified such that attention along  $r$  is sufficient by taking advantage of the equivariance property of the network.

Let  $p$  be the input of a roto-translational convolution  $f_R : \mathbb{Z}^2 \times \Theta \times \Lambda_0 \rightarrow \mathbb{Z}^2 \times \Theta \times \Lambda_1$  as defined in Eq. 7.3, and  $\Theta$  be the set of rotations by  $\theta_r$  degrees:  $\Theta = \{\theta_r = r \frac{2\pi}{r_{\max}}\}_{r=1}^{r_{\max}}$ . Let  $f_R(p)(u) \in \mathbb{R}^{r_{\max} \times \Lambda_1}$  be the matrix consisting of the  $r_{\max}$  oriented responses for each  $\lambda \in \Lambda_1$  learned representation at a certain position  $u$ . Since the vectors  $f_R(p)(u, \lambda) \in \mathbb{R}^{r_{\max}}$ ,  $\lambda \in \Lambda_1$  permute cyclically as a result of the rotation equivariance property of  $f_R$ , it is mandatory to ensure equivariance to cyclic permutations for each  $f_R(p)(u, \lambda)$  during the course of the attention procedure (see Section 7.3).

At first sight, one is inclined to think that there is no connection between multiple vectors  $f_R(p)(u, \lambda)$  in  $f_R(p)(u)$ , and, therefore, in order to exploit co-occurrences, one must impose additional constraints along the  $\lambda$  axis. However, there is indeed an implicit restriction in  $f_R(p)(u)$  along  $\lambda$  resulting from the rotation equivariance property of the mapping  $f_R$ , which we can take advantage from to simplify the problem at hand. Con-



**Figure F.1:** Synchronous movement of feature mappings and attention masks as a function of input rotation in the group  $p4$  ( $r_{\max} = 4$ ).

sider, for instance, the input  $\theta_i p$ , a  $\theta_i$ -rotated version of  $p$ . By virtue of the equivariance property of  $f_R$ , we have (locally) that  $f_R(\theta_i p) = \mathcal{P}^i(f_R(p))$ . Furthermore, we know that this property must hold for all the learned feature representations  $f_R(p)(u, \lambda), \forall \lambda \in \Lambda_1$ . Resultantly, we have that:

$$f_R(\theta_i p)(u, r, \lambda) = \mathcal{P}^i(f_R(p)(u, r, \lambda)), \quad \forall \lambda \in \Lambda_1. \quad (\text{F.1})$$

In other words, if one of the learned mappings  $f_R(p)(u, r, \lambda)$  experiences a permutation  $\mathcal{P}^i$  along  $r$ , *all* the learned representations  $f_R(p)(u, r, \lambda), \forall \lambda \in \Lambda_1$  must experience the exact same permutation  $\mathcal{P}^i$  as well. Resultantly, the equivariance property of the mapping  $f_R$  ensures that *all* the  $\Lambda_1$  learned feature representations  $f_R(p)(u, \lambda)$  “*move synchronously*” as a function of input rotation  $\theta_i$ .

Likewise, if we apply a cyclic equivariant attention mechanism  $\mathcal{A}_\lambda^C$  independently on top of each  $\lambda$  learned representation  $f_R(p)(u, \lambda)$ , we obtain that the relation

$$\mathcal{A}_\lambda^C(f_R(\theta_i p))(u, r, \lambda) = \mathcal{P}^i(\mathcal{A}_\lambda^C(f_R(p))(u, r, \lambda)), \quad \forall \lambda \in \Lambda_1, \quad (\text{F.2})$$

must hold as well. Analogous to the case shown in Eq. F.1 and given that  $\mathcal{A}_\lambda^C$  is equivariant to cyclic permutations on the domain, we obtain that *all* the  $\Lambda_1$  learned *attention masks*  $\mathcal{A}_\lambda^C$  “*move synchronously*” as a function of input rotation  $\theta_i$  as well (see Fig. F.1).

From Eq. F.2 and Figure F.1, one can clearly see that by utilizing  $\mathcal{A}_\lambda^C$  independently along  $r$  and taking advantage from the fact that all  $\Lambda_1$  learned feature representations are tied with one another via  $f_R$ , one is able to prioritize learning of feature representations that co-occur together as opposed to the much looser formulation in Eq. F.1, where feedback is obtained from all orientations.



# Attentive Group Equivariant Convolutional Neural Networks

## G.1 Generalized Visual Self-Attention

Before we derive the constraints for general visual self-attention and prove Thm. 8.3.1 of the main article, we first motivate our definition of group equivariant visual self-attention. In this section, we explain that our definition of attentive group convolution, as given in Eq. 8.14, and reformulated in Eq. G.5, essentially describes a group equivariant linear mapping that is augmented with an additional attention function.

### G.1.1 Self-attention: From Vectors to Feature Maps

Let us first consider the general form of a linear map between respectively vector spaces (used in multi-layer perceptrons) and feature maps (used in (group) convolutional neural nets), defined as follows:

$$\text{vectors: } \mathbf{x}_c^{out} = \sum_{\tilde{c}}^{N_c} \mathbf{W}_{c,\tilde{c}} \mathbf{x}_{\tilde{c}}^{in}, \quad (\text{G.1})$$

$$\text{feat maps: } f_c^{out}(g) = \sum_{\tilde{c}}^{N_c} \int_G \Psi_{c,\tilde{c}}(g, \tilde{g}) f_{\tilde{c}}^{in}(\tilde{g}) d\tilde{g}. \quad (\text{G.2})$$

Here, the first equation describes a linear map between vectors  $\mathbf{x}^{in} \in \mathbb{R}^{N_c}$  and  $\mathbf{x}^{out} \in \mathbb{R}^{N_c}$  via matrix-vector multiplication with matrix  $\mathbf{W} \in \mathbb{R}^{N_c \times N_c}$ . The second equation describes a linear map between feature maps  $f^{in} \in (\mathbb{L}_2(G))^{N_c}$  and  $f^{out} \in (\mathbb{L}_2(G))^{N_c}$ , via a two

argument kernel  $\Psi \in \mathbb{L}_1(G \times G)^{N_c \times N_c}$ . The two argument kernel  $\Psi$  can be seen as the continuous counterpart of the matrix  $\mathbf{W}$ , and matrix-vector multiplication (sum over input indices) is augmented with an integral over the input coordinates  $\tilde{g}$ .

Keeping this form of linear mapping, we define the self-attentive map as the regular linear map augmented with attention weights computed from the input. Consequently, we formally define the self-attentive mappings as:

$$\text{vectors: } \mathbf{x}_c^{out} = \sum_{\tilde{c}}^{N_{\tilde{c}}} \mathbf{A}_{c,\tilde{c}} \mathbf{W}_{c,\tilde{c}} \mathbf{x}_c^{in}, \quad (\text{G.3})$$

$$\text{feat maps: } f_c^{out}(g) = \sum_{\tilde{c}}^{N_{\tilde{c}}} \int_G \alpha_{c,\tilde{c}}(g, \tilde{g}) \Psi_{c,\tilde{c}}(g, \tilde{g}) f_{\tilde{c}}^{in}(\tilde{g}) d\tilde{g}, \quad (\text{G.4})$$

in which the attention weights are computed from the input via some operator  $\mathcal{A}$ , i.e.,  $\mathbf{A}_{c,\tilde{c}} = \mathcal{A}[\mathbf{x}^{in}]_{c,\tilde{c}}$  in the vector case and  $\alpha_{c,\tilde{c}} = \mathcal{A}[f^{in}]_{c,\tilde{c}}$  in the case of feature maps.

### G.1.2 Equivariant Linear Maps are Group Convolutions

Now, since we want to preserve the spatial correspondences between the input and output feature maps, special attention should be paid to the continuous self-attentive mappings. In other words, these operators should be equivariant. By including an equivariance constraint on the linear mapping of Eq. G.2 we obtain a group convolution (see e.g. Bekkers [17], Cohen et al. [68], Kondor and Trivedi [194]). The derivation is as follows:

Imposing the equivariance constraint  $\mathcal{L}_g[f^{in}] \xrightarrow{\text{Eq.G.2}} \mathcal{L}_{\bar{g}}[f^{out}]$  means that for all  $\bar{g}, g \in G$  and all  $f \in \mathbb{L}_2(G)^{N_c}$  we must guarantee that:

$$\begin{aligned} \mathcal{L}_g[f^{in}] &= \mathcal{L}_{\bar{g}}[f^{out}] \\ &\Leftrightarrow \\ \int_G \Psi_{c,\tilde{c}}(g, \tilde{g}) \mathcal{L}_{\bar{g}}[f](\tilde{g}) d\tilde{g} &= \int_G \Psi_{c,\tilde{c}}(\bar{g}^{-1}g, \tilde{g}) f(\tilde{g}) d\tilde{g} \\ &\Leftrightarrow \\ \int_G \Psi_{c,\tilde{c}}(g, \tilde{g}) f(\bar{g}^{-1}\tilde{g}) d\tilde{g} &= \int_G \Psi_{c,\tilde{c}}(\bar{g}^{-1}g, \tilde{g}) f(\tilde{g}) d\tilde{g} \\ &\Leftrightarrow \\ \int_G \Psi_{c,\tilde{c}}(g, \tilde{g}) f(\bar{g}^{-1}\tilde{g}) d\tilde{g} &= \int_G \Psi_{c,\tilde{c}}(\bar{g}^{-1}g, \bar{g}^{-1}\tilde{g}) f(\bar{g}^{-1}\tilde{g}) d\tilde{g}, \end{aligned}$$

where the change of variables  $\tilde{g} \rightarrow \bar{g}^{-1}\tilde{g}$  as well as the left-invariance of the Haar measure ( $d(\bar{g}^{-1}\tilde{g}) = d\tilde{g}$ ) is used in the last step. Since this equality must hold for all  $f \in \mathbb{L}_2(G)^{N_{\tilde{c}}}$ ,  $\Psi$  should be left-invariant in both input arguments. In other words, we have that:

$$\forall \bar{g} \in G : \Psi(\bar{g}g, \bar{g}\tilde{g}) = \Psi(g, \tilde{g}).$$

Resultantly, we can always multiply both arguments with  $g^{-1}$  and obtain  $\Psi(e, g^{-1}\tilde{g})$ , which is effectively a single argument function  $\psi(g^{-1}\tilde{g}) := \Psi(e, g^{-1}\tilde{g})$  that takes as input

a relative “displacement”  $g^{-1}\tilde{g}$ . Consequently, under the equivariance constraint, Eq. G.2 becomes a group convolution:

$$f_c^{out}(g) = \sum_{\tilde{c}}^{N_c} \int_G \psi_{c,\tilde{c}}(g^{-1}\tilde{g}) f_{\tilde{c}}^{in}(\tilde{g}) d\tilde{g}.$$

### G.1.3 Proof of Theorem 1

We can apply the same type of derivation to reduce the general form of visual self-attention of Eq. G.4 to our main definition of attentive group convolution:

$$f_c^{out}(g) = \sum_{\tilde{c}}^{N_c} \int_G \alpha_{c,\tilde{c}}(g, \tilde{g}) \psi_{c,\tilde{c}}(g^{-1}\tilde{g}) f_{\tilde{c}}^{in}(\tilde{g}) d\tilde{g}. \quad (\text{G.5})$$

However, we cannot reduce attention map  $\alpha$  to a single argument function like we did for the kernel  $\Psi$  since  $\alpha$  depends on the input  $f^{in}$ . To see this consider the following:

Without loss of generality, let  $\mathfrak{A} : \mathbb{L}_2(G) \rightarrow \mathbb{L}_2(G)$  denote the attentive group convolution defined by Eq. G.5, with  $N_c = N_{\tilde{c}} = 1$ , and some  $\psi$  which in the following we omit in order to simplify our derivation. Equivariance of  $\mathfrak{A}$  implies that  $\forall_{f \in \mathbb{L}_2(G)}, \forall_{\bar{g}, g \in G}$ :

$$\begin{aligned} \mathfrak{A}[\mathcal{L}_{\bar{g}}[f]](g) &= \mathcal{L}_{\bar{g}}[\mathfrak{A}[f]](g) \\ &\Leftrightarrow \\ \mathfrak{A}[\mathcal{L}_{\bar{g}}[f]](g) &= \mathfrak{A}[f](\bar{g}^{-1}g) \\ &\Leftrightarrow \\ \int_G \mathcal{A}[\mathcal{L}_{\bar{g}}[f]](g, \tilde{g}) \mathcal{L}_{\bar{g}}[f](\tilde{g}) d\tilde{g} &= \int_G \mathcal{A}[f](\bar{g}^{-1}g, \tilde{g}) f(\tilde{g}) d\tilde{g} \\ &\Leftrightarrow \\ \int_G \mathcal{A}[\mathcal{L}_{\bar{g}}[f]](g, \tilde{g}) f(\bar{g}^{-1}\tilde{g}) d\tilde{g} &= \int_G \mathcal{A}[f](\bar{g}^{-1}g, \bar{g}^{-1}\tilde{g}) f(\bar{g}^{-1}\tilde{g}) d\tilde{g}, \end{aligned}$$

where we once again perform the variable substitution  $\tilde{g} \rightarrow \bar{g}^{-1}\tilde{g}$  at the right hand side of the last step. This must hold for all  $f \in \mathbb{L}_2(G)$  and hence:

$$\forall_{\bar{g} \in G} : \mathcal{A}[\mathcal{L}_{\bar{g}}f](g, \tilde{g}) = \mathcal{A}[f](\bar{g}^{-1}g, \bar{g}^{-1}\tilde{g}), \quad (\text{G.6})$$

which proves the constraint on  $\mathcal{A}$  as given in Thm. 1 of the main article. Just as for convolutions in Sec. G.1.2, we can turn this into a single argument function as:

$$\mathcal{A}[f](g, \tilde{g}) = \mathcal{A}[\mathcal{L}_{g^{-1}}f](e, g^{-1}\tilde{g}) =: \mathcal{A}'[\mathcal{L}_{g^{-1}}f](g^{-1}\tilde{g}), \quad (\text{G.7})$$

in which  $\mathcal{A}'$  is an attention operator that generates a single argument attention map from an input  $f$ . However, this would mean that for each  $g$  the input should be transformed via  $\mathcal{L}_{g^{-1}}$ , which does not make things easier for us. Things do get easier when we choose to attend to either the input or the output, which we discuss next.

**Corollary G.1.0.1.** *Each attention operator  $\mathcal{A}$  that generates an attention map  $\alpha : G \times G \rightarrow$*

$[0, 1]$  which is left-invariant to either one of the arguments, and thus exclusively attends either the input or output domain, satisfies the equivariance constraint of Eq. G.6, iff the operator is  $G$ -equivariant, i.e., a group convolution.

*Proof.* Left-invariant to either one of the arguments (let us now consider invariance in the first argument) means that:

$$\forall_{g, \tilde{g}} : \mathcal{A}[f](g, \tilde{g}) = \mathcal{A}[f](e, \tilde{g}),$$

and hence, we are effectively dealing with a single argument attention map, which we define as  $\mathcal{A}'[f](\tilde{g}) := \mathcal{A}(e, \tilde{g})$ . Hence, the equivariance constraint of Eq. G.6 becomes:

$$\begin{aligned} \forall_{\bar{g} \in G} : \mathcal{A}[\mathcal{L}_{\bar{g}} f](g, \tilde{g}) = \mathcal{A}[f](\bar{g}^{-1}g, \bar{g}^{-1}\tilde{g}) &\Leftrightarrow \\ \forall_{\bar{g} \in G} : \mathcal{A}'[\mathcal{L}_{\bar{g}} f](\tilde{g}) = \mathcal{A}'[f](\bar{g}^{-1}\tilde{g}) &\Leftrightarrow \\ \forall_{\bar{g} \in G} : \mathcal{A}'[\mathcal{L}_{\bar{g}} f] = \mathcal{L}_{\bar{g}}[\mathcal{A}'][f]. \end{aligned}$$

Conclusively,  $\mathcal{A}'$  must be an equivariant operator.  $\square$

The derivation of the Eq. G.6 together with the proof of Corollary G.1.0.1 completes the proof of Theorem 8.3.1.

#### G.1.4 Equivariance Proof of the Proposed Visual Attention

In this section we revisit the proposed attention mechanisms and prove that they indeed satisfy Thm. 1 of the main article. Recall the general formulation of attentive group convolution given in Eq. G.5. Inspired by Woo et al. [431], we reduce the computation load by factorizing the attention map  $\alpha$  into channel and spatial components:

$$\alpha_{c, \tilde{c}}(g, \tilde{g}) = \alpha^C(x, h, \tilde{h}) \alpha_{c, \tilde{c}}^C(h, \tilde{h}),$$

where  $\alpha^C$  attends to both input and output channels as well as input and output poses  $h, \tilde{h} \in H$ , and spatial attention attends to the output domain  $g = (x, h) \in G$  for all input poses  $\tilde{h} \in H$  but does not change for input spatial positions  $\tilde{x} \in \mathbb{R}^d$ . We denote the operators  $\mathcal{A}^C, \mathcal{A}^X$  utilized to compute the attention maps as  $\alpha^C = \mathcal{A}^C[f]$  and  $\alpha^X = \mathcal{A}^X[f]$ .

##### Channel attention

We compute channel attention via:<sup>1</sup>:

$$\begin{aligned} \mathcal{A}^C[f](h, \tilde{h}) &= \varphi^C[s^C[\tilde{f}[f]]](h, \tilde{h}) \\ &= \sigma([\mathbf{W}_2(h^{-1}\tilde{h}) \cdot [\mathbf{W}_1(h^{-1}\tilde{h}) \cdot s_{\text{avg}}^C(h, \tilde{h})]^+]^+ [\mathbf{W}_2(h^{-1}\tilde{h}) \cdot [\mathbf{W}_1(h^{-1}\tilde{h}) \cdot s_{\text{max}}^C(h, \tilde{h})]^+]^+) \end{aligned} \quad (\text{G.8})$$

with

$$\tilde{f}_{c, \tilde{c}}(x, h, \tilde{h}) := [\tilde{f}_{\tilde{c}} \star_{\mathbb{R}^d} \mathcal{L}_h[\psi_{c, \tilde{c}}]](x, \tilde{h}) \quad (\text{G.9})$$

---

<sup>1</sup>In the main text, we write Eqs. G.8, G.10 in convolution form by which only attention to  $\tilde{h}$  is considered. However, in the rot-MNIST experiments we compute attention based on  $\tilde{f}$  and apply attention to both  $h$  and  $\tilde{h}$  via Eqs. G.8, G.10.

the intermediary result from the convolution between the input  $f$  and the  $h$ -transformation of the filter  $\psi$ ,  $\mathcal{L}_h[\psi]$  before pooling over  $\tilde{c}$  and  $\tilde{h}$ .  $s_{\text{avg}}^{\mathcal{C}}$  and  $s_{\text{max}}^{\mathcal{C}}$  denote respectively average and max pooling over the  $x$  coordinate.

Here, we apply a slight abuse of notation with  $\tilde{f}[f]$  and  $s^{\mathcal{C}}[\tilde{f}]$  in order to keep track of the dependencies. In order to proof equivariance of the attention operator  $\mathcal{A}^{\mathcal{C}}$  we need to proof that  $\forall \bar{g} \in G : \mathcal{A}^{\mathcal{C}}[\mathcal{L}_{\bar{g}}[f]](h, \tilde{h}) = \mathcal{A}^{\mathcal{C}}[f](\bar{h}^{-1}h, \bar{h}^{-1}\tilde{h})$ , with  $\bar{g} = (\bar{x}, \bar{h})$ . To this end, we first identify the equivariance and invariance properties of the functions used in Eq. G.8.

From Eq. G.9 we see that the intermediate convolution result  $\tilde{f}$  is equivariant via  $\tilde{f}[\mathcal{L}_{\bar{g}}[f]](\tilde{x}, \tilde{h}) = \tilde{f}[f](\bar{g}^{-1}\tilde{x}, \bar{h}^{-1}h, \bar{h}^{-1}\tilde{h})$ . For the statistics operators  $s^{\mathcal{C}}$  we have invariance w.r.t. translations due to the pooling over  $x$ , and equivariance w.r.t. parameter  $\bar{h}$  via  $s^{\mathcal{C}}[\tilde{f}[\mathcal{L}_{\bar{g}}[f]]](h, \tilde{h}) = s^{\mathcal{C}}[\tilde{f}[f]](\bar{h}^{-1}h, \bar{h}^{-1}\tilde{h})$ . Now, we propagate the transformation on the input and compute the result of  $\mathcal{A}^{\mathcal{C}}[\mathcal{L}_{\bar{g}}[f]](g, \tilde{g})$ . That is, we compute the left-hand side of the constraint given in Eq. G.6, where, for brevity, we omit the  $s_{\text{max}}^{\mathcal{C}}$  term:

$$\mathcal{A}^{\mathcal{C}}[\mathcal{L}_{\bar{g}}[f]](g, \tilde{g}) = \mathbf{W}_2(h^{-1}\tilde{h}) \cdot [\mathbf{W}_1(h^{-1}\tilde{h}) \cdot s_{\text{avg}}^{\mathcal{C}}(\bar{h}^{-1}h, \bar{h}^{-1}\tilde{h})]^+.$$

The right-hand side of Eq. G.6 is given by:

$$\mathcal{A}^{\mathcal{C}}[f](\bar{g}^{-1}g, \bar{g}^{-1}\tilde{g}) = \mathbf{W}_2(h^{-1}\tilde{h}) \cdot [\mathbf{W}_1(h^{-1}\tilde{h}) \cdot s_{\text{avg}}^{\mathcal{C}}(\bar{h}^{-1}h, \bar{h}^{-1}\tilde{h})]^+.$$

and hence, Eq. G.6 is satisfied for all  $\bar{g} \in G$ . Resultantly,  $\mathcal{A}^{\mathcal{C}}$  is a valid attention operator.

### Spatial attention

The spatial attention map  $\alpha^{\mathcal{X}}$  is computed via:

$$\begin{aligned} \alpha^{\mathcal{X}}(g, \tilde{h}) &= \mathcal{A}^{\mathcal{X}}[f](g, \tilde{h}) \\ &= \varphi^{\mathcal{X}}[s^{\mathcal{X}}[\tilde{f}[f]]](g, \tilde{h}) = \sigma([s^{\mathcal{X}} \star_{\mathbb{R}^d} \mathcal{L}_h[\psi^{\mathcal{X}}]])(x, \tilde{h}) \end{aligned} \quad (\text{G.10})$$

where  $\sigma$  is a point-wise logistic sigmoid,  $\psi^{\mathcal{X}} : G \rightarrow \mathbb{R}^2$  is a group convolution filter and  $s^{\mathcal{X}}[\tilde{f}] : G \times H \rightarrow \mathbb{R}^2$  is a map of averages and maximum values taken over the channel axis at each  $g \in G$  in  $\tilde{f}$  for each  $\tilde{h} \in H$ . Note that Eq. G.10 corresponds to a group convolution up to the final pooling operation over  $\tilde{h}$ . Since the statistics operator  $s^{\mathcal{X}}$  is invariant w.r.t. translations in the input and Eq. G.10 corresponds to a group convolution (up to pooling over  $\tilde{h}$ ), we have that  $\mathcal{A}^{\mathcal{X}}$  is a valid attention operator as well.

