UNIVERSITÄT ZU LÜBECK

From the Institute for Neuro- and Bioinformatics
of the University of Lübeck
Director: Prof. Dr. rer. nat. Thomas Martinetz

Invariant Integration for Prior-Knowledge Enhanced Deep Learning Architectures

Dissertation
for Fulfillment of
Requirements
for the Doctoral Degree
of the University of Lübeck

from the Department of Computer Sciences and Technical Engineering

Submitted by
Matthias Rath
from Nagold

Lübeck 2025

First referee: PD Dr.-Ing. habil. Alexandru Paul Condurache
Second referee: Prof. Dr.-Ing. Alfred Mertins
Date of oral examination: 10.03.2025
Approved for printing. Lübeck, 03.04.2025

UNIVERSITÄT ZU LÜBECK

Aus dem Institut für Neuro- und Bioinformatik
der Universität zu Lübeck
Direktor: Prof. Dr. rer. nat. Thomas Martinetz

Invariant Integration for Prior-Knowledge Enhanced Deep Learning Architectures

Inauguraldissertation
zur
Erlangung der Doktorwürde
der Universität zu Lübeck

Aus der Sektion Informatik/Technik

vorgelegt von
Matthias Rath
aus Nagold

Lübeck, 2025

1. Berichterstatter/Berichterstatterin: PD Dr.-Ing. habil. Alexandru Paul Condurache
2. Berichterstatter/Berichterstatterin: Prof. Dr.-Ing. Alfred Mertins
Tag der mündlichen Prüfung: 10.03.2025
Zum Druck genehmigt. Lübeck, den 03.04.2025

Abstract

Incorporating prior knowledge to Deep Neural Networks is a promising approach to improve their sample efficiency by effectively limiting the search space the learning algorithm needs to cover. This reduces the amount of samples a network needs to be trained on to reach a specific performance. *Geometrical prior knowledge* is the knowledge about input transformations that affect the output in a predictable way, or not at all. It can be built into Deep Neural Networks in a mathematically sound manner by enforcing *in-* or *equivariance*.

Equivariance is the property of a map to behave predictably under input transformations. Convolutions are an example for a *translation-equivariant* map, where a translation of the input results in a shifted output. *Group-equivariant convolutions* are a generalization achieving equivariance towards more general *transformation groups* such as rotations or flips. Using group-equivariant convolutions within Neural Networks embeds the desired equivariance in addition to translations.

Invariance is a closely related concept, where the output of a function does not change when its input is transformed. Invariance is often a desirable property of a feature extractor in the context of classification. While the extracted features need to encode the information required to discriminate between different classes, they should be *invariant to intra-class variations*, i.e., to transformations that map samples within the same class subspace. In the context of Deep Neural Networks, the required *invariant* representations can be obtained with mathematical guarantees by applying group-equivariant convolutions followed by globally pooling among the group- and spatial domain. While *pooling* guarantees invariance, it also *discards information* and is thus not ideal.

In this dissertation, we investigate the *transition from equi- to invariance* within Deep Neural Networks that leverage geometrical prior knowledge. Therefore, we replace the spatial pooling operation with *Invariant Integration*, a method that guarantees invariance while adding *targeted model capacity* rather than destroying information. We first propose

an Invariant Integration Layer for *rotations* based on the *group average* calculated with monomials. The layer can be readily used within a Neural Network and allows backpropagating through it. The monomial parameters are selected either by iteratively optimizing the least-squared-error of a linear classifier, or based on neural network pruning methods.

We then *replace the monomials* with functions that are more often encountered in the context of Neural Networks such as *learnable weighted sums* or self-attention. We thereby *streamline the training procedure* of Neural Networks enhanced with Invariant Integration.

Finally, we expand Invariant Integration towards *flips and scales*, highlighting the universality of our approach. We further propose a *multi-stream architecture* that is able to *leverage invariance to multiple transformations* at once. This approach allows us to efficiently *combine multiple invariances* and select the best-fit invariant solution for the specific problem to solve.

The conducted experiments show that applying *Invariant Integration* in combination with *group-equivariant convolutions* significantly *boosts* the *sample efficiency* of Deep Neural Networks *improving* the *performance* when the amount of available *training data is limited*.

Kurzfassung

Das Einbauen von a-priori Wissen ist ein vielversprechender Ansatz, um die Dateneffizienz tiefer neuronaler Netze zu verbessern, indem der Suchraum des Lernalgorithmus verkleinert wird. Dies reduziert die Anzahl der benötigten Trainingsbeispiele, um eine bestimmte Performanz zu erreichen. *Geometrisches a-priori Wissen* beschreibt die Kenntnis von Eingangstransformationen, die den gewünschten Netzwerkausgang in vorhersehbarer Weise oder überhaupt nicht beeinflussen. Es kann in tiefe neuronale Netze eingebaut werden, indem *In-* oder *Equivarianz* erzwungen wird.

Equivarianz beschreibt die Eigenschaft einer Abbildung, sich unter Eingangstransformationen vorhersehbar zu verhalten. Faltungen sind ein Beispiel für eine *translationsequivariante* Abbildung, wobei Eingangstranslationen in Ausgangstranslationen resultieren. Die *gruppen-equivariante Faltung* ist eine Erweiterung für generelle *Gruppentransformationen*, welche Rotationen oder Spiegelungen beinhalten. Durch Nutzung der gruppen-equivarianten Faltung in tiefen neuronalen Netzen kann Equivarianz zu zusätzlichen Transformationen neben Translationen erzwungen werden.

Invarianz ist eine verwandte Eigenschaft, bei der ein Funktionsausgang sich nicht ändert, wenn der Funktionseingang transformiert wird. Invarianz ist häufig eine wünschenswerte Eigenschaft für einen Merkmalsextraktor, welcher für Klassifikation eingesetzt wird. Die extrahierten Merkmale sollen ausschließlich die Information enkodieren, die zur Unterscheidung verschiedener Klassen benötigt wird. Im Gegensatz dazu profitieren sie von *Invarianz* zu *intraklassen Variationen*, bei der ein Beispiel innerhalb desselben Klassenunterraumes transformiert wird. In tiefen neuronalen Netzen können die gewünschten invarianten Merkmale mathematisch garantiert durch gruppen-equivariante Faltungen gefolgt von globalem *Pooling* entlang der Gruppen- sowie der räumlichen Domäne erreicht werden. Obwohl *Pooling* Invarianz garantiert, *zerstört* es *Information*, weshalb es keine ideale Operation ist.

In dieser Dissertation untersuchen wir daher den *Transfer von Equivarianz zu Invarianz*

in neuronalen Netzwerken, die mit geometrischem a-priori Wissen angereichert sind. Dafür ersetzen wir die räumliche Pooling-Operation mit *invarianter Integration*. Die invariante Integration ist eine Methode, die Invarianz garantiert, während sie *gezielte Modellkapazität* hinzufügt, anstatt Informationen zu zerstören. Zunächst führen wir eine *rotations-invariante Integrationsschicht* für neuronale Netze ein, die auf dem *Gruppendurchschnitt* mit *Monomen* basiert. Diese Schicht kann direkt innerhalb von neuronalen Netzen verwendet werden, da sie kompatibel mit dem *Backpropagation*-Algorithmus ist. Dabei werden die *Monomparameter* entweder durch *iterative* Optimierung eines linearen Klassifikators mit der *Least-Squares*-Methode, oder mittels *Netzwerkpruning*-Methoden ausgewählt.

Im zweiten Schritt werden die *Monome* durch Funktionen *ersetzt*, die häufiger im Kontext neuronaler Netze eingesetzt werden, wodurch die *Trainingsprozedur vereinfacht* wird. Dies umfasst eine *lernbare, gewichtete Summe*, sowie den *Self-Attention* Algorithmus.

Schließlich wird die *invariante Integration erweitert*, um *Invarianz zu Spiegelungen und Skalierungen* zu erreichen. Dies unterstreicht die Universalität unseres untersuchten Ansatzes. Eine *mehrarmige Netzwerkarchitektur* erzielt *simultan Invarianz zu mehreren Transformationen*. Dieser Ansatz ermöglicht es, mehrere Invarianzen effizient zu kombinieren, sowie die am besten zu der zu lösenden Aufgabe passenden Invarianzeigenschaften automatisch auszuwählen.

Die durchgeführten Experimente beweisen, dass die *invariante Integration* in Kombination mit gruppen-equivarianter Faltung die *Dateneffizienz* von *tiefen neuronalen Netzen* signifikant *verbessert*. Dadurch wird eine *verbesserte Performanz* erzielt, wenn die verfügbare *Anzahl von Trainingsdaten limitiert* ist.

Danksagung

CNNs, DNNs, Invariance
GPUs, TPUs und AISTATS
GPT, LLMs und MLP
FCE, VfB, olé olé
Frei nach „Die Fantastischen Vier - MFG“

Vielen Dank an meinen Betreuer PD Dr.-Ing. habil. Alexandru Paul Condurache für die hilfreiche und intensive Betreuung, vielfältige Tipps, Diskussionen und Unterstützung. Insbesondere gilt mein Dank dem häufig zeitintensiven und -kritischen „Paper-Pingpong“ im Vorlauf von Paper-Deadlines. Weiterhin gilt mein Dank Prof. Dr.-Ing. Alfred Mertins vom Institut für Signalverarbeitung der Universität zu Lübeck, Prof. Dr. rer. nat. Thomas Martinetz vom Institut für Neuro- und Bioinformatik der Universität zu Lübeck sowie meinen Bosch-Vorgesetzten und Kollegen André Treptow, Christian Connette, Benjamin Coors, Jasmin Ebert, Lukas Enderich, Steffen Hagedorn, Julia Lust & Paul Wimmer für das Vertrauen, Korrekturlesen, hilfreiche Diksussionen und die generelle Unterstützung.

Zuletzt möchte ich mich bei meiner Familie Sonja, Bernhard, Anni & Joni sowie meiner Freundin Julia bedanken. Danke für all eure Unterstützung während der Doktorandenzeit! Nicht vergessen möchte ich auch die Jungs vom 1.FC Egenhausen, die mir sowohl auf als auch abseits des Platzes willkommene Ablenkung spendiert haben, selbst unmittelbar vor Abgabefristen.

Table of Contents

Chapter 1

Introduction

1.1 Motivation for Enforcing Invariance in Deep Neural Networks

Algorithms based on *Artificial Intelligence (AI)* have produced remarkable results in recent years from early variants of self-driving vehicles [46], beating human experts in complex games such as Go or Chess [158], automatic protein folding [86] or rendering images from text prompts [138]. More recently, Large Language Models (LLMs) were able to compile human-level responses towards a wide range of prompts including cooking recipes, fictional song lyrics or mathematical equations [121].

From a high level perspective, most if not all of the presented recent breakthroughs are based on Deep Learning (DL), a subfield of Machine Learning (ML) and therefore AI, see Figure 1.1. Recent progress in DL relies on a common core recipe: training Deep Neural Networks (DNNs) on ever growing amounts of data and compute [60]. One important field driving the progress in DL is Computer Vision (CV), which this dissertation will mainly focus on.

Despite the rapid progress, AI has failed to live up to some of its promises, which granted might have been overinflated at times. *Autonomous Driving (AD)* is one example where AI systems have not yet achieved their envisioned potential. At the time of writing this thesis, fully self-driving cars still do not roam our streets.

One approach to solve AD uses *monolithic end-to-end algorithms* that learn driving actions directly from sensor data. This approach aligns all learned components towards the overall driving objective which promises an improved performance at the cost of a more complex optimization [77]. In comparison, the *sequential, modular approach* to AD separates the environment perception from follow-up tasks such as planning or executing actions and implements each sub-task individually. While dissecting the driving task into sub-tasks can lead to misalignment, it simplifies optimization. Furthermore, the modular approach promises improved interpretability and generalization capabilities [14].

Hence, the prevalent approach to AD in production is modular and heavily relies

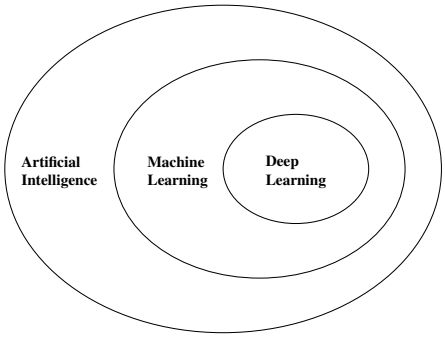


Figure 1.1: Deep Learning is a subfield of Machine Learning, which is in turn a subfield of Artificial Intelligence [60].

on prior knowledge manually built into the system. Within this setup, AI algorithms successfully handle parts of the algorithm chain implementing the driving task, such as detecting other traffic participants, road boundaries or traffic signs [46]. Nevertheless, even the existing (partially) self-driving cars only operate in limited application areas and need to be closely monitored by human drivers.

Another example for the current limitations are LLM-based chat bots. While their responses are indeed impressive and cover a wide range of expertise, LLMs at times misleadingly hallucinate knowledge and present it as facts. For example, they misleadingly assign non-existing contributions to researchers or fabricate facts when writing scientific articles [84].

In contrast to the human brain, DNNs are task-specific experts. A network that mastered the task of playing Go is unable to control a self-driving car and vice versa. Contrarily, the human brain often masters multiple tasks. General AI able to leverage in- and equivariance spanning all tasks still eludes us, despite recent advances sparked by LLM research.

In his position paper "A Path Towards Autonomous Machine Intelligence", Yann LeCun identified three major challenges that current DL systems fail to tackle [102].

- First, while current DL solutions excel at performing subconscious computations such as perceiving the environment or reading text, they have yet to learn to reason.
- Second, complex action sequences can not be planned because hierarchical representations of action plans can not be learned.
- Third, the currently used supervised and reinforcement learning algorithms *require* a *huge amount of training samples* to learn accurate and predictive models of the

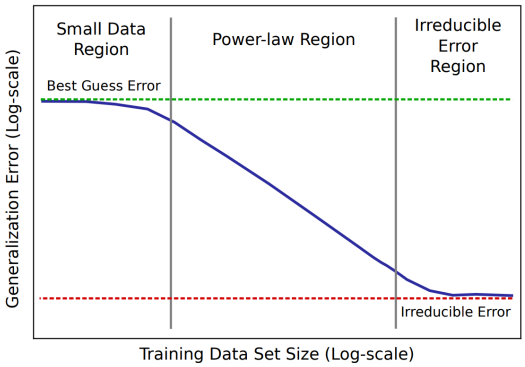


Figure 1.2: The power law relation between the generalization error and the amount of training data of supervised deep learning algorithms [71]. Modern Deep Learning algorithms mainly operate in the power law region. Hence, their performance benefits from large amounts of available training data.

world [102].

Other research provides further evidence that the performance of DL algorithms heavily depends on the amount of available data [71]. Specifically, a *power law* relates the generalization error to the amount of available training data for supervised DL algorithms, see Figure 1.2. Current DL systems are mostly trained within the power-law region. Consequently, they only achieve a satisfactory performance given enough training data. Recent progress in speech recognition and CV has mainly focused to boost the performance by further increasing the size of trained models and (pre-training) datasets [35,170]. Nonetheless, in practice it is often desirable to obtain a performant solution using only a limited amount of data.

Commonly used approaches to reduce the amount of required task-specific data are *self-supervised learning* or *transfer learning*, where the DNN is *pre-trained* on large amounts of unlabelled data from the training distribution itself or labelled data from an unrelated dataset, respectively [102]. Alternatively, *prior knowledge* can be *embedded* into the DNN. Even though those concepts can help mitigating the amount of required training data for a specific task [102], a sufficient amount of labelled training data from the target task is required to achieve an acceptable performance.

This is especially important for *real-world applications*, where the data first needs to be *collected*, *stored* and *labelled*. Especially labelling often involves tedious manual work

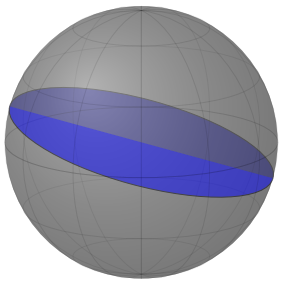


Figure 1.3: An exemplary three-dimensional solution space of a DL optimization algorithm (grey). Appropriately constraining the optimization problem by embedding prior knowledge reduces the solution space the learning algorithm has to cover to the blue, two-dimensional circle.

and is thus expensive. There are also known applications where the amount of available data for the design of ML solutions is limited, e.g., in the medical domain. Even assuming the availability of enough training data, optimizing DNNs on huge datasets requires considerable compute resources, which is both expensive and causes a non-negligible carbon footprint [128].

Due to all the above reasons, it is therefore an important challenge to *optimize* the *efficiency* of DNN training algorithms. Thus, efficient ML is concerned with obtaining a satisfactory performance when training data is scarce (*sample efficiency*) or when compute resources are limited (*compute efficiency*).

Improving the sample efficiency is particularly important from a practitioner's viewpoint. In many cases, problem-related data is scarce and needs to be carefully selected and annotated. This is in contrast to academic benchmarks where curated datasets are readily available and allow us to compare different algorithms and DNN architectures. A particularly promising direction to improve the sample efficiency of DNNs is to combine data-driven DL with *prior knowledge* about the problem, in order to reduce the search space the learning algorithm needs to parse, which is exemplarily visualized in Figure 1.3.

Combining prior knowledge with the architectures and optimization methods of DNNs is often called a *hybrid approach* that aims to combine the best of both worlds: state-of-the-art results by data-driven optimization and an improved performance when training data is limited. The embedded prior knowledge also leads to a more *explainable*, *robust* system behaviour through well-determined layers or intermediate representations, in contradiction to the commonly used approach of treating DNNs as a black box and

training them in an end-to-end fashion [107]. This simplifies the analysis of the DNN's behaviour in safety-critical cases such as robotics (including AD) or medical imaging.

Prior knowledge can be expressed in many different ways. A simple example in the context of object detection is the knowledge about the expected size of different objects in images. This is used for the anchor boxes in object detectors [137]. However, this kind of prior knowledge is hard to formalize which complicates incorporating it into DNNs in a principled manner. Another possibility is to model a prior distribution over the output of a DNN (e.g., via a Bayesian neural network) in order to predict a posterior distribution with the help of the likelihood [117].

One specific type of prior knowledge is called *geometrical prior knowledge*. It is the knowledge about certain *input transformations* that *change* the *output in a predictable way*, or not at all [174]. For CV tasks, geometrical prior knowledge covers transformations such as those that describe the change of the image context when a camera is moved to a novel viewpoint [23], or the transformations that objects can undergo in a scene. This includes the *symmetry transformations*, the set of all transformations acting on the input that leave an object unchanged, e.g., rotations, flips and translations (cf. Section 2.2.1) [17, 174]. Geometrical prior knowledge can be *built into DNN architectures* in a mathematically sound manner by *enforcing* representations that *guarantee in-* or *equivariance* to the corresponding transformation.

Equivariance is the property of a function or an operation[1] to behave predictably under transformations of its input, which will be defined in Section 2.2.2. A common example is the output function of the convolution operation, which combines two input functions into an output function, cf. Equation 2.3. A shift of any of the convolution's input functions induces a shift of the output function, thus resulting in *translation equivariance*.

Consequently, translation equivariance can be embedded into the learned features of DNNs by employing learnable layers based on convolutions. The resulting architecture is called Convolutional Neural Network (CNN) [54, 103]. Different DNN architectures are categorized based on the type of learnable layers they employ. For CV, well-known architecture types include CNNs based on convolutions, Multi-Layer Perceptrons (MLPs) based on fully connected (fc) layers [60, 144] and Transformers that employ learnable

[1] A *function* $f : \mathcal{X} \to \mathcal{Y}$ maps each value from the set $\mathcal{X}$ (the *domain*) to a single element in $\mathcal{Y}$ (the *codomain*). An *operation* on a set $\mathcal{X}$ combines one or more elements of $\mathcal{X}$ to yield another element within $\mathcal{X}$ [36].

self-attention (SA) [35, 170], see Section 2.1.1.

The increased sample efficiency gained through the translation equivariance of convolutions enables CNNs to outperform MLPs on important CV benchmarks [66, 96, 186]. While Vision Transformers (VITs) have recently achieved results that are at least on par to CNNs despite disregarding equivariance, they require large amounts of (pre-)training data and are thus less sample efficient [35, 170]. In addition, for the *practically relevant* use-case of *embedded systems*, strict resource and runtime restrictions favour the computationally cheaper convolution opposed to SA. Furthermore, current hardware is specialized on convolutions [63]. Due to the high practical relevance, we focus on CNNs in this work while drawing comparisons to Transformers where applicable.

The equivariant behaviour of convolutions can be extended to also include other *symmetry transformations* such as rotations or flips resulting in the *group-equivariant convolution (G-Conv)*, introduced in Section 2.3.1.1 [17, 173]. Slight modifications allow us to further extent these concepts to scales [162]. Employing G-Convs within DNN architectures to guarantee equivariance to rotations, flips, or scales in addition to translations increases the amount of built-in prior knowledge and thus further improves the sample efficiency [17, 162, 173]. In addition, equivariant DNNs simplify the system validation, as the responses towards the transformations with built-in equivariance are known a priori. Furthermore, the desired properties are mathematically guaranteed, not only approximated as, for example, with *data augmentation* [97].

For classification tasks, the class output should not change under input transformations that do not affect the desired prediction. Hence, representations *invariant* to those *intra-class variations* are required. The current state-of-the-art to impose invariant representations within a DNN architecture is to apply G-Convs followed by globally pooling over both the *group* as well as the spatial *domain* [17, 162, 173]. By *enforcing in- and equivariance* in a mathematically guaranteed manner *based on geometrical prior knowledge* rather than learning it from data, the solution space is successfully reduced, which regularizes learning and improves the sample efficiency. Nevertheless, while pooling guarantees invariance, it also discards information and is thus suboptimal. Therefore, replacing the pooling operation in invariant DNNs promises to further boost their sample efficiency [132–134].

Principally, the purpose of guaranteeing invariance can also be achieved by other means besides pooling. One promising example is *Invariant Integration (II)*, a method that

Figure 1.4: Max pooling (left) guarantees invariance, but only preserves a single value of the input space, effectively destroying the other available information. In contrast, adding *targeted model capacity* via an invariant extraction function $f_t(\mathbf{x})$ (right) is able to consider and weight all available information while still guaranteeing invariance.

computes invariant features based on the *group average* [78, 120, 148]. So far, II has mainly been used to extract invariant features directly from the input data which were processed by traditional machine learning classifiers [21, 114–116, 150].

In this dissertation, we instead propose a novel II layer that we apply on top of the equivariant representations learned by G-Convs within DNNs for the transition from equi- to invariant features. Thereby, *targeted model capacity* is added during the transfer instead of destroying information through pooling, as exemplarily visualized in Figure 1.4. Our proposed *II-enhanced architectures* yield deep neural feature spaces that can be optimized based on training data, as well as a well-defined map that mathematically guarantees invariance while adding targeted model capacity via II. Consequently, we further improve the sample efficiency of DNN architectures that leverage geometrical prior knowledge [132–134].

1.2 Contributions

In this thesis, we present the following contributions

1. We provide an overview of the related literature concerning DNNs enhanced with geometrical prior knowledge based on our survey paper *"Boosting Deep Neural Networks with Geometrical Prior Knowledge: A Survey"* [135].
2. We propose an *Invariant Integration Layer* based on the group average calculated with monomials. A small modification of the input values enables to use II within a DNN. Additionally, the backpropagation algorithm can be routed through the layer, allowing for an end-to-end optimization. We apply our novel layer to the *group of 2D rotations* and select the monomials using an iterative algorithm based on

the Least Square Error (LSE) of a linear classifier. Applying the II layer enforces transformation-invariance to the learned representations while adding targeted model capacity [132].

3. Subsequently, we *improve* the 2D rotation-*II layer*. First, we replace the iterative *monomial selection* with *pruning* methods to enable training our II-enhanced DNNs more natively in an end-to-end fashion. Second, we investigate *replacing the monomials* with well-known DNN functions such as SA, a MLP or a learnable weighted sum (WS). Both approaches allow to apply II within bigger networks and to real-world datasets. Our rotation-II enhanced networks outperform both standard as well as invariant CNNs without II, specifically in the limited data domain [133].
4. Finally, we expand II *beyond rotations* towards *flips and scales*. To achieve scale-invariance, we divide scale-homogeneous features obtained using translation-II. This highlights the generality of our approach and extends its applicability towards problems where more sophisticated invariances are required. We introduce a *multi-stream architecture* that *combines multiple invariances* by handling each invariance within a dedicated stream. A standard convolutional stream allows to cover features where the invariant streams are too strict or even misspecified. Combining multiple streams allows our network to automatically learn the best-fit invariance for the specific problem to solve. Our II-enhanced multi-stream networks thus outperform related equivariant CNNs in all data domains, while adding targeted model capacity [134].

1.3 Outline

In this section, we briefly summarize the further contents of this thesis.

Chapter 2 introduces the necessary background to follow our contributions. This includes a brief introduction to DNNs and why incorporating geometrical prior knowledge is beneficial (Section 2.1). Section 2.2 introduces the mathematical framework that forms the backbone of geometrical prior knowledge, mainly Group Theory. Section 2.3 presents state-of-the-art approaches to embed geometrical prior knowledge into DNN architectures and training algorithms based on our contribution [135]. Section 2.4 discusses the transfer from equivariant to invariant features. It then motivates, why replacing the commonly

used pooling approach with II is beneficial for state-of-the-art invariant classification DNNs. Finally, Invariant Integration itself is introduced.

Chapter 3 summarizes our contributions from reference [132]. Namely, it introduces the II layer – a novel DNN layer based on the group average that can be used within group-equivariant CNNs to obtain invariant representations. The group average is solved using monomials, which are simple functions whose parameters can be chosen solving an iterative algorithm. The novel layer is then applied to 2D rotations.

In Chapter 4, the II layer is further expanded in order to be applicable towards more real-world DNN architectures and tasks. Therefore, the iterative monomial selection is replaced with pruning-based approaches. In addition, we investigate variants of the II layer that entirely replace the monomials with well-established functions from DL literature such as a MLP, a WS or SA. This chapter summarizes reference [133].

Chapter 5 is based on reference [134], where the II layer is applied beyond rotations towards flips and scales. This allows to use II-enhanced DNNs within manifolds of applications, where multiple invariances improve the sample efficiency. A multi-stream architecture is introduced to combine these multiple invariances effectively. Thereby, the DNN is able to automatically learn the best-fit invariance for the problem to solve.

Lastly, Chapter 6 summarizes our contributions from previous chapters and gives a brief outlook on possible future research directions.

Chapter 2

Deep Learning with Geometrical Priors

In this chapter, we introduce the theoretical background underlying the contributions of this thesis. This includes the basics of DL and how geometrical prior knowledge can help amend some of its shortcomings (Section 2.1). Section 2.2 formally defines geometrical prior knowledge using the mathematical notion of groups. It also discusses, how to leverage this type of prior knowledge by enforcing in- or equivariance. Section 2.3 describes state-of-the-art methods that enhance DNNs with geometrical prior knowledge. Finally, we introduce II and present a path to improve the transfer from equi- to invariant features within DNNs by leveraging II (Section 2.4).

2.1 Boosting Deep Neural Networks with Geometrical Prior Knowledge

In this section we start by reviewing the basics of DNNs. We briefly introduce the two main components of DNNs: the *architecture* consisting of different *layers* (Section 2.1.1) and the *training recipe* based on optimizing *loss functions* via *gradient descent* by *backpropagation* (Section 2.1.2). Using the example of CNNs, we then show how adding prior knowledge helps the learning algorithm of DNNs to achieve satisfactory solutions (Section 2.1.3). Finally, we provide an intuition about how geometrical prior knowledge can be generalized beyond the convolutional layer (Section 2.1.3.1).

2.1.1 Deep Neural Network Architecture

The core ingredients of modern DNN architectures are *learnable layers*, which combine *linear matrix multiplications* with *non-linear, non-learnable activation functions*. This combination enables DNNs to model complex relationships [60].

Feed-forward DNNs, the focus of this dissertation, consist of multiple *layers* f_l for

$l = 1, \dots, L$ that are sequentially applied to compute the output $f(x)$

$$f(x) = f_L(f_{L-1}(\dots f_1(x)\dots)), \tag{2.1}$$

where the number of layers L is called the *depth* of the network. Intermediate layers, called *hidden layers*, return a *latent representation* of the data [60].

Each layer typically consists of a learnable component based on linear matrix multiplications followed by a non-linear activation function σ, which enables DNNs to learn intricate non-linear relations between the in- and outputs. As a result, DNNs can approximate any desired non-linear relation, given sufficient training data [60].

Different established DNN architectures are distinguished by the primary type of learnable layers they employ:

- fc layers are the building blocks of MLPs,
- convolutional layers are the core of CNNs and
- SA is the basis of Transformers.

These learnable layers will be presented in Sections 2.1.1.1 - 2.1.1.3.

Preliminary: Defining 2D Inputs as Matrices or Functions In the following, we will focus on image-like inputs $\mathbf{x}$ defined on the two-dimensional grid $\mathbb{Z}^2$ with height H and width W. These can be represented in two different variants depending on the context:

1. The matrix representation $\mathbf{x} \in \mathbb{R}^{H \times W}$, where each matrix contains $H \cdot W$ real-valued entries.
2. The function representation $\mathbf{x} : \mathbb{Z}^2 \to \mathbb{R}$, where $u \mapsto \mathbf{x}(u)$ assigns a value $\mathbf{x}(u) \in \mathbb{R}$ to each discrete position $u \in \mathbb{Z}^2$ in the two-dimensional grid.

As long as the input function is defined on the regular grid $\mathbb{Z}^2$, both representations are equivalent, allowing for an easy transition between them. However, the functional definition enables inputs to be defined on arbitrary input spaces beyond the constraints of grid-based matrices. For example, using $\mathbb{R}^2$ instead of $\mathbb{Z}^2$ allows to model continuous input spaces. Furthermore, it is essential for several applications including convolutions (Section 2.1.1.2), group theory (Section 2.2.1), the related work enforcing geometrical

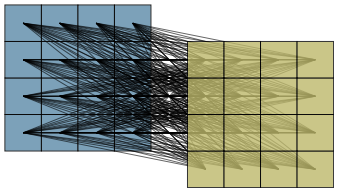


Figure 2.1: Illustration of a two-dimensional fc layer processing the input (blue) with learnable weights (black connections) to obtain the output (yellow). Every input is connected to all outputs, where each connection represents a unique learnable weight.

prior knowledge (Section 2.3) and our contributions (Chapters 3-5). If not obvious from the context, we will define the inputs accordingly throughout our thesis.

Additionally, inputs often consist of spatially aligned data organized into multiple *channels*. For example, an RGB image has $C = 3$ *input channels*, one for each colour. The matrix and function representation then become $\mathbf{x} \in \mathbb{R}^{H \times W \times C}$ and $\mathbf{x} : \mathbb{Z}^2 \to \mathbb{R}^C$, respectively. Since many concepts are applied independently to each channel, we occasionally omit the channel dimension to simplify our explanations.

2.1.1.1 The Fully-Connected Layer

The fc layer implements a linear matrix multiplication processing inputs $\mathbf{x} \in \mathbb{R}^n$ followed by adding a learnable bias term $\mathbf{b} \in \mathbb{R}^m$ and a *non-linear activation function* σ

$$f_l(\mathbf{x}) = \sigma(\mathbf{W}^T \mathbf{x} + b), \tag{2.2}$$

where $\mathbf{W} \in \mathbb{R}^{n \times m}$ is a learnable linear weight matrix connecting each input to every output. This design, based on the Neuron and Perceptron models [111, 144], uses the learnable parameters $\mathbf{W}$ and $\mathbf{b}$ to capture linear relationships, while the activation function σ introduces non-linearity, enabling the network to represent complex, non-linear patterns.

When a DNN only applies fc layers, it is called MLP [60]. Although fc layers can process multi-dimensional inputs by vectorizing them (e.g., $\text{vec} : \mathbb{R}^{H \times W \times C_i} \to \mathbb{R}^{HWC_i}$), this approach significantly increases the computational and parameter complexity. Particularly, the learnable weights $\mathbf{W} \in \mathbb{R}^{HWC_i \times HWC_o}$ and the bias $\mathbf{b} \in \mathbb{R}^{HWC_o}$ require $H^2W^2C_iC_o + HWC_o$ parameters [60].

Figure 2.1 illustrates the connections of a two-dimensional fc layer, where a feature map of size 4×4 already results in $4^2 \cdot 4^2 = 256$ learnable connections for $C_i = C_o = 1$

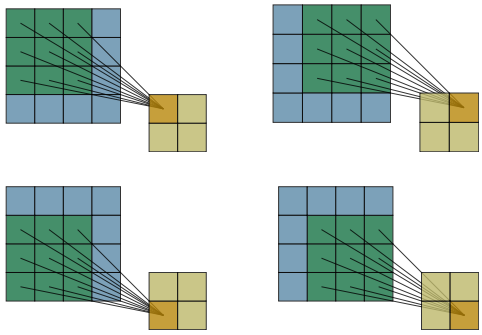


Figure 2.2: Illustration of a convolutional layer with kernel size 3×3. The kernel (green) is translated over all input locations (blue) to calculate the output (yellow). The weights (black connections) are shared among all input locations.

(excluding the bias). Hence, more parameter-efficient layers such as convolutions (cf. Section 2.1.1.2) are commonly preferred for multi-dimensional inputs.

2.1.1.2 The Convolutional Layer

Even though a DNN consisting of fc layers can learn arbitrary non-linear relations, it is beneficial to use layers with *structured connections* in order to reduce the amount of learnable parameters [60]. Thereby, the search-space the learning algorithm needs to explore is reduced. A well-known example are CNNs which replace the fc matrix multiplication with convolutions of the input with a learnable kernel ψ shared among all input locations [54, 103], as visualized in Figure 2.2.

In the following, we mathematically introduce convolutions and prove their *translation equivariance* property. The one-dimensional *continuous convolution* $(x * \psi)(t)$ is an operation defined on the input function $x : \mathbb{R} \to \mathbb{R}$ and the kernel function $\psi : \mathbb{R}^k \to \mathbb{R}$ with *kernel size* k [66, 103]

$$(x * \psi)(t) = \int x(\tau)\psi(t-\tau)d\tau. \tag{2.3}$$

In practice, DL frameworks commonly implement the *cross-correlation* $(x \star \psi)(t)$ rather than the convolution [1, 127]. Both definitions are equivalent up to a flipped kernel for real-valued inputs and kernels

$$(x \star \psi)(t) = \int x(\tau)\psi(\tau-t)d\tau = \int x(\tau)\psi(-(t-\tau))d\tau. \tag{2.4}$$

We follow the related literature and refer to the correlation function as convolution in the context of DNNs [1, 60].

For many practical applications such as CNNs for CV, the *discrete two-dimensional convolution* is applied [54, 103]. Here, the integral reduces to a discrete sum, while the functions are now defined as $\mathbf{x} : \mathbb{Z}^2 \to \mathbb{R}$ and $\psi : \mathbb{Z}^{k\times k} \to \mathbb{R}$, cf. Section 2.1.1

$$f_l(\mathbf{x})(u) = (\mathbf{x} \star \psi)(u) = \sum_v \mathbf{x}(v)\psi(v-u), \tag{2.5}$$

with computational complexity $\mathcal{O}(k^2HW)$. Intuitively, the convolution is a sum over all input elements within the kernel's spatial extent $k \times k$ weighted by the corresponding kernel element. The kernel is shifted over the entire input space $u \in U \subset \mathbb{Z}^2$ to compute the result at each point of the input.

Whereas fc layers learn an individual weight connecting each input to every output (*global connectivity*), convolutions apply the weights locally within the neighbourhood defined by the kernel (*local connectivity*)[1]. Furthermore, by shifting the kernel over the input, the weights are shared among all locations, which is called *translational weight tying*, cf. Figure 2.2. Consequently, convolutions can be viewed as a linear matrix multiplication with shared weights.

Due to the local connectivity and weight sharing, convolutional layers require significantly less parameters compared to fc layers. For a kernel of size $k \times k$, convolutions use k^2 parameters in comparison to the H^2W^2 parameters of a fc layer, where $k \ll H, W$. The difference of learnable weights is visually highlighted when comparing the connections in Figures 2.1 and 2.2, where the convolutional layer with kernel size $k = 3$ has 9 learnable weights compared to the 256 learnable weights of the fc layer.

Sharing the weights among all input positions enables CNNs to recognize patterns independently of their location. Additionally, if an input pattern is shifted, the corresponding output is shifted as well. Therefore, CNNs are *equivariant* to *translations* [54, 60, 103].

Theorem 2.1.1. *Given an input function $\mathbf{x} : \mathbb{Z}^2 \to \mathbb{R}$, a convolutional kernel $\psi : \mathbb{Z}^{k\times k} \to \mathbb{R}$ and the translation operator $\mathcal{T}_t$ acting on the input via $\mathcal{T}_t\,\mathbf{x}(u) = \mathbf{x}(u-t)$ with $u, t \in \mathbb{Z}^2$.*

Convolutions are translation-equivariant $(\mathcal{T}_t\,\mathbf{x} \star \psi)(u) = \mathcal{T}_t(\mathbf{x} \star \psi)(u)$. A shift $\mathcal{T}_t$ of the input leads to a mathematically guaranteed shift of the output.

[1] Convolutions may exhibit global connectivity by using a kernel of size $H \times W$.

Proof.

$$(\mathcal{T}_t \mathbf{x} \star \psi)(u) = \sum_v \mathbf{x}(v-t)\psi(v-u) \tag{2.6}$$

$$= \sum_v \mathbf{x}(v)\psi(v-u+t) \tag{2.7}$$

$$= \sum_v \mathbf{x}(v)\psi(v-(u-t)) \tag{2.8}$$

$$= (\mathbf{x} \star \psi)(u-t) = \mathcal{T}_t(\mathbf{x} \star \psi)(u). \tag{2.9}$$

□

2.1.1.3 Visual Self-Attention

Recently, *Transformers*, which utilize *self-attention* as their core learnable component, have achieved remarkable results when scaling up the model size and the amount of training data of DNNs [35, 170].

Attention is a non-linear, learnable operation that solves an information retrieval problem by computing *attention scores* $\mathbf{A}$ based on the *similarity* between *queries* $\mathbf{Q}$ and *keys* $\mathbf{K}$ used to weigh and extract the *values* $\mathbf{V}$. Using the *dot-product attention* score $\mathbf{A} = \text{softmax}(\mathbf{Q}\mathbf{K}^T)$, attention is defined as [170]

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}(\mathbf{Q}\mathbf{K}^T)\mathbf{V}. \tag{2.10}$$

The attention scores $\mathbf{A}$ explicitly encode, which values $\mathbf{V}$ the layer attends to [170]. In the case of SA, the queries $\mathbf{Q}$, keys $\mathbf{K}$ and values $\mathbf{V}$ are all extracted from the input $\mathbf{x}$ itself. Consequently, SA computes the attention scores by relating the input to itself. In contrast, the more general attention mechanism can also adjust the weights based on other input maps, such as output values from other DNN layers. SA and Transformers were initially introduced to DNNs in the context of machine translation and LLMs [170]. They have since been adapted for processing 2D images $\mathbf{x} \in \mathbb{Z}^{H \times W \times C_i}$, resulting in *visual SA* [35].

Visual SA defines the input pixels as $T = H \cdot W$ individual *tokens* and reshapes the input to $\tilde{\mathbf{x}} \in \mathbb{R}^{T \times C_i}$. The attention scores $\mathbf{A} \in \mathbb{R}^{T \times T}$ are computed based on three learnable matrix multiplications with the value matrix $\mathbf{W}_V \in \mathbb{R}^{C_i \times C_h}$, the key matrix $\mathbf{W}_K \in \mathbb{R}^{C_i \times C_h}$

and the query matrix $\mathbf{W}_Q \in \mathbb{R}^{C_i \times C_h}$ [35]

$$\mathrm{SA}(\mathbf{x}) = \mathrm{softmax}(\mathbf{A})\tilde{\mathbf{x}}\mathbf{W}_V \text{ with } \mathbf{A} = \tilde{\mathbf{x}}\mathbf{W}_Q(\tilde{\mathbf{x}}\mathbf{W}_K)^T. \tag{2.11}$$

Expansions to visual SA include adding *positional encodings* to the input tokens, allowing the model to learn spatial relationships between them [155]. Moreover, multi-head self-attention (MH-SA) computes multiple SA layers in parallel and combines the outputs using a further learnable matrix multiplication [35, 170].

To reduce the computational complexity, modern VITs compute MH-SA based on non-overlapping image patches of size $k \times k$ instead of per input pixel [35]. This results in an input $\mathbf{x}_P \in \mathbb{R}^{T \times k^2 C_i}$ with $T = \frac{HW}{k^2}$ flattened patches of size $k^2 C_i$. Thereby, the computational complexity required to calculate the attention score reduces from $\mathcal{O}((HW)^2)$ to $\mathcal{O}((\frac{HW}{k^2})^2)$, which enables using SA within large-scale Transformers for vision tasks [35].

Comparing Transformers to CNNs In contrast to convolutions that act within a local kernel size and can only learn global relations gradually with network depth [66, 103], SA provides a global field-of-view. Here, the spatial relations between the input tokens are entirely learned rather than hardcoded a priori [35]. SA enables the model to *attend to* any part of the input by dynamically adjusting the attention weights based on the input itself [35, 170]. Specifically, SA calculates *input-dependent weights* $w(\mathbf{x}) = \mathrm{softmax}(\mathbf{A})$ resulting in the output $\mathbf{y} = w(\mathbf{x})\mathbf{x}\mathbf{W}_v$. In contrast, fc layers and convolutions only update their weights during network training using Stochastic Gradient Descent (SGD) (cf. Section 2.1.2), but do not exhibit any input dependence. Instead, their output is given by $\mathbf{y} = \mathbf{W}^T\mathbf{x}$, where the weights $\mathbf{W}$ remain fixed during inference.

While SA is more flexible compared to convolutions, it encodes less prior knowledge. Thus, SA layers are successful in the large data domain but often require enormous amount of pre-training data to outperform their convolutional counterparts. For example, a VIT needs to be pre-trained on the ImageNet-21k dataset containing 15 million images with 21k classes to reach the performance of a CNN on ImageNet [35]. Furthermore, the computational complexity of convolutions only linearly depends on the input size (cf. Equation 2.5). In combination with highly optimized embedded framework implementations, convolutions are faster than SA on embedded hardware and thus often preferred for solving real-time CV problems [63].

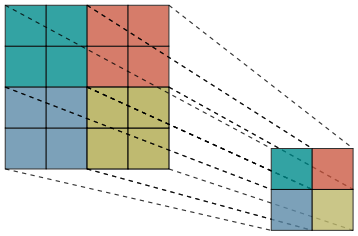


Figure 2.3: Illustration of a pooling operation with kernel size and stride 2×2.

As far as this contribution is concerned, our research focuses on improving the sample efficiency and obtaining good results in the limited data domain for real-world embedded CV models. In contrast, Transformers achieve remarkable scaling effects in domains where both large datasets and compute resources are available.

2.1.1.4 Other Common Layers

Pooling *Pooling* layers are non-learnable layers that gradually reduce the spatial extent of two-dimensional representations in DNNs. Thereby pooling significantly reduces the memory requirements and increases the *receptive field* of subsequent DNN layers, allowing the network to learn more global features with depth. Pooling extracts a value from a specified neighbourhood of size $k \times k$ via an *aggregation function*, e.g. the maximum or the average. The output size is then reduced by shifting the neighbourhood over the entire input with stride s, where $s = k$ is commonly used. Applied in this way, pooling introduces *invariance* to *small translations* [60, 188]. When applied globally to equivariant representations, pooling can also achieve invariance with respect to (w.r.t.) other transformations, such as rotations, see Section 2.4. Figure 2.3 illustrates an example of applying two-dimensional pooling to an input $\mathbf{x} \in \mathbb{Z}^{H \times W}$.

For multi-channel inputs, pooling is applied independently per channel [60]. Practitioners commonly prefer max pooling over average pooling. While it is slightly less well-defined, specifically w.r.t. its gradient flow (cf. Section 2.1.2), it empirically achieves a better performance [66, 96].

Batch Normalization *Batch Normalization* normalizes the activations of a layer based on the mean μ and standard deviation σ aggregated over all samples processed in a training *batch* (see Section 2.1.2) with a small numerical constant $\varepsilon \ll 1$ preventing

division by 0 [81]

$$\tilde{\mathbf{x}} = \frac{\mathbf{x} - \mu}{\sigma + \varepsilon}. \tag{2.12}$$

Batch Normalization approximately normalizes the value range the following layer needs to process towards zero mean and unit standard deviation. This stabilizes the gradient flow throughout the DNN during network optimization [81], which in turn speeds up the training as stable gradients permit using larger learning rates (cf. Section 2.1.2) [60]. To circumvent cases where zero mean and unit standard deviation are too restrictive, Batch Normalization includes learnable parameters that allow to scale and shift the outputs accordingly. During inference, where no batch dimension is available, the mean and standard deviation are replaced with an exponential moving average computed during training [81].

2.1.2 Deep Neural Network Training

In Section 2.1.1, we presented layers based on learnable linear matrix multiplications and non-linear, non-learnable functions as the basic building blocks of DNNs. However, we did not yet discuss how DNNs can *learn*, i.e., how their *weights* are *optimized* based on training data. This is commonly achieved by minimizing a *loss function* using *gradient descent* and *backpropagation*.

Optimizing the Loss with Gradient Descent Loss functions $\mathcal{L}(\mathbf{y}, f(\mathbf{x};\theta))$ quantify the error between a DNN's prediction $\hat{\mathbf{y}} = f(\mathbf{x};\theta)$ where $\theta = \{\mathbf{W}_1, \mathbf{b}_1, \ldots, \mathbf{W}_L, \mathbf{b}_L\}$ contains all learnable parameters, and the *target* or *ground truth* $\mathbf{y}$. Thereby, the loss measures how well the DNN performs. Common loss functions include the *mean-squared error* for continuous targets (*regression*) and the *cross-entropy loss* for categorical targets (*classification*) [60].

To train DNNs, the parameters θ are initialized randomly and optimized by minimizing the loss [60]

$$\min_{\theta} \mathcal{L}(\mathbf{y}, f(\mathbf{x};\theta)). \tag{2.13}$$

Typically, this is done using the *Gradient Descent* algorithm. Gradient descent minimizes the loss sequentially by updating the parameters following the direction of steepest descent represented by the gradient with *learning rate* or *step size* η [60]. For a single

weight $\mathbf{W}_l$, the update equation from step t to $t+1$ is

$$\mathbf{W}_l^{t+1} = \mathbf{W}_l^t - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{W}_l^t}. \tag{2.14}$$

For a DNN, the gradient of the loss w.r.t. each individual parameter needs to be calculated

$$\nabla_\theta \mathcal{L} = (\frac{\partial \mathcal{L}}{\partial \mathbf{W}_1}, \frac{\partial \mathcal{L}}{\partial \mathbf{b}_1}, \ldots, \frac{\partial \mathcal{L}}{\partial \mathbf{W}_L}, \frac{\partial \mathcal{L}}{\partial \mathbf{b}_L}). \tag{2.15}$$

Therefore, each network layer including its non-linearities needs to be *differentiable almost everywhere* [60]. The loss and its gradients are computed using a *training dataset* $\mathcal{D}_{\text{train}} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^N$ that consists of input-output combinations. In theory, it is possible to compute all gradients explicitly based on the entire training data at once. In practice, however, the available compute memory limits the amount of training data that can be processed at once. This problem can be avoided by using the SGD algorithm [60].

SGD iteratively calculates the gradients and weight updates using subsets of the training data $\mathcal{D}_{\text{train}}$ called *batches* rather than on the full dataset. This approach allows to effectively train a DNN given limited compute capabilities by avoiding to compute all network responses and gradients simultaneously [60]. The computation requirements can be controlled based on the available hardware via the *batch size*. Consequently, SGD is commonly preferred over standard gradient descent for DNN optimization [60, 117]. Advanced optimizers can be used to improve the stability and efficiency of training. *Adam*, for example, expands SGD via an adaptive learning rate for each parameter based on the moving average of its gradients and their squared values [89].

Backpropagation Even when using SGD, computing all gradients throughout a DNN is a likewise demanding task, particularly for DNNs with many layers. Calculating all gradients at once (as defined by Equation 2.15) is both analytically implausible as well as computationally expensive [60]. Fortunately, the *chain rule* of differentiation $\frac{\partial f(g(x))}{\partial x} = \frac{\partial f(g(x))}{\partial g(x)} \frac{\partial g(x)}{\partial x}$ provides an alternative path towards iteratively computing gradients throughout the network [60, 117].

In DNNs, the chain rule reveals that the gradient of each layer depends only on the gradients of the layer connected to its outputs and the gradient of the layer itself. Consequently, the gradients can be propagated throughout the network in reverse order,

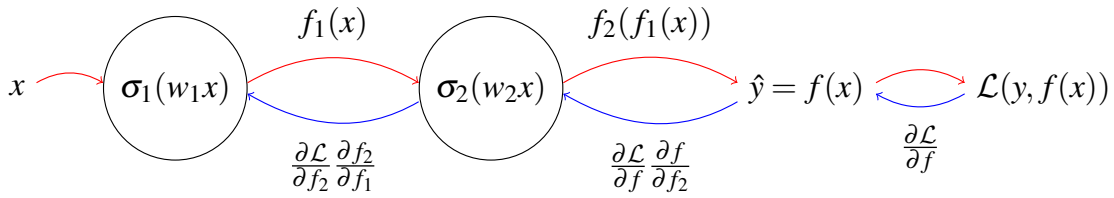


Figure 2.4: Computational graph of a DNN with $f(x) = \sigma_2(w_2\sigma_1(w_1x))$. The *forward pass* (red) iteratively computes latent representations, predictions and losses throughout the DNN, while the *backward pass* (blue) is used to compute and propagate the gradients required for optimization iteratively in reverse direction.

starting at the output and traversing towards the input [60]. This is known as *backward pass* and illustrated in blue in Figure 2.4. The overall algorithm for computing all gradients in this manner is called *backpropagation*. Similarly, computing the network responses $f_l(x)$ based on the input data is called *forward pass* (red in Figure 2.4) [60, 117].

The backpropagation algorithm avoids storing all gradients at once, since only the necessary gradients for each layer are stored during the backward pass. This significantly reduces memory usage. In contrast, computing all gradients simultaneously would require storing intermediate results for the entire network.

Moreover, the backward *flow of gradients* highlights important design decisions for DNNs. If gradients shrink towards 0 (*vanishing gradients*), early network layers fail to learn. If gradients grow towards ∞ (*exploding gradients*), training becomes unstable. These issues are closely tied to the choice of non-linearities, which need to provide stable gradients [60]. Additional layers such as Batch Normalization can help mitigate these issues (cf. Section 2.1.1.4). Together, SGD and backpropagation form the foundation of efficient DNN training [60].

2.1.3 Embedding Geometrical Prior Knowledge in DNNs

Based on the example of CNNs, we will now highlight how DNNs benefit from embedding geometrical prior knowledge. As shown in Section 2.1.1.2, the learnable convolutions used in CNNs mathematically guarantee *translation equivariance*. The equivariance property is directly related to the structure embedded into the convolutional layer: sharing the learnable network weights among all translational locations of the input induces translation equivariance [54, 103]. Hence, convolutions provide a principled approach to

build prior knowledge about transformations that affect the desired output in a predictable way into DNN architectures [54, 60, 103].

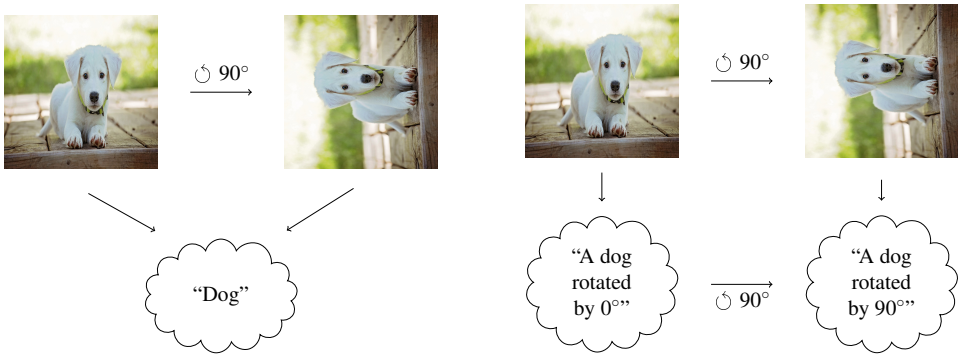
Translation equivariance is an important *inductive bias* that allows CNNs to outperform MLPs on relevant CV benchmarks [66, 96, 103]. Intuitively, the large number of DNN parameters creates a high-dimensional solution space, making it challenging for the learning algorithm to find an optimal solution. Adding problem-specific, reasonable constraints, e.g., the structured connections built-in via convolutions, greatly reduces the dimensionality of the solution space, cf. Figure 1.3. By enforcing translation equivariance via convolutions, CNNs no longer need to learn this property from data, but can focus their model capacity on learning other relevant features. This directly improves the *sample efficiency* of the learning algorithm, which results in a better performance given limited training data [17, 54, 103].

In addition to a better sample efficiency, the built-in prior knowledge promises to mitigate further disadvantages of modern DNNs, specifically when they are applied in safety-critical fields such as the medical domain or AD. Here, DNNs need to predict in an explainable and robust manner such that their predictive capabilities meet the safety-relevant validation criteria. Rather than treating the DNN as a black box, prior knowledge built in via in- or equivariance provides exact predictability of the desired outputs under the respective input transformation [17].

2.1.3.1 Generalizing Geometrical Prior Knowledge beyond Translations

While geometrical prior knowledge generally exhibits the benefits mentioned above, the standard convolution merely provides a way to embed it for 2D translations. Naturally, finding a principled methodology such that DNNs cover prior knowledge about symmetry transformations *beyond translations* is desired. Therefore, the core principle of how convolutions embed geometrical prior knowledge can be expanded to other transformations by forcing the output or learned features of DNNs to be *in- or equivariant* [174].

Let us first define in- and equivariance. A function is *invariant* w.r.t. a transformation, if its output does not change under transformations of the input. Comparably, the function is *equivariant*, if input transformations induce predictable transformations in the output space. This concept can be formally defined using the mathematical notion of a *group* that covers translations as well as further transformations such as rotations or flips [145, 174].



(a) Invariance (b) Equivariance

Figure 2.5: Visual example of in- and equivariant predictions. This illustration incorporates a license-free image of a dog [130].

Figure 2.5 exemplarily shows the difference between a rotation-invariant and a rotation-equivariant prediction. The mathematical definitions of in- and equivariance w.r.t. general groups will be introduced in Section 2.2.2.

While enforcing in- or equivariance provides the means to embed geometrical prior knowledge, we still need to introduce the precise methods to actually enforce the desired properties. One straight-forward, commonly used approach to obtain a DNN that *approximates* in- or equivariance to input transformations is *data augmentation*, see Section 2.3.2.1. Here, the DNN inputs and targets are randomly transformed during training [60]. While data augmentation is flexible and straightforward to implement, the DNN only learns to approximate the desired in- or equivariance properties. Consequently, the prior knowledge is not incorporated in a mathematically guaranteed way [17, 174].

Nevertheless, multiple approaches mathematically *guarantee equivariance* similarly to convolutions for the case of two-dimensional translations. Importantly, convolutions can be generalized to enforce equivariance towards more sophisticated transformations, such as scales, flips or rotations. This generalized function is called *group-equivariant convolution* and will be introduced in detail in Section 2.3.1.1 [17, 174]. Further methods that leverage geometrical prior knowledge either approximately or strictly will be introduced in Section 2.3.

Invariant features are desirable for classification, where the predicted output should not change, when the input is transformed, cf. Figure 2.5a. The common way to obtain *invariant* features within DNNs is to learn equivariant representations which in turn are

reduced to be invariant, e.g., via *pooling*. Thereby, the early learnable layers can still leverage the full equivariance properties to detect common patterns such as edges and corners in arbitrary orientations. At the same time, the deeper network layers benefit from the more restrictive invariant features that simplify the final prediction task [17, 174].

Two properties are important for the transfer function from equi- to invariance. First, the function needs to guarantee the desired invariance. Second, the transfer function needs to preserve or even enhance the *separability* of the feature space that is used for the final prediction task [17, 149]. As an illustrative example, simply setting all values to 0 would achieve invariance, albeit at the cost of entirely destroying the separability. Preserving or enhancing the separability while guaranteeing invariance is exactly where our thesis aims to improve upon the commonly used pooling operations by adding *targeted model capacity*, see Figure 1.4. For the purpose of enforcing invariance within DNNs, we introduce II on top of learned equivariant features (cf. Section 2.4 and Chapters 3 - 5).

To summarize, geometrical prior knowledge can be embedded to DNNs by forcing the learned representations and outputs to be *in-* or *equivariant* w.r.t. the known symmetry transformations. This principle can be expanded beyond the translations covered by standard convolutions. The most common way to obtain guaranteed equivariant representations are G-Convs, that generalize the convolution towards transformation groups. For classification tasks, the desired invariance is enforced by learning equivariant representations, which are reduced to an invariant representation via pooling. Depending on the problem to solve, in- and equivariant representations promise an improved network interpretability, as the responses to input transformations are mathematically well defined. Additionally, the sample efficiency of the learning algorithm is improved via the embedded structure of the network which reduces the search space the learning algorithm needs to explore. Consequently, adding geometrical prior knowledge improves the stability and efficiency of DNN training.

2.2 Formalizing Geometrical Prior Knowledge using Group Theory

In this section, we introduce the mathematical concepts fundamental to understanding both the related work that incorporates geometric priors into DNNs (Section 2.3) and the core contributions of this dissertation (Chapters 3 - 5).

We begin by *formally defining geometric prior knowledge* related to symmetry transformations. To this end, we introduce the mathematical concept of a *group* and explore how groups *act* on arbitrary sets X, i.e., how group elements transform those sets (Section 2.2.1). This foundation enables us to *extend geometric priors* to transformations *beyond translations*. Based on groups and group actions, we formally define *in- and equivariance w.r.t. transformation groups*, establishing a clear pathway towards incorporating geometric priors with rigorous mathematical guarantees (Section 2.2.2). We present examples of 2D transformation groups relevant to our contributions (Section 2.2.3). Finally, we explain how *groups act on the features* of equivariant CNNs through different *group representations*, which describe the linear actions of groups on *vector spaces*. The choice of representation imposes design constraints on these vector spaces, determining how equivariant features are encoded and transformed (Section 2.2.4). To clarify these definitions, we use the group of 90° rotations on a two-dimensional grid as a guiding example. Finally, Section 2.2.5 briefly presents *transformation-steerable filters*.

We focus on the definitions and properties essential to follow our contributions. Particularly, we restrict our explanations to real-valued grids. Further, we will focus on two-dimensional inputs and transformations to simplify our explanations. Additional definitions referenced in the related work (Section 2.3) are provided in Appendix A. We direct interested readers to references [64, 145, 174] for a comprehensive foundation in group theory and group representation theory beyond the scope of this thesis.

2.2.1 Transformation Groups

As introduced in Section 1.1, the set of all transformations that leave an object unchanged is called *symmetry transformations*. This includes geometrical transformations such as translations, rotations or reflections. The set of all symmetry transformations equipped with composition forms a *group* [174].

Definition 2.2.1. *A **group** consists of a set of elements G and a group operation $\circ$, which combines two elements of the group to form a third $a \circ b = c$ with $a, b, c \in G$. A group fulfils the following axioms [145]*

1. *Closure: The result of combining any two group elements via the group operation is part of the group $a \circ b \in G \quad \forall\, a, b \in G$.*

2. *Associativity: The order in which subsequent group operations are performed does not matter* $(a \circ b) \circ c = a \circ (b \circ c) \quad \forall\, a,b,c \in G$.
3. *Identity: The group contains an identity element* $e \in G$ *such that* $a \circ e = e \circ a = a \;\; \forall\, a \in G$.
4. *Invertibility: For each group element, the corresponding inverse element is part of the group, such that the group operation results in the identity* $\forall\, a \in G \;\; \exists\, a^{-1} \in G$ *s.t.* $a \circ a^{-1} = e$.