## G.2 Extended Implementation Details

In this section we provide extended details over our implementation. For the sake of completeness and reproducibility, we summarize the exact training procedures utilized during our experiments. Moreover, we delve into some important changes performed to some network architectures during our experiments to ensure *exact* equivariance, and shed light into their importance for our equivariant attention maps.

### G.2.1 General Observations

We utilize PyTorch for our implementation. Any missing parameter specification in the following sections can be safely considered to be the default value of the corresponding parameter. For batch normalization layers, we utilize  $\text{eps}=0.00002$  similarly to Cohen and Welling [65].

### G.2.2 rot-MNIST

For rotational MNIST, we utilized the same backbone network as in Cohen and Welling [65]. During training we utilize Adam [185], batches of size 128, weight decay of 0.0001, learning rate of 0.001, drop-out rate of 0.3 and perform training for 100 epochs. Importantly and contrarily to Cohen and Welling [65], we consistently experience improvements when utilizing drop-out and therefore we do not exclude it for any model.

### G.2.3 CIFAR-10

It is not clear from Cohen and Welling [65], Springenberg et al. [365] which batch size is used in their experiments. For our experiments, we always utilize batches of size 128.

#### All-CNN

We utilize the All-CNN-C structure of Springenberg et al. [365]. Analogously to Cohen and Welling [65], Springenberg et al. [365], we utilize stochastic gradient descent, weight decay of 0.001 and perform training for 350 epochs. We utilize a grid search on the set  $\{0.01, 0.05, 0.1, 0.25\}$  for the learning rate and report the best obtained performance. Furthermore, we reduce the learning rate by a factor of 10 at epochs 200, 250 and 300.

#### ResNet44

Similar to Cohen and Welling [65], we utilize stochastic gradient descent, learning rate of 0.05 and perform training for 300 epochs. Furthermore, we reduce the learning rate by a factor of 10 at epochs 50, 100 and 150.

### G.2.4 PCam

During training on the PatchCamelyon dataset, we utilize Adam [185], batches of size 64, weight decay of 0.0001, learning rate of 0.001 and perform training for 100 epochs. Furthermore, we reduce the learning rate by a factor of 2 after 20 epochs of no improvement in the validation loss.

## G.3 Effects of Stride and Input Size on Equivariance

Theoretically seen, the usage of stride during pooling and during convolution is of no relevance for the equivariance properties of the corresponding mapping [65]. However,

we see that in practice stride can affect equivariance for specific cases as is the case for our experiments on CIFAR-10.

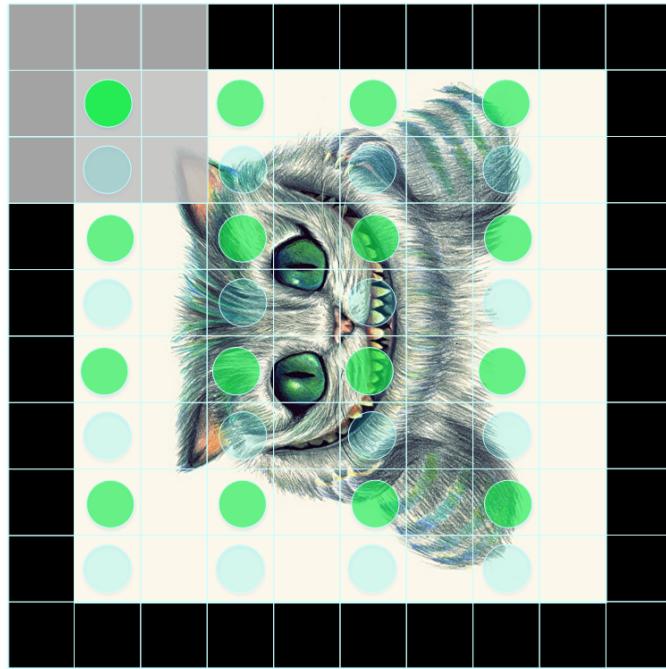
Consider the convolution between an input of even size and a small 3x3 filter as shown in Fig. G.1a. Via group convolutions, we can ensure that the output of the original input and a rotated one (Fig. G.1b) will be exactly equal (up to the same rotation). Importantly however, note that for Fig. G.1, the local support of the filter, i.e., the input section with which the filter is convolved at a particular position, *is not equivalent* for rotated versions of the input (denoted by blue circles for the non-rotated case and by green circles for the rotated case). As a result, despite the group convolution itself being equivariant, the responses of both convolutions do not entirely resemble one another and, consequently, the depicted strided group convolution is *not* exactly equivariant.

It is important to highlight that this behaviour is just exhibited for the special case when the residual between the used stride and the input size is even. Unfortunately, this is the case both for the ResNet44 as well as the All-CNN networks utilized in our CIFAR-10 experiments. However, as neighbouring pixels are extremely correlated with one another, the effects of this phenomenon are not of much relevance for the classification task itself. As a matter of fact, it can be interpreted as a form of data augmentation by skipping intermediary pixel values. Consequently, we can say that these networks are *approximately equivariant*.

Importantly, this phenomenon does affect the resulting equivariant attention maps generated via attentive group convolutions as shown in Fig. G.2. As these networks are only equivariant in an approximate manner, the generated attention maps are slightly deformed versions of one another for multiple orientations. To alleviate this problem, we replace all strided convolutions in the All-CNN and ResNet44 architectures by conventional convolutions with stride=1, followed by spatial max pooling. As a result, we can produce exactly equivariant attention maps as shown in Figs. 8.7, 8.8 and G.3.

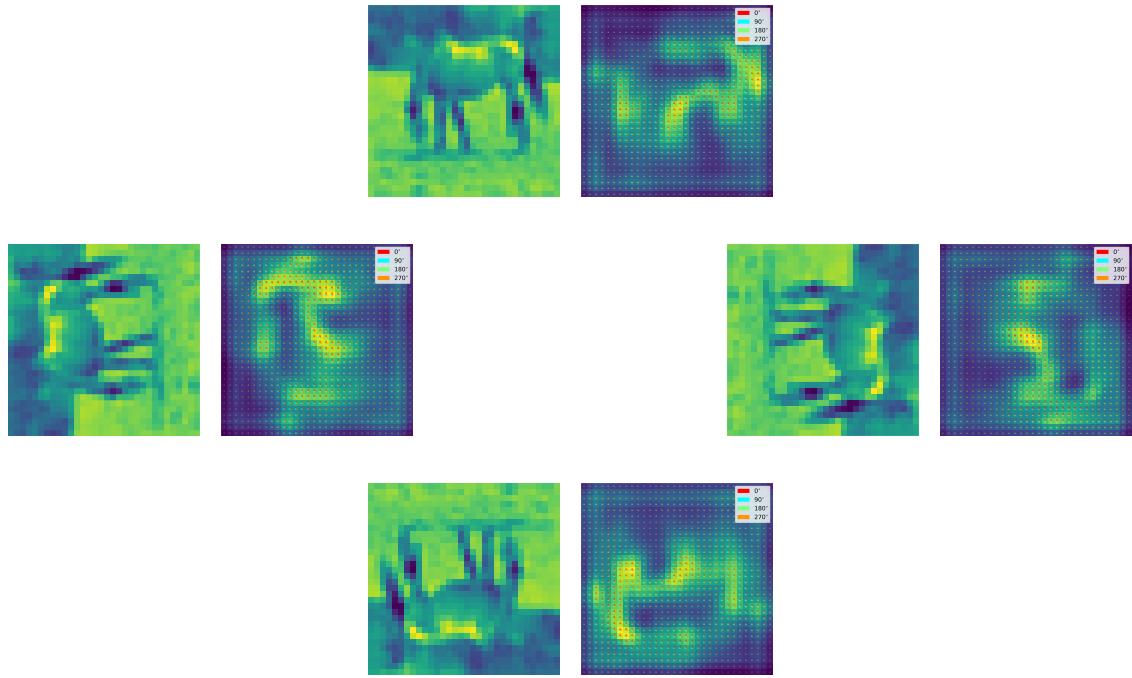


(a)

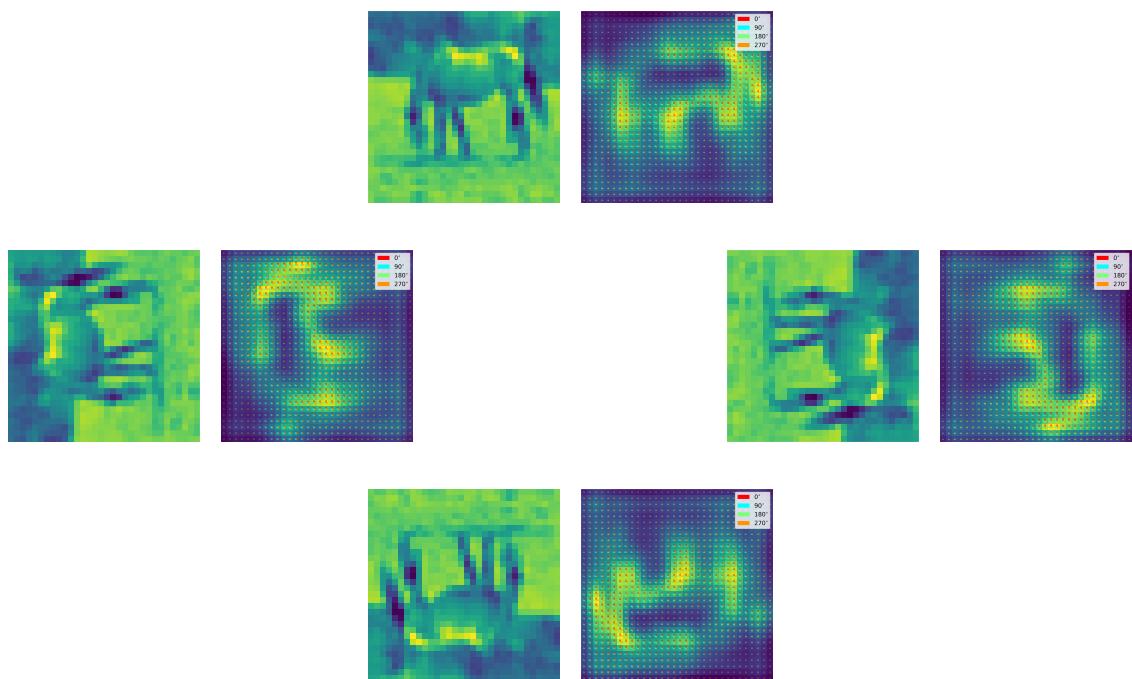


(b)

**Figure G.1:** Effect of stride and input size on exact equivariance. Although group convolutions are ensured to be group equivariant, in practice, if the residual between the stride and the input size is even, as it's the case for the networks utilized in the CIFAR-10 experiments, equivariance is only approximate. This has important effects on equivariant attention maps (Fig. G.2).



**Figure G.2:** Examples of equivariant attention maps under approximate equivariance. Note that the attention map around the horse’s back changes for different orientations.



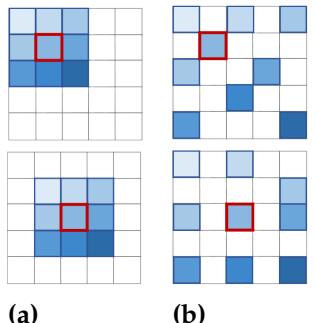
**Figure G.3:** Examples of equivariant attention maps under the exact equivariance regime.





# Group Equivariant Stand-Alone Self-Attention

## H.1 Convolution and self-attention: A graphical comparison



**Figure H.1:** Parameter usage in convolutional kernels (Fig. H.1a) and self-attention (Fig. H.1b). Given a budget of 9 parameters, a convolutional filter ties these parameters to specific positions. Subsequently, these parameters remain static regardless of (i) the query input position and (ii) the input signal itself. Self-attention, on the other hand, does not tie parameters to any specific positions at all. As a result, provided enough heads, self-attention is more general than convolutions, as it can represent any convolutional kernel, e.g., Fig. H.1a, as well as several other functions defined on its receptive field.

## H.2 Concepts From Group Theory

**Definition H.2.0.1 (Group).** A group is a tuple  $(\mathcal{G}, \cdot)$ , where  $\mathcal{G}$  is a set and  $\cdot : \mathcal{G} \times \mathcal{G} \rightarrow \mathcal{G}$  is a binary operation on  $\mathcal{G}$ , such that (i) the set is closed under this operation, (ii) the operation is associative, i.e.,  $(g_1 \cdot g_2) \cdot g_3 = g_1 \cdot (g_2 \cdot g_3)$ ,  $g_1, g_2, g_3 \in \mathcal{G}$ , (iii) there exists an identity element  $e \in \mathcal{G}$  s.t.  $\forall g \in \mathcal{G}$  we have  $e \cdot g = g \cdot e = g$ , and (iv) for each  $g \in \mathcal{G}$ , there exists an inverse  $g^{-1}$  s.t.  $g \cdot g^{-1} = e$ .

**Definition H.2.0.2 (Subgroup).** Let  $(\mathcal{G}, \cdot)$  be a group. A subset  $\mathcal{H}$  of  $\mathcal{G}$  is a subgroup of  $\mathcal{G}$  if  $\mathcal{H}$  is nonempty and closed under the group operation and inverses (i.e.,  $h_1, h_2 \in \mathcal{H}$  implies that  $h_1^{-1} \in \mathcal{H}$  and  $h_1 \cdot h_2 \in \mathcal{H}$ ). If  $\mathcal{H}$  is a subgroup of  $\mathcal{G}$  we write  $\mathcal{H} \leq \mathcal{G}$ .

**Definition H.2.0.3 (Semi-direct product and affine groups).** In practice, one is mainly interested in the analysis of data defined on  $\mathbb{R}^d$ , and, consequently, in groups of the form  $\mathcal{G} = \mathbb{R}^d \rtimes \mathcal{H}$ , resulting from the semi-direct product ( $\rtimes$ ) between the translation group  $(\mathbb{R}^d, +)$  and an arbitrary (Lie) group  $\mathcal{H}$  that acts on  $\mathbb{R}^d$ , e.g., rotation, scaling, mirroring, etc. This family of groups is referred to as affine groups and their group product is defined as:

$$g_1 \cdot g_2 = (x_1, h_1) \cdot (x_2, h_2) = (x_1 + h_1 \odot x_2, h_1 \cdot h_2), \quad (\text{H.1})$$

with  $g = (x_1, h_1)$ ,  $g_2 = (x_2, h_2) \in \mathcal{G}$ ,  $x_1, x_2 \in \mathbb{R}^d$  and  $h_1, h_2 \in \mathcal{H}$ . The operator  $\odot$  denotes the action of  $h \in \mathcal{H}$  on  $x \in \mathbb{R}^d$ , and it describes how a vector  $x \in \mathbb{R}^d$  is modified by elements  $h \in \mathcal{H}$ .

**Definition H.2.0.4 (Group representation).** Let  $\mathcal{G}$  be a group and  $\mathbb{L}^2(X)$  be a space of functions defined on some vector space  $X$ . The (left) regular group representation of  $\mathcal{G}$  is a linear transformation  $\mathcal{L} : \mathcal{G} \times \mathbb{L}^2(X) \rightarrow \mathbb{L}^2(X)$ ,  $(g, f) \mapsto \mathcal{L}_g[f] := f(g^{-1} \odot x)$ , that shares the group structure via:

$$\mathcal{L}_{g_1} \mathcal{L}_{g_2}[f] = \mathcal{L}_{g_1 \cdot g_2}[f] \quad (\text{H.2})$$

for any  $g_1, g_2 \in \mathcal{G}$ ,  $f \in \mathbb{L}_2(X)$ . That is, concatenating two such transformations, parameterized by  $g_1$  and  $g_2$ , is equivalent to a single transformation parameterized by  $g_1 \cdot g_2 \in \mathcal{G}$ . If the group  $\mathcal{G}$  is affine, the group representation  $\mathcal{L}_g$  can be split as:

$$\mathcal{L}_g[f] = \mathcal{L}_x \mathcal{L}_h[f], \quad (\text{H.3})$$

with  $g = (x, h) \in \mathcal{G}$ ,  $x \in \mathbb{R}^d$  and  $h \in \mathcal{H}$ . Intuitively, the representation of  $\mathcal{G}$  on a function  $f$  describes how the function as a whole, i.e.,  $f(x), \forall x \in X$ , is transformed by the effect of group elements  $g \in \mathcal{G}$ .

### H.3 Actions and Representations of Groups Acting on Homogeneous Spaces for Functions Defined on Sets

In this section we show that the action of a group  $\mathcal{G}$  acting on a homogeneous space  $\mathcal{X}$  is well defined on sets  $\mathcal{S}$  gathered from  $\mathcal{X}$ , and that it induces a group representation of functions defined on  $\mathcal{S}$ .

Let  $\mathcal{S} = \{i\}$  be a set and  $\mathcal{X}$  be a homogeneous space on which the action of  $\mathcal{G}$  is well-defined, i.e.,  $gx \in \mathcal{X}, \forall g \in \mathcal{G}, \forall x \in \mathcal{X}$ . Since  $\mathcal{S}$  has been gathered from  $\mathcal{X}$ , there exists an injective map  $x : \mathcal{S} \rightarrow \mathcal{X}$ , that maps set elements  $i \in \mathcal{S}$  to unique elements  $x_i \in \mathcal{X}$ . That is, there exists a map  $x : i \mapsto x_i$  that assigns an unique value  $x_i \in \mathcal{X}$  to each  $i \in \mathcal{S}$  corresponding to the coordinates from which the set element has been gathered.

Since the action of  $\mathcal{G}$  is well defined on  $\mathcal{X}$ , it follows that the left regular representation (Def. H.2.0.4)  $\mathcal{L}_g[f^\mathcal{X}](x_i) := f^\mathcal{X}(g^{-1}x_i) \in L_Y(\mathcal{X})$  of functions  $f^\mathcal{X} \in L_Y(\mathcal{X})$  exists

and is well-defined. Since  $x$  is injective, the left regular representation  $\mathcal{L}_g[f^{\mathcal{X}}](x_i) = f^{\mathcal{X}}(g^{-1}x_i)$  can be expressed uniquely in terms of set indices as  $\mathcal{L}_g[f^{\mathcal{X}}](x_i) = f^{\mathcal{X}}(g^{-1}x_i) = f^{\mathcal{X}}(g^{-1}x(i))$ . Furthermore, its inverse  $x^{-1} : \mathcal{X} \rightarrow \mathcal{S}$ ,  $x^{-1} : x_i \rightarrow i$  also exist and is well-defined. As a consequence, points  $x_i \in \mathcal{X}$  can be expressed uniquely in terms of set indices as  $i = x^{-1}(x_i), i \in \mathcal{S}$ . Consequently, functions  $f^{\mathcal{X}} \in L_y(\mathcal{X})$  can be expressed in terms of functions  $f \in L_y(\mathcal{S})$  by means of the equality  $f^{\mathcal{X}}(i) = f(x^{-1}(x_i))$ . Resultantly, we see that the group representation  $\mathcal{L}_g[f^{\mathcal{X}}](x_i) = f^{\mathcal{X}}(g^{-1}x(i))$  can be described in terms of functions  $f \in L_y(\mathcal{S})$  as:

$$\mathcal{L}_g[f^{\mathcal{X}}](x_i) = f^{\mathcal{X}}(g^{-1}x(i)) = f^{\mathcal{X}}(g^{-1}x(i)) = f(x^{-1}(g^{-1}x(i))) = \mathcal{L}_g[f](i),$$

with a corresponding group representation on  $L_y(\mathcal{S})$  given by  $\mathcal{L}_g[f](i) = f(x^{-1}(g^{-1}x(i)))$ , and an action of group elements  $g \in \mathcal{G}$  on set elements  $i$  given by  $gi := x^{-1}(gx(i))$ .

## H.4 The Case of Non-Unimodular Groups: Self-Attention on the Scale-Translation Group

The lifting and group self-attention formulation provided in Sec. 9.5.2 are only valid for unimodular groups. That is, for groups whose action does not change the volume of the objects they act upon, e.g., rotation, mirroring, etc. Non-unimodular groups, however, do modify the volume of the acted objects [17]. The most relevant non-unimodular group for this work is the scale group  $\mathcal{H} = (\mathbb{R}_{>0}, \times)$ . To illustrate why this distinction is important, consider the following example:

Imagine we have a circle on  $\mathbb{R}^2$  of area  $\pi r^2$ . If we rotate, mirror or translate the circle, its size is kept constant. If we increase its radius by a factor  $h \in \mathbb{R}_{>0}$ , however, its size would increase by  $h^2$ . Imagine that we have an application for which we would like to recognize this circle regardless of any of these transformations by means of self-attention. For this purpose, we define a neighborhood  $\mathcal{N}$  for which the original circle fits perfectly. Since the size of the circle is not modified for any translated, rotated or translated versions of it, we would still be able to detect the circle regardless of these transformations. If we scale the circle by a factor of  $h > 1$ , however, the circle would fall outside of our neighborhood  $\mathcal{N}$  and hence, we would not be able to recognize it.

A solution to this problem is to scale our neighborhood  $\mathcal{N}$  in a proportional way. That is, if the circle is scaled by a factor  $h \in \mathbb{R}_{>0}$ , we scale our neighborhood by the same factor  $h$ :  $\mathcal{N} \rightarrow h\mathcal{N}$ . Resultantly, the circle would fall within the neighborhood for any scale factor  $h \in \mathbb{R}_{>0}$ . Unfortunately, there is a problem: self-attention utilizes summations over its neighborhood. Since  $\sum_{i \in h\mathcal{N}} i > \sum_{i \in \mathcal{N}} i$ , for  $h > 1$ , and  $\sum_{i \in h\mathcal{N}} i < \sum_{i \in \mathcal{N}} i$ , for  $h < 1$ , the result of the summations would still differ for different scales. Specifically, this result would always be bigger for larger versions of the neighborhood. This is problematic, as the response produced by the same circle, would still be different for different scales.

In order to handle this problem, one utilizes a normalization factor proportional to the change of size of the neighborhood considered. This ensures that the responses are

equivalent for any scale  $\hbar \in \mathbb{R}_{>0}$ . As a result, we obtain that  $\sum_{i \in \hbar_1 \mathcal{N}} (\hbar_1^2)^{-1} i = \sum_{i \in \hbar_2 \mathcal{N}} (\hbar_2^2)^{-1} i$ ,<sup>1</sup>  $\forall \hbar_1, \hbar_2 \in \mathbb{R}_{>0}$ .