By convention, groups are commonly denoted by their set G and the group operation is omitted $a \circ b = ab$ [174]. We will follow these conventions throughout this dissertation. In the case of transformation groups, each group element $g \in G$ represents a specific transformation from the set, defined by particular parameters.

Definition 2.2.2. *The* ***group order*** $|G|$ *is the number of elements contained in the set G. Groups containing a finite number* $|G| < \infty$ *of elements are called* ***finite groups*** *[145].*

Definition 2.2.3. *A* ***subgroup****, denoted* $H \subset G$*, is any non-empty subset of the group that itself fulfils the group axioms. Thus, any subgroup needs to contain the inverse* $h, h^{-1} \in H$*, the identity* $e \in H$ *and the composition* $hk \in H \;\; \forall\, h,k \in H$ *[145].*

A group subset that does not contain the identity element, the inverse of each element or every combination of any two of its elements does consequently not form a subgroup.

Group Actions For our work, it is particularly important to understand how transformation groups affect the inputs and representations of DNNs. *Group actions* provide a formal definition of how group elements $g \in G$ act on arbitrary sets X, thus offering a framework for understanding how transformations *act on* the *inputs and representations*.

Definition 2.2.4. *Let G be a group and X be a set. A* ***group action*** $\mathfrak{T}$ *is a function*

$$\mathfrak{T} : G \times X \to X, \quad \textit{denoted by} \quad (g,x) \mapsto \mathfrak{T}(g,x), \tag{2.16}$$

that satisfies the axioms of associativity and identity. Particularly, the ***left group action*** *acts on the set from the left*

$$G \times X \to X, \quad \textit{denoted by} \quad (g,x) \mapsto \mathcal{T}_g x. \tag{2.17}$$

For the left group action, the axioms are given by [145, 174]

- *associativity:* $\mathcal{T}_g(\mathcal{T}_h x) = \mathcal{T}_{gh} x \quad \forall\, g,h \in G,\ x \in X$ *and*
- *identity:* $\mathcal{T}_e x = x \quad \forall\, x \in X.$

The *right group action* $\tilde{\mathcal{T}}_g$ can be defined analogously with the group acting on the input from the right $G \times X \to X$, denoted by $(g,x) \mapsto x\tilde{\mathcal{T}}_g$. Any right group action can be transformed into a left group action by defining $\mathcal{T}_g x = x\tilde{\mathcal{T}}_{g^{-1}}$ [145]. We will thus focus on left group actions without the loss of generality.

2.2.2 In- & Equivariance

Group Theory is the basis to mathematically describe geometrical symmetries. It is used to formally define *in- and equivariance* w.r.t. transformation groups [174].

Definition 2.2.5. *Let $f : X \to Y$ be a function and G be a transformation group acting on the input set X via a left group action $G \times X \to X$, $(g,x) \mapsto \mathcal{T}_g x$. Similarly, another left group action $G \times Y \to Y$, $(g, f(x)) \mapsto \mathcal{T}'_g f(x)$ acts on the output set Y.*
f is ***equivariant*** *w.r.t. G, if there exists an explicit relationship between transformations $\mathcal{T}_g$ of the function's input and the corresponding transformation $\mathcal{T}'_g$ of its output.*

$$f(\mathcal{T}_g x) = \mathcal{T}'_g f(x) \quad \forall\, g \in G,\ x \in X. \tag{2.18}$$

Intuitively, this means that transforming the input with $\mathcal{T}_g$ and applying f is equivalent to applying f on the untransformed input and transforming the result with $\mathcal{T}'_g$. It is important to note, that $\mathcal{T}_g$ and $\mathcal{T}'_g$ do not need to be the same transformation. For example, rotating the input might induce a shift in the output space rather than a rotation [17, 174].

Definition 2.2.6. *A function $f : X \to Y$ is* ***invariant*** *w.r.t. G, if its output is unaffected by input transformations, i.e., the group action in the output space $\mathcal{T}'_g$ is the identity $\mathbb{1}$.*

$$f(\mathcal{T}_g x) = \mathbb{1}[f(x)] = f(x) \quad \forall\, g \in G,\ x \in X. \tag{2.19}$$

In short, invariance is a special case of equivariance, where the output is unaffected by input transformations $g \in G$ [174].

(a) Translation (b) 90° rotation

Figure 2.6: The translation group $T(2)$ and the group of 90° rotations $\mathrm{SO}(2,4)$ acting on two-dimensional inputs can be expressed as a permutation p of the pixels.

A common simplified notation replaces both left group actions $\mathcal{T}_g$ and $\mathcal{T}'_g$ with the group element $g \in G$ [174]. The definitions of equi- and invariance then become

$$f(gx) = gf(x) \quad \forall\, g \in G,\, x \in X, \tag{2.20}$$

and

$$f(gx) = f(x) \quad \forall\, g \in G,\, x \in X, \tag{2.21}$$

respectively.

We have now formally defined in- and equivariance, which allows us to enforce geometric prior knowledge in a principled manner, as discussed in Section 2.1.3.1. Methods that impose in- or equivariance to DNNs will be introduced in Section 2.3.

2.2.3 Examples for Transformation Groups

So far, we have introduced groups as an abstract algebraic concept that acts on arbitrary sets. In this section, we will provide illustrative examples of transformations acting on two-dimensional inputs $\mathbf{x} : \mathbb{R}^2 \to \mathbb{R}$, using the functional definition of inputs introduced in Section 2.1.1. We first provide examples for transformation groups, followed by two examples for transformations that violate at least one of the group axioms.

Example 1: The Group of Translations $\mathrm{T}(2)$ A simple example for a transformation group are *translations*, where the set $\mathbb{Z}^2$ is combined via the group action of addition $\mathbf{a} \circ \mathbf{b} = \mathbf{a} + \mathbf{b}$. The group is closed, as $\mathbf{a} + \mathbf{b} \in \mathbb{Z}^2$. The identity element of the group is $\mathbf{e} = (0,0)^T = \mathbf{0} \in \mathbb{Z}^2$ and the inverse element is $\mathbf{a}^{-1} = -\mathbf{a}$ [145]. An example for the translation group acting on an image is shown in Figure 2.6a. For T(2), the group action

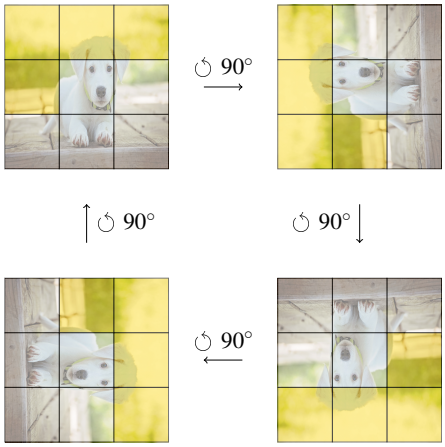


Figure 2.7: The group $\mathrm{SO}(2,4)$ includes all 90° rotations of a two-dimensional grid.

$\mathcal{T}_t$ on a two-dimensional input $\mathbf{x} : \mathbb{Z}^2 \to \mathbb{R}$ is

$$\mathcal{T}_t \mathbf{x}(u) = \mathbf{x}(u-t) \quad \text{with} \quad u,t \in \mathbb{Z}^2. \tag{2.22}$$

Example 2: The General Linear Group $\mathrm{GL}(\mathrm{n},\mathbb{R})$ The *general linear group* $\mathrm{GL}(n,\mathbb{R})$ consists of all real-valued, invertible $n \times n$-matrices $\mathbf{A} \in \mathbb{R}^{n \times n}$ and the matrix multiplication $\mathbf{A} \circ \mathbf{B} = \mathbf{AB}$ as the group operation. The inverse element is $\mathbf{A}^{-1}$ with $\mathbf{A}\mathbf{A}^{-1} = \mathbf{I}$. Here, $\mathbf{I}$ is the $n \times n$ identity matrix and also represents the identity element [145]. The group action $\mathcal{T}_\mathbf{A}$ of $\mathrm{GL}(2,\mathbb{R})$ on a two-dimensional input $\mathbf{x} : \mathbb{R}^2 \to \mathbb{R}$ is

$$\mathcal{T}_\mathbf{A} \mathbf{x}(u) = \mathbf{x}(\mathbf{A}^{-1}u) \tag{2.23}$$

The general linear group covers all transformation that act on the input via matrix multiplication. This includes many important subgroups such as rotations and flips.

Example 3: The Group of Rotations $\mathrm{SO}(2)$ The set of two-dimensional rotations $\phi \in [0, 2\pi)$ forms the *special orthogonal group* $\mathrm{SO}(2)$ [145, 174]. The identity element e is a rotation by 0, while the inverse $a^{-1} = -a$ is a rotation of the same angle in the opposite direction. The group operation $\circ$ combines two angles via $a \circ b = (a+b) \bmod 2\pi$, ensuring the result remains within the interval $[0, 2\pi)$. For example, $(\pi + \pi) \bmod 2\pi = 0$. Since the set of rotations is bounded, i.e., restricted to a fixed interval, SO(2) is a *compact group*. In contrast, translations are not compact, as they can take arbitrarily large values.

SO(2) *acts* on a two-dimensional input $\mathbf{x} : \mathbb{R}^2 \to \mathbb{R}$ via [145, 174]

$$\mathcal{T}_\phi \mathbf{x}(u) = \mathbf{x}(\mathbf{R}^{-1}u) \quad \text{with} \quad \mathbf{R} = \begin{pmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{pmatrix}. \tag{2.24}$$

Finite subgroups with N rotations in $[0, \frac{2\pi}{N}, \ldots, \frac{(N-1)2\pi}{N}]$ are denoted as SO$(2,N)$. For example, SO$(2,4)$ consisting of 90° rotations, is illustrated in Figure 2.7.

Example 4: The Group of Rotations, Translations and Flips E(2) The *Euclidean group* E(2) covers all two-dimensional transformations that preserve the Euclidean distance, which includes translations, rotations and flips. Each group element is a tuple $(\mathbf{t}, \mathbf{O})$ that contains a translation vector $\mathbf{t} = (t_u, t_v)^T$ and a rotation-flip-matrix

$$\mathbf{O} = \begin{pmatrix} \cos\phi & -f\sin\phi \\ \sin\phi & f\cos\phi \end{pmatrix}, \tag{2.25}$$

with $f \in \{-1, 1\}$, where -1 indicates a reflection along a line through the origin with slope $\tan\phi$ [145]. The group operation

$$(\mathbf{t}_1, \mathbf{O}_1) \circ (\mathbf{t}_2, \mathbf{O}_2) = (\mathbf{t}_1 + \mathbf{O}_1\mathbf{t}_2, \mathbf{O}_1\mathbf{O}_2) \tag{2.26}$$

reflects that rotations and flips on the 2D grid transform subsequent translations, while the overall rotation and reflection are unaffected by translations[2] [174]. Analogously, the inverse element is $(\mathbf{t}, \mathbf{O})^{-1} = (-\mathbf{O}^{-1}\mathbf{t}, \mathbf{O}^{-1})$ [64].

An element $(\mathbf{t}, \mathbf{O}) \in$ E(2) acts on two-dimensional inputs via [64]

$$\mathcal{T}_{t,\mathbf{O}} \mathbf{x}(u) = \mathbf{x}(\mathbf{O}^{-1}(u - \mathbf{t})) \quad \text{with} \quad \mathbf{t} = \begin{pmatrix} t_u \\ t_v \end{pmatrix}. \tag{2.27}$$

Example 5: The Permutation Group p The *group of permutations* p contains all permutations of a set X, where the group operation is the function composition $(a \circ b)(x) = a(b(x))$, with $x \in X$. The group action maps each element of the set onto another, such that all elements in the set persist. This can be denoted by a *permutation matrix*, that

[2] This can be formally derived by applying the semi-direct matrix product, cf. Appendix A.1.1.

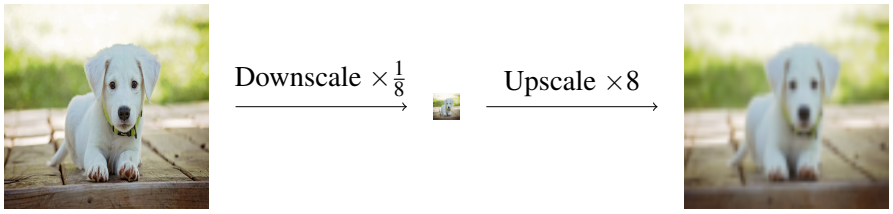


Figure 2.8: 2D scale transformations can only be modelled as a semigroup since the information loss during downscaling prevents inverting the transformation. This is exemplarily shown for an image, where the upscaled version on the right appears blurry due to missing the high frequency information lost during downscaling.

explicitly denotes how each element is mapped from its initial position to another, unique position [145]. For example, the permutation matrix

$$\mathbf{P} = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \end{pmatrix}$$

maps element 1 onto position 2, element 2 onto position 3 and element 3 onto position 1. The identity element is a permutation matrix that maps each element onto itself, while the inverse is given by exchanging the two rows of the permutation matrix [145].

The permutation group covers all translations and rotations lying on a regular grid as subgroups, as these are special cases of permuted pixels. In the case of two-dimensional images the regular grid includes all integer-valued translations, flips and 90° rotations, i.e., $T(2) \subset \mathrm{p}$, $\mathrm{SO}(2,4) \subset \mathrm{p}$, $\mathrm{O}(2,4) \subset \mathrm{p}$, see Figure 2.6.

Non-Example 1: 2D Rotations in a Limited Set The set of rotations within the range $\mathcal{R} = [0, \alpha]$ with $\alpha < 2\pi$ does not form a group, as it violates the axiom of closure. Combining two elements $\gamma = (\alpha + \beta) \bmod 2\pi$ with $0 < \beta < 2\pi - \alpha$ results in $\gamma \notin \mathcal{R}$, which is outside of the defined set.

Non-Example 2: The Semigroup of 2D Scales Scales are an important transformation that naturally occurs in images, for example, caused by variable camera-to-object distances. Scale transformations are non-invertible due to the information loss when down-scaling an image, which is visualized in Figure 2.8. Thus, they can only be modelled as a *semigroup*. The scale semigroup is the set $s \in \mathbb{R}^+$ with the semigroup operation $a \circ b = ab$.

Semigroups still adhere to the first two group axioms: closure and associativity [145]. However, the missing invertibility is important when enforcing symmetries to scales within DNNs, which will be discussed in Section 2.3.1.1 [182] and Chapter 5 [134].

The action of the two-dimensional scale semigroup on inputs $\mathbf{x} : \mathbb{R}^2 \to \mathbb{R}$ is [182]

$$\mathcal{T}_s \mathbf{x}(u) = \mathbf{x}(s^{-1}u) \quad \forall\, s > 0 \quad \text{with } u \in \mathbb{R}^2. \tag{2.28}$$

2.2.4 Group Representations

Group representations provide a foundational framework towards designing in- or equivariant DNNs. The choice of group representation determines, how the features of equivariant CNNs (Section 2.3.1.1) transform under group actions.

In this section, we will again focus on the 2D Euclidean space and real-valued signals. Nevertheless, the presented concepts can be extended to more general spaces and complex-valued inputs. For a detailed discussion of group representation theory, we refer the reader to references [38, 64, 173, 174].

We start by defining group representations ρ formally.

Definition 2.2.7. *Let G be a group with operation $\circ$ and H be a group with operation $\star$. A function $h : G \to H$ is a **group homomorphism**, if [145],*

$$h(a \circ b) = h(a) \star h(b) \;\forall\, a, b \in G. \tag{2.29}$$

Intuitively, a group homomorphism h is a function between groups that preserves the algebraic group structure, i.e., applying h on the combination of two group elements $a \circ b$ in the group G is equivalent to combining the result of h applied to each individual element in G [64, 145].

Definition 2.2.8. *A subgroup $H \subset G$ is a **closed subgroup**, if the following property holds: If $\mathbf{A}_m$ is any sequence of matrices in G, and $\mathbf{A}_m$ converges to a matrix $\mathbf{A}$, then either $\mathbf{A} \in G$ or $\mathbf{A}$ is not invertible. A **matrix Lie group** $G \subseteq \mathrm{GL}(n, \mathbb{C})$ is a closed subgroup of $\mathrm{GL}(n, \mathbb{C})$.*

All matrix groups $G \subseteq \mathrm{GL}(n, \mathbb{R})$ considered in this thesis are also matrix Lie groups [64].

Definition 2.2.9. *Let G be a matrix Lie group and let $\mathcal{V} = \mathbb{K}^n$ be a finite-dimensional vector space with $\mathbb{K} \in \{\mathbb{R}, \mathbb{C}\}$. A **group representation** ρ is a Lie group homomorphism*

$$\rho : G \to \mathrm{GL}(n, \mathbb{K}), \quad \textit{denoted as} \quad g \mapsto \rho(g) \tag{2.30}$$

that maps the group G to the general linear group $GL(n, \mathbb{K})$ [64].

Thus, a group representation maps each group element to an invertible $n \times n$ matrix, where n is determined by the vector space $\mathcal{V}$ corresponding to the representation ρ. Group representations describe *linear group actions*, denoted by $\rho(g)$, on *vector spaces* $\mathcal{V} = \mathbb{K}^n$ through matrix multiplication [64].

Vector Fields In Section 2.1.1, we defined the inputs (and features) of CNNs as functions $\mathbf{x} : \mathbb{R}^2 \to \mathbb{R}$. This definition corresponds to a *scalar field* assigning a scalar to each input position. For simplicity, we may omit the channel dimension since groups act identically on each channel [174]. To understand the action of group representations, we now model the inputs as vector fields $\mathbf{x} : \mathbb{R}^2 \to \mathcal{V}$, where each point in the input space is associated with a *feature vector* from the vector space $\mathcal{V}$. The choice of representation ρ determines the type of the vector field $\mathcal{V} \in \{\mathbb{R}^n, \mathbb{C}^n\}$ and its transformation behaviour [174].

The Representation-Induced Group Action on Vector Fields Matrix groups $G \subseteq GL(n, \mathbb{R})$ act on vector fields $\mathbf{x} : \mathbb{R}^2 \to \mathcal{V}$ according to the *ρ-induced action* $\mathcal{T}_g^\rho$ which depends on the representation ρ and the group element $g \in G$. The *ρ-induced action* is given by[3] [174]

$$\mathcal{T}_g^\rho \mathbf{x}(u) = \rho(g)\mathbf{x}(g^{-1}u), \quad u \in \mathbb{R}^2. \tag{2.31}$$

This action consists of two elements:

1. a spatial transformation $\mathbf{x}(g^{-1}u) = \mathcal{T}_g \mathbf{x}(u)$ (see examples in Section 2.2.3) and
2. a *linear action* on the feature vector in $\mathcal{V}$, determined by $\rho(g)$ [174].

In this context, the representation $\rho(g)$ defines *how each feature vector at every input position transforms* via a linear matrix multiplication. The representation also directly

[3] Deriving this relation lies beyond the scope of this dissertation. For details, refer to reference [174], Section 4.2. The action $\mathcal{T}_g$ is itself a representation (a linear action on vector fields), specifically the *induced representation* of the group G on an Euclidean vector space [174].

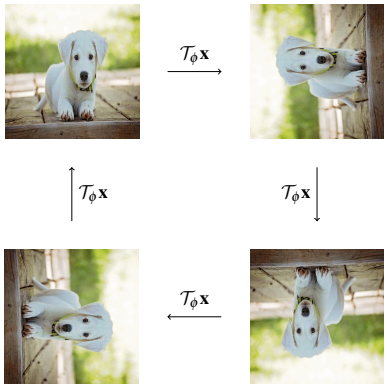


Figure 2.9: The trivial representation of SO(2,4). Each 90° rotation $\mathcal{T}_\phi^{\rho_{\text{triv}}}$ rotates the input $\mathbf{x} : \mathbb{R}^2 \to \mathbb{R}$ spatially. The values of the *feature vectors* at each position (corresponding to the vector space $\mathcal{V} = \mathbb{R}$) do not change.

implies the *type* of these *feature vectors* according to the *corresponding vector space* $\mathcal{V}$. In contrast, the spatial transformation is independent of the chosen representation.

2.2.4.1 Examples for Group Representations & their Actions on Vector Fields

We will now introduce three common representation types used in the in- and equivariant DNN literature as well as for our contributions (Chapter 3 - 5). These representations form the basis to categorize different approaches that enforce in- or equivariance within DNNs (see Table 2.1 and Section 2.3.1.1).

We will focus on how these *representations* ρ *determine* the *transformation behaviour of inputs or features* that encode *equivariance* w.r.t. the group *according to the representation*. Additionally, the representation defines the *type of the corresponding vector space* $\mathcal{V}$ [19, 174]. We use the group of two-dimensional 90° rotations $\text{SO}(2,4)$ as an illustrative example for each representation type.

Example 1: The Trivial Representation The *trivial group representation* maps all group elements to the identity matrix $\rho_{triv}(g) = \mathbf{I}$ and imposes a corresponding *scalar field* $\mathbf{x} : \mathbb{R}^2 \to \mathbb{R}$, where $\mathcal{V} = \mathbb{R}$. The induced representation acting on the scalar field is [174]

$$\mathcal{T}_g^{\rho_{\text{triv}}} \mathbf{x}(u) = \rho_{\text{triv}}(g)\mathbf{x}(g^{-1}u) = \mathbf{x}(g^{-1}u) = \mathcal{T}_g \mathbf{x}(u). \tag{2.32}$$

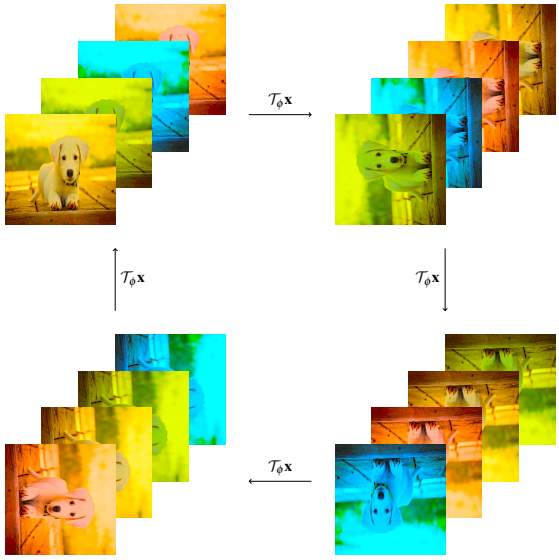


Figure 2.10: The vector field corresponding to the regular representation of SO(2,4) with $|G| = 4$. Each rotation $\mathcal{T}_\phi^{\rho_{\text{reg}}}$ with $\phi = \frac{\pi}{2}$ rotates the input $\mathbf{x}_{\text{reg}} : \mathbb{R}^2 \to \mathbb{R}^{|G|}$ spatially. In addition, the feature vectors defined in the corresponding vector space $\mathcal{V} = \mathbb{R}^{|G|}$ are permuted along the *group dimension*. This is visualized by the different colours, each corresponding to a specific element along the group dimension.

Consequently, the trivial representation acts by transforming the spatial domain. The scalar values themselves do not change, i.e., information about the transformation itself is not encoded [64, 174]. For $\text{SO}(2,4)$, the trivial representation consequently imposes the vector space $\mathcal{V} = \mathbb{R}$ where a rotation $\mathcal{T}_\phi$ rotates the output but does not change the feature vectors, see Figure 2.9.

Example 2: The Regular Representation The *regular group representation* is determined by permutations of its group elements. It induces a vector field $\mathbf{x}_{\text{reg}} : \mathbb{R}^2 \to \mathbb{R}^{|G|}$, with $\mathcal{V} = \mathbb{R}^{|G|}$, where the *group dimension* of the vector space has the size of the group order $|G|$. By definition, the regular representation only exists for finite groups. The induced representation acting on the regular vector field is [174]

$$\mathcal{T}_g^{\rho_{\text{reg}}} \mathbf{x}_{\text{reg}}(u) = \rho_{\text{reg}}(g)\mathbf{x}_{\text{reg}}(g^{-1}u), \tag{2.33}$$

where $\rho_{\text{reg}}(g)$ is a matrix *permuting* the feature vectors *along the group dimension* and $g^{-1}u$ spatially transforms the input [64, 174]. Thereby, the group dimension encodes the

information about the orientation for each vector, in contrast to the trivial representation, where the scalar features remain unchanged.

For $\mathrm{SO}(2,4)$, the regular representation results in $\mathbf{x}_{\text{reg}} : \mathbb{R}^2 \to \mathbb{R}^4$. A rotation permutes the feature vector along the group dimension through ρ_{reg} in addition to the spatial rotation via $g^{-1}u$. This is visualized in Figure 2.10, where each entry along the group dimension is highlighted by a separate colour. For example, the permutation matrices of the regular representation for a rotation by 0 and by $\pi/2$ are

$$\rho_{\text{reg}}(0) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \mathbf{I} \quad \text{and} \quad \rho_{\text{reg}}(\pi/2) = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}, \tag{2.34}$$

respectively [174].

Example 3: Irreducible Representations A particularly important type of representation are the irreducible representations (irreps). Let us start by defining irreps formally.

Definition 2.2.10. *Let ρ be a finite-dimensional representation of a matrix Lie group G acting on the vector space $\mathcal{V}$. A **subspace** $\mathcal{W}$ of $\mathcal{V}$ is called **invariant** w.r.t. ρ, if [64]*

$$\rho(g)w \in \mathcal{W} \; \forall \; w \in \mathcal{W}, \; g \in G. \tag{2.35}$$

*A representation ρ is an **irrep**, if the only invariant subspaces of $\mathcal{V}$ w.r.t. ρ are the trivial subspaces $\mathcal{V}$ and $\{0\}$ [64].*

By definition, a representation ρ is irreducible, if there exists no proper subspace $\mathcal{W} \subset \mathcal{V}$ in the vector space $\mathcal{V}$ that remains unchanged under $\rho(g)$ for any group element g. In other words, a vector space $\mathcal{V}$ associated with an irrep can not further be decomposed into smaller invariant subspaces, i.e., there is no subspace within the vector space not affected by the representation. Consequently, an irrep encodes equivariance efficiently [64, 174].

Let us now focus on the *induced action* of an *irrep* ρ_{irrep} on *vector fields* $\mathbf{x}_{\text{irrep}} : \mathbb{R}^2 \to \mathcal{V}$. The induced action $\mathcal{T}_g^{\rho_{irrep}}$ on such an input is

$$\mathcal{T}_g^{\rho_{irrep}} \mathbf{x}_{\text{irrep}}(u) = \rho_{\text{irrep}}(g)\mathbf{x}_{\text{irrep}}(g^{-1}u), \tag{2.36}$$

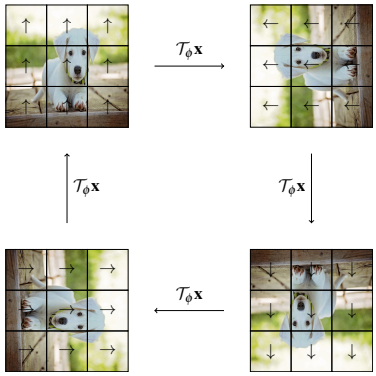


Figure 2.11: The vector field corresponding to the irrep of SO(2,4) with order $k = 1$. Each 90° rotation $\mathcal{T}_\phi^{\rho_{\text{irrep}}}$ acting on the vector field $\mathbf{x}_{irrep} : \mathbb{R}^2 \to \mathbb{C}$ induces a spatial rotation and a rotation of the complex orientation by the same angle, as visualized by the arrows. The complex orientation is defined *per pixel*, exemplarily shown by a 3×3 grid.

where $\rho_{\text{irrep}}(g)$ encodes the transformation of the feature vector and $g^{-1}u$ is a spatial transformation [174].

SO(2) has infinitely many two-dimensional irreps of *order* $k \in \mathbb{N}^+$ that act on the corresponding vector space $\mathcal{V} = \mathbb{R}^2$ by [174]

$$\rho_{\text{irrep}}(k\phi) = \begin{pmatrix} \cos(k\phi) & -\sin(k\phi) \\ \sin(k\phi) & \cos(k\phi) \end{pmatrix}. \tag{2.37}$$

The case $k = 0$ is not handled above, since it results in the *trivial representation* as the only one-dimensional irrep, where the corresponding vector space is reduced to $\mathcal{V} = \mathbb{R}$.

For all irreps, the corresponding vector space can alternatively be defined as a complex-valued vector space $\mathcal{V} = \mathbb{C}$, where an input $\mathbf{x} = \mathbf{a}e^{i\phi}$ is represented by the amplitude $\mathbf{a} : \mathbb{R}^2 \to \mathbb{R}$ encoding feature values and the complex phase $\phi : \mathbb{R}^2 \to i\mathbb{R}$ encoding local orientation for each input location [181]. Using this complex vector space, the irreps of SO(2) can be denoted as

$$\rho_{\text{irrep}}(k\phi) = e^{ik\phi}, \tag{2.38}$$

where $\rho_{\text{irrep}}(0\phi) = e^0 = 1$ results in the trivial representation. Figure 2.11 illustrates a vector field corresponding to the irrep of SO(2) with order $k = 1$. In this case, a rotation acts both spatially and on the complex phase by the same angle.

Irreducible representations efficiently encode the group structure. Unlike the regular

representation, which requires storing $|G| = 4$ values to represent discrete 90° rotations, an irrep of $\mathrm{SO}(2)$ encodes continuous orientation information through the complex phase using only two values. This enables the efficient representation of continuous equivariance, supporting accurate computation and compact storage of equivariant responses. In contrast to the trivial representation, irreps with order $k > 0$ capture the rotation information rather than discarding it [174, 181]. Since any representation of a finite group can be decomposed into a direct sum of irreps (cf. Appendix A.2), they are an important foundation of group representation theory. Finding all irreps of a specific group is an interesting aspect, as it allows to compose any arbitrary representation [64, 174].

2.2.5 Transformation-Steerable Filters

Steerable filters allow to calculate transformed versions of filters in closed-form. The initially proposed variant can be synthesized in arbitrary rotated versions f^θ using a linear combination of a finite number of *basis filters* ψ_i [51]. In the two-dimensional case, rotation-steerable filters are defined in the polar coordinate system with radial coordinate $r = \sqrt{x^2 + y^2}$ and angular coordinate $\phi = \arctan(\|\frac{y}{x}\|)$

$$f^\theta(r, \phi) = \sum_{i=1} w_i(\theta)\psi_i(r, \phi), \tag{2.39}$$

where $w_i(\theta)$ are called interpolation functions that modify the basis filters towards the desired orientation θ. Steerable filters can be calculated in arbitrary rotated versions analytically without suffering from sampling artefacts. This is important for CV tasks, where multiple rotated versions of a filter are applied or learned frequently [51].

The concept of steerable filters can be extended to transformation groups G, enabling the calculation of arbitrarily group-transformed filters without introducing sampling artefacts [98, 162, 173, 176]. A transformed steerable filter defined on the group elements $g \in G$ can be computed in closed form for any transformation $h \in G$ as a linear combination of basis filters

$$f^h(g) = \sum_{i=1} w_i(h)\psi_i(g). \tag{2.40}$$

The basis for steerable filters is group-specific and can be constructed using specific functions such as the *harmonic functions* for SO(2) and SO(3) (cf. Appendix A.2.1)

[98, 162, 173, 175, 176]. The application of steerable filters within equivariant DNNs will be presented in further detail in Section 2.3.1.1.

2.3 Enforcing Geometrical Priors in Deep Neural Networks

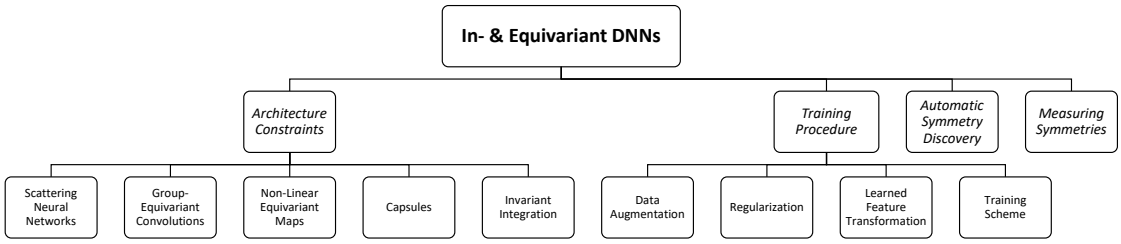


Figure 2.12: (Figure 1 in Rath & Condurache [135]) Taxonomy of methods leveraging Geometrical Prior Knowledge for DNNs.

In this section, we introduce the current state-of-the-art methods to incorporate geometrical prior knowledge to DNNs. A general taxonomy of the in- and equivariant DNN literature is shown in Figure 2.12. Following this taxonomy, we present related work that achieves *guaranteed equivariance* via *architecture restrictions* (Section 2.3.1), learns *approximate equivariance* by adapting the *training procedure* (Section 2.3.2), automatically *discovers in- or equivariance* from data (Section 2.3.3) and *measures equivariance* properties (Section 2.3.4). This section is based on our survey paper *"Boosting Deep Neural Networks with Geometrical Prior Knowledge: A Survey"* [135], which will not further be cited. Due to its fundamental role to our contributions (Chapter 3-5), we discuss II separately in Section 2.4.1.

2.3.1 Architecture Constraints

In- or equivariance can be incorporated into a DNN in a mathematically guaranteed way by restricting the architecture or the learnable filters. CNNs share a convolutional kernel among all positions of the input image or feature map and thereby achieve translation equivariance [54, 103], as proven in Section 2.1.1.2. This concept is one of the key properties responsible for the success of CNNs in various practical applications and can be expanded beyond translations.

2.3.1.1 Group-Equivariant Convolutional Neural Networks

Group-Equivariant Convolutional Neural Networks (G-CNNs) enforce equivariance to symmetric transformations by replacing the standard convolution with the generalized G-Conv [17]. In this section, we first introduce the G-Conv by deriving it from the standard convolution. Afterwards, we present related work investigating the theory underlying G-CNNs as well as applications to different transformation groups and input domains.

Recall the two-dimensional discrete standard convolution from Equation 2.5

$$(\mathbf{x} \star \psi)(u) = \sum_{v \in \mathbb{Z}^2} \mathbf{x}(v)\psi(v-u). \tag{2.41}$$

The convolution kernel ψ of spatial extent v is shifted (translated) over the domain $\mathbb{Z}^2$ of the input function $\mathbf{x}$. Theorem 2.1.1 states that this induces equivariance to translations.

The convolution operation can be generalized to arbitrary groups G, resulting in the G-Conv. The *continuous G-Conv* is defined as

$$(\mathbf{x} \star_G \psi)(h) = \int_{g \in G} \mathbf{x}(g)\psi(h^{-1}g)\, d\mu(g), \tag{2.42}$$

where μ is the left Haar measure [17, 174]. The G-Conv is computed on the domain of the group G. Consequently, its input $x : G \to \mathbb{R}$ and filters $\psi : G \to \mathbb{R}$ are defined on the group G, meaning they assign a value to each group element [17, 174]. The shift $-u$, i.e., the action of the translation group, in Formula 2.5 is replaced with the action of the transformation group $h^{-1} \in G$. Thereby, the G-Conv applies the kernel in all transformed versions included in the group rather than only shifting it over the input. Consequently, the standard convolution is a special case for the group of translations. For the example of SO(2,4), the regular G-Conv rotates the kernel with all possible 90° rotations and stores the results along an additional dimension called *group dimension* (cf. Figure 2.10), corresponding to the regular representation [17].

Similarly to the translational convolution, the discrete variant of the G-Conv is given by reducing the integral to a discrete sum [17]

$$(\mathbf{x} \star_G \psi)(h) = \sum_{g \in G} \mathbf{x}(g)\psi(h^{-1}g). \tag{2.43}$$

Theorem 2.3.1. *Given an input* $\mathbf{x} : G \to \mathbb{R}$*, a convolutional kernel* $\psi : G \to \mathbb{R}$ *and the left group action* $\mathcal{T}_{\tilde{g}}$ *acting on the input via* $\mathcal{T}_{\tilde{g}}\mathbf{x}(h) = \mathbf{x}(\tilde{g}^{-1}h)$ *with* $\tilde{g}, h \in G$*.*

Group-equivariant convolutions are equivariant to transformations of the group

$$(\mathcal{T}_{\tilde{g}}\mathbf{x} \star_G \psi)(h) = \mathcal{T}'_{\tilde{g}}(\mathbf{x} \star_G \psi)(h). \tag{2.44}$$

A group action $\mathcal{T}_{\tilde{g}}$ *acting on the input leads to a mathematically guaranteed group action* $\mathcal{T}'_{\tilde{g}}$ *acting on the output.*

Proof.

$$\begin{aligned}((\mathcal{T}_{\tilde{g}}\mathbf{x}) \star_G \psi)(h) &= \sum_{g \in G} \mathbf{x}(\tilde{g}^{-1}g)\psi(h^{-1}g) \\ &= \sum_{g \in G} \mathbf{x}(g)\psi(\tilde{g}h^{-1}g) \\ &= \sum_{g \in G} \mathbf{x}(g)\psi((\tilde{g}^{-1}h)^{-1}g) \\ &= (\mathbf{x} \star_G \psi)(\tilde{g}^{-1}h) \\ &= \mathcal{T}'_{\tilde{g}}(\mathbf{x} \star_G \psi)(h).\end{aligned} \tag{2.45}$$

□

Since the input data of DNNs is not defined on the group domain (e.g., images are defined on the regular grid $\mathbb{Z}^2$), a *lifting* convolutional layer is defined, that lifts the input from the regular grid $u \in \mathbb{Z}^2$ to the group domain $h \in G$

$$(\mathbf{x} \star_G \psi)(h) = \sum_{u \in \mathbb{Z}^2} \mathbf{x}(u)\psi(h^{-1}u), \tag{2.46}$$

where $(\mathbf{x} \star_G \psi) : G \to \mathbb{R}$ is defined on the group domain and $\psi : \mathbb{Z}^2 \to \mathbb{R}$ is a learnable filter defined on the regular grid $\mathbb{Z}^2$ [17].

Theoretical Foundations G-CNNs have first been introduced for rotation equivariant classification [17] (cf. Section 2.3.1.1). Further research provides theoretical foundations that enable a classification of existing equivariant G-CNNs, allows to expand them towards general groups and domains or mathematically proves their properties.

DNNs are provably equivariant, if each operation within the network enforces or preserves equivariance [17]. Besides the learnable convolutions, this includes other layers such as non-linearities or pooling, which preserve equivariance when being conducted in a point-wise manner or when acting on a subgroup $H \subset G$ [17]. A DNN is provably equivariant to the action of a compact group G if and only if each layer implements the generalized G-Conv (see Equation 2.43) [93], or if and only if the group action explains the symmetries within the network's parameter matrices [136]. The latter insight allows to derive a parameter-sharing scheme to enforce in- or equivariance [136].

G-Convs can be classified based on their symmetry group G and the base space H the features are attached to [93], which can be expanded to also consider the field type ρ, corresponding to the choice of group representation (cf. Section 2.2.4) [19]. G-Convs are not only a necessary and sufficient condition for equivariant DNNs, but also represent the most general equivariant linear map [19].

The feature spaces of equivariant CNNs can be defined as steerable compositions of *elementary feature types* each encoding certain transformation symmetries. The steerability directly induces a linear change of the features under input transformations. The choice of feature types directly imposes constraints on the network weights which provably reduces the number of learnable parameters [20]. Based on these insights, a PAC-Bayesian generalization bound for steerable CNNs expressed as a direct sum of irreps allows to determine the effect of chosen group size, multiplicity and type of irreps on the generalization error of G-CNNs [6]. All single-hidden-layer CNNs with ReLU activations invariant to a finite orthogonal group G can be classified based on the signed permutations of the underlying irreps. The classification enables to automatically find the best G-invariant single hidden-layer CNNs, e.g., via Neural Architecture Search [2].

A general overview over the mathematical concepts underlying G-CNNs including group representation theory, integration and harmonic analysis in non-Euclidean spaces, as well as differential geometry is provided in reference [38]. The application to SO(3) highlights the connection between spherical harmonics and irreps when calculating convolutions in the Fourier space (cf. Appendix A.2.1) [38].

Applications In this section, we present applications of G-CNNs ordered by their equivariance group. We categorize them systematically based on their symmetry group G, the base space H and the field type induced by the representation ρ in Table 2.1 following

the methodology introduced in reference [19].

2D Rotations, Translations & Flips First, we present approaches equivariant to the group E(2), which includes two-dimensional translations, rotations and flips, and its subgroups (cf. Section 2.2.3).

G-Convs have initially been introduced to DNNs for the two-dimensional group of 90° rotations and flips E(2, 4). Equivariance is achieved by applying a standard convolution with rotated and flipped filters and stacking the activations among the *group dimension*, which corresponds to the regular group representation (cf. Section 2.2.4) [17]. Bilinear interpolation expands this framework to 60°-rotations using a hexagonal grid [75], or arbitrary groups containing N discrete rotations $\mathrm{SE}(2,N) \subset \mathrm{SE}(2)$ [8].

Instead of using interpolations, the filters of E(2)-equivariant G-Convs can be restricted to a learned linear combinations of *steerable filters* [173, 176]. As presented in Section 2.2.5, arbitrary transformed versions of steerable filters can be calculated in closed-form and are therefore not affected by interpolation artefacts. For SE(2), the steerable filters basis is given by the circular harmonics [176]. A general recipe to calculate equivariant kernel bases for any subgroup and representation of E(2) can be defined based on the irreps. The equivariance constraints can be relaxed per layer, allowing earlier layers to exhibit stricter equivariance than later layers in a DNN [173].

Interpolating the filters can also be avoided by learning a filter basis and all its rotated versions jointly through rotation-invariant coefficients using a equivariance loss term [34]. Another weight-sharing approach based on polar coordinates with orbit decomposition obtains rotation-equivariant continuous convolutions via torus-valued kernels [172].

Two approaches propose SE(2)-equivariant CNNs using irreps. The maximum activation and the orientation that caused it computed via G-Conv can be stored in a *vector field* [110]. Alternatively, the filters can be restricted to circular harmonics with learnable coefficients in the polar domain, which are inherently related to the irreps (cf. Appendix A.2.1). Both, the filters and the activations are defined in the complex domain, where the modulus is invariant to rotations, while the phase stores the equivariant information about the orientation. Hence, equivariance to *continuous* 2D rotations is achieved [181].

Multiple approaches apply E(2)-equivariant convolutions beyond supervised learning with CNNs. The regular SO(2, N)-G-Conv has been used within equivariant Generative Adversarial Networks (GANs) [32] and Variational Autoencoders (VAEs) to disentangle

Table 2.1: (Table 2 in Rath & Condurache [135]) Overview of in- and equivariant CNNs classified by the equivariance group G, the base space G/H and the vector field type imposed by the chosen representation ρ. Based on [19].

Reference	Equivariance group	Base space	Vector field type
[54, 103]	$T(2)$	$\mathbb{Z}^2$	regular
[17]	$\mathbf{SE}(2,4)$, $\mathbf{E}(2,4)$	$\mathbb{Z}^2$	regular
[20]	$\mathbf{SE}(2,4)$, $\mathbf{E}(2,4)$	$\mathbb{Z}^2$	irrep & regular
[33]	$\mathbf{SE}(2,4)$	$\mathbb{Z}^2$	regular
[141]	$\mathbf{SE}(2,4)$	$\mathbb{Z}^2$	regular
[8]	$\mathbf{SE}(2,N)$	$\mathbb{Z}^2$	regular
[110]	$\mathbf{SE}(2,N)$	$\mathbb{R}^2$	irrep & regular
[176]	$\mathbf{SE}(2,N)$	$\mathbb{R}^2$	regular
[34]	$\mathbf{SE}(2,N)$	$\mathbb{R}^2$	regular
[181]	$\mathbf{SE}(2)$	$\mathbb{R}^2$	irrep
[172]	$\mathbf{SE}(2)$	$\mathbb{R}^2$	trivial & regular
[173]	$\mathbf{E}(2)$	$\mathbb{R}^2$	any
[18]	$\mathbf{SO}(3)$	S^2	regular
[39, 41, 43]	$\mathbf{SO}(3)$	S^2	trivial
[129]	$\mathbf{SO}(3)$	S^2	trivial
[29]	$\mathbf{SO}(3)$	S^2	trivial
[92]	$\mathbf{SO}(3)$	S^2	irrep
[85]	$\mathbf{SO}(3)$	S^2	irrep
[42]	$\mathbf{SO}(3)$	S^2	irrep
[178]	D_4, D_{4h}, $\mathbf{SE}(3,4)$, $\mathbf{E}(3,4)$	$\mathbb{Z}^3$	regular
[180]	$\mathbf{SE}(\mathbf{3},\mathbf{4})$, T_4, V	$\mathbb{Z}^3$	regular
[175]	$\mathbf{SE}(3,N)$	$\mathbb{R}^3$	irrep
[3]	$\mathbf{SE}(3,N)$	$\mathbb{R}^3$	regular & trivial
[184]	$T(2) \rtimes S$	$\mathbb{R}^2$	regular
[87]	$T(2) \rtimes S$	$\mathbb{R}^2$	trivial
[109]	$T(2) \rtimes S$	$\mathbb{R}^2$	irrep & regular
[182]	$T(2) \rtimes S$	$\mathbb{R}^2$	regular
[190]	$T(2) \rtimes S$	$\mathbb{R}^2$	regular
[59]	$T(2) \rtimes S$	$\mathbb{R}^2$	regular & trivial
[160, 162]	$T(2) \rtimes S$	$\mathbb{R}^2$	regular
[82]	$\mathbf{SE}(2) \rtimes S$	$\mathbb{R}^2$	irrep
[106]	$\mathbf{SE}(2) \rtimes S$	$\mathbb{R}^2$	regular & trivial
[166]	$\mathbf{SE}(3)$	$\mathcal{G}$	irrep & regular
[5]	$\mathbf{E}(3)$	$\mathcal{G}$	irrep
[10]	$\mathbf{E}(3)$	$\mathcal{G}$	irrep
[147]	$\mathbf{E}(N)$	$\mathcal{G}$	trivial
[76]	$\mathbf{E}(3)$	$\mathcal{G}$	trivial
[16]	Local Gauge G	Manifold	irrep
[27]	Local Gauge-G	Mesh	irrep
[152]	$\mathbf{SE}(3) \rtimes S$ & Local Gauge-G	S^2	irrep
[7]	$T(2) \rtimes H$	$\mathbb{R}^2$	regular
[48]	G	$\mathbb{R}^n$	regular
[49]	$\mathbf{GL}(n)$	$\mathbb{R}^n$	any
[98]	G	X	any
[13]	G	X	any

the learned invariant object representation from the equivariant knowledge about its orientation [118, 179]. E(2)-equivariant steerable filters can further be used within the learnable decoder of Gaussian and Conditional Neural Processes to embed equivariance to the learned posterior distributions [74].

3D Rotations and Translations Equivariance can also be enforced beyond two-dimensional signals, representations and transformations. However, as finite groups grow exponentially with increasing dimension, sophisticated analytical solutions gain importance. Especially for larger groups, the irreps provide a more efficient way to encode the outputs in the group domain compared to the regular representation which grows with the group size.

First, we present work which achieves equivariance to the group of 3D rotations SO(3) for inputs defined on the *sphere* S^2, which for example include spherical images or global climate models. A SO(3)-equivariant spherical convolution can be obtained using the Group Fast Fourier transformation followed by calculating the regular G-Conv in Fourier space via multiplication [18]. Alternatively, the trivial representation can be used to calculate the G-Conv in the Fourier domain [39]. Applications of SO(3)-equivariant spherical convolutions cover 3D shape recognition [18, 39], semantic segmentation of spherical images [41], encoding equivariant shape information from multi-view images [43] and unsupervised 3D shape correspondence [163].

The spherical convolution can be approximated by treating the discrete sphere as a graph, applying graph convolutions and restricting the learned filters to be radial to achieve equivariance to 3D rotations. While this provides less exact equivariance properties, the convolution is computationally more efficient compared to spherical convolutions [129]. The number of connected neighbours on the approximated sphere directly controls the trade-off between equivariance guarantees and computational complexity [29]. Comparably, the sphere can be processed as a mesh, i.e., a unstructured grid, where rotation-equivariance is guaranteed by applying a mesh convolution that involves a learnable linear combination of parametrized differential operators [85].

N-body networks provide a general CNN architecture that encodes equivariance to 3D rotations in Fourier space via the irreps and computing the non-linearities directly in the Fourier space based on the Clebsch-Gordan transformation (cf. Appendix A.2) [91]. The Clebsch-Gordan decomposition provides a non-linearity within SO(3)-equivariant

spherical CNNs, which increases computational efficiency by calculating both the convolutions and the Clebsch-Gordan transformation in the Fourier space. Hence, the frequent calculation of forward and inverse Fourier transformations after each convolutional layer is avoided [92]. Similarly to the 2D case, the learned filters of spherical CNNs can be restricted to complex-valued, spin-weighted spherical harmonic functions to induce continuous equivariance to SO(3). By using spin-weighted functions, the rotation information is encoded in the phase of the complex-valued activations. Thus, lifting the input to SO(3) is avoided which reduces the computational complexity [42]. Rotation-equivariant neural vector fields can be learned via a conditional latent code that represents the desired rotation around the vertical sphere axis [55].

Multiple proposals achieve equivariance to the group of 3D rotations and translations SE(3) for *Euclidean 3D inputs*. The regular G-Conv can be expanded to finite subsets of SE(3) by rotating the filters and storing all responses among the group dimension, similarly to the 2D case [17]. Rather than using the group of all transformations, appropriate subgroups such as Klein's four group V containing only 4 rotations can be used to improve the computational efficiency [178, 180]. Steerable filters can also be expanded to the 3D case by using learnable linear combinations of the spherical harmonics as convolutional kernels [175]. Local 3D rotation invariance can be obtained using a single rotation-equivariant convolution with steerable filter weights followed by orientation max pooling and global spatial average pooling. The obtained invariance is localized by the chosen kernel size [3]. A general algorithmic framework to compute any E(3)-equivariant CNN is provided by reference [57].

Scale Scales are an important transformation that naturally occurs in images, as pointed out in Section 2.2.3. However, since scales only form a semigroup, specific measures need to be introduced to achieve equivariance. Additionally, the equivariance properties are limited to a certain range because the scales semigroup is also non-cyclic.

Scale *invariance* can be achieved by sharing scaled versions of a convolutional filter obtained via bi-linear interpolation among a group *column*, similar to the regular G-Conv for 2D rotations. Scaling a feature results in a column flip of the responses. Instead of using global pooling, the activations of all columns are concatenated and processed by fc layers [184]. Contrarily, the input of each convolutional layer can be scaled using bilinear interpolation instead of the filters. By storing only the maximum response, local scale-

invariance is obtained for each layer [87]. This approach can be expanded by additionally storing the information about which scale caused the maximum response at each location in a vector field to obtain equivariance [109].

In order to achieve mathematically guaranteed equivariance for semigroups, the G-Conv needs to be expanded accordingly by transforming the input instead of the filter [182], similarly to reference [87]. A transformation acting on the input again induces a shift along the group axis of the semigroup convolution's output. To achieve equivariance to integer-valued scales, the image can be defined within a scale-space consisting of multiple blurred versions processed by the semigroup convolution [182].

Ignoring boundary effects, the regular G-Conv also achieves equivariance for semigroups. One possibility to achieve scale-equivariance are joint convolutions across the translation and scale group with convolutional filters constructed from two fixed, separable Fourier-Bessel bases with trainable expansion coefficients [190]. To avoid interpolation artefacts, the G-Conv can again be used in combination with scale-steerable filters [59, 160, 162]. The filters are learnable linear combinations of a filter basis constructed from log-radial harmonics [59] or two-dimensional Hermite polynomials with a Gaussian envelope [162]. Alternatively, the discrete filter basis can be learned by directly minimizing the equivariance error for non-integer valued scales or continuous functions [160]. Scale-equivariant convolutions with steerable filters have successfully been applied to siamese tracking [161].

Further approaches achieve scale-equivariance using wavelet filters for time-series [139], scale-rotation equivariance via a SO(2)-equivariant Autoencoder (AE) and non-linearities based on scale-coupled bases [82], or scale-invariant features by processing multiple scaled inputs with a standard CNN and applying the scale G-Conv on top [106].

Graph Neural Networks Graph Neural Networks (GNNs) are DNNs designed to process data defined on a graph $\mathcal{G}$. By design, GNNs are permutation-in- or equivariant, i.e., the order of graph nodes does not affect the desired output. The graph convolution is commonly computed in the spectral domain [12] with localized filters [28] and can be approximated in an compute-efficient way [90]. GNNs with multiple stacked layers can also be interpreted as a message passing DNN, since the information is aggregated from the neighbouring nodes at each layer. Message passing GNNs can be convolutional, but can also use other functions for the feature aggregation. Similarly to DNNs operating on

Euclidean inputs, in- or equivariance to other transformations than permutations generally improves the sample efficiency and performance of GNNs [5, 10, 76, 147, 166].

Representations of point clouds, i.e., unconnected graphs, that are locally equivariant to 3D rotations, translations and point permutations, can be learned using continuous convolutions with filters restricted to the spherical harmonics. The point-wise convolutions are calculated using the equivariant tensor product given by the Clebsch-Gordan coefficients and the irreps of SO(3) resulting in vector-valued responses per point [166]. Alternatively, the convolutional filters can be restricted to spherical harmonics with a learnable radial profile and phase offset to achieve equivariance to E(3) [5]. A non-linear generalization of E(3)-steerable G-Convs computed via the Clebsch-Gordan tensor product allows to construct steerable E(3)-equivariant message passing GNNs [10]. Equivariance to E(n) can be enforced without spherical harmonics or tensor-valued intermediate representations by considering the relative squared distance between nodes in the edge function and sequentially updating the relative positions of all particles. This approach allows to scale the equivariance beyond 3 dimensions [147]. By adjusting the adjacency-matrix of graph CNNs, in- or equivariance to isometric transformations is embedded in a computationally efficient manner, allowing to scale equivariant GNNs up to large input graphs required for physical models [76].

Arbitrary Groups & Inputs G-Convs are not restricted to Euclidean spaces, but can be expanded towards arbitrary input domains or groups. Equivariance to local gauge transformations on arbitrary manifolds can be achieved by expanding the G-Conv to only depend on the intrinsic geometry of the manifold [16]. Using graph convolutions with anisotropic kernels obtains gauge-equivariance on meshes allowing to appropriately capture the mesh geometry rather than modelling it as a graph [27]. A hierarchy of symmetries including local gauge transformations as well as global rotations and scalings of pixelized spheres can be achieved via equivariant maps based on systems of blocks and an equivariant padding operation [152].

The G-Conv layer can be expanded to general Lie groups via a surjective exponential map based on the Lie algebra, allowing the application to arbitrary continuous data, including images or point clouds. To implement equivariance for a specific group, the group exponential and logarithm maps need to be solved within the general layer framework [48]. A standard convolution with kernels parametrized as B-spline basis

functions defined on a group's Lie algebra induces equivariance to groups that are a semidirect product of the translation group and arbitrary Lie groups [7]. G-steerable kernels can be designed for arbitrary compact groups G and a homogeneous base space X by an expansion of the Wigner-Eckart theorem. Those G-steerable kernels consist of learnable bases, Clebsch-Gordan coefficients and harmonic basis functions. Equivariance to the semidirect product of the group G and the translation group $T(n)$ is achieved by applying a standard convolution using the derived kernels [98]. This approach can be further expanded to non-homogeneous base spaces by defining a G-steerable basis over the whole space rather than using the harmonic basis defined on orbits of the group [13].

A Lie group-equivariant conditional Neural Process can be induced by using a functional equivariant representation in the encoder and Lie G-Convs [48] in the decoder [88]. Similarly, VAEs achieve equivariance to Lie groups by applying the exponential map derived via the Lie algebra [191]. Finally, a general algorithm allows to design equivariant MLPs for any arbitrary group representation and matrix Lie group. The equivariance constraints of the learnable kernels are solved via a singular value decomposition of infinitesimal or discrete Lie algebra generators [49].

2.3.1.2 Non-Linear Equivariant Maps

While G-Convs are the most general linear map guaranteeing equivariance, non-linear operations such as SA used within Transformers have recently gained significant research interest [35, 170]. Similarly to convolutions, those non-linear operations can be adapted to guarantee in- or equivariance as well. Additionally, the in- or equivariance properties of the general non-linearities used within DNNs have been investigated.

Attention The rotation G-Conv can be combined with SA by multiplying the convolutional output with attention scores to obtain equivariant data-dependent filters [33]. Alternatively, attention can be applied on the activations of a G-Conv along the group-dimension to learn co-occurring transformations from the dataset. Thereby, knowledge about the relative orientation between different parts can be leveraged without disrupting the global equivariance properties [141]. A general group-equivariant SA layer can be derived by modifying the positional encodings to be invariant to transformation groups and lifting the computed representations to the group domain [140]. However, while

group-equivariant SA networks achieve better performance than their non-equivariant counterparts, they fail to outperform equivariant convolutional architectures in the limited data domain [140]. The group-equivariant SA can be expanded to general Lie groups and general input domains by first lifting the input to the group itself with a learnable map before applying the SA modules directly on the group [79]. Combining a modified SE(3)-equivariant Tensor Field Network [166] with learnable linear invariant attention weights results in a SE(3)-equivariant Transformer for point clouds [52]. An iterative variant allows to apply this Transformer to dynamically changing graphs utilizing non-fixed basis functions [53]. A gauge-equivariant SA operation that is also invariant to global rotations can be obtained by constraining the learnable matrices of the attention layer using a Taylor series expansion and a projection onto local coordinate systems, respectively [69]. A general parameter-, data- and compute-efficient equivariant layer that includes both group SA and G-Convs as special cases is proposed by reference [68]. The layer is decoupled into a dynamic kernel generation method based on the input features and a feature encoder that guarantees equivariance by only depending on the relative positions of input pairs [68].

Non-Linearities The non-linearities of group-equivariant DNNs need to be adapted such that the equivariance information encoded by the G-Convs is preserved (cf. Section 2.3.1.1). For the example of regular group representations, the non-linearities need to be applied in a point-wise manner such that the whole group channel is processed equivalently [17]. Nevertheless, the applied non-linearities and down-sampling operations only preserve equivariance ignoring boundary effects. Consequently, the effects of those layers on the equivariance properties can be assessed. Reference [183] measures the loss of in- or equivariance suffered by applying pooling or strided convolutions within equivariant DNNs and proposes an adapted pooling version that achieves exact in- or equivariance. Alternatively, non-linearities based on the Fast Fourier Transformation yield exact equivariance for polynomial non-linearities and approximate solutions with tunable parameters for other non-linearities [50].

2.3.1.3 Scattering Neural Networks

In- or equivariance properties can be enforced to functions by using a well-defined map with *fixed filters* that yields the desirable properties. The hand-crafted *scattering transformation* consists of a convolution of the input signal with a family of wavelets computed by rotating and scaling a complex-valued mother-wavelet followed by a modulus non-linearity and an averaging filter. The scattering features achieve invariance w.r.t. translations, are stable to small deformations and robust albeit not invariant to other transformations such as rotations [11]. Moreover, they closely resemble the learned features of the early layers of CNNs for image classification [125].

Scattering can be expanded to guarantee invariance beyond translations w.r.t. compact Lie groups G [108]. This expansion has been conducted for the 2D group of translations and rotations SE(2) [124]. Cascading multiple scattering transformations with global scale-space averaging achieves invariance to translations, rotations and scales while still guaranteeing stability to small deformations [157].