In the example above we have provided an intuitive description of the (left invariant) Haar measure  $d\mu(\hbar)$ . As its name indicates, it is a measure defined on the group, which is invariant over all group elements  $\hbar \in \mathcal{H}$ . For several unimodular groups, the Haar measure corresponds to the Lebesgue measure as the volume of the objects the group acts upon is kept equal, i.e.,  $d\mu(\hbar) = d\hbar$ .<sup>2</sup> For non-unimodular groups, however, the Haar measure requires a normalization factor proportional to the change of volume of these objects. Specifically, the Haar measure corresponds to the Lebesgue measure times a normalization factor  $\hbar^d$ , where  $d$  corresponds to the dimensionality of the space  $\mathbb{R}^d$  the group acts upon [17, 322], i.e.,  $d\mu(\hbar) = \frac{1}{\hbar^d} d\hbar$ .

In conclusion, in order to obtain group equivariance to non-unimodular groups, the lifting and group self-attention formulation provided in Eq. 9.14, 9.16 must be modified via normalization factors proportional to the group elements  $\hbar \in \mathcal{H}$ . Specifically, they are redefined as:

$$m_{\mathcal{G}\uparrow}^r[f, \rho](i, \hbar) = \varphi_{\text{out}} \left( \bigcup_{h \in [H]} \sum_{j \in \hbar \mathcal{N}(i)} \frac{1}{\hbar^d} \sigma_j \left( \langle \varphi_{\text{qry}}^{(h)}(f(i)), \varphi_{\text{key}}^{(h)}(f(i) + \mathcal{L}_\hbar[\rho](i, j)) \rangle \right) \varphi_{\text{val}}^{(h)}(f(j)) \right) \quad (\text{H.4})$$

$$m_{\mathcal{G}}^r[f, \rho](i, \hbar) = \varphi_{\text{out}} \left( \bigcup_{h \in [H]} \sum_{\tilde{\hbar} \in \mathcal{H}} \sum_{\substack{(j, \hat{\hbar}) \in \hbar \mathcal{N}(i, \tilde{\hbar}) \\ + \mathcal{L}_\hbar[\rho](i, \tilde{\hbar}, j)}} \frac{1}{\hbar^{d+1}} \sigma_{j, \hat{\hbar}} \left( \langle \varphi_{\text{qry}}^{(h)}(f(i, \tilde{\hbar})), \varphi_{\text{key}}^{(h)}(f(j, \hat{\hbar})) \rangle \right) \varphi_{\text{val}}^{(h)}(f(j, \hat{\hbar})) \right). \quad (\text{H.5})$$

The factor  $d + 1$  in Eq. H.5 results from the fact that the summation is now performed on the group  $\mathcal{G} = \mathbb{R}^d \rtimes \mathcal{H}$ , an space of dimensionality  $d + 1$ . An interesting case emerges when global neighborhoods are considered, i.e., s.t.  $\mathcal{N}(i) = \mathcal{S}$ ,  $\forall i \in \mathcal{S}$ . Since  $\hbar \mathcal{N}(i) = \mathcal{N}(i) = \mathcal{S}$  for any  $\hbar > 1$ , approximation artifacts are introduced. It is not clear if it is better to introduce normalization factors in these situations or not. An in-depth investigation of this phenomenon is left for future research.

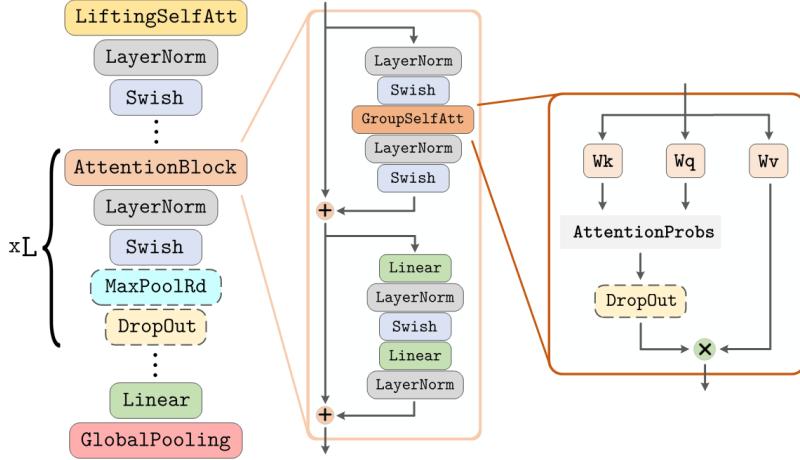
#### H.4.1 Current empirical aspects of scale equivariant self-attention

Self-attention suffers from quadratic memory and time complexity proportional to the size of the neighborhood considered. This constraint is particularly important for the scale group, for which these neighborhoods grow as a result of the group action. We envisage two possible solutions to this limitation left out for future research:

The most promising solution is given by incorporating recent advances in efficient self-attention in group self-attention, e.g., Choromanski et al. [59], Katharopoulos et al. [181], Kitaev et al. [189], Wang et al. [418], Zaheer et al. [458]. By reducing the quadratic complexity of self-attention, the current computational constraints of scale equivariant self-attention can be (strongly) reduced. The resulting architectures would be comparable to

<sup>1</sup>The squared factor in  $\hbar_1^2$  and  $\hbar_2^2$  appears as a result that the neighborhood growth is quadratic in  $\mathbb{R}^2$ .

<sup>2</sup>This is why this subtlety is often left out in group equivariance literature.



**Figure H.2:** Graphical description of group self-attention networks. Dot-lined blocks depict optional blocks. Linear layers are applied point-wise across the feature map. Swish non-linearities [300] and LayerNorm [8] are used all across the network. **GlobalPooling** consists of max-pool over group elements followed by spatial mean-pool.

Bekkers [17], Romero et al. [322], Sosnovik et al. [362] in terms of their functionality and the discretizations they can manage.

The second option is to draw a self-attention analogous to Worrall and Welling [433], where scale equivariance is implemented via dilated convolutions. One might consider an analogous to dilated convolutions via “sparse” dilations of the self-attention neighborhood. As a result, scale equivariance can be implemented while retaining an equal computational cost for all group elements. Importantly however, this strategy is viable for a dyadic set of scales only, i.e., a set of scales given by a set  $\{2^j\}_{j=0}^{j_{\max}}$ , and thus, less general than the scale-equivariant architectures listed before.

## H.5 Experimental Details

In this section we provide extended details over our implementation as well as the exact architectures and optimization schemes used in our experiments. All our models follow the structure shown in Fig. H.2 and vary only in the number of blocks and channels. All self-attention operations utilize 9 heads. We utilize PyTorch for our implementation. Any missing specification can be safely considered to be the PyTorch default value

### H.5.1 Rotated MNIST

For rotational MNIST we use a group self-attention network composed of 5 attention blocks with 20 channels. We utilize automatic mixed precision during training to reduce memory requirements. `attention_dropout_rate` and `value_dropout_rate` are both set to 0.1. We train for 300 epochs and utilize the Adam optimizer, batch size of 8, weight decay of 0.0001 and learning rate of 0.001.

### H.5.2 CIFAR-10

For CIFAR-10 we use a group self-attention network composed of 6 attention blocks with 96 channels for the first two blocks and 192 channels for the rest. `attention_dropout_rate` and `value_dropout_rate` are both set to 0.1. We use dropout on the input with a rate of 0.3 and additional dropout blocks of rate 0.2 followed by spatial max-pooling after the second and fourth block. We did not use automatic mixed precision training for this dataset as it made all models diverge. We perform training for 350 epochs and utilize stochastic gradient descent with a momentum of 0.9 and cosine learning rate scheduler with base learning rate 0.01 [238]. We utilize a batch size of 24, weight decay of 0.0001 and He's initialization.

### H.5.3 PatchCamelyon

For PatchCamelyon we use a group self-attention network composed of 4 attention blocks with 12 channels for the first block, 24 channels for the second block, 48 channels for the third and fourth blocks and 96 channels for the last block. `attention_dropout_rate` and `value_dropout_rate` are both set to 0.1. We use an additional max-pooling block after the lifting block to reduce memory requirements. We did not use automatic mixed precision training for this dataset as it made all models diverge. We perform training for 100 epochs, utilize stochastic gradient descent with a momentum of 0.9 and cosine learning rate scheduler with base learning rate 0.01 [238]. We utilize a batch size of 8, weight decay of 0.0001 and He's initialization.

## H.6 Proofs

*Proof of Proposition 9.4.1.* If the self-attention formulation provided in Eqs. 9.3, 9.8, is permutation equivariant, then it must hold that  $m[\mathcal{L}_\pi[f]](i) = \mathcal{L}_\pi[m[f]](i)$ . Consider a permuted input signal  $\mathcal{L}_\pi[f](i) = f(\pi^{-1}(i))$ . The self-attention operation on  $\mathcal{L}_\pi[f]$  is:

$$\begin{aligned}
m[\mathcal{L}_\pi[f]](i) &= \varphi_{\text{out}} \left( \bigcup_{h \in [H]} \sum_{j \in \mathcal{S}} \sigma_j \left( \langle \varphi_{\text{qry}}^{(h)}(\mathcal{L}_\pi[f](i)), \varphi_{\text{key}}^{(h)}(\mathcal{L}_\pi[f](j)) \rangle \right) \varphi_{\text{val}}^{(h)}(\mathcal{L}_\pi[f](j)) \right) \\
&= \varphi_{\text{out}} \left( \bigcup_{h \in [H]} \sum_{j \in \mathcal{S}} \sigma_j \left( \langle \varphi_{\text{qry}}^{(h)}(f(\pi^{-1}(i))), \varphi_{\text{key}}^{(h)}(f(\pi^{-1}(j))) \rangle \right) \varphi_{\text{val}}^{(h)}(f(\pi^{-1}(j))) \right) \\
&= \varphi_{\text{out}} \left( \bigcup_{h \in [H]} \sum_{\pi(j) \in \mathcal{S}} \sigma_{\pi(j)} \left( \langle \varphi_{\text{qry}}^{(h)}(f(i)), \varphi_{\text{key}}^{(h)}(f(j)) \rangle \right) \varphi_{\text{val}}^{(h)}(f(j)) \right) \\
&= \varphi_{\text{out}} \left( \bigcup_{h \in [H]} \sum_{\bar{j} \in \mathcal{S}} \sigma_{\bar{j}} \left( \langle \varphi_{\text{qry}}^{(h)}(f(i)), \varphi_{\text{key}}^{(h)}(f(\bar{j})) \rangle \right) \varphi_{\text{val}}^{(h)}(f(\bar{j})) \right) \\
&= m[f](\bar{i}) = m[f](\pi^{-1}(i)) \\
&= \mathcal{L}_\pi[m[f]](i).
\end{aligned}$$

Here we have used the substitution  $\bar{i} = \pi(i)$  and  $\bar{j} = \pi(j)$ . Since the summation is defined over the entire set we have that  $\sum_{\pi(j) \in \mathcal{S}} [\cdot] = \sum_{\bar{j} \in \mathcal{S}} [\cdot]$ . Conclusively, we see that

$m[\mathcal{L}_\pi[f]](i) = \mathcal{L}_\pi[m[f]](i)$ . Hence, permutation equivariance indeed holds.  $\square$

**Proof of Claim 9.4.2. Permutation equivariance.** If the self-attention formulation provided in Eq. 9.10 is permutation equivariant, then it must hold that  $m[\mathcal{L}_\pi[f], \rho](i) = \mathcal{L}_\pi[m[f, \rho]](i)$ . Consider a permuted input signal  $\mathcal{L}_\pi[f](i) = f(\pi^{-1}(i))$ . The self-attention operation on  $\mathcal{L}_\pi[f]$  is given by:

$$\begin{aligned} m[\mathcal{L}_\pi[f], \rho](i) \\ = \varphi_{\text{out}} \left( \bigcup_{h \in [H]} \sum_{j \in \mathcal{N}(i)} \sigma_j \left( \langle \varphi_{\text{qry}}^{(h)}(\mathcal{L}_\pi[f](i) + \rho(i)), \varphi_{\text{key}}^{(h)}(\mathcal{L}_\pi[f](j) + \rho(j)) \rangle \right) \varphi_{\text{val}}^{(h)}(\mathcal{L}_\pi[f](j)) \right) \end{aligned}$$

As discussed in Sec. 9.3.2, since there exists permutations in  $\mathbb{S}_N$  able to send elements  $j$  in  $\mathcal{N}(i)$  to elements  $\tilde{j}$  outside of  $\mathcal{N}(i)$ , it is trivial to show that Eq. 9.10 is not equivariant to  $\mathbb{S}_N$ . Consequently, in order to provide a more interesting analysis, we consider global attention here, i.e., cases where  $\mathcal{N}(i) = \mathcal{S}$ . As shown for Proposition 9.4.1, this self-attention instantiation is permutation equivariant. Consequently, by considering this particular case, we are able to explicitly analyze the effect of introducing absolute positional encodings into the self-attention formulation. We have then that:

$$\begin{aligned} m[\mathcal{L}_\pi[f], \rho](i) \\ = \varphi_{\text{out}} \left( \bigcup_{h \in [H]} \sum_{j \in \mathcal{S}} \sigma_j \left( \langle \varphi_{\text{qry}}^{(h)}(f(\pi^{-1}(i)) + \rho(i)), \varphi_{\text{key}}^{(h)}(f(\pi^{-1}(j)) + \rho(j)) \rangle \right) \varphi_{\text{val}}^{(h)}(f(\pi^{-1}(j))) \right) \\ = \varphi_{\text{out}} \left( \bigcup_{h \in [H]} \sum_{\pi(\tilde{j}) \in \mathcal{S}} \sigma_{\tilde{j}} \left( \langle \varphi_{\text{qry}}^{(h)}(f(\tilde{i}) + \rho(\pi(\tilde{i}))), \varphi_{\text{key}}^{(h)}(f(\tilde{j}) + \rho(\pi(\tilde{j}))) \rangle \right) \varphi_{\text{val}}^{(h)}(f(\tilde{j})) \right) \\ = \varphi_{\text{out}} \left( \bigcup_{h \in [H]} \sum_{\tilde{j} \in \mathcal{S}} \sigma_{\tilde{j}} \left( \langle \varphi_{\text{qry}}^{(h)}(f(\tilde{i}) + \rho(\pi(\tilde{i}))), \varphi_{\text{key}}^{(h)}(f(\tilde{j}) + \rho(\pi(\tilde{j}))) \rangle \right) \varphi_{\text{val}}^{(h)}(f(\tilde{j})) \right) \end{aligned}$$

Here we have used the substitution  $\tilde{i} = \pi(i)$  and  $\tilde{j} = \pi(j)$ . Since the summation is defined over the entire set we have that  $\sum_{\pi(\tilde{j}) \in \mathcal{S}} [\cdot] = \sum_{\tilde{j} \in \mathcal{S}} [\cdot]$ . Since  $\rho(\pi(\tilde{i})) \neq \rho(\tilde{i})$  and  $\rho(\pi(\tilde{j})) \neq \rho(\tilde{j})$ , we are unable to reduce the expression further towards the form of  $m[f, \rho](\tilde{i})$ . Hence, we conclude that absolute position-aware self-attention is not permutation equivariant.

**Translation equivariance.** If the self-attention formulation provided in Eq. 9.10, is translation equivariant, then it must hold that  $m[\mathcal{L}_y[f], \rho](i) = \mathcal{L}_y[m(f, \rho)](i)$ . Consider a

translated input signal  $\mathcal{L}_y[f](i) = f(x^{-1}(x(i) - y))$ . Self-attention on  $\mathcal{L}_y[f]$  is given by:

$$\begin{aligned}
m[\mathcal{L}_y[f], \rho](i) &= \varphi_{\text{out}} \left( \bigcup_{h \in [H]} \sum_{j \in \mathcal{N}(i)} \sigma_j \left( \langle \varphi_{\text{qry}}^{(h)}(\mathcal{L}_y[f](i) + \rho(i)), \varphi_{\text{key}}^{(h)}(\mathcal{L}_y[f](j) + \rho(j)) \rangle \right) \varphi_{\text{val}}^{(h)}(\mathcal{L}_y[f](j)) \right) \\
&= \varphi_{\text{out}} \left( \bigcup_{h \in [H]} \sum_{j \in \mathcal{N}(i)} \sigma_j \left( \langle \varphi_{\text{qry}}^{(h)}(f(x^{-1}(x(i) - y)) + \rho(i)), \varphi_{\text{key}}^{(h)}(f(x^{-1}(x(j) - y)) + \rho(j)) \rangle \right) \varphi_{\text{val}}^{(h)}(f(x^{-1}(x(j) - y))) \right) \\
&= \varphi_{\text{out}} \left( \bigcup_{h \in [H]} \sum_{x^{-1}(x(\bar{j}) + y) \in \mathcal{N}(x^{-1}(x(\bar{i}) + y))} \sigma_{x^{-1}(x(\bar{j}) + y)} \left( \langle \varphi_{\text{qry}}^{(h)}(f(\bar{i}) + \rho(x^{-1}(x(\bar{i}) + y))), \varphi_{\text{key}}^{(h)}(f(\bar{j}) + \rho(x^{-1}(x(\bar{j}) + y))) \rangle \right) \varphi_{\text{val}}^{(h)}(f(\bar{j})) \right) \\
&= \varphi_{\text{out}} \left( \bigcup_{h \in [H]} \sum_{x^{-1}(x(\bar{j}) + y) \in \mathcal{N}(x^{-1}(x(\bar{i}) + y))} \sigma_{x^{-1}(x(\bar{j}) + y)} \left( \langle \varphi_{\text{qry}}^{(h)}(f(\bar{i}) + \rho^P(x(\bar{i}) + y)), \varphi_{\text{key}}^{(h)}(f(\bar{j}) + \rho^P(x(\bar{j}) + y)) \rangle \right) \varphi_{\text{val}}^{(h)}(f(\bar{j})) \right)
\end{aligned}$$

Here, we have used the substitution  $\bar{i} = x^{-1}(x(i) - y) \Rightarrow i = x^{-1}(x(\bar{i}) + y)$  and  $\bar{j} = x^{-1}(x(j) - y) \Rightarrow j = x^{-1}(x(\bar{j}) + y)$ . Since the area of summation remains equal to any translation  $y \in \mathbb{R}^d$ , we have that:

$$\sum_{x^{-1}(x(\bar{j}) + y) \in \mathcal{N}(x^{-1}(x(\bar{i}) + y))} [\cdot] = \sum_{x^{-1}(x(\bar{j}) + y) \in \mathcal{N}(x^{-1}(x(\bar{i})))} [\cdot] = \sum_{\bar{j} \in \mathcal{N}(\bar{i})} [\cdot].$$

Hence, we can further reduce the expression above as:

$$\begin{aligned}
m[\mathcal{L}_y[f], \rho](i) &= \varphi_{\text{out}} \left( \bigcup_{h \in [H]} \sum_{\bar{j} \in \mathcal{N}(\bar{i})} \sigma_{\bar{j}} \left( \langle \varphi_{\text{qry}}^{(h)}(f(\bar{i}) + \rho^P(x(\bar{i}) + y)), \varphi_{\text{key}}^{(h)}(f(\bar{j}) + \rho^P(x(\bar{j}) + y)) \rangle \right) \varphi_{\text{val}}^{(h)}(f(\bar{j})) \right)
\end{aligned}$$

Since,  $\rho(\bar{i}) + y \neq \rho(\bar{i})$  and  $\rho(\bar{j}) + y \neq \rho(\bar{j})$ , we are unable to reduce the expression further towards the form of  $m(f, \rho)(\bar{i})$ . Consequently, we conclude that the absolute positional encoding does not allow for translation equivariance either.  $\square$

**Proof of Claim 9.4.3.** If the self-attention formulation provided in Eq. 9.11 is translation equivariant, then it must hold that  $m^r[\mathcal{L}_y[f], \rho](i) = \mathcal{L}_y[m^r[f, \rho]](i)$ . Consider a translated input signal  $\mathcal{L}_y[f](i) = f(x^{-1}(x(i) - y))$ . Self-attention on  $\mathcal{L}_y[f]$  is given by:

$$\begin{aligned}
m^r[\mathcal{L}_y[f], \rho](i) &= \varphi_{\text{out}} \left( \bigcup_{h \in [H]} \sum_{j \in \mathcal{N}(i)} \sigma_j \left( \langle \varphi_{\text{qry}}^{(h)}(\mathcal{L}_y[f](i)), \varphi_{\text{key}}^{(h)}(\mathcal{L}_y[f](i) + \rho(i, j)) \rangle \right) \varphi_{\text{val}}^{(h)}(\mathcal{L}_y[f](j)) \right) \\
&= \varphi_{\text{out}} \left( \bigcup_{h \in [H]} \sum_{j \in \mathcal{N}(i)} \sigma_j \left( \langle \varphi_{\text{qry}}^{(h)}(f(x^{-1}(x(i) - y))), \varphi_{\text{key}}^{(h)}(f(x^{-1}(x(j) - y) + \rho(i, j))) \rangle \right) \varphi_{\text{val}}^{(h)}(f(x^{-1}(x(j) - y))) \right) \\
&= \varphi_{\text{out}} \left( \bigcup_{h \in [H]} \sum_{x^{-1}(x(\bar{j}) + y) \in \mathcal{N}(x^{-1}(x(\bar{i}) + y))} \sigma_{x^{-1}(x(\bar{j}) + y)} \left( \langle \varphi_{\text{qry}}^{(h)}(f(\bar{i})), \varphi_{\text{key}}^{(h)}(f(\bar{j}) + \rho(x^{-1}(x(\bar{i}) + y), x^{-1}(x(\bar{j}) + y))) \rangle \right) \varphi_{\text{val}}^{(h)}(f(\bar{j})) \right)
\end{aligned}$$

Here, we have used the substitution  $\bar{i} = x^{-1}(x(i) - y) \Rightarrow i = x^{-1}(x(\bar{i}) + y)$  and  $\bar{j} = x^{-1}(x(j) - y) \Rightarrow j = x^{-1}(x(\bar{j}) + y)$ .