The scattering transformation can also be used within DNN architectures by replacing the early convolutional layers with fixed scattering transformations, thereby embedding invariance to the lower-level features [122, 126]. In combination with a spatial aggregation function, the input size of hybrid scattering CNNs can be reduced to lower the parameter count and required training time [123]. Compared to conventional CNNs, hybrid scattering networks learn less general filters in the early layers, impeding their ability to learn more complex shapes. This discrepancy can be resolved by replacing the spatial averaging of the scattering transformation with a method that preserves the spatial domain [24]. Another scattering DNN learns a dictionary of scattering coefficients and processes it with a MLP allowing to scale scattering-based DNNs to big datasets such as ImageNet [187].

Even though hybrid scattering networks use the scattering transformation within learned DNNs, the scattering operation itself is still hand-crafted and fixed. Two methods aim to learn coefficients within the scattering operation itself via backpropagation by either multiplying its output with learnable weights [25], or learning the parameters of the scattering transformation's Mother wavelet by relaxing the constraints on the wavelet family [56]. In both cases, the learnable scattering transformation is used to replace the early layers of a conventional CNN similarly to reference [122].

2.3.1.4 Capsules

Capsules are a specific DNN architecture designed to explicitly learn in- and equivariant representations. Capsules disentangle their representations into an invariant component that predicts the presence probability of a specific entity and an equivariant component learning its instantiation parameters, e.g., its pose. A specific routing algorithm determines how information is propagated among different capsules arranged in a capsule network [72]. A variation of capsules encodes the instantiation parameters via the orientation and the existence probability via the length of a vector allowing to define the routing algorithm based on the scalar product of corresponding vectors [146]. The learned representations can be disentangled by learning an invariant entity presence via a logistic unit and an equivariant pose matrix, that allows to compute transformation matrices between capsules [73]. Further, capsules can be applied to AEs [94].

So far, the presented capsule networks only learn or approximate the equivariant pose matrices. Combined with SO(2)-G-Convs, capsules achieve a guaranteed invariant presence probability and an equivariant pose prediction. However, since the pose vector is defined on the group domain, it prevents the capsule from extracting arbitrary pose information, e.g., about lighting [105]. Alternatively, G-Convs can be used within capsules to learn the manifold of legal pose-variations, rather than pair-wise relationships between capsules. A provably equivariant routing procedure guarantees to preserve the learned part-whole relationship under transformations. By no longer embedding the pose information in the group domain, non-geometrical properties can be encoded [171].

2.3.1.5 Other Architecture Restriction Methods

Another method to embed equivariance is to apply a fixed kernel among arbitrary transformations of the input, computed via kernel-based interpolation schemes. Equivariance is embedded by sharing the weights among each point of the symmetry feature map [58].

2.3.2 Training Procedure

Beyond modifying the architecture, another variant to enforce in- or equivariance to DNNs is to adapt the training procedure. In contrast to the architecture restrictions presented in section 2.3.1, however, the obtained equivariance properties are approximately learned

rather than mathematically guaranteed.

2.3.2.1 Data Augmentation

Data augmentation is a simple method to approximate in- or equivariance in DNNs. Formally, the training samples $\mathbf{x}$ and targets $\mathbf{y}$ are transformed with using a set of transformations $\mathcal{T}$ such that the target encodes the desired transformation behaviour. For example, if invariance is desired the target remains unchanged for arbitrary input transformations. Random transformation are applied during training, allowing the DNN to generalize across the defined transformation set by learning from augmented sample-target pairs. Effectively, data augmentation artificially increases the amount of available training samples [60].

Data augmentation often significantly improves the performance and robustness of DNNs and is easy to implement, as only the input and corresponding output transformations need to be known. The transformations are not restricted to groups, which enables an easy generalization to sophisticated augmentations such as occlusions, in-painting, noise, lighting changes or even adding or removing objects from a scene [60, 156].

Nevertheless, DNNs trained with data augmentation do not guarantee the desired in- or equivariance but rather learn it during training. Therefore, additional model capacity is required, e.g., in early CNN layers, where multiple transformed versions of the same filter are frequently learned. The desired symmetry properties are also not guaranteed *per layer*, but only *globally* for the entire model. Moreover, the exact transformation set used for the augmentations can be hard to optimize, while misspecified hyper-parameters (HPs) can even decrease the performance. Finally, defining the augmentation still requires prior knowledge about the desired in- or equivariance properties [97, 99, 174].

A closely related approach that aims to increase the robustness of DNNs is *test time augmentation*. Here, multiple transformed versions of the input are processed by the same network during inference. The network predictions are then aggregated, e.g., via averaging or a Bayesian Neural Network. While this approach increases robustness, multiple parallel computations increase the memory and runtime footprint during inference [99, 154].

Data augmentation can easily be used in conjunction with the earlier presented approaches that guarantee equivariance. This allows to combine mathematically guaranteed equivariance to well-defined transformation groups with robustness to more complicated

transformations that cannot be modelled as groups [17].

2.3.2.2 Symmetry Regularization

A possibility to improve the transformation robustness learned with data augmentation is to add a *symmetry regularization* loss that enforces the desired in- or equivariance properties on the latent representations or outputs. Similarly to data augmentation, the regularization term can be applied to transformations that cannot be modelled as groups, as long as the desired effect on the learned representations or output is known.

Symmetry regularizations can be applied by transforming pairs of transformed inputs and defining a loss based on the Kullback-Leibler divergence [22], or based on the mean-squared Euclidean distance in latent space [4]. Furthermore, the worst-case transformation set a DNN needs to be robust to can be modelled as spatial adversarial attacks to derive a adversarial defence regularization term [185]. Alternatively, a regularization term can be defined based on the group transformation's *defining action* that should be preserved under transformations, e.g., distance for the Euclidean groups [151].

2.3.2.3 Training Scheme

A modified training scheme enables a VAE to learn an invariant class representation and equivariant information specific to individual samples. The decoder computes an invariant canonical class latent variable based on samples from the same class excluding the input. The sample-specific equivariant latent variable smoothly transforms the canonical class sample into the specific instance, encoding arbitrary smooth transformations [45].

2.3.2.4 Learned Feature Transformations

A transformation-invariant feature space can alternatively be obtained by estimating the transformations acting on the input and re-transforming the signal to its *canonical* form. This *learned feature transformations* can be applied on a DNN's input or feature space.

The exponential map of a transformation group allows to warp the input into a space where transformations result in translations. Thus, standard convolutions in the warped space achieve the desired equivariance in the input space. While this approach is computationally cheaper than G-Convs, it is limited to Abelian groups with a maximum of two

parameters [70].

Spatial Transformer Networks (STNs) implicitly learn invariance to affine transformations including translation, rotation and scale. Therefore, a localization network estimates affine transformation parameters used to transform the input back into its untransformed, *canonical* version. By employing differentiable image sampling methods, backpropagation can be used to train both the DNN and the localization network at once [83]. This concept can be expanded towards Polar Transformer Networks (PTNs) which predict the origin for a log-polar transformation of the input. The log-polar representation is invariant w.r.t. the predicted object origin, while rotations and scales induce shifts in the polar space, which is the *canonical coordinate system* of rotations and scales. Consequently, standard CNN can be used to achieve rotation- and scale-equivariant predictions [40]. Equivariant Transformer Networks (ETNs) further generalize STNs towards arbitrary continuous groups. In contrast to STNs, the input of the localization network is transformed to the *canonical coordinate system* where the desired symmetry transformation acts via translations. Thereby, the transformation parameters are estimated in an equivariant manner using standard convolutions. In comparison to PTNs, ETNs only use the canonical coordinate system for the localization network rather than for the entire DNN [165].

2.3.3 Discovering In- & Equivariance from Data

Rather than guaranteeing in- or equivariance via fixed, pre-determined architecture restrictions or training procedures, the desired properties can be learned directly from data. On the one hand, this allows to discover symmetries automatically, which is helpful if the required prior knowledge is not available. On the other hand, this approach does not strictly leverage prior knowledge – and will at most achieve the performance of algorithms with guaranteed equivariance, if the built-in symmetries are present in the data.

Invariances for Gaussian process models can be learned by modelling transformations as priors and optimizing the kernel parameters via a variational lower bound using backpropagation [169]. *Augerino* learns the parameters of affine data augmentations through backpropagation enabled via the reparametrization trick and a regularization term [9]. This approach can be expanded to transformation sets by defining augmentations as differentiable layers embedded into the DNN, which allows to learn the range of transformations and their importance from data [143]. Alternatively, data augmentation

parameters can be optimized based on their marginal likelihood [80].

While the previous approaches focus on data augmentation, the parameters of modified G-Convs can also be automatically learned from training data. Restricting the kernels of a CNN to a Lie algebra basis learned from training data provides equivariance to continuous groups without discretization or summing over irreps [30]. An evidence lower bound for the marginal likelihood of the parameters of affine invariance groups allows to learn the present symmetry transformation parameters directly from training data using Monte Carlo sampling. However, this bound is restricted to single-layer DNNs and continuous groups [168]. The built-in equivariance constraints of equivariant DNNs can be relaxed by adding unconstrained residual paths. While the network is biased towards the equivariant solution, it can benefit from the unconstrained model capacity when the desired symmetry is misspecified [47]. *Partial* G-Convs achieve approximate equivariance to subsets of groups G. Therefore, a Monte Carlo approximation of the G-Conv is calculated where the group elements are sampled from learnable probability distributions. The desired equivariance subset can be learned per layer, allowing varying constraints throughout the DNN [142]. Another generalization of the G-Conv allows to relax strict symmetry constraints using non-stationary kernels depending on the absolute input group-element. During training, the layer can interpolate between a non-equivariant linear product, a strict equivariant convolution and a strict invariant map which allows to relax the built-in equivariance [167].

Moreover, meta-learning allows to learn equivariance to symmetry groups shared among multiple tasks via parameter-sharing patterns [189]. For time series predictions, restricting the transitions between time steps to be linear uncovers symmetries by block-diagonalization the transition matrices [112].

2.3.4 Measuring Equivariance

Another important aspect is to measure the in- or equivariance properties of DNNs. The Local Equivariance Error measures the equivariance properties of DNNs based on the Lie derivative, showing that a lower equivariance error correlates with a better task performance. Notably, non-equivariant models such as VITs achieve lower equivariance errors than their equivariant counterparts with enough data and the correct training procedure, as the non-linearities within equivariant models are often break exact equivariance

guarantees due to aliasing effects [62]. The G-empirical equivariance deviation measures a DNN's failure to achieve in- or equivariance. Models trained via data augmentation only achieve global in- or equivariance, while restricted architectures obtain layer-wise equivariance and achieve better generalization to out-of-distribution setups [97]. The Lie group generators calculated using the training data and the derivative of the model quantify the learned symmetries of DNNs without specifying a set of transformations beforehand. Due to its expensive calculation, this method is limited to small-scale MLPs, where larger models with fine-tuning achieve tighter equivariance guarantees [113].

2.3.5 Summary & Analysis

This section reviewed related work that incorporates in- or equivariance into DNNs based on architecture constraints (Section 2.3.1) and adapted training procedures (Section 2.3.2), automatically discovers in- or equivariance (Section 2.3.3) or measures the desired properties (Section 2.3.4). In the following, we summarize these approaches and briefly analyse their advantages and limitations.

Restricting DNN architectures enforces geometrical prior knowledge to DNNs with mathematical guarantees, thereby reducing the training algorithm's search space. *G-Convs* enforce equivariance to transformation groups G to a learnable linear map. By applying G-Convs in CNNs, the sample efficiency is improved, which enables state-of-the-art performance in tasks where in- or equivariance is required. However, G-Convs suffer from a computational overhead and are limited to transformations that can be modelled as (often finite) groups. This excludes transformations such as viewpoint changes in images or lighting. Furthermore, G-CNNs rely on pooling to achieve invariant representations, which discards available information. Incorporating in- or equivariance to multiple transformation groups at once is another area which needs further investigation.

Non-linear equivariant maps, such as SA, extend equivariance guarantees beyond G-Convs. Similarly to G-Convs, these approaches increase the sample efficiency, at the downside of a higher computational complexity compared to their non-equivariant counterparts. Approaches with *fixed filters*, like the *scattering transformation*, mathematically guarantee in- and equivariance but restrict DNNs from learning task-specific, informative representations.

Capsules disentangle object representations into an invariant object information and

equivariant pose parameters, allowing to model transformations such as viewpoint-changes or lighting not covered by transformation groups. However, the desired properties are learned rather than guaranteed. Adaptations leveraging group symmetry are restricted in terms of transformation efficiency or regarding the equivariant pose information. Additionally, capsule networks are comparably hard to train as they rely on an intricate routing algorithm during inference and optimization. Consequently, they fail to reach the baseline performances provided by CNNs or Transformers.

Adapting the *training procedure* rather than restricting the architecture provides a flexible alternative to learn in- or equivariance. *Data augmentation* is straightforward to implement, adaptable towards a variety of transformations and can be combined with other approaches. However, it does not provide any mathematical guarantees and thus fails to restrict the solution space of the learning algorithm. Additionally, the equivariance is only learned for the network as a whole, not guaranteed at each layer. Enhancements like *symmetry regularization* compute additional losses to increase the transformation robustness of the learned representations,but add training overhead by processing transformed pairs of inputs.

Learned Feature Transformations, such as STNs, re-transform the inputs to their untransformed, canonical form using estimated parameters. Thus, invariance is learned from data rather than enforced manually. Variants that combine this approach with equivariant convolutions achieve learned invariance alongside additional equivariance guarantees. Feature transformation layers can easily be added to existing DNNs and be applied to intermediate representations. However, while not restricted to transformation groups, their invariance depends on correctly estimating the transformation parameters and is thus not mathematically guaranteed.

Automatic symmetry discovery methods learn and embed in- or equivariance properties into DNNs directly from training data. This mitigates the problem of overly restricted models due to misspecified invariances and allows to incorporate invariance when the required prior knowledge is not available. Nevertheless, models with learned equivariance fail to outperform methods with correctly specified built-in equivariance.

Equivariance Measures evaluate, if the desired symmetry properties are sufficiently enforced by a DNN or its layers. These measures allow to identify weak spots of in- or equivariant DNNs, e.g., pooling layers that may be detrimental to invariance guarantees due to aliasing effects.

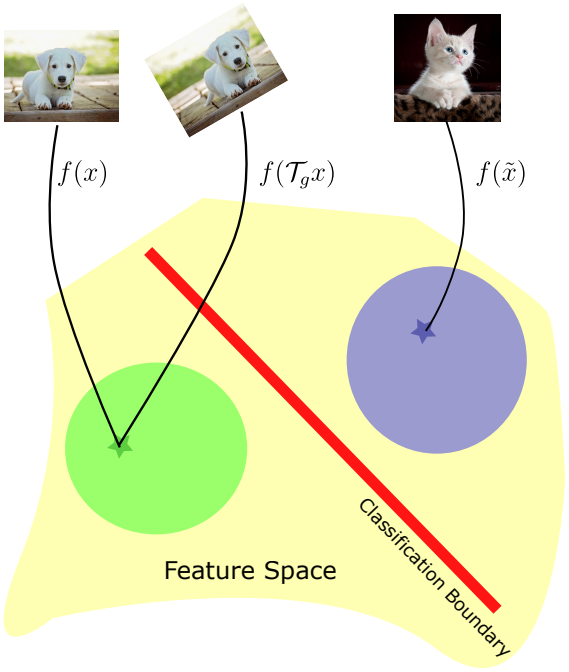


Figure 2.13: An invariant, but separable feature space for a classification DNN distinguishing cats from dogs. Applying an image transformation $\mathcal{T}_g$ does not change the computed feature $f(x) = f(\mathcal{T}_g x)$. However, the features of different classes are separated $f(x) \neq f(\tilde{x})$ which allows for a perfect classification with a linear classification boundary (red). This illustration incorporates license-free images of a dog [130] and a cat [164].

Overall, geometrical prior knowledge enhances state-of-the-art DNNs by improving the sample efficiency of the training algorithm and the interpretability of the computed responses. While the related literature provides manifolds of possibilities to enforce prior knowledge about symmetry transformations, we specifically focus on achieving *invariant representations for image classification networks* in our contributions (Chapters 3 - 5).

2.4 Invariant Representations in Neural Networks for Classification

Let us first recall, why an *invariant* feature space is beneficial for DNNs, e.g., employed for image classification. An exemplary ideal feature space for image classification that distinguishes cats from dogs is illustrated in Figure 2.13.

Ideally, the feature values do not change, if an image of a dog is rotated, translated or flipped, i.e., the feature space is invariant to these transformations. Contrarily, an image

of a cat is mapped to a distinct area where all variations of cats are encoded. In other words, an ideal feature space for classification is invariant w.r.t. *intra-class variations* but separable w.r.t. *extra-class variations*. Only the information required to distinguish different object classes is contained while all irrelevant information is dismissed. A linear classifier can then perfectly separate the image classes in the invariant feature space [148]. Enforcing invariance via geometrical prior knowledge rather than learning it simplifies distinguishing relevant from irrelevant information [174].

While an invariant feature space simplifies the final classification problem, it is also a rather restrictive property, as the information along the axes of invariance are entirely discarded. This means that the knowledge about the transformation itself is lost. Consequently, invariance is only desirable for deeper layers of DNNs close to the final classification layer. In contrast, earlier layers benefit from equivariance where local, lower-level features such as edges or corners need to be detected in different orientations [17, 173]. Additionally, the information about those orientations is useful, e.g., to combine multiple edges to object contours. Thus, it makes sense to learn a equivariant feature space that is transferred to invariance before the final, global object classification [17].

The related work presented in Section 2.3 achieves in- or equivariance within DNNs in two different ways. Adaptive training methods such as data augmentation or regularization terms yield approximate in- or equivariance globally for the entire network [97]. On the other hand, equivariant learnable layers such as G-Convs enforce equivariant features at each layer [17]. Generally, G-Convs outperform the approximate approaches due to their stricter, layer-wise properties. Especially for the case of invariant image classification, CNNs based on the regular G-Convs with transformation-steerable filters achieve state-of-the-art results [160, 162, 173, 176]. Additionally, G-Convs are the most general linear map that guarantees equivariance [93]. Consequently, they provide the core component towards building state-of-the-art DL models that enforce geometrical priors.

For problems where *invariance* is required, the learned equivariant features need to be transferred towards an invariant representation. In related equivariant DNN literature, the transfer to invariance is achieved through *max pooling* among both the spatial and group domains, see Figure 2.14 [17,160,162,173,176]. While max pooling guarantees invariance, it also destroys valuable information encoded in the equivariant feature representation.

Consequently, our contributions investigate how to achieve state-of-the-art invariant representations within DNNs. We combine equivariant G-Convs with a dedicated opera-

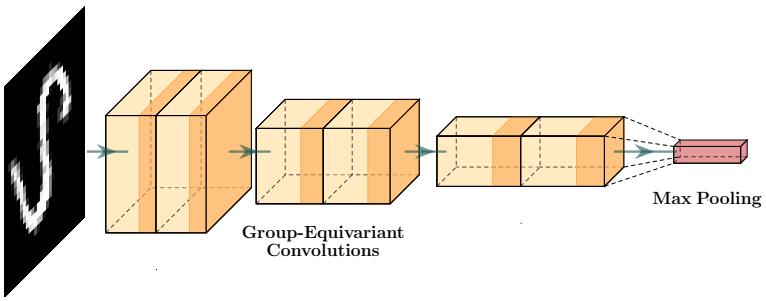


Figure 2.14: The current state-of-the-art to learn *invariant representations* includes learnable group-equivariant convolutional layers (yellow) followed by max pooling among the group and spatial domains (red).

tion that enhances the transfer from equi- to invariance beyond the capabilities of pooling. Thereby, we effectively improve the second main building block of state-of-the-art DNNs enforcing geometrical priors via invariance. By adding *targeted model capacity* using a learnable map, the available *information* encoded in the equivariant feature space is *enhanced* rather than destroyed during the *transfer to invariance*. This promises a better classification performance, especially when training data is scarce. Specifically, we investigate to use variants of *Invariant Integration (II)* for this purpose [132–134], which we will introduce in Section 2.4.1.

2.4.1 Invariant Integration

Invariant Integration is an approach to construct a *complete feature space* w.r.t. a transformation [78, 120, 148, 149]. In the context of ML, it has first been applied to pattern recognition tasks in reference [148]. A feature space $\mathcal{F}$ is *complete*, if all equivalent patterns according to a transformation group G acting on the input space X are mapped to the same point in $\mathcal{F}$, whereas all non-equivalent patterns are mapped to distinct points. This means that the feature space is invariant but preserves discriminative capacities along other modes of variation, as shown in Figure 2.13. Mathematical conditions for the existence of complete feature spaces in pattern recognition tasks are provided in reference [148].

2.4.1.1 Mathematical Definition

II computes a complete feature space for finite groups based on the *group average* over a polynomial function f

$$A[f](\mathbf{x}) := \int_{g\in G} f(\mathcal{T}_g \mathbf{x}) \mathrm{d}\mu(g), \tag{2.47}$$

where $\mathrm{d}\mu$ denotes the Haar measure with $\int_G \mathrm{d}\mu(g) = 1$ [148, 149].

For finite groups of order $|G|$, the group average is reduced to a sum over all group elements [149, 150]

$$A[f](\mathbf{x}) := \frac{1}{|G|} \sum_{g\in G} f(\mathcal{T}_g \mathbf{x}). \tag{2.48}$$

Theorem 2.4.1. *Let G be a group, let $\mathbf{x} : \mathbb{R}^n \to \mathbb{R}$ be an input. Given a polynomial function f and the left group action $\mathcal{T}_g$ acting on the input via $\mathcal{T}_g \mathbf{x}(u) = \mathbf{x}(g^{-1}u)$ with $u, g \in G$. Invariant Integration is invariant to transformations of the group G, i.e., $A[f](\mathcal{T}_g \mathbf{x}(u)) = A[f](\mathbf{x})$.*

Proof.

$$\begin{aligned} A[f](\mathcal{T}_g \mathbf{x}) &= \int_{h\in G} f(\mathcal{T}_h \mathcal{T}_g \mathbf{x}) d\mu(h) && (2.49)\\ &= \int_{hg\in G} f(\mathcal{T}_h \mathcal{T}_g \mathbf{x}) d\mu(hg) && (2.50)\\ &= \int_{\tilde{g}\in G} f(\mathcal{T}_{\tilde{g}} \mathbf{x}) d\mu(\tilde{g}) && (2.51)\\ &= A[f](\mathbf{x}), && (2.52) \end{aligned}$$

□

where the change of variable in the integral is valid due to left invariance of the Haar measure $d\mu(g) = d\mu(hg)$.

One important design-choice for II is the polynomial function f that needs to ensure the separability of the feature space. The set of all possible monomials $m(\mathbf{x})$ with monomial exponents b_i provides a good choice for the function f as it forms a finite basis of the signal space representing the complete set of invariants [120, 149].

$$m(\mathbf{x}) = x_1^{b_1} x_2^{b_2} \ldots x_N^{b_N} \quad \text{with} \quad b_1 + b_2 + \ldots + b_N \leq \|G\|, \tag{2.53}$$

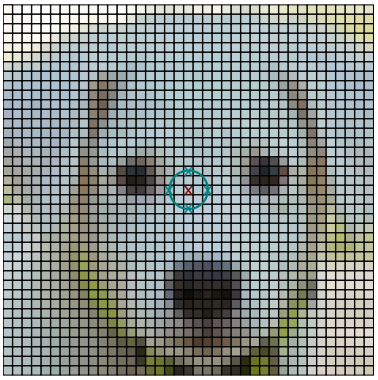


Figure 2.15: SO(2)-II using a monomial with $d_1 = 0, d_2 = 2$ and $\Phi = 4$ rotations. All values processed by the group average (blue) lie on a circle around the centre point (red) [150].

Limiting the number of product factors via a *monomial order K* results in

$$m(\mathbf{x}) = \prod_{i=1}^{K} x_i^{b_i} \quad \text{with} \quad \sum_i b_i \leq |G|. \tag{2.54}$$

The upper bound for the number of all possible monomials with order K is $\binom{K+|G|}{K}$ [149]. While this bound represents a basis for the full set of invariants, calculating it is computationally inefficient, especially for larger groups. For practical applications, it is thus important to reduce the number of monomials by carefully selecting a meaningful subset. Thereby, the amount of required compute is decreased while the separability of the invariant feature space is maintained [150]. Weak G-commutative maps can be applied to the feature space to further improve the separability. Those maps maintain the invariance properties while enhancing the distance between non-invariant modes of variation [148].

Generally, II and the group average are closely related to G-Convs (cf. Equations 2.42 and 2.48). Particularly, one special case of II is equivalent to a lifting G-Conv followed by global Average Pooling, which we will prove in Section 4.3.2.1.

2.4.1.2 Application to SO(2)

II with monomials has been applied to two-dimensional images for the group of translations and rotations [150]. For this case, it is straightforward to use pixels and their neighbours for the monomial factors x_i, $i = 1, \ldots, K$ to embed the locality of the regular

grid into the invariant feature calculation. Consequently, the monomials can be defined via the distance $d_i = \sqrt{(u_i - u_1)^2 + (v_i - v_1)^2}$ between the centre pixel (u_1, v_1) and the neighbour (u_i, v_i) with $d_1 = 0$. For Φ discrete 2D rotations and UV translations, II with monomials based on neighbouring pixels results in [150]

$$A[m](x) = \frac{1}{UV\Phi} \sum_{u,v} \sum_{\phi} \prod_{i=1}^{K} \mathbf{x}(u + d_i \cos(\phi), v + d_i \sin(\phi))^{b_i}. \tag{2.55}$$

The inner sum applies the product of the centre pixel with a specific neighbour to all neighbours lying on a circle of radius d_i, which is shown in Figure 2.15. The outer sum computes the translation average using each input position as the centre [150].

2.4.1.3 Invariant Feature Extraction with Invariant Integration

SO(2)-II, as introduced in the previous section, can be applied to greyscale images to extract invariant features for object classification using a nearest neighbour classifier. The extracted features are globally invariant and stable to local transformations including small object overlaps. However, while the obtained features are invariant, additional maps are needed to improve the separability of the feature space, rather than applying II directly to the input [150].

II can be expanded to non-compact groups and continuous signals. Generally, combining invariance to multiple compact subgroups induces invariance to more general groups. For example, a quotient of homogeneous features invariant to the special unitary group $\mathrm{SU}(n, \mathbb{C})$, which includes all unitary, complex-valued $n \times n$ matrices with determinant 1, results in features invariant to the general linear group $GL(n, \mathbb{C})$ [149].

Applying II on top of Fourier transformed input images achieves invariance to multiple transformations at once. The Fourier descriptors of human contours are invariant to colour changes, starting point, rotation and translation. Applying rotation-II in the Fourier space guarantees invariance w.r.t. anthropomorphic changes, i.e., different size and build among humans. Overall, this *chain of invariant transformations* achieves an induced invariance in the input domain and can be used for human event detection with a Support Vector Machine (SVM) [21]. Further, II with monomials can be applied beyond images to Fourier transformed audio signals to extract robust, invariant features for speaker-independent speech recognition [114–116].

The group average (Equation 2.47) is further used beyond the II framework. *TI-Pooling* is a special case of the group average, where a CNN replaces the polynomial function f. For each transformation the network should be invariant to, the network computes a forward-pass with shared weights. The responses are then aggregated using max pooling. While this approach guarantees transformation-invariance, it is computationally expensive as it requires $|G|$ forward passes per input, both during training and inference [99]. For larger, intractable groups the group average can be solved by integrating over a subset of the group called *frame*. The *frame average* allows to apply the group average beyond two-dimensional data, e.g., to point clouds and graph DNNs using the entire DNN instead of the polynomial f [131]. The group average has also been employed to prove a non-zero improvement of the generalization ability of in- and equivariant DNNs, if the embedded equivariance is present in the target distribution. Based on this insight, a regularization term is provided to learn the desired invariance [37].

2.4.2 Applying Invariant Integration within Deep Neural Networks

As demonstrated in this chapter, the state-of-the-art method to embed *geometrical prior knowledge* into DNNs is to enforce *transformation equivariance* to the learned representations using G-Convs. If *invariance* is desired, the equivariant representation can be reduced to an invariant one via pooling. While pooling guarantees invariance, it destroys parts of the information encoded in the learned equivariant features.

Contrarily, II constructs a complete, invariant feature space w.r.t. transformation groups G. So far, II has mostly been used to extract invariant features used in conjunction with traditional ML classifiers such as SVMs. Nevertheless, it provides a general framework to obtain provably invariant features based on any input representation.

Consequently, this thesis investigates how to combine G-Convs with II in invariant DNNs. We aim to leverage II for the transfer from equi- to invariance within DNNs for object classification in CV. Thereby, we enhance the potential of equivariant features obtained via G-Convs with a dedicated operation that enforces the desired invariances. Rather than destroying valuable learned information with pooling, *targeted model capacity* is added to the DNN during the transfer from equi- to invariance. This promises to further improve the sample efficiency of the learning algorithm for tasks where invariance is required such as image classification [132–134].

Chapter 3

The Invariant Integration Layer

In our first contribution, we show how to incorporate prior knowledge to a DNN architecture in a principled manner. We enforce feature space invariances using a novel layer based on II. This allows us to construct a complete feature space invariant to finite transformation groups. We apply our proposed layer to explicitly insert invariance properties for vision-related classification tasks, demonstrate our approach for the case of rotation invariance and report state-of-the-art performance on the Rotated-MNIST dataset. Our method is especially beneficial when training with limited data. This chapter summarizes our paper *"Invariant Integration in Deep Convolutional Feature Space"* [132]. The following sections will not further cite our own contribution.

3.1 Invariant Integration in Deep Convolutional Feature Space

As motivated in Section 2.4.2, we aim to use II as a dedicated operation to enhance the transfer from equi- to invariance in DNNs. Therefore, we propose a novel DNN layer based on II and embed it into a CNN architecture. The II layer builds upon an equivariant backbone to yield an invariant feature representation (see Figure 3.1). We demonstrate that the backpropagation algorithm is applicable w.r.t. the II layer's parameters and input. Hence, its parameters as well as preceding layers can be optimized en bloc. We test our method on the Rotated-MNIST dataset [100] and demonstrate state-of-the-art performance. The largest performance gains are achieved in the limited data regime, where only few labelled training samples are available.

3.2 Related Work

The work related to the II layer can be split into rotation-equivariant CNNs, discussed in Section 2.3.1.1 and II, as introduced in Section 2.4.1.

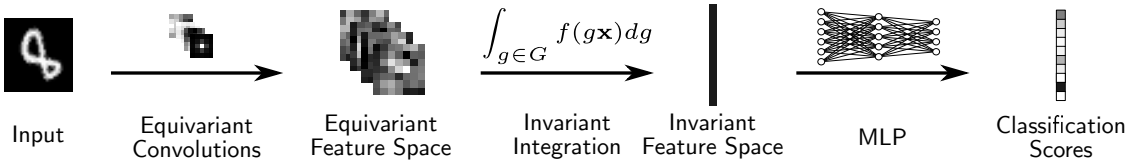


Figure 3.1: (Figure 1 in Rath & Condurache [132]) We apply II in a learned equivariant feature space to form a feature space invariant to rotations and translations and use a MLP for classification.

3.3 A DNN Architecture based on Invariant Integration

In this section, we introduce a DNN architecture that combines G-Convs to extract rotation-equivariant features (cf. Section 2.3.1.1) with II to obtain an invariant feature space (cf. Section 2.4.1). The proposed architecture is visualized in Figure 3.1

3.3.1 Rotation-Invariant Integration in Image Space

We apply II using monomials to the group $\mathrm{SE}(2)$ covering rotations and translations in 2D image space (cf. Equation 2.55) [150]. Our input is defined as $\mathbf{x} \in \mathbb{R}^{H\times W\times C}$, where H, W and C are the image's width, height and number of channels. We apply the group average independently to each channel $c = 1,\dots,C$ and reformulate to

$$A[f](\mathbf{x}) = \sum_u \sum_v \int_\phi m(\mathbf{x}(u,v;\phi))\, d\phi \tag{3.1}$$

$$= \sum_u \sum_v \int_\phi \prod_{i=1}^{K} \mathbf{x}(u + d_i \sin(\phi), v + d_i \cos(\phi))^{b_i}\, d\phi, \tag{3.2}$$

with the monomial distances d_i, the pixel offsets u and v and the orientation angle ϕ. The output is of dimension $\mathbb{R}^{C\times M}$, where M is the number of monomials. Contrarily to previous work, we choose to apply rotation-II directly in the convolutional feature space obtained with G-Convs rather than directly on the input images.

3.3.2 Monomial Selection

Since computing all possible monomials is computationally expensive, the question arises, how to choose a meaningful subset. While the exponents b_i can be optimized using

backpropagation (see Chapter 3.3.4), choosing the monomial orders K and their distances is non-trivial.

We use an iterative approach for feature selection called *Feature Finding Neural Network (FFNN)* proposed by Gramss [61]. It has successfully been used to select the monomials for II in automatic speech recognition by Mueller and Mertins [116] and is based on iteratively ranking monomial combinations according to the *LSE* of a linear classifier. While using a linear classifier facilitates good generalization ability, the LSE can be computed in closed-form, which enables an efficient calculation.

We adapt this method by utilizing the closed-form solution of a linear classifier to rank the feature combinations according to the classifiers' validation accuracies instead of the LSE. The iterative selection is stopped, if the validation accuracy has not improved within the past ten iterations.

3.3.3 Architecture and Training Process

The architecture for a classification network using the proposed II layer is shown in Figure 3.1. The II layer is used to construct an invariant feature space w.r.t. transformations of the convolutional feature space it acts on. To ensure that the whole DNN is invariant, the convolutional feature space must be equivariant w.r.t. transformations of the DNN's input. Thus, we use an equivariant backbone built out of G-Convs followed by max pooling among the group elements. Rotations of the input induce rotations in the resulting feature space, i.e., it is equivariant to rotations. Consequently, we apply the II layer on top to build an complete invariant feature space (see Figure 3.1). Finally, we use fc layers to obtain the classification scores. In comparison, the G-Conv-based networks proposed in [176, 181] use max- or average-pooling over both the group and the spatial domain to obtain invariance based on the equivariant convolutional feature space.

During training, we first train the baseline network without the II layer. Thereafter, we apply II on the features of the last convolutional layer and select M monomials using our iterative selection approach. Finally, we re-train the entire network including the II layer and the preceding equivariant convolutional layers.

3.3.4 Backpropagation for Invariant Integration Layers

To enable using II for the construction of a complete feature space in combination with DNNs it is inevitable that the proposed feature transformation is differentiable w.r.t. both its parameters and inputs. The gradients of a monomial w.r.t. a single input x_j and w.r.t. the learnable exponents b_j are respectively defined as

$$\frac{\partial m(\mathbf{x})}{\partial x_j} = b_j x_j^{b_j-1} \prod_{i=1, i\neq j}^{K} x_i^{b_i}, \tag{3.3}$$

$$\frac{\partial m(\mathbf{x})}{\partial b_j} = \log(x_j) x_j^{b_j} \prod_{i=1, i\neq j}^{K} x_i^{b_i}. \tag{3.4}$$

Investigating these formulas, it is evident that we need to enforce $x_i \neq 0 \; \forall i$, because

$$\lim_{x_i \to 0} \frac{\partial m(\mathbf{x})}{\partial x_j} = 0 \quad \text{and} \quad \lim_{x_i \to 0} \frac{\partial m(\mathbf{x})}{\partial b_j} = 0, \tag{3.5}$$

would lead to vanishing gradients backpropagated to earlier layers and to the exponents of the II layer, respectively.

We further restrict the input values to $\mathbb{R}^+$ because multiplying by a negative number alters the sign of the output, hence leading to great shifts in output space and thus unstable gradients. Finally, we note that the identity element of a multiplication is 1, while a multiplication with values close to 0 would eradicate the result. Following the aforementioned insights, we choose to shift our input features

$$\tilde{\mathbf{x}} = \max(\varepsilon, \mathbf{x} - x_{\min} + 1), \tag{3.6}$$

with $0 < \varepsilon \ll 1$. In our implementation, we shift the input per channel, while $x_{\min}$ is the minimal value that occurred within that channel when processing the training data. Lastly, we apply the max-function to avoid values smaller than ε during inference.

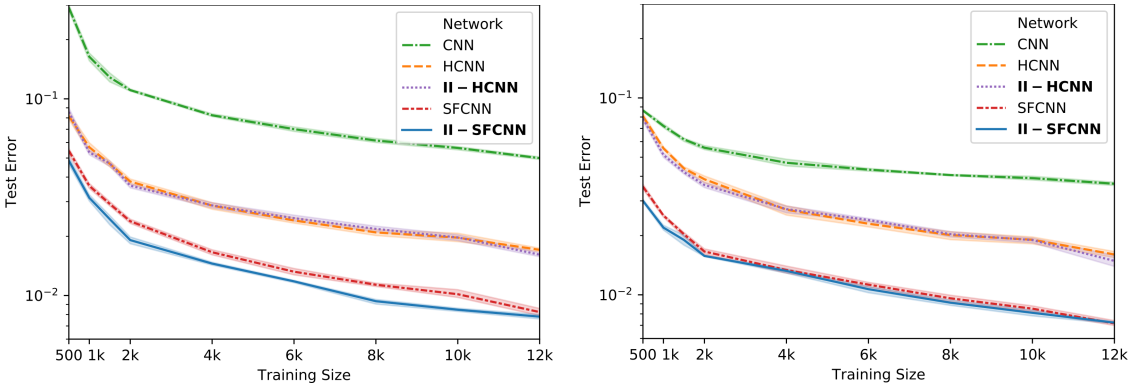


Figure 3.2: (Figure 2 in Rath & Condurache [132]) Mean Test Error (MTE) using limited training data without (*left*) and with (*right*) data augmentation with random rotations. The transparent corridors represent the results' standard deviation. Methods with II-prefix replace the final spatial pooling layer of the corresponding network with II to achieve invariance.

3.4 Experiments & Discussion

We use our approach to enforce rotation invariance for the recognition of randomly rotated hand-written digits on the Rotated-MNIST dataset [100]. It contains 10k training, 2k validation and 50k test images. We optimize the HPs on the validation set. Thereafter, we retrain on the training and validation sets to report the final results on the test set, similarly to references [176, 181].

For the equivariant backbone, we use two different approaches that are equivariant to rotations in image space. First, we apply harmonic convolutions (Harmonic Filter CNN (HCNN)) proposed by Worrall et. al which employ G-Convs based on the irreps of the continuous 2D rotation group [181]. We use the modulus of the final complex convolutional layer as the input to our II layer, which corresponds to the trivial representation (cf. Section 2.2.4). Secondly, we use the Rotation-Equivariant Steerable Filter CNN (SFCNN) proposed by Weiler et. al which uses regular G-Convs with steerable filters [176]. We apply max pooling along the angle dimension of the final convolutional layer to again achieve the trivial representation and apply channel-wise II. We empirically choose to use $M = 5$ monomials and integrate over 8 angles using bilinear interpolation to obtain a good trade-off between performance and computational complexity.

We compare different approaches using the MTE over ten independent training runs.

Table 3.1: (Table 1 in Rath & Condurache [132]) MTE [%] on Rotated-MNIST with and without using augmentations with random rotations. Methods with II-prefix expand the corresponding network by replacing the final spatial pooling layer with II to achieve invariance.

Method	Augmentation	
	×	✓
CNN [17, 181]	5.130	3.772
HCNN [181]	1.730	1.606
II-HCNN	1.548	1.377
SFCNN [176]	0.880	**0.714**
II-SFCNN	**0.799**	0.722

Table 3.1 shows the results obtained by training on the entire training and validation set with and without data augmentation using random input rotations. We evaluate a standard CNN as well as the HCNN and the SFCNN with and without the II layer. When combining II with the HCNN, we achieve a relative improvement of 10.52 % without and 14.26 % with data augmentation. The II-SFCNN improves the baseline by 9.2 %. However, when training with augmentations, the SFCNN achieves the best performance, but only by a small margin.

Figure 3.2 shows the MTE and the standard deviation when training with limited data. When using only a subset of the available training data, we achieve significant improvements over the state-of-the-art baseline algorithms. It seems that enforcing the sought invariances on top of an equivariant feature space by using II instead of the simpler pooling operation is particularly beneficial when only limited data is available. We believe this is related to the fact that a small training set typically does not include the variability needed to properly capture the sought invariances without explicitly enforcing them. Moreover, our II layer adds targeted model capacity rather than discarding valuable parts of the information available in the learned features. Additionally, we observe that the performance boost is bigger when applying the II layer to the SFCNN because its feature representation is equivariant to sampled rotations, while the HCNN guarantees equivariance to continuous rotations. Enforcing the invariance with the II layer seems to mitigate those sampling effects.

3.5 Conclusion

In this initial contribution, we proposed a novel layer based on II which allows us to incorporate prior knowledge expressed as the need to obtain features invariant to certain geometrical transformations of the input, e.g., rotations. We apply it to enforce rotation invariance in a CNN architecture. We show competitive performance when training on the full dataset and state-of-the-art results in the low data regime. This indicates that our approach enables a better transition from equivariant to invariant features than pooling, specifically when the learned feature representation is not exactly equivariant, e.g., due to sampling effects. Our method is especially helpful, when training with fewer training samples, which indicates that the added, targeted model capacity increases the expressibility compared to spatial max pooling while still guaranteeing invariant features.

Chapter 4

Scaling the Invariant Integration Layer Towards Real-World Applications

In our second contribution, the II layer is adapted to enable adding it to DNNs based on a more streamlined training procedure. Therefore, we propose to replace the iterative *monomial selection* with pruning-based approaches. Moreover, we investigate replacing the monomials with alternative functions f. The investigated candidates are well-established functions from DL literature such as a Multi-Layer Perceptron (MLP), a weighted sum (WS) or self-attention (SA). This enables to apply the II layer within large-scale CNN architectures such as Wide-ResNets (WRNs) [186]. On various real-world image classification tasks, using II-enhanced DNNs improves the sample efficiency. This is demonstrated by state-of-the-art results on SVHN, CIFAR-10 & STL-10, specifically in the limited data domain. This chapter summarizes our contribution *"Improving the Sample-Complexity of Deep Classification Networks with Invariant Integration"* [133], which will not further be cited in the following sections.

4.1 Streamlining Invariant Integration within Deep Neural Networks

In chapter 3, we have shown that II provides a principled approach to embed geometrical prior knowledge into DNN architectures. Specifically, explicitly enforcing rotation-invariance by means of II instead of using a global pooling operation among the spatial dimensions improves the sample efficiency of rotation-equivariant CNNs used for classification tasks despite adding parameters, hence improving generalization [132]. All previous methods including our method from Chapter 3 used II in combination with monomials which are hard to optimize with common DNN training methods. The monomial parameters are chosen using an iterative method based on the LSE of a linear classifier before the DNN can be trained. Thus, the method so far relies on an expensive pre-training step that reduces its applicability towards real-world problems.

Consequently, in this chapter we investigate how to adapt the rotation-II framework in order to improve its applicability. We explicitly investigate the transition between in- and equivariant features within group-equivariant CNNs for the case of 2D rotations and replace the spatial pooling operation by II. We start by introducing a *novel monomial selection* algorithm based on *pruning*. We then replace the monomials altogether using simple, well-known DNN functions such as a weighted sum, a MLP or self-attention instead. Both variants *streamline the training* algorithm of II-enhanced DNNs as the *monomial selection is simplified or no longer required*. In addition, a connection between II using a weighted sum and regular G-Convs is established.

Our second contribution is the first application of II within large-scale CNN architectures such as WRNs used for real-world image classification [186]. We build upon our previous contributions and combine rotation-II with an equivariant backbone based on steerable G-Convs. Our II-enhanced networks achieve state-of-the-art results in the limited and full data regime of Rotated-MNIST and SVHN, where rotations are responsible for most of the relevant variability. Furthermore, we demonstrate very good performance in limited data regimes on CIFAR-10 and STL-10, where other modes of intraclass variation are present besides rotations. Thus, II successfully *improves the sample efficiency* in practically relevant problem settings demonstrating its general applicability.

4.2 Related Work

Our second contribution builds upon the II layer introduced in Chapter 3 [132]. The related work includes group-equivariant CNNs, see Section 2.3.1.1 and II, presented in Section 2.4.1.

4.3 Method

In chapter 3, II was first applied as a layer within a deep equivariant neural network. First, let us recall the general formulation of the group average

$$A[f](\mathbf{x}) = \int_{g \in G} f(\mathcal{T}_g \mathbf{x}) d\mu(g). \tag{4.1}$$

Specifically, the introduced II layer was applied to the group of two dimensional

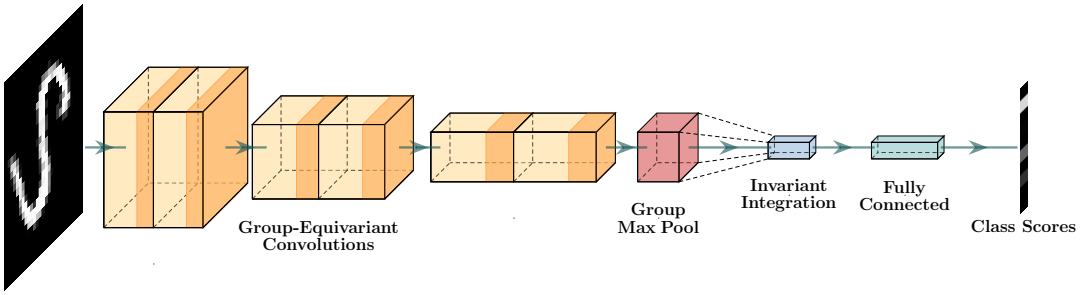


Figure 4.1: (Figure 1 in Rath & Condurache [133]) General invariant DNN architecture using II. The architecture includes group convolutions (orange) and group pooling (red) creating an equivariant representation, an II layer enforcing invariance (blue), and fc layers (green).

rotations and translations using monomials for the function f

$$A[m](\mathbf{x}) = \frac{1}{UV\Phi} \sum_{u,v,\phi} \prod_{i=1}^{K} \mathbf{x}(u + \cos(\phi)d_i, v + \sin(\phi)d_i)^{b_i}. \tag{4.2}$$

In this section, we will first discuss, how to select an appropriate subset of monomials in Equation 4.2. Thereafter, we will present alternatives for the function f within the group average (Equation 4.1) with the goal of avoiding the monomial selection altogether.

We apply our layer within DNNs consisting of a backbone of regular G-Convs with steerable filters. Group max-pooling is applied to achieve an equivariant feature space transforming according to the trivial representation (cf. Section 2.2.4) which is then transferred to invariance with the II layer. Finally, fc layers are employed to calculate the classification scores. This general invariant DNN architecture is depicted in Figure 4.1.

4.3.1 Monomial Selection

While the II layer boosts the sample efficiency when learning invariant representations, it introduces additional parameters which need to be carefully designed. One of them is the selection of a meaningful set of parameters d_i and b_i that define the monomials needed to obtain the invariant representation. This step is necessary since the amount of all possible monomials satisfying $\sum_i b_i \leq |G|$ is too extensive.

In chapter 3, an iterative approach is used based on the LSE solution of a linear

classifier. While this solution is easy to compute, the iterative selection is time-consuming and computationally expensive. Moreover, the base network without II needs to be pre-trained which requires additional computations and prevents training the full network from scratch.

Consequently, we investigate alternative approaches for the monomial selection. Two approaches are introduced and explained in the following that both enable training the network end-to-end from scratch and are computationally inexpensive compared to the iterative approach.

4.3.1.1 Random Selection

First, we randomly select the n_m monomials by sampling both the exponents and distances from uniform distributions. This approach is fast because it only requires a single random sampling operation and serves as a baseline to evaluate other selection methods.

4.3.1.2 Pruning Selection

Alternatively, the monomial selection can be formulated as selecting a subset containing $n_m \ll M$ possible monomial parameters. Consequently, it is closely related to the field of pruning in DNNs [177]. Pruning aims to reduce the amount of connections or neurons within DNN architectures in order to decrease the computational complexity while maintaining the best possible performance. We compare two pruning algorithms: a magnitude- and a connectivity-based approach.

Magnitude-Based Approach Han et al. determine the importance of connections in DNNs by pre-training the network for τ epochs and sorting the weights of all layers by their magnitude $|w_{ij}|$ [65]. This approach is applied iteratively keeping the γ highest-ranked connections at each step $i = 1, \ldots, I$ until the final pruning-ratio γ^I is reached.

Since we aim to prune monomials instead of single connections, we sum the absolute value of weights connected to a single monomial, i.e., all weights of the first fc layer following the II layer

$$s_j = \frac{1}{C_i C_o} \sum_{k=1}^{C_i} \sum_{l=1}^{C_o} |w_{klj}|, \tag{4.3}$$

where C_i is the number of input channels before II is applied, C_o is the number of neurons

in the fc layer and j selects the connections belonging to the j^{th} monomial. Following reference [65], we apply the pruning iteratively. In each step, we keep the n_i monomials with the highest calculated score s_j. We do not re-initialize our network randomly in between iterative steps but re-load the pre-trained weights from the previous step.

Connectivity-Based Approach We examine a second pruning approach based on the initial connectivity of weights inspired by reference [104]. All monomial output connections are multiplied with an indicator mask $c \in \{0,1\}^M$ using the Hadamard product $c \odot w_l$. Here, w_l is the weight vector of the fc layer following the II step. Setting an individual value c_j to zero results in deleting all connections $w_{l,j}$ connected to monomial j. Consequently, the effect of deactivating a monomial can be estimated w.r.t the training loss L by calculating the *connection sensitivity*

$$s_j = \frac{\partial L(c, w_l; \mathcal{D})}{\partial c_j} \tag{4.4}$$

for each monomial using backpropagation. The n_m monomials with the highest connection sensitivity are kept. The derivative is calculated using the training dataset $\mathcal{D}$.

The connectivity-based approach can either be used directly after initializing the DNN, or after some pre-training steps. Additionally, it can either be used iteratively or in a single step.

4.3.1.3 Initial Selection

We investigate two different approaches for the initial selection of $M \gg n_m$ monomials. In addition to a purely random selection, we design a catalogue-based initial selection in which all possible distance combinations are guaranteed to be involved in the initial set. In both cases, we sample the exponents randomly from a uniform distribution.

4.3.2 Replacing the Monomials

In addition to the novel monomial selection algorithm, we investigate alternatives for the monomials used to calculate the group average (Equation 4.1). We apply the proposed functions to the group of discrete 2D rotations and compare monomials to well-utilized DNN functions such as a WS, a MLP and SA.

4.3.2.1 Weighted Sum

One possibility for f is to use a weighted sum where the weights are a learnable kernel ψ applied at each group element $g \in G$ transforming the input $\mathbf{x} : \mathbb{Z}^2 \to \mathbb{R}$. We obtain

$$A[\mathrm{WS}](\mathbf{x}) = \frac{1}{|G|} \sum_{g \in G} \sum_{y \in \mathbb{Z}^2} \mathbf{x}(y)\, \mathcal{T}_g\, \psi(y) = \frac{1}{|G|} \sum_{g \in G} \sum_{y \in \mathbb{Z}^2} \mathbf{x}(y) \psi(g^{-1} y). \tag{4.5}$$

For 2D-rotations, this results in translating and rotating the kernel using all group elements $g \in \mathrm{SO}(2)$. We implement two different versions of II with WS. First, we apply a global convolutional filter, i.e., the kernel size is equivalent to the size of the input feature map (*Global-WS*). Secondly, we use local filters with kernel size k which we apply at all spatial locations (u, v) and all orientations ϕ (*Local-WS*).

Relation to Group Convolutions In the following we show the close connection between II using a WS and the regular G-Conv introduced in reference [17]. Recall the formulation of the discrete lifting G-Conv of an image $\mathbf{x} : \mathbb{Z}^2 \to \mathbb{R}$ and a filter $\psi : \mathbb{Z}^{k \times k} \to \mathbb{R}$

$$[x \star_G \psi](g) = \sum_{y \in \mathbb{Z}^2} \mathbf{x}(y) \psi(g^{-1} y). \tag{4.6}$$

The regular lifting G-Conv followed by global average pooling $A_G\{\cdot\}$ among all group elements is

$$A_G\{[\mathbf{x} \star_G \psi](\cdot)\} = \frac{1}{|G|} \sum_{g \in G} \sum_{y \in \mathbb{Z}^2} \mathbf{x}(y) \psi(g^{-1} y), \tag{4.7}$$

which is exactly the same formulation as Equation 4.5. Thus, using a regular lifting convolution and applying global average pooling can be formulated as a special case of II.

4.3.2.2 Multi-Layer Perceptron

Another possibility for f is a MLP which consists of multiple fc layers and non-linearities σ. In combination with the rotation-group average, we obtain

$$A[\mathrm{MLP}](\mathbf{x}) = \frac{1}{UV\Phi} \sum_{u,v,\phi} \sigma(\mathbf{W}_l \cdots \sigma(\mathbf{W_1}\, \mathcal{T}_\phi\, \mathbf{x}_\mathcal{N})), \tag{4.8}$$

where $\mathcal{N}$ defines the neighbourhood of a pixel located at (u, v) enabling to apply the MLP locally. We parametrize the MLP with 3 layers and use Rectified Linear Unit (ReLU) non-linearities for σ.

4.3.2.3 Self-Attention

Finally, we insert a SA module into the II framework. For this definition, we define the inputs in matrix form $\mathbf{x} \in \mathbb{R}^{H \times W}$ (cf. Section 2.1.1). Recall the definition of visual SA from Section 2.1.1.3. $\mathrm{SA}(\mathbf{x})$ is calculated by defining the pixels of the input image or feature space as $T = H \cdot W$ individual tokens $\mathbf{x} \in \mathbb{R}^{HW \times C_i}$ with C_i values and learning attention scores $\mathbf{A} \in \mathbb{R}^{T \times T}$. This includes three learnable matrices: the value matrix $\mathbf{W}_V \in \mathbb{R}^{C_i \times C_h}$, the key matrix $\mathbf{W}_K \in \mathbb{R}^{C_i \times C_h}$ and the query matrix $\mathbf{W}_Q \in \mathbb{R}^{C_i \times C_h}$. Visual SA is defined as

$$\mathrm{SA}(\mathbf{x}) = \mathrm{softmax}(\mathbf{A})\mathbf{x}\mathbf{W}_V \text{ with } \mathbf{A} = \mathbf{x}\mathbf{W}_Q(\mathbf{x}\mathbf{W}_K)^T. \tag{4.9}$$

To incorporate positional information between the individual pixels, we use relative encodings $\mathbf{P}$ between query pixel $\mathbf{x}_i$ and key pixel $\mathbf{x}_j$ resulting in [155]

$$\mathbf{A}_{\mathbf{i},\mathbf{j}} = x_i\mathbf{W}_Q((x_j + \mathbf{P}_{x_j - x_i})\mathbf{W}_K)^T. \tag{4.10}$$

We embed this formulation into the II framework by transforming the input $\mathbf{x}$ using bilinear interpolation and apply the group average over all results.

$$A[\mathrm{SA}](\mathbf{x}) = \frac{1}{|G|}\sum_{g \in G} \mathrm{softmax}(\mathcal{T}_g\mathbf{A})\,\mathcal{T}_g\mathbf{x}\mathbf{W}_V, \tag{4.11}$$

where $\mathcal{T}_g\mathbf{A}$ denotes calculating the attention scores using the transformed input. We also investigate MH-SA where H self-attention layers are calculated, concatenated and processed by a learnable linear matrix multiplication with weights $\mathbf{W}_o \in \mathbb{R}^{HC_h \times C_o}$. This formulation is related to reference [140], where opposed to our approach equivariance is enforced using adapted positional encodings.

Table 4.1: (Table 1 in Rath & Condurache [133]) MTE of different monomial selection types on Rotated-MNIST using an II-enhanced SFCNN [132]. ✓ indicates full pre-training, ◐ iterative pre-training for a small number of epochs and x pruning at initialization.

Selection	Pre-Train	Init.	MTE [%]
SFCNN	-	-	0.714 ± 0.022
-	x	Random	0.751 ± 0.032
LSE	✓	Random	0.687 ± 0.012
Connectivity	x	Random	0.758 ± 0.0025
Connectivity	◐	Catalogue	0.708 ± 0.010
Connectivity	◐	Random	0.705 ± 0.027
Magnitude	◐	Catalogue	0.704 ± 0.022
Magnitude	◐	Random	0.677 ± 0.031

4.4 Experiments & Discussion

We evaluate the different setups on Rotated-MNIST, SVHN, CIFAR-10 and STL-10. For each dataset, we choose a baseline architecture, assume that the feature extraction network is highly optimized and focus on the role of the II layer. We keep the number of parameters for the equivariant networks constant by adapting the number of channels per layer (see Appendix B.1.1). We conduct experiments using the full training data, but more importantly limited subsets to investigate the sample efficiency of the different variants. When training on limited datasets, we keep the number of total training iterations constant and adapt all HPs depending on epochs, such as learning rate decay, accordingly. All data subsets are sampled randomly with constant class ratios and are equal among all architectures. We optimized the HPs using Bayesian Optimization with Hyperband [44] and a train-validation split of 80:20. Implementation details and HPs can be found in Appendix B.1.

4.4.1 Evaluating Monomial Selection

We evaluate the monomial selection methods on Rotated-MNIST, a dataset for handwritten digit recognition with randomly rotated inputs including 12k training and 50k testing greyscale-images [100]. Therefore, we train a rotation-equivariant CNN with

Table 4.2: (Table 2 in Rath & Condurache [133]) MTE on limited subsets of Rotated-MNIST using SFCNN as baseline [176].

			Number/Percentage of samples					
G-Conv	II	f	500/4.2%	1k/8.3%	2k/17%	4k/33%	6k/50%	12k/100%
x	x	-	8.635	7.205	5.586	4.684	4.324	3.664 ± 0.082
✓	x	Pooling	3.543	2.529	1.660	1.337	1.126	0.714 ± 0.022
✓	✓	Monomials	3.115	2.194	1.593	1.322	1.068	**0.677** ± 0.031
✓	✓	Global-WS	3.120	2.294	1.614	1.200	1.004	0.712 ± 0.027
✓	✓	Local-WS	3.168	2.292	1.612	1.186	1.032	**0.688** ± 0.032
✓	✓	MLP	3.250	2.310	1.652	1.242	1.024	0.732 ± 0.023
✓	✓	MH-SA	3.178	2.268	1.666	1.294	1.038	0.710 ± 0.022
✓	✓	G-Conv+Avg-Pool	3.352	2.542	1.836	1.346	1.128	0.782 ± 0.012
E(2)-CNN, Rotation								0.705 ± 0.025
E(2)-CNN, Rotation & Flips								**0.682** ± 0.022

steerable filters (SFCNN) with five convolutional and three fc layers where we insert II in between [132, 176]. For all layers, we use $n_\alpha = 16$ rotations. Table 4.1 shows the performance of the different monomial selection algorithms. We perform five runs for each dataset size using data augmentation with random rotations and report the MTE and the standard deviation for the full dataset.

The results in Table 4.1 indicate that for the monomial selection the magnitude-based pruning with random pre-selection outperforms both the LSE baseline and the connectivity-pruning approach. Random initial selection outperforms the catalogue-based approach. Furthermore, it is evident that the monomial selection algorithm plays a key part and allows a relative performance increase of up to 10.9% compared to a purely random monomial selection. Therefore, we use randomly initialized magnitude-based pruning with pre-training for all following monomial experiments.

4.4.2 Evaluating Alternatives to Monomials on Digits

We further use Rotated-MNIST to evaluate the monomial replacement candidates using the training setup from above on full and limited datasets (Table 4.2). We observe that all variants of II outperform the baseline SFCNN utilizing pooling and a standard seven layer CNN trained with data augmentation (as used for comparison in [17]). Especially in the limited-data domain, II-enhanced networks achieve a better performance despite

Table 4.3: (Table 3 in Rath & Condurache [133]) MTE on limited subsets of SVHN using WRN16-4 [186] as baseline.