$y) \Rightarrow j = x^{-1}(x(\bar{j}) + y)$ . By using the definition of  $\rho(i, j)$  we can further reduce the expression above as:

$$\begin{aligned} &= \varphi_{\text{out}} \left( \bigcup_{h \in [H]} \sum_{x^{-1}(x(\bar{j})+y) \in \mathcal{N}(x^{-1}(x(\bar{i})+y))} \left( \langle \varphi_{\text{qry}}^{(h)}(f(\bar{i})), \varphi_{\text{key}}^{(h)}(f(\bar{j}) + \rho^P(x(\bar{j}) + y - (x(\bar{i}) + y))) \rangle \right) \varphi_{\text{val}}^{(h)}(f(\bar{j})) \right) \\ &= \varphi_{\text{out}} \left( \bigcup_{h \in [H]} \sum_{x^{-1}(x(\bar{j})+y) \in \mathcal{N}(x^{-1}(x(\bar{i})+y))} \left( \langle \varphi_{\text{qry}}^{(h)}(f(\bar{i})), \varphi_{\text{key}}^{(h)}(f(\bar{j}) + \rho^P(x(\bar{j}) - x(\bar{i}))) \rangle \right) \varphi_{\text{val}}^{(h)}(f(\bar{j})) \right) \\ &= \varphi_{\text{out}} \left( \bigcup_{h \in [H]} \sum_{x^{-1}(x(\bar{j})+y) \in \mathcal{N}(x^{-1}(x(\bar{i})+y))} \left( \langle \varphi_{\text{qry}}^{(h)}(f(\bar{i})), \varphi_{\text{key}}^{(h)}(f(\bar{j}) + \rho(\bar{i}, \bar{j})) \rangle \right) \varphi_{\text{val}}^{(h)}(f(\bar{j})) \right) \end{aligned}$$

Since the area of the summation remains equal to any translation  $y \in \mathbb{R}^d$ , we have that:

$$\sum_{x^{-1}(x(\bar{j})+y) \in \mathcal{N}(x^{-1}(x(\bar{i})+y))} [\cdot] = \sum_{x^{-1}(x(\bar{j})) \in \mathcal{N}(x^{-1}(x(\bar{i})))} [\cdot] = \sum_{\bar{j} \in \mathcal{N}(\bar{i})} [\cdot].$$

Resultantly, we can further reduce the expression above as:

$$\begin{aligned} m^r[\mathcal{L}_y[f], \rho](i) &= \varphi_{\text{out}} \left( \bigcup_{h \in [H]} \sum_{\bar{j} \in \mathcal{N}(\bar{i})} \sigma_{\bar{j}} \left( \langle \varphi_{\text{qry}}^{(h)}(f(\bar{i})), \varphi_{\text{key}}^{(h)}(f(\bar{j}) + \rho(\bar{i}, \bar{j})) \rangle \right) \varphi_{\text{val}}^{(h)}(f(\bar{j})) \right) \\ &= m^r[f, \rho](\bar{i}) = m^r[f, \rho](x^{-1}(x(i) - y)) \\ &= \mathcal{L}_y[m^r[f, \rho]](i) \end{aligned}$$

We see that indeed  $m^r[\mathcal{L}_y[f], \rho](i) = \mathcal{L}_y[m^r[f, \rho]](i)$ . Hence, the relative positional encoding allows for translation equivariance. We emphasize that this is a consequence of the fact that  $\rho(x^{-1}(x(\bar{i}) + y), x^{-1}(x(\bar{j}) + y)) = \rho(\bar{i}, \bar{j})$ ,  $\forall y \in \mathbb{R}^d$ . That is, it comes from the fact that relative positional encoding is invariant to the action of the translation group.  $\square$

**Proof of Claim 9.5.1.** If the lifting self-attention formulation provided in Eq. 9.11 is  $\mathcal{G}$ -equivariant, then it must hold that  $m_{\mathcal{G}}^r[\mathcal{L}_g[f], \rho](i, h) = \mathcal{L}_g[m_{\mathcal{G}}^r[f, \rho]](i, h)$ . Consider a  $g$ -transformed input signal  $\mathcal{L}_g[f](i) = \mathcal{L}_y \mathcal{L}_{\tilde{h}}[f](i) = f(x^{-1}(\tilde{h}^{-1}(x(i) - y)))$ ,  $g = (y, \tilde{h})$ ,  $y \in \mathbb{R}^d$ ,  $\tilde{h} \in \mathcal{H}$ . The lifting group self-attention operation on  $\mathcal{L}_g[f]$  is given by:

$$\begin{aligned} &m_{\mathcal{G}}^r[\mathcal{L}_y \mathcal{L}_{\tilde{h}}[f], \rho](i, h) \\ &= \varphi_{\text{out}} \left( \bigcup_{h \in [H]} \sum_{j \in \mathcal{N}(i)} \sigma_j \left( \langle \varphi_{\text{qry}}^{(h)}(\mathcal{L}_y \mathcal{L}_{\tilde{h}}[f](i)), \varphi_{\text{key}}^{(h)}(\mathcal{L}_y \mathcal{L}_{\tilde{h}}[f](i) + \mathcal{L}_h[\rho](i, j)) \rangle \right) \varphi_{\text{val}}^{(h)}(\mathcal{L}_y \mathcal{L}_{\tilde{h}}[f](j)) \right) \\ &= \varphi_{\text{out}} \left( \bigcup_{h \in [H]} \sum_{j \in \mathcal{N}(i)} \sigma_j \left( \langle \varphi_{\text{qry}}^{(h)}(f(x^{-1}(\tilde{h}^{-1}(x(i) - y)))), \varphi_{\text{key}}^{(h)}(f(x^{-1}(\tilde{h}^{-1}(x(j) - y))) + \mathcal{L}_h[\rho](i, j)) \rangle \right) \varphi_{\text{val}}^{(h)}(f(x^{-1}(\tilde{h}^{-1}(x(j) - y)))) \right) \\ &= \varphi_{\text{out}} \left( \bigcup_{h \in [H]} \sum_{x^{-1}(\tilde{h}x(\bar{j})+y) \in \mathcal{N}(x^{-1}(\tilde{h}x(\bar{i})+y))} \left( \langle \varphi_{\text{qry}}^{(h)}(f(\bar{i})), \varphi_{\text{key}}^{(h)}(f(\bar{j}) + \mathcal{L}_h[\rho](x^{-1}(\tilde{h}x(\bar{i})+y), x^{-1}(\tilde{h}x(\bar{j})+y))) \rangle \right) \varphi_{\text{val}}^{(h)}(f(\bar{j})) \right) \end{aligned}$$

Here we have used the substitution  $\bar{i} = x^{-1}(\tilde{h}^{-1}(x(i) - y)) \Rightarrow i = x^{-1}(\tilde{h}x(\bar{i}) + y)$  and  $\bar{j} = x^{-1}(\tilde{h}^{-1}(x(j) - y)) \Rightarrow j = x^{-1}(\tilde{h}x(\bar{j}) + y)$ . By using the definition of  $\rho(i, j)$  we can further reduce the ex-

pression above as:

$$\begin{aligned}
&= \varphi_{\text{out}} \left( \bigcup_{h \in [H]} \sum_{x^{-1}(\tilde{h}x(\bar{j})+y) \in \mathcal{N}(x^{-1}(\tilde{h}x(\bar{i})+y))} \sigma_{x^{-1}(\tilde{h}x(\bar{j})+y)} \left( \langle \varphi_{\text{qry}}^{(h)}(f(\bar{i})), \varphi_{\text{key}}^{(h)}(f(\bar{j})) \rangle \right. \right. \\
&\quad \left. \left. + \rho^P(h^{-1}(\tilde{h}x(\bar{j})+y) - h^{-1}(\tilde{h}x(\bar{i})+y))) \right) \varphi_{\text{val}}^{(h)}(f(\bar{j})) \right) \\
&= \varphi_{\text{out}} \left( \bigcup_{h \in [H]} \sum_{x^{-1}(\tilde{h}x(\bar{j})+y) \in \mathcal{N}(x^{-1}(\tilde{h}x(\bar{i})+y))} \sigma_{x^{-1}(\tilde{h}x(\bar{j})+y)} \left( \langle \varphi_{\text{qry}}^{(h)}(f(\bar{i})), \varphi_{\text{key}}^{(h)}(f(\bar{j})) \rangle \right. \right. \\
&\quad \left. \left. + \rho^P(h^{-1}\tilde{h}(x(\bar{j}) - x(\bar{i}))) \right) \varphi_{\text{val}}^{(h)}(f(\bar{j})) \right) \\
&= \varphi_{\text{out}} \left( \bigcup_{h \in [H]} \sum_{x^{-1}(\tilde{h}x(\bar{j})+y) \in \mathcal{N}(x^{-1}(\tilde{h}x(\bar{i})+y))} \sigma_{x^{-1}(\tilde{h}x(\bar{j})+y)} \left( \langle \varphi_{\text{qry}}^{(h)}(f(\bar{i})), \varphi_{\text{key}}^{(h)}(f(\bar{j}) + \mathcal{L}_{\tilde{h}^{-1}h}[\rho](\bar{i}, \bar{j})) \rangle \right) \varphi_{\text{val}}^{(h)}(f(\bar{j})) \right)
\end{aligned}$$

Since, for unimodular groups, the area of summation remains equal for any  $g \in \mathcal{G}$ , we have that:

$$\sum_{x^{-1}(\tilde{h}x(\bar{j})+y) \in \mathcal{N}(x^{-1}(\tilde{h}x(\bar{i})+y))} [\cdot] = \sum_{x^{-1}(\tilde{h}x(\bar{j})+y) \in \mathcal{N}(x^{-1}(\tilde{h}x(\bar{i})))} [\cdot] = \sum_{x^{-1}(x(\bar{j})+y) \in \mathcal{N}(x^{-1}(x(\bar{i})))} [\cdot] = \sum_{\bar{j} \in \mathcal{N}(\bar{i})} [\cdot].$$

Resultantly, we can further reduce the expression above as:

$$\begin{aligned}
m_{\mathcal{G}}^r[\mathcal{L}_y \mathcal{L}_{\tilde{h}}[f], \rho](i, h) &= \varphi_{\text{out}} \left( \bigcup_{h \in [H]} \sum_{\bar{j} \in \mathcal{N}(\bar{i})} \sigma_{\bar{j}} \left( \langle \varphi_{\text{qry}}^{(h)}(f(\bar{i})), \varphi_{\text{key}}^{(h)}(f(\bar{j}) + \mathcal{L}_{\tilde{h}^{-1}h}[\rho](\bar{i}, \bar{j})) \rangle \right) \varphi_{\text{val}}^{(h)}(f(\bar{j})) \right) \\
&= m_{\mathcal{G}}^r[f, \rho](\bar{i}, \tilde{h}^{-1}h) = m_{\mathcal{G}}^r[f, \rho](x^{-1}(\tilde{h}^{-1}(x(i) - y)), \tilde{h}^{-1}h) \\
&= \mathcal{L}_y \mathcal{L}_{\tilde{h}}[m_{\mathcal{G}}^r[f, \rho]](i, h).
\end{aligned}$$

We see indeed that  $m_{\mathcal{G}}^r[\mathcal{L}_y \mathcal{L}_{\tilde{h}}[f], \rho](i, h) = \mathcal{L}_y \mathcal{L}_{\tilde{h}}[m_{\mathcal{G}}^r[f, \rho]](i, h)$ . Consequently, we conclude that the lifting group self-attention operation is group equivariant. We emphasize once more that this is a consequence of the fact that  $\mathcal{L}_g[\rho](i, j) = \rho(i, j)$ ,  $\forall g \in \mathcal{G}$ . In other words, it comes from the fact that the positional encoding used is invariant to the action of elements  $g \in \mathcal{G}$ .  $\square$

**Proof of Claim 9.5.2.** If the group self-attention formulation provided in Eq. 9.11 is  $\mathcal{G}$ -equivariant, then it must hold that  $m_{\mathcal{G}}^r[\mathcal{L}_g[f], \rho](i, h) = \mathcal{L}_g[m_{\mathcal{G}}^r[f, \rho]](i, h)$ . Consider a  $g$ -transformed input signal  $\mathcal{L}_g[f](i, \tilde{h}) = \mathcal{L}_y \mathcal{L}_{\tilde{h}}[f](i, \tilde{h}) = f(\rho^{-1}(\tilde{h}^{-1}(\rho(i) - y)), \tilde{h}\tilde{h})$ ,  $g = (y, \tilde{h})$ ,  $y \in \mathbb{R}^d$ ,  $\tilde{h} \in \mathcal{H}$ . The group self-attention operation on  $\mathcal{L}_g[f]$  is given by:

$$\begin{aligned}
m_{\mathcal{G}}^r[\mathcal{L}_y \mathcal{L}_{\tilde{h}}[f], \rho](i, h) &= \varphi_{\text{out}} \left( \bigcup_{h \in [H]} \sum_{\tilde{h} \in \mathcal{H}} \sum_{(j, \hat{h}) \in \mathcal{N}(i, \tilde{h})} \sigma_{j, \hat{h}} \left( \langle \varphi_{\text{qry}}^{(h)}(\mathcal{L}_y \mathcal{L}_{\tilde{h}}[f](i, \tilde{h})), \varphi_{\text{key}}^{(h)}(\mathcal{L}_y \mathcal{L}_{\tilde{h}}[f](j, \hat{h})) \rangle \right. \right. \\
&\quad \left. \left. + \mathcal{L}_{\tilde{h}}[\rho](i, \tilde{h}), (j, \hat{h})) \right) \varphi_{\text{val}}^{(h)}(\mathcal{L}_y \mathcal{L}_{\tilde{h}}[f](j, \hat{h})) \right) \\
&= \varphi_{\text{out}} \left( \bigcup_{h \in [H]} \sum_{\tilde{h} \in \mathcal{H}} \sum_{(j, \hat{h}) \in \mathcal{N}(i, \tilde{h})} \sigma_{j, \hat{h}} \left( \langle \varphi_{\text{qry}}^{(h)}(f(x^{-1}(\tilde{h}^{-1}(x(i) - y)), \tilde{h}^{-1}\tilde{h})), \varphi_{\text{key}}^{(h)}(f(x^{-1}(\tilde{h}^{-1}(x(j) - y)), \tilde{h}^{-1}\hat{h})) \rangle \right. \right. \\
&\quad \left. \left. + \mathcal{L}_{\tilde{h}}[\rho](i, \tilde{h}), (j, \hat{h})) \right) \varphi_{\text{val}}^{(h)}(f(x^{-1}(\tilde{h}^{-1}(x(j) - y)), \tilde{h}^{-1}\hat{h})) \right) \\
&= \varphi_{\text{out}} \left( \bigcup_{h \in [H]} \sum_{\tilde{h}\tilde{h}' \in \mathcal{H}} \sum_{(x^{-1}(\tilde{h}x(\bar{j})+y), \tilde{h}\tilde{h}') \in \mathcal{N}(x^{-1}(\tilde{h}x(\bar{i})+y), \tilde{h}\tilde{h}')} \sigma_{x^{-1}(\tilde{h}x(\bar{j})+y), \tilde{h}\tilde{h}'} \left( \langle \varphi_{\text{qry}}^{(h)}(f(\bar{i}, \tilde{h}')), \varphi_{\text{key}}^{(h)}(f(\bar{j}, \hat{h}') + \mathcal{L}_{\tilde{h}}[\rho](x^{-1}(\tilde{h}x(\bar{i})+y), \tilde{h}\tilde{h}')) \rangle \right. \right. \\
&\quad \left. \left. + (x^{-1}(\tilde{h}x(\bar{j})+y), \tilde{h}\hat{h}')) \right) \varphi_{\text{val}}^{(h)}(f(\bar{j}, \hat{h}')) \right)
\end{aligned}$$

Here we have used the substitutions  $\bar{i} = x^{-1}(\tilde{h}^{-1}(x(i) - y)) \Rightarrow i = x^{-1}(\tilde{h}x(\bar{i}) + y)$ ,  $\tilde{h}' = \tilde{h}^{-1}\tilde{h}$ ,

and  $\bar{j} = x^{-1}(\bar{h}^{-1}(x(j) - y)) \Rightarrow i = x^{-1}(\bar{h}x(\bar{i}) + y)$ ,  $\hat{h}' = \bar{h}^{-1}\bar{h}$ . By using the definition of  $\rho((i, \tilde{h}), (j, \hat{h}))$  we can further reduce the expression above as:

$$\begin{aligned} &= \varphi_{\text{out}} \left( \bigcup_{h \in [H]} \sum_{\bar{h}\hat{h}' \in \mathcal{H}} \sum_{(x^{-1}(\bar{h}x(\bar{j}) + y), \bar{h}\hat{h}') \in \mathcal{N}(x^{-1}(\bar{h}x(\bar{i}) + y), \bar{h}\hat{h}')} \left( \langle \varphi_{\text{qry}}^{(h)}(f(\bar{i}, \tilde{h}')), \varphi_{\text{key}}^{(h)}(f(\bar{j}, \hat{h}')) \right. \right. \\ &\quad \left. \left. + \rho^P(\bar{h}^{-1}(\bar{h}x(\bar{j}) + y - (\bar{h}x(\bar{i}) + y)), \bar{h}^{-1}\bar{h}\hat{h}'^{-1}\hat{h}')) \right) \varphi_{\text{val}}^{(h)}(f(\bar{j}, \hat{h}')) \right) \\ &= \varphi_{\text{out}} \left( \bigcup_{h \in [H]} \sum_{\bar{h}\hat{h}' \in \mathcal{H}} \sum_{(x^{-1}(\bar{h}x(\bar{j}) + y), \bar{h}\hat{h}') \in \mathcal{N}(x^{-1}(\bar{h}x(\bar{i}) + y), \bar{h}\hat{h}')} \left( \langle \varphi_{\text{qry}}^{(h)}(f(\bar{i}, \tilde{h}')), \varphi_{\text{key}}^{(h)}(f(\bar{j}, \hat{h}')) \right. \right. \\ &\quad \left. \left. + \rho^P(\bar{h}^{-1}\bar{h}(x(\bar{j}) - x(\bar{i}), \tilde{h}'^{-1}\hat{h}')) \right) \varphi_{\text{val}}^{(h)}(f(\bar{j}, \hat{h}')) \right) \\ &= \varphi_{\text{out}} \left( \bigcup_{h \in [H]} \sum_{\bar{h}\hat{h}' \in \mathcal{H}} \sum_{(x^{-1}(\bar{h}x(\bar{j}) + y), \bar{h}\hat{h}') \in \mathcal{N}(x^{-1}(\bar{h}x(\bar{i}) + y), \bar{h}\hat{h}')} \left( \langle \varphi_{\text{qry}}^{(h)}(f(\bar{i}, \tilde{h}')), \varphi_{\text{key}}^{(h)}(f(\bar{j}, \hat{h}')) \right. \right. \\ &\quad \left. \left. + \mathcal{L}_{\bar{h}^{-1}\bar{h}}[\rho](\bar{i}, \tilde{h}', \bar{j}, \hat{h}')) \right) \varphi_{\text{val}}^{(h)}(f(\bar{j}, \hat{h}')) \right) \end{aligned}$$

Furthermore, since for unimodular groups the area of summation remains equal for any transformation  $g \in \mathcal{G}$ , we have that:

$$\begin{aligned} \sum_{(x^{-1}(\bar{h}x(\bar{j}) + y), \bar{h}\hat{h}') \in \mathcal{N}(x^{-1}(\bar{h}x(\bar{i}) + y), \bar{h}\hat{h}')} [\cdot] &= \sum_{(x^{-1}(\bar{h}x(\bar{j}) + y), \bar{h}\hat{h}') \in \mathcal{N}(x^{-1}(\bar{h}x(\bar{i}) + y), \bar{h}\hat{h}')} [\cdot] \\ &= \sum_{(x^{-1}(x(\bar{j})), \hat{h}') \in \mathcal{N}(x^{-1}(x(\bar{i})), \hat{h}')} [\cdot] = \sum_{(\bar{j}, \hat{h}') \in \mathcal{N}(\bar{i}, \tilde{h}')} [\cdot]. \end{aligned}$$

Additionally, we have that  $\sum_{\bar{h}\hat{h}' \in \mathcal{H}} [\cdot] = \sum_{\tilde{h}' \in \mathcal{H}} [\cdot]$ . Resultantly, we can further reduce the expression above as:

$$\begin{aligned} m_{\mathcal{G}}^r[\mathcal{L}_y \mathcal{L}_{\bar{h}}[f], \rho](i, h) &= \varphi_{\text{out}} \left( \bigcup_{h \in [H]} \sum_{\bar{h}' \in \mathcal{H}} \sum_{(\bar{j}, \hat{h}') \in \mathcal{N}(\bar{i}, \tilde{h}')} \left( \langle \varphi_{\text{qry}}^{(h)}(f(\bar{i}, \tilde{h}')), \varphi_{\text{key}}^{(h)}(f(\bar{j}, \hat{h}')) \right. \right. \\ &\quad \left. \left. + \mathcal{L}_{\bar{h}^{-1}\bar{h}}[\rho](\bar{i}, \tilde{h}', \bar{j}, \hat{h}')) \right) \varphi_{\text{val}}^{(h)}(f(\bar{j}, \hat{h}')) \right) \\ &= m_{\mathcal{G}}^r[f, \rho](\bar{i}, \bar{h}^{-1}\bar{h}) = m_{\mathcal{G}}^r[f, \rho](x^{-1}(\bar{h}^{-1}(x(i) - y)), \bar{h}^{-1}\bar{h}) \\ &= \mathcal{L}_y \mathcal{L}_{\bar{h}}[m_{\mathcal{G}}^r[f, \rho]](i, h). \end{aligned}$$

We see that indeed  $m_{\mathcal{G}}^r[\mathcal{L}_y \mathcal{L}_{\bar{h}}[f], \rho](i, h) = \mathcal{L}_y \mathcal{L}_{\bar{h}}[m_{\mathcal{G}}^r[f, \rho]](i, h)$ . Hence, we can conclude that the group self-attention operation is group equivariant. We emphasize once more that this is a consequence of the fact that  $\mathcal{L}_g[\rho](i, \tilde{h}, j, \hat{h}) = \rho(i, \tilde{h}, j, \hat{h})$ ,  $\forall g \in \mathcal{G}$ . That is, it comes from the fact that the positional encoding used is invariant to the action of elements  $g \in \mathcal{G}$ .  $\square$





# Scale-Translation Equivariant Learning From Raw Time-Series

## I.1 Group and group action

**Group.** A group is an ordered pair  $(\mathcal{G}, \cdot)$  where  $\mathcal{G}$  is a set and  $\cdot : \mathcal{G} \times \mathcal{G} \rightarrow \mathcal{G}$  is a binary operation on  $\mathcal{G}$ , such that (i) the set is closed under this operation, (ii) the operation is associative, i.e.,  $(g_1 \cdot g_2) \cdot g_3 = g_1 \cdot (g_2 \cdot g_3)$ ,  $g_1, g_2, g_3 \in \mathcal{G}$ , (iii) there exists an identity element  $e \in \mathcal{G}$  such that  $\forall g \in \mathcal{G}$  we have  $e \cdot g = g \cdot e = g$ , and (iv) for each  $g \in \mathcal{G}$ , there exists an inverse  $g^{-1}$  such that  $g \cdot g^{-1} = e$ .