			Number/Percentage of samples					
G-Conv	II	f	1k/1.3%	5k/6.9%	10k/14%	50k/69%	73k/100%	#Param.
x	x	-	12.72	6.37	4.96	3.29	3.00 ± 0.01	2.75M
✓	x	Pooling	11.15	5.52	4.46	3.25	2.89 ± 0.09	2.76M
✓	✓	Monomials	**10.67**	5.45	4.51	3.10	2.79 ± 0.03	2.78M
✓	✓	Global-WS	11.37	6.45	4.96	3.32	2.95 ± 0.07	2.83M
✓	✓	Local-WS	**10.70**	**5.04**	**4.31**	**3.00**	**2.69** ± 0.01	2.77M

adding more parameters. Consequently, II successfully improves the sample efficiency and thereby improves the generalization ability. We conjecture that this is due to the II layer better preserving information that effectively contributes to successful classification compared to spatial pooling, i.e., II explicitly enforces invariance without afflicting other relevant information.

For all practical purposes, monomial-based II performs on par with the alternative functions which enable a streamlined training procedure. Thus, it seems possible to replace the monomials with other functions in order to avoid the monomial selection step while maintaining the performance. This would further reduce the training time and at the same time provide a setup in which the II layer can be optimally tuned and adapted to the other layers in the network. All proposed functions are well-known in DL literature which supports the practical deployment. To show that the benefits of II do not only stem from additional model capacity but from effectively leveraging prior knowledge, we add an additional steerable G-Conv and perform average pooling as a special case of II (G-Conv+Avg-Pool) which performs clearly inferior.

We outperform the E(2)-CNNs [173] when they only incorporate invariance to rotations and achieve comparable results when they use a bigger invariance group including flips. The WS approach shows the most promising results among the different monomial replacement candidates.

We also conduct experiments on SVHN in order to assess the performance of II on real-world datasets that do not involve artificially induced global invariances. It contains 73k training and 10k test samples of single digits from house numbers in its core dataset [119]. We use WRN16-4 as baseline [186] and conduct experiments on the full dataset and

Table 4.4: (Table 4 in Rath & Condurache [133]) MTE on limited subsets of CIFAR-10 using WRN28-10 [186] as baseline.

			Number/Percentage of samples				
G-Conv	II	f	100/0.2%	1k/2%	10k/20%	50k/100%	#Param.
x	x	-	71.69	37.61	**9.08**	**3.89** $\pm$ 0.02	36.5M
✓	x	Pooling	76.54	37.29	12.68	4.71 $\pm$ 0.04	36.7M
✓	✓	Monomials	**69.42**	**29.83**	11.15	4.60 $\pm$ 0.12	36.8M
✓	✓	Local-WS	72.72	32.10	10.45	4.54 $\pm$ 0.15	36.9M
E(2)-WRN28-10						***2.91***	$\sim$37M

limited subsets (Table 4.3). We compare the WRN to a SF-WRN and to II-enhanced networks based on monomials, global- and local-WS with $k = 3$. For all following experiments, we use $n_\alpha = 8$ angles for the steerable convolutions as well as II and perform three runs per network and dataset size.

The II-based approach generally outperforms both the standard WRN16-4 as well as the equivariant baseline which achieves invariance using pooling. This proves that II is useful for real-world setups with non-transformed input data and can be applied to complex DNN architectures such as WRNs. The monomial and local-WS approach seem to perform best among all dataset sizes, with local-WS achieving slightly better results. We believe this is due to the fact that the architecture using this newly proposed function can be trained more efficiently. Additionally, training can be conducted in a single run without intermediate pruning steps since the monomial selection is avoided. Global-WS achieves worse results over all dataset sizes. Generally we assume that differences in performance among various methods over data size have to do with the trade-off between how good a specific architecture is able to leverage the prior knowledge on rotation invariance and how good it is able to learn and preserve other relevant invariance cues contained in real-world datasets such as colour changes or illuminations.

4.4.3 Object Classification on Real-World Natural Images

To evaluate our approach on more complex classification settings including more variability, we use CIFAR-10 and STL-10. CIFAR-10 is an object classification dataset with 50k training and 10k test RGB-images [95]. STL-10 is a subset of ImageNet containing 5k

Table 4.5: (Table 5 in Rath & Condurache [133]) MTE on STL-10 using WRN16-8 [186] as baseline.

G-Conv	II	f	MTE[%]	#Param.
x	x	-	12.74 ± 0.23	10.97M
✓	x	Pooling	12.51 ± 0.33	10.83M
✓	✓	Monomials	10.84 ± 0.46	10.85M
✓	✓	Local-WS	$\mathbf{10.09} \pm 0.21$	10.92M
E(2)-WRN16-8			9.80 ± 0.40	12.0M
SES-WRN16-8			***8.51***	11.0M

labelled training images from 10 classes [15]. It is commonly used as a benchmark for semi-supervised learning and classification with limited training data.

We use WRN28-10 and WRN16-8 as baseline architecture, respectively and test II with monomials and local-WS with $k = 3$. For CIFAR-10, we train on full data as well as on limited subsets using standard data augmentation with random crops and flips (Table 4.4). For STL-10 (Table 4.5), we use random crops, flips and cutout [31].

On CIFAR-10, we notice two developments: While our networks outperform the WRN28-10 in the limited-data domain, indicating an improved sample efficiency, they are unable to achieve better results in large-data regimes (Table 4.4). Networks employing II achieve a better performance than the pooling counterpart among all dataset sizes indicating that II better preserves the information needed for a successful classification leading to a better sample efficiency.

Local-WS performs on par or slightly worse than the monomials. We conjecture that on bigger dataset sizes, our approach with its focus on rotation-invariance does not capture the complex local object-related invariant cues needed for successful classification as good as a standard WRN. We remark that for SVHN, relevant invariance cues besides rotation are rather global (e.g., colour, illumination, noise), while for CIFAR these are also local and object-related (e.g., perspective changes, occlusions). Thus, our method handles global invariances well while needing additional steps to handle local invariances other than rotation.

The E(2)-WRNs [173] achieve better results than our networks in this setup. Contrarily to our approach, the equivariance restrictions are softened with network depth and a bigger invariance group including flips is used, thus addressing more local invariances.

Nevertheless, the approach can be combined with II in the future.

On STL-10, both II-enhanced networks outperform the pooling baseline and the standard WRNs. The local-WS approach outperforms the monomial counterpart. On this basis, we conclude that for all practical purposes, II based on local-WS delivers best results while being simpler to train than the monomial variant. Again, other methods incorporating invariance to other groups such as the general E(2)-CNNs [173] or scales (Scale-Equivariant Steerable Filter CNN (SES-CNN), [162]) achieve better results than our purely rotation-invariant network. This is intuitive since samples from ImageNet involve variability from an even greater source of different transformations than CIFAR-10. Consequently, the invariance cues that need to be captured by a classifier are even more complex.

4.5 Conclusion

In this contribution, we focused on leveraging prior knowledge about invariance to geometrical transformations for classification problems. Therefore, we adapted the II framework by introducing a novel monomial selection algorithm and replacing the monomials with different functions such as a weighted sum, a MLP, and self-attention. Replacing the monomials enabled a streamlined training of II-enhanced DNNs by avoiding the pre-training and selection step. This allows to optimally tune and adapt all algorithmic components at once promoting the application of II to complex real-world datasets and architectures such as WRNs.

Our method explicitly enforces invariance which we see among the key factors to be taken into consideration by a feature-extraction engine for successful classification, especially for real-world applications, where data is often limited. Assuming that rotation invariance is required, we have shown how to design a DNN based on II to leverage this prior knowledge. In comparison to the standard approach, we replace spatial max pooling by a dedicated layer which explicitly enforces invariance while increasing the network's expressibility. To enable the network to capture other invariance cues in particular of global nature we use trainable weights as well.

We have demonstrated state-of-the-art sample efficiency on datasets from various real-world setups. We achieve state-of-the-art results on all data regimes on image classification

tasks when the targeted invariances (i.e., rotation) generate the most intraclass variance, as in the case of Rotated-MNIST and SVHN. On Rotated-MNIST, we even outperform the E(2)-CNN which also includes invariance to flips.

On CIFAR-10 and STL-10, we show top performance in limited-data regimes for image classification tasks where various other transformations besides rotation are responsible for the intraclass variance. At the same time, the performance in the full-data regime is better than the pooling baseline, which shows that our approach is able to effectively make use of prior knowledge and introduce rotation invariance without afflicting other learned invariances. Specifically, II based on monomials and a local-WS achieve the best and most stable performance and consistently outperform the baseline, which uses group and spatial pooling, as well as standard convolutional architectures.

For the choice of the II function, Local-WS performs similarly or better than monomials while being easier to apply and optimize due to avoiding the monomial selection step. It is different to simply adding an additional group-equivariant layer and performing average pooling among rotations and spatial locations because group pooling is performed before applying the II layer. Compared to TI-Pooling [99], our method explicitly guarantees invariance within a single forward pass. In contrast, TI-Pooling approximates invariance by pooling among the responses of a non-equivariant network which requires one forward pass per group element.

Chapter 5

Invariance to Multiple Transformations at Once

Many highly relevant practical use-cases require simultaneous invariance to multiple transformations to achieve optimal prediction results with DNNs. In our final contribution, we hence expand the II layer introduced in Chapter 3 beyond rotations towards flips and scales. Based on the adjustments from chapter 4, scale-II can be defined based on monomials or a weighted sum, which again enables a streamlined training procedure. Since scale transformations only form a semigroup, achieving scale invariance based on II requires an expansion of the general II framework. To capture invariance to multiple transformations within a single DNN, we propose a multi-stream architecture (see Figure 5.1). Invariance to each relevant transformation is covered by a dedicated stream. By combining the features before the final classification with a learnable map and adding a *standard* convolutional stream, the network can automatically learn, which invariants are the most relevant for the classification task to be solved. This allows our multi-stream invariant DNN to obtain new state-of-the-art results in all data regimes, but especially when training data is scarce. Specifically, we report a new state-of-the-art on STL-10, a limited subset of the general object classification dataset ImageNet. This chapter summarizes our contribution *"Deep Neural Networks with Efficient Guaranteed Invariances"* [134], which will not further be cited in the upcoming sections.

5.1 Expanding Invariant Integration to Multiple Transformations

As discussed in Chapters 3 & 4, II is a method to construct a complete feature space w.r.t. a transformation group G [148]. So far, II replaced the global spatial pooling operation for rotation-invariant image classification DNNs. For tasks where invariance to *rotations* is desired, this resulted in an improved sample efficiency by efficiently leveraging the available prior knowledge while adding targeted model capacity [132, 133].

In our final contribution, we expand II to include geometrical prior knowledge about further transformations. We first adapt rotation-II to also include *flips* and achieve invari-

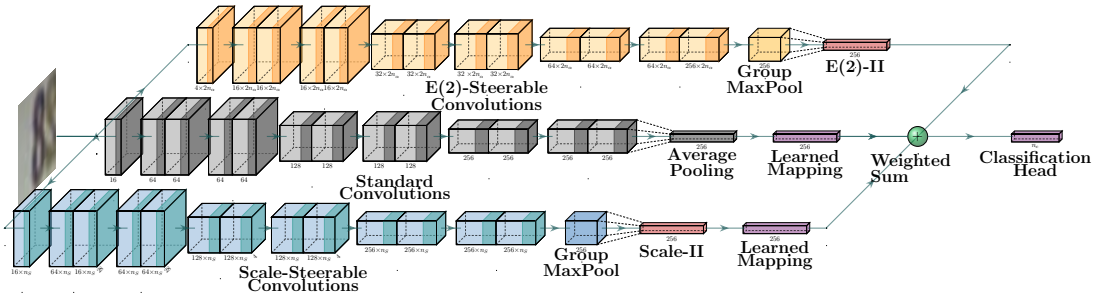


Figure 5.1: (Figure 1 in Rath & Condurache [134]) Triple-stream invariant WRN16-4 architecture. Includes standard convolutions (grey), rotation-flip-steerable convolutions (E(2), orange), scale-steerable convolutions (blue), II layers (red), a WS (green) and fc layers (purple). Residual shortcut connections are omitted for clarity.

ance to the 2D Euclidean group $\mathrm{E}(2)$. We then expand II to achieve invariance to *scales*, thus covering a larger set of transformations. Thereby, we show that the II framework can be applied to general transformation (semi-)groups in order to improve the sample efficiency of equivariant CNNs for classification tasks.

Common group-equivariant DNNs incorporate prior knowledge about a *single transformation*. It is an open challenge how to proceed when *multiple symmetries* are involved because it may be impossible to solve the constraints needed to design transformation-steerable filters depending on the involved groups. Even when avoiding the constraints via interpolation methods, simply expanding the regular equivariant G-Convs is computationally inefficient since the representation grows multiplicatively. For example, for a kernel with 8 rotations and 4 scales, we would have to store $8 \cdot 4 = 32$ responses per kernel.

We address the issue of multiple invariances within a single architecture that effectively combines several streams, each one with specific invariances enforced by G-Convs followed by II (see Figure 5.1). This significantly extends the practical applicability of the II framework. Through our *multi-stream architecture*, we propose a novel approach that allows our network to learn the best possible combination of invariant features among multiple transformations at once. We evaluate our approach on Scaled-MNIST and on the real-world datasets SVHN, CIFAR-10 and STL-10. On STL-10 using only labelled data, we report new state-of-the-art results.

5.2 Related Work

Chapters 3 and 4 applied II within DNNs for the case of rotations [132, 133]. Our third contribution expands the II layer towards further transformations including flips and scales. The related work includes group-equivariant CNNs that enforce equivariance to rotations, flips and scales (Section 2.3.1.1) as well as II (Section 2.4.1). Further, our multi-stream architecture compares to other approaches that encode in- or equivariance to rotations and scales at the same time.

5.2.1 Simultaneous Invariance to Rotations and Scales

PTNs transform inputs into the polar coordinate system using an estimated object centre to achieve equivariance to rotations and scales, see Section 2.3.2.4 [40]. Working in polar coordinates, although advantageous for rotations, may prove to be detrimental to translation equivariance. Indeed PTNs are by design invariant to translation, but it is not clear how much other relevant information is destroyed. At the same time, STNs in general cannot offer invariance guarantees, as they rely on the localization network to learn the correct transformation [83]. The invariance is not fully guaranteed but approximated w.r.t. the estimated transformation (STNs) or object centre (PTNs). The same is valid for deformable [26] and tiled convolutions [101].

Our multi-stream architecture differs from these approaches, as the invariance is guaranteed per stream. Furthermore, an additional stream using standard convolutions allows to recover relevant information that might be destroyed by enforced invariance.

5.3 Method

In this section, we extend rotation-II using a WS to the E(2)-group involving rotations and flips (Section 5.3.1). We further show why II cannot be straightforwardly applied to scale transformations and introduce an alternative to obtain scale-invariants based on II (Section 5.3.2). We then propose a scale-invariant CNN including Scale-II (Section 5.3.3). Finally, we introduce a multi-stream DNN architecture that efficiently combines invariances to multiple transformations within a single DNN (Section 5.3.4).

5.3.1 E(2)-Invariant Integration

Recall the rotation-II formulation using a weighted sum from Chapter 4

$$A[\mathrm{WS}](\mathbf{x}) = \frac{1}{N_\phi UV} \sum_{\phi,t} \sum_{y \in \mathbb{Z}^2} \mathbf{x}(y)\, \mathcal{T}_\phi\, \psi(y-t). \tag{5.1}$$

Equation 5.1 can straightforwardly be expanded to each discrete subgroup of E(2) (as used by reference [173]), which contains flips $\mathcal{T}_f$ in addition to rotations $\mathcal{T}_\phi$ and translations $\mathcal{T}_t\, \psi(y) = \psi(y-t)$

$$A[\mathrm{WS}](\mathbf{x}) = \frac{1}{2N_\phi UV} \sum_{f,\phi,t} \sum_{y \in \mathbb{Z}^2} \mathbf{x}(y)\, \mathcal{T}_\phi\, \mathcal{T}_f\, \psi(y-t). \tag{5.2}$$

5.3.2 Scale-Invariant Integration

In images, objects naturally appear at different scales, e.g., due to variable camera-to-object distances. Hence, DNNs for object classification or detection benefit from invariance to scales. In comparison to the rotation group, discrete scale transformations are not circular and non-invertible due to the loss of information during down-scaling. Thus, scales do not satisfy all group axioms and can only be modelled as a semigroup.

As a consequence, the common approach to achieve invariant representations (cf. Section 2.4) using G-Convs followed by pooling among the group and spatial domains does not necessarily yield invariance for the scale group. In fact, average pooling over the spatial dimension of scale-equivariant features does not lead to invariant, but rather homogeneous features, i.e., scaling by s modifies the output by multiplying it with s^2. Consequently, we propose to use an adapted variant of II in combination with a scale-equivariant CNN to obtain scale-invariant features.

Schulz-Mirbach demonstrated that it is impossible to construct invariants by integrating over the scale semigroup while at the same time achieving separability [148]. This prohibits constructing a complete feature space w.r.t. scales using the standard II approach. Nevertheless, the group average w.r.t. translations is a homogeneous function w.r.t. scales when using polynomials [149]. This means that the effect of scaling by s on the features obtained using translation-II is defined as $A[f](\mathcal{T}_s\, \mathbf{x}) = s^K A[f](\mathbf{x})$ where the order K is defined by the polynomial order of f with the scale operator $\mathcal{T}_s\, \mathbf{x}(y) = \mathbf{x}(s^{-1}y)\ \forall s > 0$.

As shown in reference [149], a complete feature space w.r.t. the scale-translation semigroup $G_S = T(2) \rtimes S$ can be calculated by dividing homogeneous functions of the same order. When using *monomials m*, one resulting *scale-invariant integral* is given by dividing monomials of the same order $\sum b_{1,i} = \sum b_{2,i}$ with $t \in \mathbb{Z}^2$

$$A_{G_S}[m](\mathbf{x}) = \frac{A_T[m_1](\mathbf{x})}{A_T[m_2](\mathbf{x})} = \frac{\sum_t \prod_i \mathbf{x}(t-d_{1,i})^{b_{1,i}}}{\sum_t \prod_i \mathbf{x}(t-d_{2,i})^{b_{2,i}}}. \tag{5.3}$$

The special case of translation-II that combines values within a fixed neighbourhood using a learnable WS as function f results in a standard convolution followed by average pooling and is equivalent to a polynomial of order 1. Consequently, we can choose a divisor of the same homogeneous order to obtain a scale-invariant representation and use the mean of the feature map. We thus introduce the *WS-based scale-group average* $A_{G_S}[\mathrm{WS}]$ based on average pooling over a standard convolution without bias with translations $\mathcal{T}_t$, $y,t \in \mathbb{Z}^2$ divided by mean

$$A_{G_S}[\mathrm{WS}](\mathbf{x}) = \frac{A_T[\mathrm{WS}](\mathbf{x})}{\sum_y \mathbf{x}(y)} = \frac{\sum_y \sum_t \mathbf{x}(y)\psi(y-t)}{\sum_y \mathbf{x}(y)}. \tag{5.4}$$

Theorem 5.3.1. *The WS-based scale-group average* $A_{G_S}[WS](\mathbf{x}) = \frac{A_T[WS](\mathbf{x})}{\sum_y \mathbf{x}(y)}$ *is invariant to scales s with* $s > 0$ *acting on the input via* $\mathcal{T}_s\mathbf{x}(y) = \mathbf{x}(s^{-1}y)$*, i.e.,*

$$A_{G_S}[WS](\mathcal{T}_s\mathbf{x}) = A_{G_S}[WS](\mathbf{x}).$$

Proof. Since translation-II and the mean are both homogeneous w.r.t. scales with factor s^2, it is easy to see that dividing them leads to a scale-invariant solution

$$A_{G_S}[\mathrm{WS}](\mathcal{T}_s\mathbf{x}) = \frac{A_T[f](\mathcal{T}_s\mathbf{x}(y))}{\sum_y \mathcal{T}_s\mathbf{x}(y)} = \frac{s^2 A_T[f](\mathbf{x})}{s^2\sum_y \mathbf{x}(y)} = \frac{A_T[f](\mathbf{x})}{\sum_y \mathbf{x}(y)} = A_{G_S}[\mathrm{WS}](\mathbf{x}). \tag{5.5}$$

□

5.3.3 Application to DNNs

Inspired by our work on rotations in chapters 3 & 4, we apply E(2)- and Scale-II (Formulas 5.2 - 5.4) on top of the corresponding equivariant features learned using regular G-Convs

with steerable filters: E(2)-SFCNNs [173] and Discrete Scale Convolution (DISCO) [160], which provide the state-of-the-art results for the respective transformation groups. Those features are processed via max pooling among the group dimension to obtain a trivial equivariant representation. II then replaces the spatial pooling operation to yield invariant features that are in turn processed by the final classification layers.

For E(2)-II, we use the WS approach. For Scale-II, we investigate both proposed variants. For the monomial variant, we follow the procedure from Section 4.3.1 and randomly select monomials that are iteratively pruned during training to find the most relevant ones. We ensure the same monomial order between dividend and divisor by normalizing the divisor's exponents with $\frac{\sum_i b_{1,i}}{\sum_i b_{2,i}}$. Additionally, $x_i > 0$ ensures a differentiable solution. For Scale-II with WS, each convolution depends on all input channels. Consequently, we divide by the mean over all input channels for the WS-based II. Moreover, it is important that $\sum_y \mathbf{x}(y) > 0$. Hence, we apply both II variants to the output of a ReLU layer minimized to a small value $\varepsilon > 0$.

5.3.4 Multi-Stream Invariance

To obtain features with guaranteed invariance to multiple transformations, we implement a DNN architecture with multiple streams. Each stream is invariant to a dedicated transformation enforced using G-Convs and II (see Figure 5.1). While multiple transformations could in theory be embedded into a single network, the regular representations the network needs to process would grow with $\mathcal{O}(N \cdot M)$ for group sizes N and M. In contrast, using a dedicated stream per transformation only increases the number of representations by $\mathcal{O}(N+M) \ll \mathcal{O}(N \cdot M)$.

The input is processed separately by each transformation-invariant stream using G-Convs with steerable filters. II is applied on top of the learned scale- and E(2)-equivariant features to obtain invariant representations. For the standard convolution stream (std.) we use average pooling. Thus, we have $\mathbf{x}_j \in \mathbb{R}^{C_j}$ with $j \in \{\text{e2}, \text{scale}, \text{std}\}$. We combine two or three streams, each one invariant to either rotations and flips, scales or translations (std.), using two steps. First, we map all features to the same dimension using a learnable linear map $\tilde{\mathbf{x}}_j = \mathbf{W_j}\mathbf{x_j}$ with $\mathbf{W_j} \in \mathbb{R}^{C_{\text{map}} \times C_j}$. We then combine these streams

via a normalized learnable WS, e.g., for the case of three streams

$$\mathbf{x}_{\text{combined}} = \mathbf{w}_{\text{e2}} \odot \tilde{\mathbf{x}}_{\text{e2}} + \mathbf{w}_{\text{scale}} \odot \tilde{\mathbf{x}}_{\text{scale}} + \mathbf{w}_{\text{std}} \odot \tilde{\mathbf{x}}_{\text{std}} \tag{5.6}$$

with $\mathbf{w}_j \in \mathbb{R}^{C_{\text{map}}}$ initialized to 1 and normalized s.t. $(\mathbf{w}_{\text{e2}} + \mathbf{w}_{\text{scale}} + \mathbf{w}_{\text{std}})_i = 1$ with $i = 1, \ldots, C_{\text{map}}$ and the Hadamard product $\odot$. This is inspired by the learnable channel-wise scaling used in Batch Normalization layers [81]. Our combination approach allows to combine the invariant features with an explicitly learned factor s.t. the network can learn, which invariances are the most relevant for the corresponding task.

Each stream is pre-trained individually. We then combine the architecture, freeze all convolutional weights and only train the linear maps $\mathbf{W}_{\text{j}}$, the WS $\mathbf{w}_j$ and the classification weights $\mathbf{w}_{\text{out}} \in \mathbb{R}^{C_{\text{map}} \times n_c}$ with n_c classes. The combination head worked best, when mapping the other streams to the E(2)-output, i.e., $C_{\text{map}} = C_{\text{e2}}$ and keeping $\mathbf{W}_{\text{e2}}$ fixed as the identity. Section 5.4.4.4 provides results with different combinations, e.g., mapping all streams or concatenation. Our dedicated training procedure allows to fine-tune each stream individually, which provides a good initialization point for the combination head and further improves the sample efficiency compared to a full end-to-end training (cf. Section 5.4.4.5). While this causes an overhead at train time, all operations can easily be fused into a single network during inference.

5.4 Experiments & Discussion

We evaluate scale-II on Scaled-MNIST [159] and the invariant multi-stream networks on SVHN [119], CIFAR-10 [95] and STL-10 [15]. We use the full dataset and limited subsets with N_t samples to assess the sample efficiency. The subsets are sampled with constant class balance and are the same for all variants. For Scaled-MNIST, we use $N_t \in \{10, 50, 100, 500, 1k, 5k, 10k, 12k\}$. For SVHN and CIFAR-10 we use $N_t \in \{100, 500, 1k, 5k, 10k, 50k\}$.

For SVHN, CIFAR-10 and STL-10 we use WRNs as backbone [186]. We use the respective *standard data augmentations*: scales for Scaled-MNIST, no augmentations for SVHN, shifts and crops for CIFAR-10, and shifts, crops and Cutout [31] for STL-10. For all single-streams, the number of trainable parameters is constant. We report results for the multi-stream architectures with full streams and constant number of parameters.

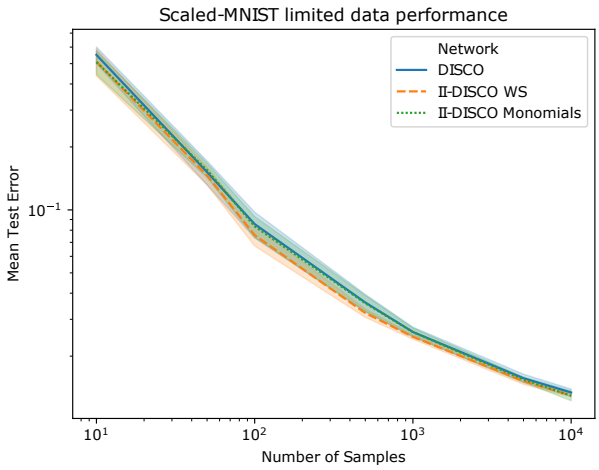


Figure 5.2: (Figure 2 in Rath & Condurache [134]) Log. Test Error (TE) on Scaled-MNIST subsets.

Table 5.1: (Table 1 in Rath & Condurache [134]) Invariance Error Δ when obtaining a scale-invariant representation using the respective layer directly on the input and a randomly initialized scale-equivariant network.

Layer	Input	CNN
Average Pooling	0.222	0.090
Mixed Pooling	0.017	0.039
Scale-II Monomials	1.24e-4	**2.64e-7**
Scale-II WS	**2.97e-9**	5.94e-3

We optimize all HPs using a 80:20 validation split and Bayesian Optimization with Hyperband [44]. For all II-WS-layers, we use $k = 3$ and a constant number of in- and output channels. For II with monomials, we use an iterative pruning-based selection with $n_m = \{25, 12, 5\}$ monomial pairs after 0, 5 and 10 epochs following Section 4.3.1. The exact HPs, optimization settings and network architectures can be found in Appendix B.2. If not mentioned otherwise, we report the mean and standard deviation over three training runs.

Table 5.2: (Table 2 in Rath & Condurache [134]) TE on Scaled-MNIST using a CNN with 5 layers, data augmentation with random scales and an upsampling layer to double the input size.

Method	Scale-II	TE [%]
CNN	-	1.60 ± 0.09
SES-CNN	-	1.42 ± 0.07
DISCO	-	1.35 ± 0.05
II-DISCO	Monomials	$\mathbf{1.30} \pm 0.06$
II-DISCO	WS	$\mathbf{1.30} \pm 0.02$

5.4.1 Evaluating Scale-Invariant Integration

We evaluate our scale-II layer on the Scaled-MNIST dataset, which consists of handwritten digits artificially scaled with factor $s \in [0.3, 1]$. We use the architecture from references [162] (SES-CNN) and [160] (DISCO) built of three convolutional and two dense layers using $n_S = 4$ scales and replace the final global pooling layer with the scale-II layer. Following references [160, 162], we compare the invariance error of the scale-II layers to the methods they use to obtain scale-invariant representations: a mixed pooling approach including average and max pooling for Scaled-MNIST and average pooling for STL-10. The invariance error is a simplified version of the equivariance error given as

$$\Delta = \frac{1}{S} \sum_{s} \frac{|\psi(\mathbf{x}) - \psi(\mathcal{T}_s \mathbf{x})|_2^2}{|\psi(\mathbf{x})|_2^2}. \tag{5.7}$$

We compute Δ when directly processing the input, i.e., ψ is the II- or pooling-layer and when ψ is the randomly initialized CNN including the respective layer using 100 samples from Scaled-MNIST scaled with $s \in [0.5, 0.55, ..., 1.0]$. The results in Table 5.1 show that scale-II guarantees invariance as opposed to the pooling approaches. While Scale-II with WS achieves a better error directly on the input, the monomial variant is slightly better when applied within the randomly initialized DNN.

We then evaluate the performance of our scale-II layer on Scaled-MNIST for classification. Here, scale-invariance is paramount to obtain correct results because the test set contains more variability than the training set, thus benefiting scale-invariant algorithms. Table 5.2 shows the results on the full dataset, Figure 5.2 with limited training data. We report mean and standard deviation over the six pre-defined data splits. II outperforms

Table 5.3: (Table 3 in Rath & Condurache [134]) TE on SVHN, CIFAR-10 and STL-10. WRN16-4, WRN28-10 and WRN16-8 are used as baseline architecture. * indicates constant number of parameters.