**Subgroup.** Given a group  $(\mathcal{G}, \cdot)$ , we say that a subset  $\mathcal{H}$  is a subgroup of  $\mathcal{G}$  if the tuple  $(\mathcal{H}, \cdot)$  also complies to the group axioms. For example, the set of rotations by  $90^\circ$ ,  $\mathcal{H}=\{0^\circ, 90^\circ, 180^\circ, 270^\circ\}$ , is a subgroup of the continuous rotation group as it also complies to the group axioms.

**Group action.** Let  $\mathcal{G}$  be a group and  $\mathcal{X}$  be a set. The (left) group action of  $\mathcal{G}$  on  $\mathcal{X}$  is a function

$$\mathcal{A} : \mathcal{G} \times \mathcal{X} \rightarrow \mathcal{X}, \quad \mathcal{A}_g : x \mapsto x', \tag{I.1}$$

such that for any  $g_1, g_2 \in \mathcal{G}$ ,  $\mathcal{A}_{g_2 g_1} = \mathcal{A}_{g_2} \circ \mathcal{A}_{g_1}$ . In other words, the action of  $\mathcal{G}$  on  $\mathcal{X}$  describes how the elements in the set  $x \in \mathcal{X}$  are transformed by elements  $g \in \mathcal{G}$ . For brevity,  $\mathcal{A}_g(x)$  is written as  $gx$ .

## I.2 Equivariance properties of time-frequency transforms

### I.2.1 The Fourier transform

The Fourier transform represents a function with finite energy  $f \in L^2(\mathbb{R})$  as a sum of complex sinusoidal waves  $e^{i\omega t} = \cos \omega t + i \sin \omega t$ :

$$f(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \hat{f}(\omega) e^{i\omega t} d\omega,$$

where,  $\hat{f}(\omega)$  depicts the amplitude of each component  $e^{i\omega t}$  in  $f$ . The *Fourier transform*  $\mathcal{F}$  is defined as:

$$\mathcal{F}[f](\omega) = \hat{f}(\omega) = \langle f, e^{i\omega t} \rangle = \int_{-\infty}^{\infty} f(t) e^{-i\omega t} dt.$$

In other words, the Fourier transform encodes  $f$  into a time-frequency dictionary  $\mathcal{D} = \{e^{i\omega t}\}_{\omega \in \mathbb{R}}$ .

**Input translation.** Let  $\mathcal{L}_{t_0}[f](t) = f(t - t_0)$  be a translated version of  $f$ . Its Fourier transform is given by:

$$\begin{aligned} \mathcal{F}[\mathcal{L}_{t_0}[f]](\omega) &= \int_{-\infty}^{\infty} f(t - t_0) e^{-i\omega t} dt \Big| \tilde{t} = t - t_0; d\tilde{t} = dt \\ &= \int_{-\infty}^{\infty} f(\tilde{t}) e^{-i\omega(\tilde{t}+t_0)} d\tilde{t} = e^{-i\omega t_0} \int_{-\infty}^{\infty} f(\tilde{t}) e^{-i\omega \tilde{t}} d\tilde{t} = e^{-i\omega t_0} \mathcal{F}[f](\omega) \end{aligned} \quad (\text{I.2})$$

In other words, a translation of  $t_0$  corresponds to a phase modulation of  $e^{-i\omega t_0}$  in the frequency domain.

**Input scaling.** Let  $\mathcal{L}_{s_0}[f](t) = f(s_0^{-1}t)$ ,  $s_0 \in \mathbb{R}_{>0}$ , be a scaled version of  $f$ . Its Fourier transform equals:

$$\begin{aligned} \mathcal{F}[\mathcal{L}_{s_0}[f]](\omega) &= \int_{-\infty}^{\infty} f(s_0^{-1}t) e^{-i\omega t} dt \Big| \tilde{t} = s_0^{-1}t; d\tilde{t} = s_0^{-1}dt \\ &= \int_{-\infty}^{\infty} f(\tilde{t}) e^{-i\omega(s_0\tilde{t})} d(s_0\tilde{t}) = s_0 \int_{-\infty}^{\infty} f(\tilde{t}) e^{-i(s_0\omega)\tilde{t}} d\tilde{t} \\ &= s_0 \mathcal{F}[f](s_0\omega) = s_0 \mathcal{L}_{s_0^{-1}}[\mathcal{F}[f]](\omega) \end{aligned} \quad (\text{I.3})$$

In other words, we observe that a dilation on the time domain produces a compression in the Fourier domain times the inverse of the dilation.

**Simultaneous input translation and scaling.** Following the same derivation procedure, we can show the behavior of the Fourier transform to simultaneous translations and dilations of the input:

$$\mathcal{F}[\mathcal{L}_{s_0} \mathcal{L}_{t_0}[f]](\omega) = s_0 e^{-i\omega t_0} \mathcal{F}[f](s_0\omega) = e^{-i\omega t_0} s_0 \mathcal{L}_{s_0^{-1}}[\mathcal{F}[f]](\omega). \quad (\text{I.4})$$

This corresponds to the superposition of the previously exhibited behaviours.

**Effect of input transformations on the spectral density.** The spectral density of a function  $f \in L^2(\mathbb{R})$  is given by  $|\mathcal{F}[f](\omega)|^2$ . Input translations and dilations produce the

following transformations:

$$|\mathcal{F}[\mathcal{L}_{t_0}[f]](\omega)|^2 = |\mathcal{F}[f](\omega)|^2 \quad (\text{I.5})$$

$$|\mathcal{F}[\mathcal{L}_{s_0}[f]](\omega)|^2 = |s_0|^2 |\mathcal{L}_{s_0^{-1}}[\mathcal{F}[f]](\omega)|^2 \quad (\text{I.6})$$

**Equivariance and invariance properties of the Fourier transform.** From Eq. I.2 we can see that the Fourier transform is translation equivariant as it encodes translations of the input as a phase modulation of the output. In addition, it is also scale equivariant (Eq. I.3), as it encodes dilations of the input as a modulation of the frequency components in the output. We can prove that the Fourier transform is dilation and translation equivariant by showing that the output transformations  $e^{-i\omega t_0}$  and  $s_0 \mathcal{L}_{s_0^{-1}}$  are group representations of the translation and scaling group in the Fourier space.

**Group representation.** Let  $\mathcal{G}$  be a group and  $f$  be a function on a given functional space  $L_V(\mathcal{X})$ . The (left) regular representation of  $\mathcal{G}$  is a linear transformation  $\mathcal{L} : \mathcal{G} \times L_V(\mathcal{X}) \rightarrow L_V(\mathcal{X})$  which extends group actions to functions on  $L_V(\mathcal{X})$  by:

$$\mathcal{L}_g : f \rightarrow f', \quad f'(\mathcal{A}_g(x)) = f(x) \Leftrightarrow f'(x) = f(g^{-1}x),$$

such that for any  $g_1, g_2 \in \mathcal{G}$ ,  $\mathcal{L}_{g_2 g_1} = \mathcal{L}_{g_2} \circ \mathcal{L}_{g_1}$ . In other words, the group representation describes how a function on a functional space  $f \in L_V(\mathcal{X})$  is modified by the effect of group elements  $g \in \mathcal{G}$ .

We can show that the combination of input translations  $t_0, t_1 \in \mathbb{R}$  or dilations  $s_0, s_1 \in \mathbb{R}_{>0}$  produces a transformation on the Fourier domain that preserves the group structure. In other words, that the transformations previously outlined are group representations. Specifically, for  $\mathcal{L}_{t_1}[\mathcal{L}_{t_0}[f]]$  and  $\mathcal{L}_{s_1}[\mathcal{L}_{s_0}[f]]$  it holds:

$$\mathcal{F}[\mathcal{L}_{t_1}[\mathcal{L}_{t_0}[f]]](\omega) = e^{-i\omega t_1} e^{-i\omega t_0} \mathcal{F}[f](\omega) = e^{-i\omega(t_1+t_0)} \mathcal{F}[f](\omega) = \mathcal{L}_{t_1+t_0}^{\text{Fourier}}[\mathcal{F}[f]](\omega)$$

$$\mathcal{F}[\mathcal{L}_{s_1}[\mathcal{L}_{s_0}[f]]](\omega) = s_1 \mathcal{L}_{s_1^{-1}}[s_0 \mathcal{L}_{s_0^{-1}}[\mathcal{F}[f]]](\omega) = (s_0 s_1) \mathcal{F}[f](s_1 s_0 \omega) = \mathcal{L}_{s_1 s_0}^{\text{Fourier}}[\mathcal{F}[f]](\omega)$$

with  $\mathcal{L}_t^{\text{Fourier}}[\mathcal{F}[f]](\omega) = e^{-i\omega t} \mathcal{F}[f](\omega)$  the representation of the Fourier transform for the translation group, and  $\mathcal{L}_s^{\text{Fourier}}[\mathcal{F}[f]](\omega) = s \mathcal{F}[f](s\omega)$  the representation of the Fourier transform for the dilation group.

Unfortunately, the resulting group representations rapidly become cumbersome especially in the presence of several input components. In addition, although the calculation of the spectral density leaves the scale equivariance property of the transformation unaffected, Eq. I.5 shows that it reduces translation equivariance of the Fourier transform to *translation invariance*. This is why the Fourier transform is commonly considered not to carry positional information.

### I.2.2 The short-time Fourier transform

The short-time Fourier transform of a signal  $f \in L^2(\mathbb{R})$  is given by:

$$\mathcal{S}[f](t, \omega) = \int_{-\infty}^{+\infty} f(\tau) w(\tau - t) e^{-i\omega\tau} d\tau.$$

In other words, it encodes the input  $f$  into a time-frequency dictionary  $\mathcal{D} = \{\phi_{t,\omega}\}, \phi_{t,\omega} = w(\tau - t) e^{-i\omega\tau}$ .

**Input translation.** Let  $\mathcal{L}_{t_0}[f](\tau) = f(\tau - t_0)$  be a translated version of  $f$ . Its short-time Fourier transform is given by:

$$\begin{aligned} \mathcal{S}[\mathcal{L}_{t_0}[f]](t, \omega) &= \int_{-\infty}^{\infty} f(\tau - t_0) w(\tau - t) e^{-i\omega\tau} d\tau \Big| \tilde{\tau} = \tau - t_0; d\tilde{\tau} = d\tau \\ &= \int_{-\infty}^{\infty} f(\tilde{\tau}) w(\tilde{\tau} + t_0 - t) e^{-i\tilde{\tau}(t+t_0)} d\tilde{\tau} = e^{-i\tilde{\tau}t_0} \int_{-\infty}^{\infty} f(\tilde{\tau}) w(\tilde{\tau} - (t - t_0)) e^{-i\omega\tilde{\tau}} d\tilde{\tau} \\ &= e^{-i\omega t_0} \mathcal{S}[f](t - t_0, \omega) = e^{-i\omega t_0} \mathcal{L}_{t_0}[\mathcal{S}[f]](t, \omega) \end{aligned} \quad (\text{I.7})$$

In other words, a translation by  $t_0$  in the time domain, corresponds to a shift by  $t_0$  on the time axis of the short-time Fourier transform, and an additional phase modulation of  $e^{-i\tilde{\tau}t_0}$  similar to that of the Fourier transform (Eq. I.2).

**Input scaling.** Let  $\mathcal{L}_{s_0}[f](\tau) = f(s_0^{-1}\tau)$ ,  $s_0 \in \mathbb{R}_{>0}$ , be a scaled version of  $f$ . Its short-time Fourier transform is given by:

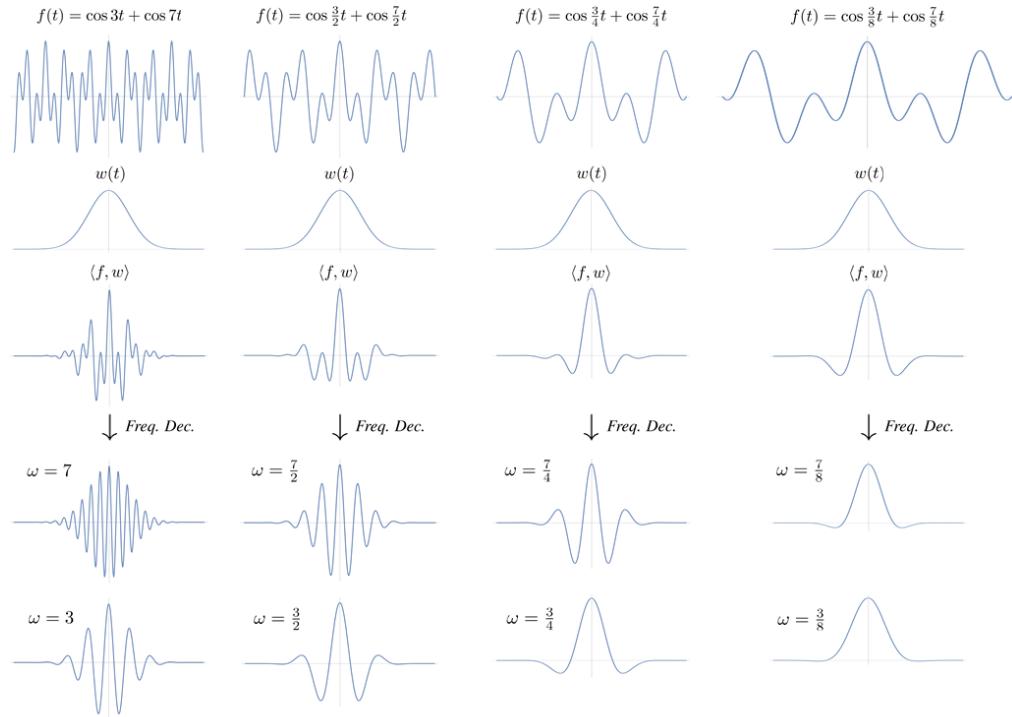
$$\begin{aligned} \mathcal{S}[\mathcal{L}_{s_0}[f]](t, \omega) &= \int_{-\infty}^{\infty} f(s_0^{-1}\tau) w(\tau - t) e^{-i\omega\tau} d\tau \Big| \tilde{\tau} = s_0^{-1}\tau; d\tilde{\tau} = s_0^{-1}d\tau \\ &= \int_{-\infty}^{\infty} f(\tilde{\tau}) w(s_0\tilde{\tau} - t) e^{-i\omega(s_0\tilde{\tau})} d(s_0\tilde{\tau}) = s_0 \int_{-\infty}^{\infty} f(\tilde{\tau}) w(s_0\tilde{\tau} - t) e^{-i(s_0\omega)\tilde{\tau}} d\tilde{\tau} \Big| t = s_0^{-1}s_0\tilde{\tau} \\ &= s_0 \int_{-\infty}^{\infty} f(\tilde{\tau}) w(s_0(\tilde{\tau} - s_0^{-1}t)) e^{-i(s_0\omega)\tilde{\tau}} d\tilde{\tau} \Big| w(s\tau) \approx w(\tau) \\ &\approx s_0 \int_{-\infty}^{\infty} f(\tilde{\tau}) w(\tilde{\tau} - s_0^{-1}t) e^{-i(s_0\omega)\tilde{\tau}} d\tilde{\tau} \approx s_0 \mathcal{S}[f](s_0^{-1}t, s_0\omega) \end{aligned} \quad (\text{I.8})$$

In other words, a dilation in the time domain produces a compression in the frequency domain analogous to the Fourier transform (Eq. I.3). However, it is important to note that we rely on the approximate  $w(x) \approx w(sx)$  to arrive to the final expression. Nevertheless, it is important to note that this approximate *does not generally hold in practice*. This approximation implies that the window function  $w$  is invariant to scaling, which holds only for increasing window sizes, i.e., when the short-term Fourier transform starts to approximate the (global) Fourier transform.

**Simultaneous input translation and scaling.** Following the same derivation procedure, we can show the behavior of the short-time Fourier transform to simultaneous translations and scaling. We have that:

$$\mathcal{S}[\mathcal{L}_{s_0}\mathcal{L}_{t_0}[f]](t, \omega) = s_0 e^{-i\omega t_0} \mathcal{S}[f](s_0^{-1}(t - t_0), s_0\omega) \quad (\text{I.9})$$

**Effect of input transformations on the spectrogram.** The spectrogram of a function  $f \in L^2(\mathbb{R})$  is given by  $|\mathcal{S}[f](t, \omega)|^2$ . Input translations and dilations produce the following



**Figure I.1:** Scale equivariance of the short-time Fourier transform. Consider a function  $f(t) = \cos \omega_1 t + \cos \omega_2 t$  composed of two frequencies  $\omega_1=3$  and  $\omega_2=7$ , and a window function  $w(t)$ , with which the short-time Fourier transform is calculated. For relatively high frequencies (left column), the dot-product of  $f$  and  $w$ ,  $\langle f, w \rangle$ , is able to capture sufficient spectral information from  $f$  to correctly extract the frequencies  $\omega_1, \omega_2$  from it. However, for dilated versions of the same signal  $f$  (right columns) obtained by reducing the frequency of the spectral components  $\omega_1, \omega_2$  of  $f$ , the capacity of the dot-product  $\langle f, w \rangle$  to capture the spectral information in the input gradually degrades and, eventually, is entirely lost. Consequently, scale equivariance holds (approximately) for scales for which *all* of the spectral components of the signal  $f$  lie within the range of the window  $w$ .

transformations:

$$|\mathcal{F}[\mathcal{L}_{t_0}[f]](t, \omega)|^2 = |\mathcal{L}_{t_0}[\mathcal{S}[f]](t, \omega)|^2 \quad (\text{I.10})$$

$$|\mathcal{F}[\mathcal{L}_{s_0}[f]](t, \omega)|^2 = |s_0|^2 |\mathcal{S}[f](s_0^{-1}t, s_0\omega)|^2 \quad (\text{I.11})$$

**Equivariance and invariance properties of the short-time Fourier transform.** The short-time Fourier transform is *approximately* translation and scale equivariance in a manner similar to that of the Fourier transform. In contrast to the Fourier transform, however, it decomposes input translations into a translation  $t - t_0$  and a phase shift  $e^{-i\omega t_0}$  in the output (Eq. I.7). This decomposition can be interpreted as a rough estimate  $t - t_0$  signifying the position in which the window  $w$  is localized, and a fine grained localization within that window given by the phase shift  $e^{-i\omega t_0}$  indicating the relative position of the pattern within the window  $\mathcal{L}_{(t-t_0)}[w](\tau)$ .

Equivariance to dilations is analogous to the Fourier transform up to the fact that time and frequency are now jointly described. However, since the window itself does not scale with the sampled frequency –as is the case in wavelet transforms–, exact equivariance is not obtained. Note that equivariance to dilations is only approximate, and is restricted to the set of scales that can be detected with the width of the window used (see Fig. I.1 for a visual explanation). Since this is not generally the case, the short-time Fourier transform is *not scale equivariant*.

The calculation of the spectrogram leaves the scale equivariance property of the transformation unaffected and is equivalent in a join manner to the scale equivariance property of the Fourier transform (Eq. I.8). Differently however, Eq. I.10 shows that translation equivariance is partially preserved and only information about the phase shift within the window is lost. This is why the short-time Fourier transform is said to carry positional information, i.e., to be (approximately) translation equivariant.

### I.2.3 The Wavelet Transform

The wavelet transform of a signal  $f \in L^2(\mathbb{R})$  is given by:

$$\mathcal{W}[f](t, s) = \langle f, \psi_{t,s} \rangle = \int_{-\infty}^{+\infty} f(\tau) \frac{1}{\sqrt{s}} \psi^*\left(\frac{\tau-t}{s}\right) d\tau,$$

and is equivalent to encoding  $f$  into a time-frequency dictionary  $\mathcal{D} = \{\psi_{t,s}\}_{t \in \mathbb{R}, s \in \mathbb{R}_{>0}}$ ,  $\psi_{t,s}(\tau) = \frac{1}{\sqrt{s}} \psi^*\left(\frac{\tau-t}{s}\right)$ .

**Input translation.** Let  $\mathcal{L}_{t_0}[f](\tau) = f(\tau - t_0)$  be a translated version of  $f$ . Its wavelet transform is given by:

$$\begin{aligned} \mathcal{W}[\mathcal{L}_{t_0}[f]](t, s) &= \int_{-\infty}^{\infty} f(\tau - t_0) \sqrt{s}^{-1} \psi^*(s^{-1}(\tau - t)) d\tau \Big| \tilde{\tau} = \tau - t_0; d\tilde{\tau} = d\tau \\ &= \int_{-\infty}^{\infty} f(\tilde{\tau}) \sqrt{s}^{-1} \psi^*(s^{-1}(\tilde{\tau} + t_0 - t)) d\tilde{\tau} = \int_{-\infty}^{\infty} f(\tilde{\tau}) \sqrt{s}^{-1} \psi^*(s^{-1}(\tilde{\tau} - (t - t_0))) d\tilde{\tau} \\ &= \mathcal{W}[f](t - t_0, s) = \mathcal{L}_{t_0} \mathcal{W}[f](t, s) \end{aligned} \quad (\text{I.12})$$

In other words, a translation of the input produces an equivalent translation in the wavelet domain.

**Input scaling.** Let  $\mathcal{L}_{s_0}[f](t) = f(s_0^{-1}t)$  be a scaled version of  $f$ . The corresponding wavelet transform is:

$$\begin{aligned}\mathcal{W}[\mathcal{L}_{s_0}[f]](t, s) &= \int_{-\infty}^{\infty} f(s_0^{-1}t) \sqrt{s}^{-1} \psi^*(s^{-1}(\tau - t)) d\tau \Big| \tilde{t} = s_0^{-1}\tau; d\tilde{t} = s_0^{-1}d\tau \\ &= \int_{-\infty}^{\infty} f(\tilde{t}) \sqrt{s}^{-1} \psi^*(s^{-1}(s_0\tilde{t} - t)) d(s_0\tilde{t}) \Big| t = s_0^{-1}s_0\tilde{t} \\ &= \int_{-\infty}^{\infty} f(\tilde{t}) \sqrt{s}^{-1} s_0 \psi^*(s^{-1}s_0(\tilde{t} - s_0^{-1}t)) d\tilde{t} \Big| s_0 = \sqrt{s_0^{-1}s_0^{-1}}^{-1} \\ &= \sqrt{s_0} \int_{-\infty}^{\infty} f(\tilde{t}) \sqrt{s_0^{-1}s^{-1}}^{-1} \psi^*\left((s_0^{-1}s)^{-1}(\tilde{t} - s_0^{-1}t)\right) d\tilde{t} \\ &= \sqrt{s_0} \mathcal{W}[f](s_0^{-1}t, s_0^{-1}s) = \sqrt{s_0} \mathcal{L}_{s_0} \mathcal{W}[f](t, s)\end{aligned}\quad (\text{I.13})$$

In other words, a dilation  $s_0$  in the input domain produces an *equivalent* dilation in the wavelet domain on both components  $(t, s)$ , multiplied by a factor  $\sqrt{s_0}$ . That is, the wavelet transform is translation equivariant.

**Simultaneous input translation and scaling.** Following the same procedure, we can show the behavior of the wavelet transform to simultaneous translations and dilations of the input:

$$\mathcal{W}[f(s_0^{-1}(\tau - t_0))](t, s) = \sqrt{s_0} \mathcal{W}[f](s_0^{-1}(t - t_0), s_0^{-1}s) = \sqrt{s_0} \mathcal{L}_{t_0} \mathcal{L}_{s_0} \mathcal{W}[f](t, s) \quad (\text{I.14})$$

We observe that the Wavelet transform is the only time-frequency transform that respects equivariance with equivalent group representations in the input and output.

**Effect of input transformations on the scalogram.** The scalogram of a function  $f \in L^2(\mathbb{R})$  is given by  $|\mathcal{W}[f](u, s)|^2$ . Input translations and dilations produce the following transformations on the scalogram:

$$|\mathcal{W}[\mathcal{L}_{t_0}[f]](u, s)|^2 = |\mathcal{L}_{t_0}[\mathcal{W}[f]](u, s)|^2 \quad (\text{I.15})$$

$$|\mathcal{W}[\mathcal{L}_{s_0}[f]](u, s)|^2 = |\mathcal{L}_{s_0}[\mathcal{W}[f]](u, s)|^2 \quad (\text{I.16})$$

In other words the scalogram is exactly equivariant to both translations and dilations.

**Equivariance and invariance properties of the wavelet transform.** From Eq. I.12, we can see that the wavelet transform is *exactly equivariant to translations* and the group representation of the output space equals that of the input space. Furthermore, translation equivariance is preserved in the scalogram as well (Eq. I.15). Similarly, scale equivariance is preserved on the wavelet transform up to a multiplicative factor (Eq. I.13). However, the scalogram preserves both translation and dilation equivariance exactly (Eq. I.16).

We emphasize that the group representation on the output space resembles that of the input space. This behavior leads to much more straightforward group representations than that exhibited by the Fourier transform and the short-time Fourier transform. Ad-

ditionally, exact scale equivariance is only obtained on the scalogram (Eq. I.16), whilst for the wavelet transform it is retained up to multiplicative factor (Eq. I.13). This behavior elucidates the fact that time-frequency transforms have been optimized for energy density representations rather than for the time-frequency representations themselves.

### I.3 Experimental details

Whenever possible, we use existing code for the baselines of our wavelet networks as a starting point for the general infrastructure of our model. Specifically, we utilize the PyTorch implementation provided in <https://github.com/philipperemy/very-deep-convnets> and <https://github.com/kyungyunlee/sampleCNN-pytorch> as baseline for the US8K experiments and the MTAT experiments Lee et al. [211], respectively. By doing so, we aim to preserve the reproducibility of the experiments in the baseline papers during our own experiments, as some important training factors are not specified in the baseline papers, e.g., the learning rate used in Dai et al. [75]. Unfortunately, Abdoli et al. [2] do not provide code and we were forced to interpret some of the ambiguities in the paper, e.g., the pooling type utilized in the pooling layers and the loss metric used.

Any omitted parameters can safely be considered to be the default values in PyTorch 1.5.0. Our experiments are carried out in a Nvidia TITAN RTX GPU.

#### I.3.1 UrbanSound8K

**Wn-Nets.** We use a sampling rate of 22.05kHz as opposed to the 8kHz used in [75]. An early study that indicated that some classes were indistinguishable for the human ear at this sampling rate.<sup>1</sup> We zero-pad signals shorter than 4 seconds so that all input signals have a constant length of 80200 samples. Following the implementation of Dai et al. [75], we utilize the Adam optimizer [185] with `lr=1e-2` and `weight_decay=1e-4`, and perform training on the official first 9 folds and test on the 10<sup>th</sup> fold. We noticed that reducing the learning rate from  $1e-2$  to  $1e-3$  increased the performance of our W-Nets. The reported results of the W-Net variants are obtained with this learning rate.

We utilize batches of size 16 and perform training for 400 epochs. The learning rate is reduced by half after 20 epochs of no improvement in validation loss. The Wn-nets used are specified in Table I.1. See Dai et al. [75, Tab. 1] for comparison.

**W-1DCNN.** Following Abdoli et al. [2], we utilize a sampling rate of 16kHz during our experiments. We zero-pad signals shorter than 4 seconds so that all input signals have a constant length of 64000 samples. Following the experimental description of the paper, we utilize the AdaDelta optimizer [460] with `lr=1.0` and perform training in a 10-fold cross validation setting as described in Sec. 10.6. We use batches of size 100 and perform training for 100 epochs. We utilize the 50999-1DCNN variant of Abdoli et al. [2], as it is

---

<sup>1</sup>See [link removed for the sake of the double-blind review process].

**Table I.1:**  $Wn$ -networks. W3-Net (0.219M) denotes a 3-layer network with 0.219M parameters.  $[79/4, 150, 3]$  denotes a group convolutional layer with a nominal kernel size of 79 samples, 150 filters and 3 scales, with a stride of 4. Stride is omitted for stride 1 (e.g.,  $[3, 150, 3]$  has stride 1). Each convolutional layer uses batch normalization right after the convolution, after which ReLU is applied. Following the findings of Romero et al. [318, Appx. C] on the influence of stride in the equivariance of the network, we replace strided convolutions by normal convolutions, followed by spatial pooling.  $[\dots] \times k$  denotes  $k$  stacked layers and double layers in brackets denote residual blocks as defined in [75, Fig. 1b]. In each of the levels of convolutional layers and residual blocks, the first convolution of the first block has scale 3 and the remaining convolutional layers at that level has scale 1.

| W3-NET<br>(0.219M)                     | W5-NET<br>(0.558M)     | W11-NET<br>(1.806M)                                       | W18-NET<br>(3.759M)                                       | W34-NET<br>(4.021M)                                     |
|--|------------------------|---|---|---|
| INPUT: 80200x 1 TIME-DOMAIN WAVEFORM   |                        |   |   |   |
| <i>Lifting Layer ( 9 scales)</i>       |                        |   |   |   |
| $[79/4, 150]$                          | $[79/4, 74]$           | $[79/4, 51]$  | $[79/4, 57]$  | $[79/4, 45]$  |
| MAXPOOL: 4x1 (OUTPUT: 80200x 9x 1)     |                        |   |   |   |
| $[3, 150, 3]$                          | $[3, 74, 3]$           | $[3, 51, 3] \times 2$                                     | $[3, 57, 3] \times 4$                                     | $\begin{bmatrix} 3, 45 \\ 3, 45 \end{bmatrix} \times 3$ |
| MAXPOOL: 4x1 (OUTPUT: 80200x 7x 1)     |                        |   |   |   |
| $[3, 148, 3]$                          | $[3, 102, 3] \times 2$ | $[3, 114, 3] \times 4$                                    | $\begin{bmatrix} 3, 90 \\ 3, 90 \end{bmatrix} \times 4$   |   |
| MAXPOOL: 4x1 (OUTPUT: 80200x 5x 1)     |                        |   |   |   |
| $[3, 296, 3]$                          | $[3, 204, 3] \times 3$ | $[3, 228, 3] \times 4$                                    | $\begin{bmatrix} 3, 180 \\ 3, 180 \end{bmatrix} \times 6$ |   |
| MAXPOOL: 4x1 (OUTPUT: 80200x 3x 1)     |                        |   |   |   |
| $[3, 408, 3] \times 2$                 | $[3, 456, 3] \times 4$ | $\begin{bmatrix} 3, 360 \\ 3, 360 \end{bmatrix} \times 3$ |   |   |
| GLOBAL AVERAGE POOLING (OUTPUT: 1 x N) |                        |   |   |   |
| SOFTMAX [110] (OUTPUT: 1 x N)          |                        |   |   |   |

the variant that requires the less human engineering.<sup>2</sup>

Unfortunately, we were not able to replicate the results reported in [2] ( $83\pm1.3\%$ ) in our experiments. Our replication of [2] lead to a 10-cross fold accuracy of  $62\pm1.3\%$ , which is 21 accuracy points less relative to the results reported. We experiment with our interpretation of the mean squared logarithmic error (MSLE) loss defined in [2, Eq. 4]. However, we find that the conventional cross-entropy loss leads to better results. Consequently, all our reported results are based on training with this loss.<sup>3</sup>

The description of the Wavelet 50999-1DCNN [2] is provided in Table I.2 (see [2, Tab. 1] for comparison).

### I.3.2 MagnaTagATune

**W3<sup>9</sup>-Network.** For the experiments in the MTAT dataset, we utilize the PyTorch code provided by Lee et al. [211]. We use the data and tag preprocessing used in [211]. We utilize the SGD optimizer with `lr=1e-2, weight_decay=1e-6` and `nesterov=True`. We use batches of size 23 and perform training for 100 epochs. The learning rate is reduced by 5 after 3 epochs of no improvement in the validation loss. Early stopping is used if the learning rate drops under  $1e-7$ .

We were unable to replicate the per-class AUC results reported in [211]. Our experiments indicated a per-class AUC of 0.893 instead of the 0.905 reported in [211]. Details of the W3<sup>9</sup>-Net used are given in Table I.3 (see [211, Tab. 1] for comparison).

---

<sup>2</sup>The remaining architectures partition the input signal into overlapping windows after which the predictions of each windows are summarized via a voting mechanism. Consequently, one could argue that the 50999-1DCNN is the only variant that truly receives the raw waveform signal. Nevertheless it is not clear from the paper how the input signal of 64000 samples is reduced to 50999 samples, which is the input dimension of the raw signal for this architecture type.

<sup>3</sup>The MSLE loss in Abdoli et al. [2, Eq. 4] is defined as  $\frac{1}{N} \sum_{i=1}^N \log \frac{p_i+1}{a_i+1}^2$ , where  $p_i$ ,  $a_i$  and  $N$  are the predicted class, the actual class, and the number of samples respectively. Note, however, that obtaining the predicted class  $p_i$ , i.e.,  $p_i = \text{argmax}_o f(x_i)$ , where  $f(x_i) \in \mathbb{R}^O$  is the output of the network for a classification problem with  $O$  classes and input  $x_i$ , is a non-differentiable function. Consequently, it is not possible to train the network based on the formulation provided there. In order to train our model with this loss, we re-formulate the MSLE loss as  $\frac{1}{N} \sum_{i=1}^N \sum_{o=1}^O \log \frac{p_{i,o}+1}{a_{i,o}+1}^2$ , where  $\{a_{i,o}\}_{o=1}^O$  is a one-hot encoded version of the label  $a_i$ . That is, we measure the difference between the one-hot encoded label and the output.

**Table I.2:** Wavelet network variant of the 50999-1DCNN [2].  $[31/2, 24, 3]$  denotes a group convolutional layer with a nominal kernel size of 31 samples, 24 filters and 3 scales, with a stride of 2. FC:  $[96, 48]$  denotes a fully-connected layer with 96 input channels and 48 output channels. Each convolutional layer uses batch normalization right after the convolution followed by ReLU. All fully connected layers expect for the last one use dropout of 0.25 and ReLU. Following the findings of Romero et al. [318, Appx. C] on the influence of stride in the equivariance of the network, we replace strided convolutions with normal convolutions followed by spatial pooling. We note that the input size of our network is (presumably) different from that in [2]. Consequently, the last pooling layer utilizes a region of 5, in contrast to 4 as used in [2]. However, as it is not clear how the input dimension is reduced from 64000 to 50999 in [2] and we stick to their original sampling procedure. We interpret their poling layers as max-pooling ones.

|   |
|---|
| W-1DCNN (0.549M)                        |
| INPUT: 64000x 1                         |
| <i>Lifting Layer ( 9 scales)</i>        |
| $[63/2, 12]$                            |
| MAXPOOL: 8x1                            |
| $[31/2, 24, 3]$                         |
| MAXPOOL: 8x1                            |
| $[15/2, 48, 3]$                         |
| $[7/2, 96, 3]$                          |
| $[3/2, 408, 3]$                         |
| MAXPOOL: 5x1                            |
| FLATTEN $196 \times 6 \rightarrow 1152$ |
| FC: $[1152, 96]$                        |
| FC: $[96, 48]$                          |
| FC: $[48, 10]$                          |
| SOFTMAX                                 |

**Table I.3:** W3<sup>9</sup>-network. [3/1, 90, 3] denotes a group convolutional layer with a nominal kernel size of 3 samples, 90 filters and 3 scales, with a stride of 1. MP:3x1 denotes a max-pooling layer of size 3. FC: [360, 50] denotes a fully-connected layer with 360 input channels and 50 output channels. Each convolutional layer uses batch normalization after the convolution followed by ReLU. Dropout of 0.5 is used after the 6<sup>th</sup> and 11<sup>th</sup> layer. Following the findings of Romero et al. [318, Appx. C] on the influence of stride in the network equivariances, we replace strided convolutions by normal convolutions followed by spatial pooling.

| W-3 <sup>9</sup> NET (2.404M)    |
|----------------------------------|
| INPUT: 59049x 1                  |
| <i>Lifting Layer ( 9 scales)</i> |
| [3/3, 90]                        |
| [3/1, 90, 3], MP:3x1             |
| [3/1, 90, 1], MP:3x1             |
| [3/1, 180, 1], MP:3x1            |
| [3/1, 180, 3], MP:3x1            |
| [3/1, 180, 1], MP:3x1            |
| [3/1, 180, 1], MP:3x1            |
| [3/1, 180, 3], MP:3x1            |
| [3/1, 180, 1], MP:3x1            |
| [3/1, 360, 1], MP:3x1            |
| [3/1, 360, 3]                    |
| FC: [360, 50]                    |
| SIGMOID                          |

# J

## Learning Equivariances and Partial Equivariances from Data

### J.1 Groups, subgroups, group actions and other group theoretical concepts

**Groups.** Group theory is the mathematical language that describes symmetries. The core mathematical object is that of a *group*, and defines what it means for something to exhibit symmetries. Specifically, a group is a tuple  $(\mathcal{G}, \cdot)$  consisting of a set of transformations  $\mathcal{G}$ , and a binary operation  $\cdot$ , that exhibit the following properties: (i) closure, i.e.,  $g_1 \cdot g_2 = g_3 \in \mathcal{G}, \forall g_1, g_2 \in \mathcal{G}$  (ii) associativity, i.e.,  $g_1 \cdot (g_2 \cdot g_3) = (g_1 \cdot g_2) \cdot g_3$  for all  $g_1, g_2, g_3 \in \mathcal{G}$ , (iii) the existence of an identity element  $e \in \mathcal{G}$ , such that  $g \cdot e = e \cdot g = g$ , and (iv) the existence of an inverse  $g^{-1} \in \mathcal{G}$  for all  $g \in \mathcal{G}$ .

**Subgroups.** Given a group  $(\mathcal{G}, \cdot)$ , we say that a subset  $\mathcal{H}$  of the group  $\mathcal{G}$ , is a *subgroup* of  $\mathcal{G}$  if this subset also complies to the group axioms under the binary operation  $\cdot$ . For instance, the set of rotations by  $90^\circ$ ,  $\mathcal{H}=\{0^\circ, 90^\circ, 180^\circ, 270^\circ\}$ , is a subgroup of the rotation group  $\text{SO}(2)$ , because it also complies to the closure, associativity, identity and inverse group axioms.

**Group action.** One can define the *action of the group  $\mathcal{G}$*  on a set  $\mathcal{X}$ . This action describes how group elements  $g \in \mathcal{G}$  modify the set  $\mathcal{X}$  when the transformation is applied. For instance, the action of elements in the group of planar rotations  $\theta \in \text{SO}(2)$  on an image  $x \in \mathcal{X}$  –written  $\theta x$ –, depicts how the image  $x$  changes when the rotation  $\theta$  is applied.

**Lie groups.** A group whose elements form a smooth manifold is referred to as a *Lie*

*group.* Since  $\mathcal{G}$  is not necessarily a vector space, we cannot add or subtract group elements –the only operation defined on the group is the binary operation  $\cdot -$ . However, if the group is a Lie group, one can link the group  $\mathcal{G}$  to a vector space –tangent space at the identity  $T_e(\mathcal{G})-$ , called the *Lie algebra*. Consequently, one can readily expand group elements on the Lie algebra using a basis  $A = \sum_k a^k e_k$  and use these components for calculations. As neural networks work on vector spaces –by means of sums and products–, it is desirable to define convolutional kernels on the Lie algebra as  $\psi = \text{MLP} : \mathfrak{g} \rightarrow \mathbb{R}^{N_{\text{in}} \times N_{\text{out}}}$ , where  $N_{\text{in}}$  and  $N_{\text{out}}$  depict the input and output channels of a convolutional kernel, respectively [105].

**Relevant groups for computer vision applications.** In this work, we consider computer vision applications and thus, are mainly interested in groups that have direct effect on these applications. These groups compose the translation group  $T(2)$ , the rotation group  $SO(2)$ , the group of rotations and reflections  $O(2)$  and combinations thereof.<sup>1</sup> The actions of these groups can intuitively be understood as the translation, the rotation, and the rotation and reflection of 2D functions, respectively.

These groups can be combined by means of the *semi-direct product* ( $\rtimes$ ) to construct groups that represent combined symmetries. For instance, the 2D roto-translation group  $SE(2) = T(2) \rtimes SO(2)$  encompasses symmetries described by both translations and rotations on 2D. Similarly, we can construct a group that describes 2D symmetries given by rotations, translations and reflections  $E(2) = T(2) \rtimes O(2)$ .<sup>2</sup> Considering equivariance to these groups allows us to construct neural networks that respect the combined symmetries described by them.

## J.2 (Approximate) equivariance in partial group convolutions

### J.2.1 Partial group convolutions from the group $\mathcal{G}$ to a subset $\mathcal{S}$

The partial group convolution from signals on  $\mathcal{G}$  to signals on a subset  $\mathcal{S}$  can be interpreted as a group convolution for which the output signal outside of  $\mathcal{S}$  is set to zero. Consequently, we can calculate the equivariance difference  $\Delta_{\text{equiv}}$  in the feature representation, by calculating the difference on the subset  $\mathcal{S}$  of a group convolution with a group-transformed input ( $\mathcal{L}_w f * \psi$ ) and a group convolution with a canonical input proceeded by the same transformation on  $\mathcal{S}$ , i.e.,  $\mathcal{L}_w(f * \psi)$ .

The equivariance difference  $\Delta_{\text{equiv}}^{\text{out}}$  resulting from the effect of considering a subset  $\mathcal{S}$  in the output domain of the operation is given by:

---

<sup>1</sup>The names  $SO(2)$ ,  $O(2)$  are derived from their formal names: Special Orthogonal and Orthogonal group.

<sup>2</sup>The names  $SE(2)$ ,  $E(2)$  are derived from their formal names: Special Euclidean and Euclidean group.

$$\begin{aligned}
\Delta_{\text{equiv}}^{\text{out}} &= \left\| \int_{\mathcal{S}} \mathcal{L}_w(\psi * f)(u) d\mu_{\mathcal{G}}(u) - \int_{\mathcal{S}} (\psi * \mathcal{L}_w f)(u) d\mu_{\mathcal{G}}(u) \right\|_2^2 \\
&= \left\| \int_{w^{-1}\mathcal{S}} (\psi * f)(w^{-1}u) d\mu_{\mathcal{G}}(u) - \int_{\mathcal{S}} (\psi * f)(w^{-1}u) d\mu_{\mathcal{G}}(u) \right\|_2^2 \\
&= \left\| \int_{\mathcal{S}} (\psi * f)(u) d\mu_{\mathcal{G}}(u) - \int_{w\mathcal{S}} (\psi * f)(u) d\mu_{\mathcal{G}}(u) \right\|_2^2 \\
&= \left\| \int_{s_{\min}}^{s_{\max}} (\psi * f)(u) d\mu_{\mathcal{G}}(u) - \int_{ws_{\min}}^{ws_{\max}} (\psi * f)(u) d\mu_{\mathcal{G}}(u) \right\|_2^2 \\
&= \left\| \left( \int_{ws_{\max}}^{s_{\max}} (\psi * f)(u) d\mu_{\mathcal{G}}(u) + \int_{s_{\min}}^{ws_{\max}} (\psi * f)(u) d\mu_{\mathcal{G}}(u) \right) - \right. \\
&\quad \left. \left( \int_{s_{\min}}^{ws_{\max}} (\psi * f)(u) d\mu_{\mathcal{G}}(u) + \int_{ws_{\min}}^{s_{\max}} (\psi * f)(u) d\mu_{\mathcal{G}}(u) \right) \right\|_2^2 \\
&= \left\| \int_{ws_{\max}}^{s_{\max}} (\psi * f)(u) d\mu_{\mathcal{G}}(u) - \int_{ws_{\min}}^{s_{\min}} (\psi * f)(u) d\mu_{\mathcal{G}}(u) \right\|_2^2
\end{aligned}$$

From the first line to the second we take advantage of the equivariance property of the group convolution:  $(\mathcal{L}_w f * \psi)(u) = \mathcal{L}_w(f * \psi)(u)$ , and account for the fact that only the region within  $\mathcal{S}$  is visible at the output. We use the change of variables  $u = w^{-1}u$  from the second to third line, and specify the boundaries of  $\mathcal{S}$ ,  $(s_{\max}, s_{\min})$  from the third to the fourth line. In the fifth line we separate the integration over  $\mathcal{S}$  as a sum of two integrals which depict the same range. In the last line, we cancel out the overlapping parts of the two integrals to come to the final result.

In conclusion, the equivariance difference induced by a subset  $\mathcal{S}^{(2)}$  on the domain of the output  $\Delta_{\text{equiv}}^{\text{out}}$  is given by the difference between the part of the representation that leaves the subset  $\mathcal{S}$ , and the part that comes to replace it instead. This behaviour is illustrated in Figure 11.1.

## J.2.2 Partial group convolutions from a subset $\mathcal{S}^{(1)}$ to a subset $\mathcal{S}^{(2)}$

To isolate the effect of having a group subset as domain of the input signal  $f$ , we first consider the domain of the output to be the group, i.e.,  $\mathcal{S}^{(2)} = \mathcal{G}$ . The equivariance difference in this case is given by the difference across the entire output representation of the group convolution calculated on an input subset  $\mathcal{S}^{(1)}$  with a canonical input  $f$ , and with a group transformed input  $\mathcal{L}_w f$ .

The equivariance difference  $\Delta_{\text{equiv}}^{\text{in}}$  resulting from the effect of considering a subset  $\mathcal{S}^{(1)}$

in the input domain of the operation is given by:

$$\begin{aligned}\Delta_{\text{equiv}}^{\text{in}} &= \left\| \int_G \int_S \psi(v^{-1}u) f(v) d\mu_G(v) d\mu_G(u) - \int_G \int_S \psi(v^{-1}u) f(w^{-1}v) d\mu_G(v) d\mu_G(u) \right\|_2^2 \\ &= \left\| \int_G \left[ \int_S \psi(v^{-1}u) f(v) d\mu_G(v) - \int_S \psi(v^{-1}u) f(w^{-1}v) d\mu_G(v) \right] d\mu_G(u) \right\|_2^2 \\ &= \left\| \int_G \int_S \psi(v^{-1}u) [f(v) - f(w^{-1}v)] d\mu_G(v) d\mu_G(u) \right\|_2^2\end{aligned}$$

In other words, the equivariance difference induced by a subset  $S^{(1)}$  on the domain of the input  $\Delta_{\text{equiv}}^{\text{in}}$  is given by the difference in  $S^{(1)}$  between the input  $f$ , and the part that comes to replace it when the input is modified by a group transformation  $w$ . This behavior is illustrated in Figure 11.2.

### J.3 Equivariance property of Monte-Carlo approximations

Consider the Monte-Carlo approximation shown in the main paper:

$$(\psi * f)(u_i) = \sum_j \psi(v_j^{-1} u_i) f(v_j) \bar{\mu}_G(v_j).$$

For a transformed version of the  $\mathcal{L}_w f$ , we can show that the Monte-Carlo approximation of the group convolution is equivariant in expectation. The proof follows the same steps than Finzi et al. [105] except that the last step of the proof follows a different reason resulting from the fact that input and output elements can be sampled from different probability distributions.

For a transformed version of the  $\mathcal{L}_w f$ , we have that:

$$\begin{aligned}(\psi \hat{*} \mathcal{L}_w f)(u_i) &= \sum_j \psi(v_j^{-1} u_i) f(w^{-1} v_j) \bar{\mu}_G(v_j) \\ &= \sum_j \psi(\tilde{v}_j^{-1} w^{-1} u_i) f(\tilde{v}_j) \bar{\mu}_G(\tilde{v}_j) \\ &\stackrel{d}{=} (\psi \hat{*} f)(w^{-1} u_i) = \mathcal{L}_w(\psi \hat{*} f)(u_i)\end{aligned}$$

From the first to the second line, we use the change of variables  $\tilde{v}_j = w v_j$  and the fact that, group elements in the input domain are sampled from the Haar measure for which it holds that  $\bar{\mu}_G(v_j) = \bar{\mu}_G(\tilde{v}_j)$ . However, from the second to the third line, *we must also assume that this holds for the output domain*. That is, that the probability of drawing  $w^{-1} u_i$  is equal to that of drawing  $u_i$ . We emphasize that this is of particular importance in the partial equivariance setting as this might not be the case in general.

## J.4 Experimental details

### J.4.1 Dataset description

**Dataset availability and licensing.** We note that all the datasets used in this paper are publicly available. MNIST is available online under Creative Commons Attribution-Share Alike 3.0 license. CIFAR-10 and CIFAR-100 are available online under MIT license. PatchCamelyon is available online under MIT license.

**Rotated MNIST.** The rotated MNIST dataset [201] contains 62,000 gray-scale 28x28 hand-written digits extracted from the MNIST dataset [208] uniformly rotated on the circle. The dataset is split into training, validation and test sets of 10,000, 2,000, and 50,000 images, respectively.

**CIFAR-10 and CIFAR-100.** The CIFAR-10 dataset [195] consists of 60,000 real-world 32x32 RGB images uniformly drawn from 10 classes divided into training and test sets of 50,000 and 10,000 samples respectively. The CIFAR100 dataset [195] is similar to the CIFAR0 dataset, with the difference that images are uniformly drawn from 100 different classes. For validation purposes, we divide the training dataset of the CIFAR-10 and CIFAR-100 datasets into training and validation sets of 45,000 and 5,000 samples.

**PatchCamelyon.** The PatchCamelyon dataset [404] consists of 327,000 RGB image patches of tumorous and non-tumorous breast tissues extracted from the Camelyon16 dataset [16], where each patch was labelled as tumorous if the central region of 32x32 pixels contained at least one tumorous pixel as given by the original annotation in Bejnordi et al. [16]. The dataset is divided into train, validation and test sets of 262,144, 32,768 and 32,768 images, respectively.

### J.4.2 General remarks

**Hardware.** Our code is written in PyTorch. Our experiments were performed on NVIDIA TITAN RTX and V100 GPUs, depending on their availability and the size of the datasets.

**Network specifications.** For almost all the experiments in this paper –except those using the 13-layer CNN of Laine and Aila [200]–, we use the architecture shown in Fig. 11.3 with an initial lifting convolutional layer followed by 2 ResBlocks with full, partial or regular convolutional layers for Regular G-CNNs, Partial G-CNNs and conventional (T(2)) CNNs. All datasets use a network with 32 feature maps in the hidden layers, Batch Normalization and ReLU.

For MNIST6-M and MNIST6-180, max-pooling is performed after each of the Residual Blocks. In the case of rotMNIST, max-pooling is performed after the lifting convolutional layer and the first group convolutional layer. For CIFAR-10 and CIFAR-100, we use max-pooling after each of the residual blocks. Finally, for PatchCamelyon, we apply max-pooling after the lifting convolution as well as both residual blocks. At the end

of the network, a global max-pooling layer is used to create invariant features used for classification. These networks have approximately 460K parameters.

**The continuous group convolutional kernels.** The convolutional kernels of Partial G-CNNs are parameterized as 3-layer SIRENs with 32 hidden units. For the experiments in the main text, we use  $\omega_0=10.0$ . We compare these to other conventional nonlinearities in Appx. J.5 (Tab. J.2). In the case of (partial) group equivariant 13-layer CNNs, the convolutional kernels are constructed as a 3-layer SIREN with 8 hidden units.

### J.4.3 Hyperparameters and training details

To facilitate replicating our experiments, we provide the list of commands used for our experiments in [github.com/merlresearch/partial-gcnn/EXPERIMENTS.md](https://github.com/merlresearch/partial-gcnn/EXPERIMENTS.md)

**Optimization and learning rate schedulers.** Networks on MNIST6-180, MNIST6-M, rotMNIST, CIFAR-10 and CIFAR-100 are trained for 300 epochs and networks on PatchCamelyon are trained for 30 epochs. Furthermore, we utilize a cosine annealing scheduler and combine it with a linear learning rate warm-up for 5 epochs.

**Learning schedulers for the probability distributions  $p(u)$ .** In order to improve the stability of learning the probability distributions on the groups, we utilize a learning rate scheduler similar to that of the main network, i.e., learning rate warm-up for 5 epochs followed by a cosine annealing scheduler, but with a lower base learning rate. Specifically, we use a base learning rate for all probability distributions  $p(u)$  of  $1e-4$ .

**Hyperparameters.** We note that all hyperparameters were chosen based on the best performance of the fully equivariant G-CNNs on the validation datasets. The found hyperparameters are subsequently used for the training of our Partial G-CNNs.

We use a batch size of 64 for all networks. In the case of CIFAR-10, CIFAR-100 and PatchCamelyon datasets, we also use a weight decay of  $1e-4$ .

**13-layer CNNs.** Additionally, in the case of 13-layer CNNs we use a dropout rate of 0.3 and train for 200 epochs with batches of size 128. These settings are used on rotMNIST, CIFAR10 and CIFAR100.

## J.5 Additional Experiments

**Classification results on PatchCamelyon.** Table J.1 shows the results obtained for G-CNNs and Partial G-CNNs on the PatchCamelyon dataset [404]. Partial G-CNNs match the performance of G-CNNs in this full equivariant setting. Similar to the rotMNIST case (Fig. 11.5), the learned probability distributions over the group elements for PatchCamelyon are consistent with Regular G-CNNs.

**Convolution kernels as implicit neural representations.** Next, we validate that SIRENs are better suited to parameterize group convolutional kernels than other alternatives.

**Table J.1:** Image recognition accuracy on PatchCam dataset.

| BASE GROUP | NO. ELEMENTS | PARTIAL EQUIV. | CLASSIFICATION ACCURACY ON PATCHCAM (%) |
|------------|--------------|----------------|---|
| T(2)       | 1            | -              | 67.59                                   |
| SE(2)      | 8            | ✗              | <b>89.87</b>                            |
|            |              | ✓              | 89.07                                   |
|            | 16           | ✗              | 89.71                                   |
|            |              | ✓              | <b>90.31</b>                            |
| E(2)       | 16           | ✗              | <b>89.77</b>                            |
|            |              | ✓              | 88.13                                   |

**Table J.2:** Comparison of kernel parameterizations.

| MODEL     | NO. ELEMENTS | KERNEL TYPE | CLASSIFICATION ACCURACY (%) |              |              |
|-----------|--------------|-------------|-----------------------------|--------------|--------------|
|           |              |             | ROT-MNIST                   | CIFAR-10     | CIFAR-100    |
| SE(2)-CNN | 4            | ReLU        | 96.49                       | 59.95        | 28.01        |
|           |              | LeakyReLU   | 94.47                       | 56.19        | 27.36        |
|           |              | Swish       | 94.41                       | 66.12        | 34.20        |
|           |              | SIREN       | <b>99.10</b>                | <b>83.73</b> | <b>52.35</b> |
| SE(2)-CNN | 8            | ReLU        | 97.73                       | 68.29        | 37.81        |
|           |              | LeakyReLU   | 97.65                       | 68.94        | 36.30        |
|           |              | Swish       | 97.72                       | 69.20        | 34.10        |
|           |              | SIREN       | <b>99.17</b>                | <b>86.08</b> | <b>55.55</b> |
| SE(2)-CNN | 16           | ReLU        | 98.49                       | 66.84        | 37.72        |
|           |              | LeakyReLU   | 98.53                       | 68.01        | 38.29        |
|           |              | Swish       | 98.55                       | 65.99        | 37.72        |
|           |              | SIREN       | <b>99.24</b>                | <b>86.68</b> | <b>51.51</b> |

Tab. J.2 shows that SE(2)-CNNs with SIREN kernels consistently outperform SE(2)-CNNs with other parameterizations by a large margin on all the image benchmarks considered. SIREN kernels consistently lead to better accuracy than other existing kernel parameterizations.

**Enforcing monotonic decreasing group subsets over depth.** Once a Partial G-CNN becomes partial equivariant at some depth, the network is, in general, unable to become fully equivariant at subsequent layers.<sup>3</sup> As a consequence, using fully equivariant layers after a partially equivariant layer does not restore full equivariance.

Based on this observation, one could argue that it is beneficial to impose a monotonically decreasing size to the learned group subsets in order to prevent the at first sight meaningless situation in which the network goes back to larger group subsets. This can be encouraged with an additional *monotonic equivariance loss* term in the training loss,

<sup>3</sup>An exception to this rule is when the a layer goes back to the original input space, i.e.,  $\mathcal{S}^{(2)} = \mathcal{X}$ , and the immediately subsequent layer goes back to the full group. This case is equivalent to performing a projection along a group axis, and going back to the full group afterwards, i.e., a lifting convolution.

**Table J.3:** Results with penalty term to encourage monotonicity in the subset sizes

| GROUP | NO. ELEMENTS | ROTMNIST | CIFAR10 | CIFAR100 |
|-------|--------------|----------|---------|----------|
| SE(2) | 16           | 99.15    | 87.02   | 57.11    |
| E(2)  | 16           | 98.41    | 89.00   | 58.85    |

which penalizes bigger subsets at subsequent layers:

$$L_{\text{mon. equiv}} = \sum_{l=1}^{L-1} (\gamma_l - \max(\gamma_{l+1}, \gamma_l)). \quad (\text{J.1})$$

Here,  $\gamma_l$  represents the limit of the subset learned at the  $l$ -th layer.

Interestingly, we find that due to the reasons explained in Sec. 11.6 imposing a monotonic decrease on the learned subsets leads to slightly worse performance than an unconstrained model (see Tabs. J.3, 11.3).

# SIKS Dissertatiereeks

- 2016 01 Syed Saiden Abbas (RUN), Recognition of Shapes by Humans and Machines  
02 Michiel Christiaan Meulendijk (UU), Optimizing medication reviews through decision support: prescribing a better pill to swallow  
03 Maya Sappelli (RUN), Knowledge Work in Context: User Centered Knowledge Worker Support  
04 Laurens Rietveld (VUA), Publishing and Consuming Linked Data  
05 Evgeny Sherkhonov (UvA), Expanded Acyclic Queries: Containment and an Application in Explaining Missing Answers  
06 Michel Wilson (TUD), Robust scheduling in an uncertain environment  
07 Jeroen de Man (VUA), Measuring and modeling negative emotions for virtual training  
08 Matje van de Camp (TiU), A Link to the Past: Constructing Historical Social Networks from Unstructured Data  
09 Archana Nottamkandath (VUA), Trusting Crowdsourced Information on Cultural Artefacts  
10 George Karafotias (VUA), Parameter Control for Evolutionary Algorithms  
11 Anne Schuth (UvA), Search Engines that Learn from Their Users  
12 Max Knobbiout (UU), Logics for Modelling and Verifying Normative Multi-Agent Systems  
13 Nana Baah Gyan (VUA), The Web, Speech Technologies and Rural Development in West Africa - An ICT4D Approach  
14 Ravi Khadka (UU), Revisiting Legacy Software System Modernization  
15 Steffen Michels (RUN), Hybrid Probabilistic Logics - Theoretical Aspects, Algorithms and Experiments  
16 Guangliang Li (UvA), Socially Intelligent Autonomous Agents that Learn from Human Reward  
17 Berend Weel (VUA), Towards Embodied Evolution of Robot Organisms  
18 Albert Meroño Peñuela (VUA), Refining Statistical Data on the Web  
19 Julia Efremova (TU/e), Mining Social Structures from Genealogical Data  
20 Daan Odijk (UvA), Context & Semantics in News & Web Search  
21 Alejandro Moreno Céller (UT), From Traditional to Interactive Playspaces: Automatic Analysis of Player Behavior in the Interactive Tag Playground

- 22 Grace Lewis (VUA), Software Architecture Strategies for Cyber-Foraging Systems
- 23 Fei Cai (UvA), Query Auto Completion in Information Retrieval
- 24 Brend Wanders (UT), Repurposing and Probabilistic Integration of Data; An Iterative and data model independent approach
- 25 Julia Kiseleva (TU/e), Using Contextual Information to Understand Searching and Browsing Behavior
- 26 Dilhan Thilakarathne (VUA), In or Out of Control: Exploring Computational Models to Study the Role of Human Awareness and Control in Behavioural Choices, with Applications in Aviation and Energy Management Domains
- 27 Wen Li (TUD), Understanding Geo-spatial Information on Social Media
- 28 Mingxin Zhang (TUD), Large-scale Agent-based Social Simulation - A study on epidemic prediction and control
- 29 Nicolas Höning (TUD), Peak reduction in decentralised electricity systems - Markets and prices for flexible planning
- 30 Ruud Mattheij (TiU), The Eyes Have It
- 31 Mohammad Khelghati (UT), Deep web content monitoring
- 32 Eelco Vriezekolk (UT), Assessing Telecommunication Service Availability Risks for Crisis Organisations
- 33 Peter Bloem (UvA), Single Sample Statistics, exercises in learning from just one example
- 34 Dennis Schunselaar (TU/e), Configurable Process Trees: Elicitation, Analysis, and Enactment
- 35 Zhaochun Ren (UvA), Monitoring Social Media: Summarization, Classification and Recommendation
- 36 Daphne Karreman (UT), Beyond R2D2: The design of nonverbal interaction behavior optimized for robot-specific morphologies
- 37 Giovanni Sileno (UvA), Aligning Law and Action - a conceptual and computational inquiry
- 38 Andrea Minuto (UT), Materials that Matter - Smart Materials meet Art & Interaction Design
- 39 Merijn Bruijnes (UT), Believable Suspect Agents; Response and Interpersonal Style Selection for an Artificial Suspect
- 40 Christian Detweiler (TUD), Accounting for Values in Design
- 41 Thomas King (TUD), Governing Governance: A Formal Framework for Analysing Institutional Design and Enactment Governance
- 42 Spyros Martzoukos (UvA), Combinatorial and Compositional Aspects of Bilingual Aligned Corpora
- 43 Saskia Koldijk (RUN), Context-Aware Support for Stress Self-Management: From Theory to Practice
- 44 Thibault Sellam (UvA), Automatic Assistants for Database Exploration
- 45 Bram van de Laar (UT), Experiencing Brain-Computer Interface Control

- 
- 46 Jorge Gallego Perez (UT), Robots to Make you Happy
  - 47 Christina Weber (UL), Real-time foresight - Preparedness for dynamic innovation networks
  - 48 Tanja Buttler (TUD), Collecting Lessons Learned
  - 49 Gleb Polevoy (TUD), Participation and Interaction in Projects. A Game-Theoretic Analysis
  - 50 Yan Wang (TiU), The Bridge of Dreams: Towards a Method for Operational Performance Alignment in IT-enabled Service Supply Chains
- 
- 2017 01 Jan-Jaap Oerlemans (UL), Investigating Cybercrime
  - 02 Sjoerd Timmer (UU), Designing and Understanding Forensic Bayesian Networks using Argumentation
  - 03 Daniël Harold Telgen (UU), Grid Manufacturing; A Cyber-Physical Approach with Autonomous Products and Reconfigurable Manufacturing Machines
  - 04 Mrunal Gawade (CWI), Multi-core Parallelism in a Column-store
  - 05 Mahdieh Shadi (UvA), Collaboration Behavior
  - 06 Damir Vandic (EUR), Intelligent Information Systems for Web Product Search
  - 07 Roel Bertens (UU), Insight in Information: from Abstract to Anomaly
  - 08 Rob Konijn (VUA), Detecting Interesting Differences: Data Mining in Health Insurance Data using Outlier Detection and Subgroup Discovery
  - 09 Dong Nguyen (UT), Text as Social and Cultural Data: A Computational Perspective on Variation in Text
  - 10 Robby van Delden (UT), (Steering) Interactive Play Behavior
  - 11 Florian Kunneman (RUN), Modelling patterns of time and emotion in Twitter #anticipointment
  - 12 Sander Leemans (TU/e), Robust Process Mining with Guarantees
  - 13 Gijs Huisman (UT), Social Touch Technology - Extending the reach of social touch through haptic technology
  - 14 Shoshannah Tekofsky (TiU), You Are Who You Play You Are: Modelling Player Traits from Video Game Behavior
  - 15 Peter Berck (RUN), Memory-Based Text Correction
  - 16 Aleksandr Chuklin (UvA), Understanding and Modeling Users of Modern Search Engines
  - 17 Daniel Dimov (UL), Crowdsourced Online Dispute Resolution
  - 18 Ridho Reinanda (UvA), Entity Associations for Search
  - 19 Jeroen Vuurens (UT), Proximity of Terms, Texts and Semantic Vectors in Information Retrieval
  - 20 Mohammadbashir Sedighi (TUD), Fostering Engagement in Knowledge Sharing: The Role of Perceived Benefits, Costs and Visibility
  - 21 Jeroen Linssen (UT), Meta Matters in Interactive Storytelling and Serious Gaming (A Play on Worlds)
  - 22 Sara Magliacane (VUA), Logics for causal inference under uncertainty

- 23 David Graus (UvA), Entities of Interest — Discovery in Digital Traces
- 24 Chang Wang (TUD), Use of Affordances for Efficient Robot Learning
- 25 Veruska Zamborlini (VUA), Knowledge Representation for Clinical Guidelines, with applications to Multimorbidity Analysis and Literature Search
- 26 Merel Jung (UT), Socially intelligent robots that understand and respond to human touch
- 27 Michiel Joosse (UT), Investigating Positioning and Gaze Behaviors of Social Robots: People's Preferences, Perceptions and Behaviors
- 28 John Klein (VUA), Architecture Practices for Complex Contexts
- 29 Adel Alhuraibi (TiU), From IT-BusinessStrategic Alignment to Performance: A Moderated Mediation Model of Social Innovation, and Enterprise Governance of IT"
- 30 Wilma Latuny (TiU), The Power of Facial Expressions
- 31 Ben Ruijl (UL), Advances in computational methods for QFT calculations
- 32 Thaer Samar (RUN), Access to and Retrievability of Content in Web Archives
- 33 Brigit van Loggem (OU), Towards a Design Rationale for Software Documentation: A Model of Computer-Mediated Activity
- 34 Maren Scheffel (OU), The Evaluation Framework for Learning Analytics
- 35 Martine de Vos (VUA), Interpreting natural science spreadsheets
- 36 Yuanhao Guo (UL), Shape Analysis for Phenotype Characterisation from High-throughput Imaging
- 37 Alejandro Montes Garcia (TU/e), WiBAF: A Within Browser Adaptation Framework that Enables Control over Privacy
- 38 Alex Kayal (TUD), Normative Social Applications
- 39 Sara Ahmadi (RUN), Exploiting properties of the human auditory system and compressive sensing methods to increase noise robustness in ASR
- 40 Altaf Hussain Abro (VUA), Steer your Mind: Computational Exploration of Human Control in Relation to Emotions, Desires and Social Support For applications in human-aware support systems
- 41 Adnan Manzoor (VUA), Minding a Healthy Lifestyle: An Exploration of Mental Processes and a Smart Environment to Provide Support for a Healthy Lifestyle
- 42 Elena Sokolova (RUN), Causal discovery from mixed and missing data with applications on ADHD datasets
- 43 Maaike de Boer (RUN), Semantic Mapping in Video Retrieval
- 44 Garm Lucassen (UU), Understanding User Stories - Computational Linguistics in Agile Requirements Engineering
- 45 Bas Testerink (UU), Decentralized Runtime Norm Enforcement
- 46 Jan Schneider (OU), Sensor-based Learning Support
- 47 Jie Yang (TUD), Crowd Knowledge Creation Acceleration
- 48 Angel Suarez (OU), Collaborative inquiry-based learning

- 2018 01 Han van der Aa (VUA), Comparing and Aligning Process Representations
- 02 Felix Mannhardt (TU/e), Multi-perspective Process Mining
- 03 Steven Boses (UT), Causal Models For Well-Being: Knowledge Modeling, Model-Driven Development of Context-Aware Applications, and Behavior Prediction
- 04 Jordan Janeiro (TUD), Flexible Coordination Support for Diagnosis Teams in Data-Centric Engineering Tasks
- 05 Hugo Huirudeman (UvA), Supporting the Complex Dynamics of the Information Seeking Process
- 06 Dan Ionita (UT), Model-Driven Information Security Risk Assessment of Socio-Technical Systems
- 07 Jieting Luo (UU), A formal account of opportunism in multi-agent systems
- 08 Rick Smetsers (RUN), Advances in Model Learning for Software Systems
- 09 Xu Xie (TUD), Data Assimilation in Discrete Event Simulations
- 10 Julienka Mollee (VUA), Moving forward: supporting physical activity behavior change through intelligent technology
- 11 Mahdi Sargolzaei (UvA), Enabling Framework for Service-oriented Collaborative Networks
- 12 Xixi Lu (TU/e), Using behavioral context in process mining
- 13 Seyed Amin Tabatabaei (VUA), Computing a Sustainable Future
- 14 Bart Joosten (TiU), Detecting Social Signals with Spatiotemporal Gabor Filters
- 15 Naser Davarzani (UM), Biomarker discovery in heart failure
- 16 Jaebok Kim (UT), Automatic recognition of engagement and emotion in a group of children
- 17 Jianpeng Zhang (TU/e), On Graph Sample Clustering
- 18 Henriette Nakad (UL), De Notaris en Private Rechtspraak
- 19 Minh Duc Pham (VUA), Emergent relational schemas for RDF
- 20 Manxia Liu (RUN), Time and Bayesian Networks
- 21 Aad Slootmaker (OU), EMERGO: a generic platform for authoring and playing scenario-based serious games
- 22 Eric Fernandes de Mello Araújo (VUA), Contagious: Modeling the Spread of Behaviours, Perceptions and Emotions in Social Networks
- 23 Kim Schouten (EUR), Semantics-driven Aspect-Based Sentiment Analysis
- 24 Jered Vroon (UT), Responsive Social Positioning Behaviour for Semi-Autonomous Telepresence Robots
- 25 Riste Gligorov (VUA), Serious Games in Audio-Visual Collections
- 26 Roelof Anne Jelle de Vries (UT), Theory-Based and Tailor-Made: Motivational Messages for Behavior Change Technology
- 27 Maikel Leemans (TU/e), Hierarchical Process Mining for Scalable Software Analysis
- 28 Christian Willemse (UT), Social Touch Technologies: How they feel and how they make you feel

- 29 Yu Gu (TiU), Emotion Recognition from Mandarin Speech
- 30 Wouter Beek (VUA), The "K" in "semantic web" stands for "knowledge": scaling semantics to the web
- 
- 2019 01 Rob van Eijk (UL), Web privacy measurement in real-time bidding systems. A graph-based approach to RTB system classification
- 02 Emmanuelle Beauxis Aussalet (CWI, UU), Statistics and Visualizations for Assessing Class Size Uncertainty
- 03 Eduardo Gonzalez Lopez de Murillas (TU/e), Process Mining on Databases: Extracting Event Data from Real Life Data Sources
- 04 Ridho Rahmadi (RUN), Finding stable causal structures from clinical data
- 05 Sebastiaan van Zelst (TU/e), Process Mining with Streaming Data
- 06 Chris Dijkshoorn (VUA), Nichesourcing for Improving Access to Linked Cultural Heritage Datasets
- 07 Soude Fazeli (TUD), Recommender Systems in Social Learning Platforms
- 08 Frits de Nijs (TUD), Resource-constrained Multi-agent Markov Decision Processes
- 09 Fahimeh Alizadeh Moghaddam (UvA), Self-adaptation for energy efficiency in software systems
- 10 Qing Chuan Ye (EUR), Multi-objective Optimization Methods for Allocation and Prediction
- 11 Yue Zhao (TUD), Learning Analytics Technology to Understand Learner Behavioral Engagement in MOOCs
- 12 Jacqueline Heinerman (VUA), Better Together
- 13 Guanliang Chen (TUD), MOOC Analytics: Learner Modeling and Content Generation
- 14 Daniel Davis (TUD), Large-Scale Learning Analytics: Modeling Learner Behavior & Improving Learning Outcomes in Massive Open Online Courses
- 15 Erwin Walraven (TUD), Planning under Uncertainty in Constrained and Partially Observable Environments
- 16 Guangming Li (TU/e), Process Mining based on Object-Centric Behavioral Constraint (OCBC) Models
- 17 Ali Hurriyetoglu (RUN), Extracting actionable information from microtexts
- 18 Gerard Wagenaar (UU), Artefacts in Agile Team Communication
- 19 Vincent Koeman (TUD), Tools for Developing Cognitive Agents
- 20 Chide Groenouwe (UU), Fostering technically augmented human collective intelligence
- 21 Cong Liu (TU/e), Software Data Analytics: Architectural Model Discovery and Design Pattern Detection
- 22 Martin van den Berg (VUA), Improving IT Decisions with Enterprise Architecture

- 23 Qin Liu (TUD), Intelligent Control Systems: Learning, Interpreting, Verification
  - 24 Anca Dumitrache (VUA), Truth in Disagreement - Crowdsourcing Labeled Data for Natural Language Processing
  - 25 Emiel van Miltenburg (VUA), Pragmatic factors in (automatic) image description
  - 26 Prince Singh (UT), An Integration Platform for Synchromodal Transport
  - 27 Alessandra Antonaci (OU), The Gamification Design Process applied to (Massive) Open Online Courses
  - 28 Esther Kuindersma (UL), Cleared for take-off: Game-based learning to prepare airline pilots for critical situations
  - 29 Daniel Formolo (VUA), Using virtual agents for simulation and training of social skills in safety-critical circumstances
  - 30 Vahid Yazdanpanah (UT), Multiagent Industrial Symbiosis Systems
  - 31 Milan Jelisavcic (VUA), Alive and Kicking: Baby Steps in Robotics
  - 32 Chiara Sironi (UM), Monte-Carlo Tree Search for Artificial General Intelligence in Games
  - 33 Anil Yaman (TU/e), Evolution of Biologically Inspired Learning in Artificial Neural Networks
  - 34 Negar Ahmadi (TU/e), EEG Microstate and Functional Brain Network Features for Classification of Epilepsy and PNES
  - 35 Lisa Facey-Shaw (OU), Gamification with digital badges in learning programming
  - 36 Kevin Ackermans (OU), Designing Video-Enhanced Rubrics to Master Complex Skills
  - 37 Jian Fang (TUD), Database Acceleration on FPGAs
  - 38 Akos Kadar (OU), Learning visually grounded and multilingual representations
- 

- |      |   |
|------|---|
| 2020 | 01 Armon Toubman (UL), Calculated Moves: Generating Air Combat Behaviour  |
|      | 02 Marcos de Paula Bueno (UL), Unraveling Temporal Processes using Probabilistic Graphical Models                             |
|      | 03 Mostafa Deghani (UvA), Learning with Imperfect Supervision for Language Understanding                                      |
|      | 04 Maarten van Gompel (RUN), Context as Linguistic Bridges  |
|      | 05 Yulong Pei (TU/e), On local and global structure mining  |
|      | 06 Preethu Rose Anish (UT), Stimulation Architectural Thinking during Requirements Elicitation - An Approach and Tool Support |
|      | 07 Wim van der Vegt (OU), Towards a software architecture for reusable game components  |
|      | 08 Ali Mirsoleimani (UL), Structured Parallel Programming for Monte Carlo Tree Search   |

- 09 Myriam Traub (UU), Measuring Tool Bias and Improving Data Quality for Digital Humanities Research
- 10 Alifah Syamsiyah (TU/e), In-database Preprocessing for Process Mining
- 11 Sepideh Mesbah (TUD), Semantic-Enhanced Training Data Augmentation-Methods for Long-Tail Entity Recognition Models
- 12 Ward van Breda (VUA), Predictive Modeling in E-Mental Health: Exploring Applicability in Personalised Depression Treatment
- 13 Marco Virgolin (CWI), Design and Application of Gene-pool Optimal Mixing Evolutionary Algorithms for Genetic Programming
- 14 Mark Raasveldt (CWI/UL), Integrating Analytics with Relational Databases
- 15 Konstantinos Georgiadis (OU), Smart CAT: Machine Learning for Configurable Assessments in Serious Games
- 16 Ilona Wilmont (RUN), Cognitive Aspects of Conceptual Modelling
- 17 Daniele Di Mitri (OU), The Multimodal Tutor: Adaptive Feedback from Multimodal Experiences
- 18 Georgios Methenitis (TUD), Agent Interactions & Mechanisms in Markets with Uncertainties: Electricity Markets in Renewable Energy Systems
- 19 Guido van Capelleveen (UT), Industrial Symbiosis Recommender Systems
- 20 Albert Hankel (VUA), Embedding Green ICT Maturity in Organisations
- 21 Karine da Silva Miras de Araujo (VUA), Where is the robot?: Life as it could be
- 22 Maryam Masoud Khamis (RUN), Understanding complex systems implementation through a modeling approach: the case of e-government in Zanzibar
- 23 Rianne Conijn (UT), The Keys to Writing: A writing analytics approach to studying writing processes using keystroke logging
- 24 Lenin da Nóbrega Medeiros (VUA/RUN), How are you feeling, human? Towards emotionally supportive chatbots
- 25 Xin Du (TU/e), The Uncertainty in Exceptional Model Mining
- 26 Krzysztof Leszek Sadowski (UU), GAMBIT: Genetic Algorithm for Model-Based mixed-Integer opTimization
- 27 Ekaterina Muravyeva (TUD), Personal data and informed consent in an educational context
- 28 Bibeg Limbu (TUD), Multimodal interaction for deliberate practice: Training complex skills with augmented reality
- 29 Ioan Gabriel Bucur (RUN), Being Bayesian about Causal Inference
- 30 Bob Zadok Blok (UL), Creatief, Creatiever, Creatiefst
- 31 Gongjin Lan (VUA), Learning better – From Baby to Better
- 32 Jason Rhuggenaath (TU/e), Revenue management in online markets: pricing and online advertising
- 33 Rick Gilsing (TU/e), Supporting service-dominant business model evaluation in the context of business model innovation

- 34 Anna Bon (UM), Intervention or Collaboration? Redesigning Information and Communication Technologies for Development
- 35 Siamak Farshidi (UU), Multi-Criteria Decision-Making in Software Production
- 
- 2021 01 Francisco Xavier Dos Santos Fonseca (TUD), Location-based Games for Social Interaction in Public Space
- 02 Rijk Mercuri (TUD), Simulating Human Routines: Integrating Social Practice Theory in Agent-Based Models
- 03 Seyyed Hadi Hashemi (UvA), Modeling Users Interacting with Smart Devices
- 04 Ioana Jivet (OU), The Dashboard That Loved Me: Designing adaptive learning analytics for self-regulated learning
- 05 Davide Dell'Anna (UU), Data-Driven Supervision of Autonomous Systems
- 06 Daniel Davison (UT), "Hey robot, what do you think?" How children learn with a social robot
- 07 Armel Lefebvre (UU), Research data management for open science
- 08 Nardie Fanchamps (OU), The Influence of Sense-Reason-Act Programming on Computational Thinking
- 09 Cristina Zaga (UT), The Design of Robothings. Non-Anthropomorphic and Non-Verbal Robots to Promote Children's Collaboration Through Play
- 10 Quinten Meertens (UvA), Misclassification Bias in Statistical Learning
- 11 Anne van Rossum (UL), Nonparametric Bayesian Methods in Robotic Vision
- 12 Lei Pi (UL), External Knowledge Absorption in Chinese SMEs
- 13 Bob R. Schadenberg (UT), Robots for Autistic Children: Understanding and Facilitating Predictability for Engagement in Learning
- 14 Negin Samaeemofrad (UL), Business Incubators: The Impact of Their Support
- 15 Onat Ege Adali (TU/e), Transformation of Value Propositions into Resource Re-Configurations through the Business Services Paradigm
- 16 Esam A. H. Ghaleb (UM), Bimodal emotion recognition from audio-visual cues
- 17 Dario Dotti (UM), Human Behavior Understanding from motion and bodily cues using deep neural networks
- 18 Remi Wieten (UU), Bridging the Gap Between Informal Sense-Making Tools and Formal Systems - Facilitating the Construction of Bayesian Networks and Argumentation Frameworks
- 19 Roberto Verdecchia (VUA), Architectural Technical Debt: Identification and Management
- 20 Masoud Mansouri (TU/e), Understanding and Mitigating Multi-Sided Exposure Bias in Recommender Systems
- 21 Pedro Thiago Timbó Holanda (CWI), Progressive Indexes
- 22 Sihang Qiu (TUD), Conversational Crowdsourcing
- 23 Hugo Manuel Proença (UL), Robust rules for prediction and description

- 24 Kaijie Zhu (TU/e), On Efficient Temporal Subgraph Query Processing
- 25 Eoin Martino Grua (VUA), The Future of E-Health is Mobile: Combining AI and Self-Adaptation to Create Adaptive E-Health Mobile Applications
- 26 Benno Kruit (CWI/VUA), Reading the Grid: Extending Knowledge Bases from Human-readable Tables
- 27 Jelte van Waterschoot (UT), Personalized and Personal Conversations: Designing Agents Who Want to Connect With You
- 28 Christoph Selig (UL), Understanding the Heterogeneity of Corporate Entrepreneurship Programs
- 
- 2022 01 Judith van Stegeren (UT), Flavor text generation for role-playing video games
- 02 Paulo da Costa (TU/e), Data-driven Prognostics and Logistics Optimisation: A Deep Learning Journey
- 03 Ali el Hassouni (VUA), A Model A Day Keeps The Doctor Away: Reinforcement Learning For Personalized Healthcare
- 04 Ünal Aksu (UU), A Cross-Organizational Process Mining Framework
- 05 Shiwei Liu (TU/e), Sparse Neural Network Training with In-Time Over-Parameterization
- 06 Reza Refaei Afshar (TU/e), Machine Learning for Ad Publishers in Real Time Bidding
- 07 Sambit Praharaj (OU), Measuring the Unmeasurable? Towards Automatic Co-located Collaboration Analytics
- 08 Maikel L. van Eck (TU/e), Process Mining for Smart Product Design
- 09 Oana Andreea Inel (VUA), Understanding Events: A Diversity-driven Human-Machine Approach
- 10 Felipe Moraes Gomes (TUD), Examining the Effectiveness of Collaborative Search Engines
- 11 Mirjam de Haas (UT), Staying engaged in child-robot interaction, a quantitative approach to studying preschoolers' engagement with robots and tasks during second-language tutoring
- 12 Guanyi Chen (UU), Computational Generation of Chinese Noun Phrases
- 13 Xander Wilcke (VUA), Machine Learning on Multimodal Knowledge Graphs: Opportunities, Challenges, and Methods for Learning on Real-World Heterogeneous and Spatially-Oriented Knowledge
- 14 Michiel Overeem (UU), Evolution of Low-Code Platforms
- 15 Jelmer Jan Koorn (UU), Work in Process: Unearthing Meaning using Process Mining
- 16 Pieter Gijsbers (TU/e), Systems for AutoML Research
- 17 Laura van der Lubbe (VUA), Empowering vulnerable people with serious games and gamification
- 18 Paris Mavromoustakos Blom (TiU), Player Affect Modelling and Video Game Personalisation

- 19 Bilge Yigit Ozkan (UU), Cybersecurity Maturity Assessment and Standardisation
- 20 Fakhra Jabeen (VUA), Dark Side of the Digital Media - Computational Analysis of Negative Human Behaviors on Social Media
- 21 Seethu Mariyam Christopher (UM), Intelligent Toys for Physical and Cognitive Assessments
- 22 Alexandra Sierra Rativa (TiU), Virtual Character Design and its potential to foster Empathy, Immersion, and Collaboration Skills in Video Games and Virtual Reality Simulations
- 23 Ilir Kola (TUD), Enabling Social Situation Awareness in Support Agents
- 24 Samaneh Heidari (UU), Agents with Social Norms and Values - A framework for agent based social simulations with social norms and personal values
- 25 Anna L.D. Latour (UL), Optimal decision-making under constraints and uncertainty
- 26 Anne Dirkson (UL), Knowledge Discovery from Patient Forums: Gaining novel medical insights from patient experiences
- 27 Christos Athanasiadis (UM), Emotion-aware cross-modal domain adaptation in video sequences
- 28 Onuralp Ulusoy (UU), Privacy in Collaborative Systems
- 29 Jan Kolkmeier (UT), From Head Transform to Mind Transplant: Social Interactions in Mixed Reality
- 30 Dean De Leo (CWI), Analysis of Dynamic Graphs on Sparse Arrays
- 31 Konstantinos Traganas (TU/e), Tackling Complexity in Smart Manufacturing with Advanced Manufacturing Process Management
- 32 Cezara Pastrav (UU), Social simulation for socio-ecological systems
- 33 Brinn Hekkelman (CWI/TUD), Fair Mechanisms for Smart Grid Congestion Management
- 34 Nimat Ullah (VUA), Mind Your Behaviour: Computational Modelling of Emotion & Desire Regulation for Behaviour Change
- 35 Mike E.U. Ligthart (VUA), Shaping the Child-Robot Relationship: Interaction Design Patterns for a Sustainable Interaction

- 2023 01 Bojan Simoski (VUA), Untangling the Puzzle of Digital Health Interventions
- 02 Mariana Rachel Dias da Silva (TiU), Grounded or in flight? What our bodies can tell us about the whereabouts of our thoughts
- 03 Shabnam Najafian (TUD), User Modeling for Privacy-preserving Explanations in Group Recommendations
- 04 Gineke Wiggers (UL), The Relevance of Impact: bibliometric-enhanced legal information retrieval
- 05 Anton Bouter (CWI), Optimal Mixing Evolutionary Algorithms for Large-Scale Real-Valued Optimization, Including Real-World Medical Applications

- 06 António Pereira Barata (UL), Reliable and Fair Machine Learning for Risk Assessment
- 07 Tianjin Huang (TU/e), The Roles of Adversarial Examples on Trustworthiness of Deep Learning
- 08 Lu Yin (TU/e), Knowledge Elicitation using Psychometric Learning
- 09 Xu Wang (VUA), Scientific Dataset Recommendation with Semantic Techniques
- 10 Dennis J.N.J. Soemers (UM), Learning State-Action Features for General Game Playing
- 11 Fawad Taj (VUA), Towards Motivating Machines: Computational Modeling of the Mechanism of Actions for Effective Digital Health Behavior Change Applications
- 12 Tessel Bogaard (VUA), Using Metadata to Understand Search Behavior in Digital Libraries
- 13 Injy Sarhan (UU), Open Information Extraction for Knowledge Representation
- 14 Selma Čaušević (TUD), Energy resilience through self-organization
- 15 Alvaro Henrique Chaim Correia (TU/e), Insights on Learning Tractable Probabilistic Graphical Models
- 16 Peter Blomsma (TiU), Building Embodied Conversational Agents: Observations on human nonverbal behaviour as a resource for the development of artificial characters
- 17 Meike Nauta (UT), Explainable AI and Interpretable Computer Vision – From Oversight to Insight
- 18 Gustavo Penha (TUD), Designing and Diagnosing Models for Conversational Search and Recommendation
- 19 George Aalbers (TiU), Digital Traces of the Mind: Using Smartphones to Capture Signals of Well-Being in Individuals
- 20 Arkadiy Dushatskiy (TUD), Expensive Optimization with Model-Based Evolutionary Algorithms applied to Medical Image Segmentation using Deep Learning
- 21 Gerrit Jan de Bruin (UL), Network Analysis Methods for Smart Inspection in the Transport Domain
- 22 Alireza Shujaifar (UU), Volitional Cybersecurity
- 23 Theo Theunissen (UU), Documentation in Continuous Software Development
- 24 Agathe Balayn (TUD), Practices Towards Hazardous Failure Diagnosis in Machine Learning
- 25 Jurian Baas (UU), Entity Resolution on Historical Knowledge Graphs
- 26 Loek Tonner (TU/e), Linearly Symmetry-Based Disentangled Representations and their Out-of-Distribution Behaviour
- 27 Ghada Sokar (TU/e), Learning Continually Under Changing Data Distributions

- 28 Floris den Hengst (VUA), Learning to Behave: Reinforcement Learning in Human Contexts
- 29 Tim Draws (TUD), Understanding Viewpoint Biases in Web Search Results
- 
- 2024 01 Daphne Miedema (TU/e), On Learning SQL: Disentangling concepts in data systems education
- 02 Emile van Krieken (VUA), Optimisation in Neurosymbolic Learning Systems
- 03 Feri Wijayanto (RUN), Automated Model Selection for Rasch and Mediation Analysis
- 04 Mike Huisman (UL), Understanding Deep Meta-Learning
- 05 Yiyong Gou (UM), Aerial Robotic Operations: Multi-environment Cooperative Inspection & Construction Crack Autonomous Repair
- 06 Azqa Nadeem (TUD), Understanding Adversary Behavior via XAI: Leveraging Sequence Clustering to Extract Threat Intelligence
- 07 Parisa Shayan (TiU), Modeling User Behavior in Learning Management Systems
- 08 Xin Zhou (UvA), From Empowering to Motivating: Enhancing Policy Enforcement through Process Design and Incentive Implementation
- 09 Giso Dal (UT), Probabilistic Inference Using Partitioned Bayesian Networks
- 10 Cristina-Iulia Bucur (VUA), Linkflows: Towards Genuine Semantic Publishing in Science
- 11 withdrawn
- 12 Peide Zhu (TUD), Towards Robust Automatic Question Generation For Learning
- 13 Enrico Liscio (TUD), Context-Specific Value Inference via Hybrid Intelligence
- 14 Larissa Capobianco Shimomura (TU/e), On Graph Generating Dependencies and their Applications in Data Profiling
- 15 Ting Liu (VUA), A Gut Feeling: Biomedical Knowledge Graphs for Interrelating the Gut Microbiome and Mental Health
- 16 Arthur Barbosa Câmara (TUD), Designing Search-as-Learning Systems
- 17 Razieh Alidoosti (VUA), Ethics-aware Software Architecture Design
- 18 Laurens Stoop (UU), Data Driven Understanding of Energy-Meteorological Variability and its Impact on Energy System Operations
- 19 Azadeh Mozafari Mehr (TU/e), Multi-perspective Conformance Checking: Identifying and Understanding Patterns of Anomalous Behavior
- 20 Ritsart Anne Plantenga (UL), Omgang met Regels
- 21 Federica Vinella (UU), Crowdsourcing User-Centered Teams
- 22 Zeynep Ozturk Yurt (TU/e), Beyond Routine: Extending BPM for Knowledge-Intensive Processes with Controllable Dynamic Contexts
- 23 Jie Luo (VUA), Lamarck's Revenge: Inheritance of Learned Traits Improves Robot Evolution

- 24 Nirmal Roy (TUD), Exploring the effects of interactive interfaces on user search behaviour
- 25 Alisa Rieger (TUD), Striving for Responsible Opinion Formation in Web Search on Debated Topics
- 26 Tim Gubner (CWI), Adaptively Generating Heterogeneous Execution Strategies using the VOILA Framework
- 27 Lincen Yang (UL), Information-theoretic Partition-based Models for Interpretable Machine Learning
- 28 Leon Helwerda (UL), Grip on Software: Understanding development progress of Scrum sprints and backlogs