II	Streams	Invariance	SVHN [%]	C-10 [%]	STL-10 [%]
	PTN	Rot. & Scale	2.97	6.72	-
x	Single	Std.	2.93 ± 0.02	3.89 ± 0.08	12.02 ± 0.05
x	Single	E(2)	2.64 ± 0.05	2.91 ± 0.13	9.80 ± 0.40
x	Single	Scale	2.71 ± 0.01	4.04 ± 0.03	8.07 ± 0.08
✓	Single	E(2)	2.36 ± 0.05	2.95 ± 0.04	7.67 ± 0.07
✓	Single	Scale	2.54 ± 0.04	3.91 ± 0.12	7.92 ± 0.09
✓	Dual*	Scale & E(2)	2.20 ± 0.05	2.96 ± 0.10	6.38 ± 0.15
✓	Dual	Scale & E(2)	2.12 ± 0.11	2.75 ± 0.04	5.95 ± 0.11
✓	Triple*	Scale, E(2) & Std.	2.29 ± 0.03	2.74 ± 0.08	6.46 ± 0.08
✓	Triple	Scale, E(2) & Std.	**2.10** ± 0.07	**2.68** ± 0.03	**5.90** ± 0.05

the pooling approach on full data and in the limited sample regime highlighting the improved sample efficiency of our approach. In summary, the mixed pooling approach used by reference [160] does not guarantee scale-invariance which leads to a decreased performance. II allows for invariance guarantees and the WS-variant is easier to optimize than the monomials-variant. Hence, II with a WS outperforms the latter in the limited data domain and is used for all further experiments.

5.4.2 Multi-Stream Digit Classification

The Street View House Number (SVHN) dataset contains single digits taken from house numbers. SVHN includes digits with different colours, font types, orientations and backgrounds and is thus harder to solve than MNIST. For all experiments on SVHN, we use the core training data, a WRN16-4 architecture, $n_r = 8$ rotations for all E(2)-G-Convs and $n_S = 3$ scales for the scale stream. The II step is calculated using the same number of rotations and flips. The results on the full SVHN dataset are shown in Table 5.3 while Figure 5.3 depicts the results using limited training data.

Using II instead of pooling improves the accuracy for all invariant architectures showing that II better preserves the information when transferring to invariance. Both invariant single-stream networks outperform the standard (std.) CNN in the limited and full data

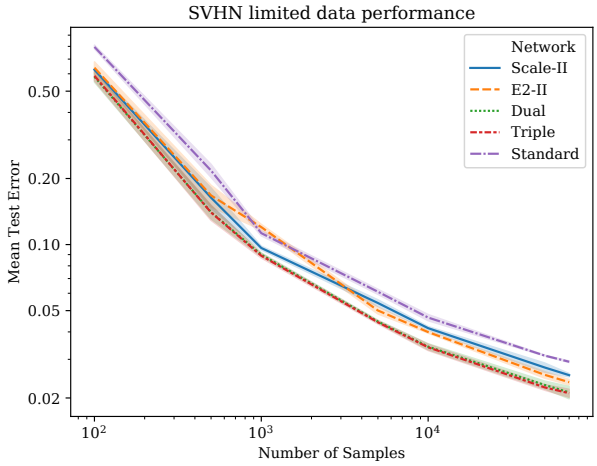


Figure 5.3: (Figure 3 in Rath & Condurache [134]) Log. TE on subsets of SVHN with full streams.

domain. This indicates that invariance to rotations and flips as well as scales is valuable information the classifier needs to learn during training – and the training data does not cover enough variability w.r.t. rotations and scales for the baseline to learn this information. The rotation- and flip-invariant DNN achieves a better sample efficiency than the scale-invariant network, indicating that rotation-invariance is more valuable for this dataset.

The multi-stream networks significantly outperform all single-stream variants including the baseline in all data regimes. The multi-stream architecture learns meaningful combinations of the generated invariants and is able to automatically choose the invariance best-fit for the training data at hand. This allows for best performances on all dataset sizes. In addition, for the rather simple task of classifying numbers, combining only a rotation- and a scale-invariant stream outperforms a standard CNN and leads to almost the same performance as with additional standard convolutions. In this problem setup, rotation- and scale-invariances seem sufficient for optimal performance and are able to recover all necessary object invariances. We conjecture that the learned features in the network's layers within the invariant streams focus on global invariances like illumination and noise while the dedicated II layer handles specific object invariances, e.g., to scales, similarly to the scattering transformation [11].

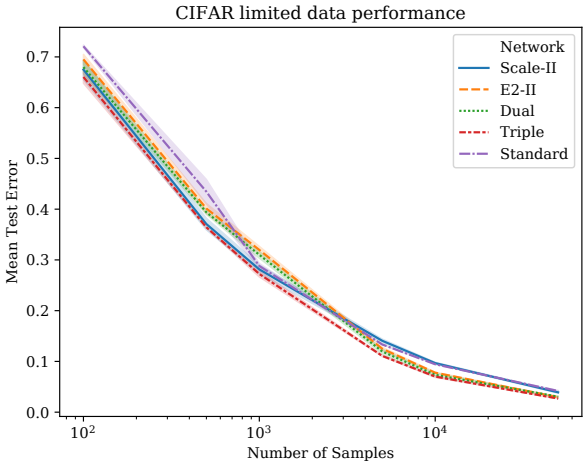


Figure 5.4: (Figure 4 in Rath & Condurache [134]) TE on subsets of CIFAR-10 with full streams.

5.4.3 Multi-Stream Object Classification

Finally, we evaluate our proposed architecture for the more complex task of object classification on CIFAR-10 and STL-10. Both datasets contain RGB images of ten different object classes. STL-10 is a subset of ImageNet containing 5k training images that is commonly used as a benchmark for the limited data performance of object classification networks. STL-10 is more challenging than CIFAR-10 since it contains bigger and more diverse images. For CIFAR-10, we use WRN28-10 and the E(2)-stream with 8, 8 and 4 rotations per residual block. For STL-10, we use WRN16-8 and the E(2)-architecture with 8, 4 and 1 rotation per residual block [173]. The E(2)-II-layer is used for 4-rotations and flips or flips-only, respectively. For both datasets, the scale-stream uses 3 scales [160]. The STL-10 and CIFAR-10 results on full data are shown in Table 5.3, CIFAR-10 results on limited data in Figure 5.4. For a fair comparison on STL-10, we adapted the official implementation of reference [160] which uses a stride of 1 in the initial convolutional layer by using stride 2 as in references [31, 173].

On CIFAR-10, we again demonstrate an increased sample efficiency of the invariant streams leading to superior performance for small dataset sizes (see Figure 5.4). While the E(2)-invariant network is able to outperform the std. baseline in the full data regime, the scale-invariant network achieves subpar performance. We interpret the latter as a sign

Table 5.4: (Table 4 in Rath & Condurache [134]) Role of the II layer for our triple-stream network on STL-10. An 'x' marks an invariant stream without II.

II		
E(2)	Scale	TE [%]
x	x	7.51 ± 0.12
x	✓	7.35 ± 0.09
✓	x	6.34 ± 0.07
✓	✓	$\mathbf{5.90} \pm 0.05$

of less variance along the scale mode in this problem setup due to the rather small images contained in CIFAR-10. Thus, other invariances are more important to decide for the correct class. The scale-invariant stream seems to be too restrictive in the sense that it is unable to learn the full set of object invariants that the baseline architecture leverages to classify the objects. On the full data, II improves the performance for the scale-invariant architecture. However, the performance on the E(2)-invariant architecture is only on par. Nevertheless, our results from Section 4.4 show that a rotation-invariant architecture using II outperforms the one with pooling in limited data regimes even when achieving slightly worse results on full data. We further investigate the advantages of II in Section 5.4.4.3. Our combined network is able to achieve the best results for all data regimes by learning to combine the best information at each dataset size. The triple stream outperforms the dual variant which indicates that the std. stream is able to capture important additional object invariances that are neglected by the restricted, invariant streams.

On STL-10, the E2-II network outperforms its counterpart without II significantly. Scale-II also slightly improves upon the baseline and II is clearly beneficial when combining multiple streams (see Table 5.4). Our multi-stream network achieves a new state-of-the-art result (cf. Table 5.3), even with constant number of parameters. This shows that incorporating prior knowledge about multiple transformations improves the performance of classification DNNs in the limited data domain, even for complex real-world datasets. Additionally, the results show that the features learned by each stream are complementary, preserved by our II layers and are effectively combined by our proposed multi-stream head. The multi-stream architecture successfully improves the sample efficiency while adding parameters – hence increasing generalization.

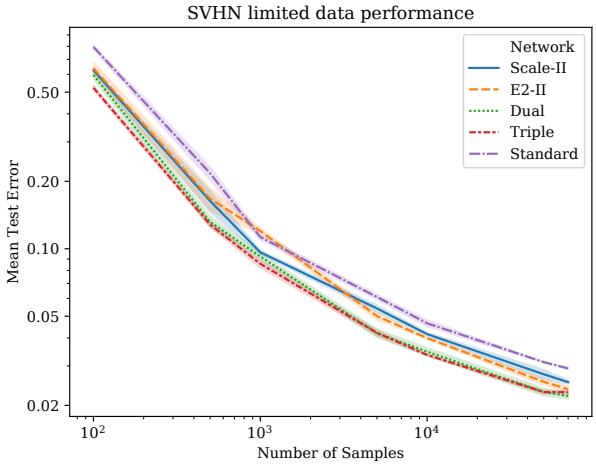


Figure 5.5: (Figure 5 in Rath & Condurache [134]) Log TE on subsets of SVHN with constant parameters.

5.4.4 Ablation Studies

We conducted extensive ablation studies concerning the number of parameters, the importance of II for the multi-stream architecture and the training procedure. The results showcase the importance of all ingredients of our multi-stream invariant network to achieve state-of-the art performance in all data regimes.

5.4.4.1 Multi-Stream: Number of Parameters

In addition to the full multi-stream architecture, we report the performance when keeping the number of parameters constant. Therefore, we shrink the number of parameters in each stream by factor 2 or 3 for the dual or triple-stream network, respectively, by naively dividing the number of channels by $\sqrt{2}$ or $\sqrt{3}$ per stream. The results are shown in Table 5.3 marked with *. Limited data results for SVHN and CIFAR-10 are shown in Figures 5.5 & 5.6.

For the multi-stream networks with constant parameters, we observe a significant performance increase in the limited data domain and when using the full dataset compared to the baseline methods. Even with our naive downscaling approach and without further fine-tuning any HPs, all single stream networks are already outperformed in limited data domains. In the future, further improvements could be achieved by investigating

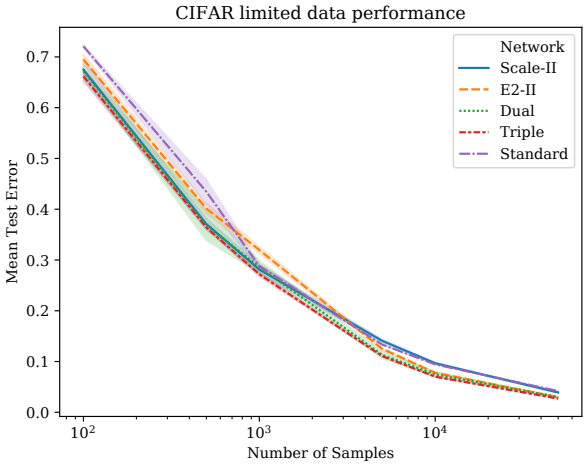


Figure 5.6: (Figure 6 in Rath & Condurache [134]) TE on subsets of CIFAR-10 with constant parameters.

more sophisticated branching methods in order to enable a better allocation of how much capacity the network should spend for each invariant stream.

While the dual stream performs slightly better on SVHN and STL-10, the triple stream achieves better performance in limited domains and on CIFAR. We believe this occurs since the individual streams of the triple-stream architecture are too thin, particularly in early layers, to cover a wide set of object invariances. At the same time, the E(2)- and the scale-stream are able to learn most object invariances the standard stream would cover when the model capacity is limited and thus benefit from additional stream-wise capacity. Nevertheless, the standard stream adds the uncovered invariances leading to a slight performance boost – specifically when training data is scarce.

To conclude, our approach achieves state-of-the-art performance on all datasets and in all data domains even with constant number of parameters. Combining multiple streams is beneficial, even without increasing the model capacity by adding parameters.

5.4.4.2 Multi-Stream: Importance of Invariance

To assess the importance of the enforced invariance, we train a triple-stream architecture consisting of three full standard streams on STL-10. We achieve a TE of 10.29% which is worse than the single-stream networks with invariance. This shows that the enforced

Table 5.5: (Table 5 in Rath & Condurache [134]) Comparison of Multi-Stream heads on STL-10 using the triple stream architecture.

Combination-Type	MTE [%]
Concat	6.82 ± 0.11
Map-E(2)	$\mathbf{5.90} \pm 0.05$
Map-Scale	7.21 ± 0.06
Map-Std	6.65 ± 0.10
Map-All	6.89 ± 0.06

invariance plays a key role for our multi-stream networks and the performance boost is not purely explained by the training procedure or added model capacity.

Training a standard WRN augmented with random rotations $\phi \in \{0°, 90°, 180°, 270°\}$ and sales $s \in [0.25, 1]$ on STL-10 achieves a TE of 21.80%, which is clearly detrimental compared to the performance without those augmentations (12.08%). Hence, layer-wise, guaranteed in- and equivariance play a key role in the improved sample efficiency of our approach.

5.4.4.3 Multi-Stream: Importance of Invariant Integration

To quantify and demonstrate the importance of the II layer, we compare our multi-stream architecture including II to variants without II on STL-10. This includes a multi-stream architecture, where only pooling is used. The results in Table 5.4 demonstrate that on a more complex classification task, in low-data regime (i.e., when the training data does not properly cover all variability present in the test data) the multi-stream approach works best when both streams use II rather than pooling.

5.4.4.4 Multi-Stream: Comparison of Combination Heads

In this section, we compare different versions of our proposed multi-stream head to a concatenation head using the triple-stream architecture on STL-10. Map-*Stream* describes which stream the respective outputs are mapped to, i.e., the respective stream uses an non-learnable identity mapping. When using Map-All, all streams use a learnable mapping at the same time. The results are shown in Table 5.5. Mapping all features to the E(2) stream and adding them via a weighted-sum leads to the best results and outperforms a

naive concatenation of features along the channel dimension.

5.4.4.5 Multi-Stream: End-to-End training

As a further ablation, we conducted a "true" end-to-end training of our dual stream with random initialization and constant number of parameters on STL-10, where we achieve 7.86 % Test Error (TE) without tuning any stream-specific HPs. While this outperforms all non-II architectures, the performance is slightly worse than a E(2)-II single stream architecture (7.67 % TE). Nevertheless, we expect the performance to be improvable beyond the pure single stream by optimizing the stream-wise HPs.

Importantly, the true end-to-end training performs worse than our more intricate training procedure (6.38 % TE). We conjecture, that optimizing each stream individually provides a good initialization point for the combination head and reduces the solution space the optimizer needs to consider. Overall, our learning strategy leads to a further improved sample efficiency of the multi-stream architecture.

5.5 Conclusion

In our final contribution, we expanded II to scale transformations and showed its effectiveness on Scaled-MNIST. Thereby, we proved the generality of our II framework to impose geometrical prior knowledge about multiple relevant input transformations.

Since Scale-II using a WS is easier to optimize than the monomial variant, we applied it in a multi-stream DNN which besides scales includes a standard convolutional and a rotation-and-flip-invariant stream. This multi-stream DNN covers a variety of practically interesting use cases as shown by an improved sample efficiency on SVHN and CIFAR-10 and new state-of-the-art results on STL-10 using only labelled data. We impose invariance to scales and rotation-and-flips in dedicated streams that also learn other global invariances and cover the remaining object invariances with a standard convolutional stream. This guarantees multiple invariances without suffering from the multiplicative increase when directly combining the groups.

Our framework is thought to leverage and honour prior knowledge. Therefore, it is focused on invariance guarantees, which may be rather restrictive in some cases. In the large data domain, Vision Transformers with less geometrical constraints commonly

outperform conventional CNNs [35]. Nevertheless, invariance guarantees improve the sample efficiency of DNNs leading to a performance boost when training data is limited. Hence, our experiments focus on small-scale datasets, where II-enhanced networks achieve a significant performance boost. While II is a general method that can be expanded beyond rotations and scales, it is restricted to transformations that can be modelled as (semi-) groups. Furthermore, we require an equivariant backbone before transferring to invariance. Nevertheless, the multi-stream network can in theory be enhanced with streams that achieve invariance without using II or G-Convs.

Chapter 6

Summary & Conclusions

In this thesis, we presented a mathematically sound framework to incorporate *geometrical prior knowledge* into DNNs by enforcing *invariance*. Thereby, we improved the *sample efficiency* and thus the performance of DNNs for image classification, particularly in the limited sample domain.

For many tasks such as image classification, DNNs benefit from features *invariant* to *intra-class variations*. This includes the task-specific *symmetry transformations* that do not affect the desired classification output. Since the symmetry transformations are often known *a priori*, in- or equivariance can be enforced rather than learned from data.

Adjusted training procedures such as *data augmentation* or STNs allow DNNs to learn in- or equivariance globally. However, these approaches approximate invariance rather than guaranteeing it. Instead, *restricting* the DNN *architecture* allows to enforce invariance to transformations in a *mathematically guaranteed* manner. Group-equivariant CNNs based on G-Convs guarantee transformation equivariant representations per layer. In related work, invariance is imposed based on the equivariant features obtained via G-Convs by *pooling* among the *group* and the *spatial* domains. This approach yields state-of-the-art performance when the embedded transformation invariance helps to solve the problem, specifically in the limited data domain [17, 162, 173].

While pooling does achieve invariance, it also *destroys information* encoded in the learned feature space. Consequently, we investigate to *replace pooling* within invariant DNNs with a dedicated operation that adds *targeted model capacity* during the transfer from equi- to invariance, rather than destroying information. Namely, we use *Invariant Integration* for this purpose. Before, II had been used as an invariant feature extractor for conventional ML classifiers such as SVMs [21, 114–116, 150]. It had not yet been combined with the learned equivariant feature spaces of DNNs obtained via G-Convs.

In Chapter 3, we propose a novel layer based on *rotation-II* using *monomials*, which directly replaces the final pooling layer of a rotation-invariant group-convolutional CNN. With some minor modifications, the *backpropagation* algorithm can be applied, thus

allowing to optimize II-enhanced DNNs end-to-end. To *select* the *monomial parameters*, we first train the baseline network without II. After applying II, we select the best-fit parameters using an *iterative approach* based on a linear classifier. Finally, we re-train the entire network including II in an end-to-end fashion. Replacing pooling with II *improves* the *sample efficiency* on the Rotated-MNIST dataset, where invariance to rotations is an artificially built-in requirement for good classification performance [132].

Chapter 4 introduces *more streamlined training algorithms* for II-enhanced networks allowing to leverage prior knowledge for larger, *real-world network architectures* and *datasets*. A novel *monomial selection* algorithm based on *pruning* avoids the pre-training step of the baseline network without II. Alternative variants of II *replace the monomials* entirely with well-known functions from DNN literature such as a MLP, SA or a learnable *WS*, thus entirely avoiding the monomial selection. Both approaches allow us to use rotation-II within WRNs for the real-world classification tasks SVHN, CIFAR and STL-10, where the latter is a limited subset of the large-scale ImageNet dataset. We demonstrate an *improved sample efficiency* compared to both standard CNNs as well as to rotation-invariant architectures based on G-Convs and pooling. II using a WS slightly outperforms the monomial approach, while allowing for a significantly simplified training algorithm, as the network can be *trained end-to-end* from scratch [133].

In Chapter 5, we *expand* the *II* framework beyond rotations to *flips and scales*. Thereby, our framework covers further invariances highly relevant for image classification tasks. As *scale* transformations only form a semigroup, we derive an *adapted variant of II* that guarantees scale-invariance. Again, we show that enhancing equivariant networks by replacing the final pooling operation with II improves the sample efficiency for tasks where scale-invariance is desired, highlighting the universality of our approach [134].

Finally, we propose a *multi-stream architecture* tailored to enforce *invariance* to *multiple transformations at once*. Thereby, multiple practically relevant invariances can be enforced within a single DNN. A *standard convolutional stream* provides the *additional capacity* and flexibility to learn object invariants not covered by *dedicated, transformation-invariant streams*. Our approach achieves state-of-the-art performance among all data domains on SVHN, CIFAR-10 and STL-10, but specifically in the limited sample domain. This indicates an *improved generalization* ability of our multi-stream architecture and its ability to automatically *select* the *best-fit invariance* for the problem at hand [134].

Overall, our results highlight the benefits of *leveraging geometrical prior knowledge*

within DNNs. By *enforcing invariance* based on equivariant features with a *dedicated operation* that adds targeted model capacity, our II-enhanced networks achieve the best of both worlds: *expressive, learned features* with strict *equivariance properties* by applying G-Convs, succeeded by an *invariant, separable feature space* obtained by leveraging II. Preserving the expressiveness of the learned features while enforcing invariance further *boosts* the *sample efficiency* of invariant DNNs. This is especially important in the limited data domain, where as much as possible of the information contained in the training data needs to be utilized.

6.1 Outlook

In the future, our approach could further be expanded by hand-crafting the function used within the II-layer rather than learning the parameters from data, similarly to the scattering transformation [11]. Additionally, our framework could be applied to larger groups, such as 3D translations and rotations. This will probably require to solve the computed integral in a more sophisticated manner, e.g., via the Monte Carlo approximation of the group average used in reference [131].

Depending on the task to solve, equivariance is more suitable than invariance as the information about the transformation itself is preserved. For example, a 3D object detector needs to be equivariant to rotations in order to correctly estimate the orientation of the detected objects. Invariance would destroy this information in the feature space, but would be beneficial for the pose-independent classification of those objects. In this case, a dedicated invariant stream employing II could handle the invariant classification task, whereas the orientation regression stream can remain equivariant.

Finally, our proposed multi-stream architecture could be combined more efficiently and optimized in an end-to-end manner, as initially investigated in Section 5.4.4. The architecture itself, e.g., the capacity used for each invariant stream, could be optimized using Neural Architecture Search. This would allow to automatically dedicate the appropriate amount of compute to the specific learning problem's most relevant invariances, reducing the development effort to carefully optimize the network architecture. Thereby, geometrical prior knowledge about multiple transformations can be built into a DNN while learning the precise importance of each invariance directly from data.

Appendix A

Group Theory Definitions

This chapter contains further definitions that supplement our brief introductions to group theory and group representation theory (Section 2.2). The definitions within this Appendix are not required to follow our main contributions, but are briefly mentioned when discussing the related work in Section 2.3.

A.1 Groups

Definition A.1.1. *A group is called an **Abelian group**, if its group operation is commutative*

$$ab = ba \quad \forall\, a, b \in G. \tag{A.1}$$

For non-Abelian groups, the group operation is *not commutative* $ab \neq ba$, i.e., the order of how the elements are applied matters [145].

Definition A.1.2. *The **orbit** $O(x)$ of a group G describes all possible actions of the group G on a set element $x \in X$*

$$O(x) = \{gx : g \in G\} = Gx \subset X. \tag{A.2}$$

Definition A.1.3. *Let G be a group. The **generating set** $S \subseteq G$ is a subset of the group set G that allows to express all group elements via a finite combination. The group generated by S is denoted as $\langle S \rangle$ [145].*

Hence, the set of generators allows to transition between all group elements [145]. The generating set is also referred to as *generators* of the group.

A.1.1 Composing Groups

To represent multiple symmetry transformations, transformation groups can be combined into a new group covering all transformations. If the transformation groups are

independent, they are combined using the *direct product*.

Definition A.1.4. *Let H be a group with operation $\star$, K be a group with operation $+$.*
*The **direct product** $G = H \times K$ is the Cartesian product $H \times K$*

$$G = H \times K = \{(h,k) |\ h \in H \text{ and } k \in K\} \tag{A.3}$$

with the group action $\circ$ of G given by

$$(h_1,k_1) \circ (h_2,k_2) = (h_1 \star h2, k_1 + k_2) \tag{A.4}$$

where the identity element is (e,e) and the inverse is $(h,k)^{-1} = (h^{-1},k^{-1})$ [145].

In the case of finite groups, the direct product produces a $|H| \times |K|$ matrix containing all possible combinations of elements $h \in H$ and $k \in K$, with the group operation applied independently to each component. For example, rotations and flips around a fixed point can be applied independently of their order and, therefore, can be combined using the direct product.

When transformations depend on each other, the direct product is not applicable. For example, when combining translations with rotations, a rotation influences a subsequent translation, i.e., rotations act on translations (cf. Section 2.2.3). In such cases, the *semidirect product* $H \rtimes K$ is used to combine dependent groups. To simplify the definition, we will restrict application of the semidirect product to the group of translations $T(n)$ and matrix subgroups $H \subseteq GL(n,\mathbb{R})$, where the components are linearly transformed via matrix multiplication[1] [174].

Definition A.1.5. *Let $T(n)$ be the translation group with the set $\mathbb{R}^n$ and operation $+$ and $H \subseteq GL(n,\mathbb{R})$ be a matrix subgroup with operation $\star$. Let γ be a group homomorphism $\gamma : H \times \mathbb{R}^n \to \mathbb{R}^n$ denoted as $(h,t) \mapsto \gamma(h,t)$ that transforms $t \in \mathbb{R}^n$ according to a matrix multiplication with a $n \times n$ matrix representing the action of H on $\mathbb{R}^n$.*

*The (outer) **semidirect product** of the translation group $T(n)$ with a matrix subgroup H, $G = T(n) \rtimes H$ is the Cartesian product $T(n) \times H$, with the group operation $\circ$ of G [145, 174]*

$$(t_1,h_1) \circ (t_2,h_2) = (t_1 + \gamma(h_1,t_2), h_1 \star h_2) \quad \text{with} \quad t_1,t_2 \in T(n) \text{ and } h_1,h_2 \in H. \tag{A.5}$$

[1] For a definition without this restriction, refer to reference [174], Definition B.2.5.

Specifically, when combining rotations and flips (forming the orthogonal group $\mathrm{O}(2)$) with translations via the semidirect product $\mathrm{E}(2)=\mathrm{T}(2)\rtimes\mathrm{O}(2)$, the homomorphism γ is

$$\gamma(\mathbf{O},\mathbf{t}) = \mathbf{Ot} \quad \forall\, \mathbf{O}\in\mathrm{O}(2),\ \mathbf{t}\in\mathbf{T}(\mathbf{2}). \tag{A.6}$$

Inserting this into the semidirect product results in the Euclidean group $E(2)$ from Section 2.2.3 [64]. The direct product is a special case of the semidirect product, where $\gamma(h,k)$ is the identity, i.e., $\gamma(h_1,k_2)=k_2$ [174].

A.2 Group Representation Theory

A.2.1 The Relation between Irreps & Harmonic Functions

The irreps of a group G are connected to the *harmonic functions*. The *Peter-Weyl theorem* establishes that any group representation can be decomposed into its *irreducible subspaces*, which are directly related to the harmonic functions [174].

For any square-integrable function f acted on by a *quotient* representation $\rho^{G/H}$ where $H\subset G$ is a subgroup, f can be represented using an *orthonormal basis* characterized by the harmonic functions. The regular representation is a special-case for this decomposition with $H=\{e\}$. In particular, for the rotation group $G=SO(2)$ with $H=\{e\}$, the harmonic basis is given by *circular harmonics* [181]

$$h_k(\phi) = e^{ik\phi}, \quad k\in\mathbb{Z}, \quad \phi\in[0,2\pi). \tag{A.7}$$

For $G=SO(3)$ with $H=SO(2)$, the basis is formed by the *spherical harmonics* [175].

These harmonics serve as the foundation for defining convolutional kernels that achieve *continuous equivariance* [181]. Furthermore, they form a filter basis that can be used to compute transformation-steerable filters, as discussed in Section 2.2.5) [51, 98, 173].

A.2.2 Combining Representations

Multiple group representations can be composed into a new one based on the *direct sum* or the *tensor product*. This is particularly interesting, because every group representation can be defined in terms of its irreps [64], cf. Section 2.2.4.1.

Definition A.2.1. *Let $\rho_1 : G \to GL(\mathcal{V}_1)$ and $\rho_2 : G \to GL(\mathcal{V}_2)$ be two representations. The **direct sum** $\rho_1 \oplus \rho_2 : G \to GL(V_1 \oplus V_2)$ is*

$$(\rho_1 \oplus \rho_2)(g) = \begin{bmatrix} \rho_1(g) & 0 \\ 0 & \rho_2(g) \end{bmatrix},$$

which acts independently on the space $V_1 \oplus V_2$ consisting of the orthogonal subspaces V_1, V_2.

Applying the direct sum on multiple representations ρ_n with $n = 1, \ldots, N$ is denoted as

$$\bigoplus_n \rho_n(g) = \rho_1(g) \oplus \rho_2(g) \oplus \ldots \oplus \rho_N(g).$$

Tensor Product & Clebsch-Gordan Decomposition

Definition A.2.2. *Let $\mathcal{V}$ and $\mathcal{W}$ be two vector spaces. The **tensor product** is a map into a new vector space $\mathcal{Z} = \mathcal{V} \otimes \mathcal{W}$ via a bilinear map $z = v \otimes w$ with $v \in \mathcal{V}, w \in \mathcal{W}, z \in \mathcal{Z}$.*

In group representation theory, the tensor product is used to combine two representations $(\rho_i, \mathcal{V}_i)$ and $(\rho_j, \mathcal{V}_j)$ of a group G into a new representation that acts on the tensor product of the vector spaces via $(\rho_i \otimes \rho_j)(g)(v_i \otimes v_j) = \rho_i(g)v_i \otimes \rho_j(g)v_j$. This is of interest, when designing arbitrary representations encoding equivariance or when studying convolutional kernel constraints that guarantee equivariance during the transfer between two arbitrary feature fields [98, 173].

The *Clebsch-Gordan decomposition* can be used to decompose the tensor product of irreps. Importantly, the tensor product of two irreps is itself not irreducible. Nevertheless, the *Clebsch-Gordan decomposition* allows to decouple the tensor product of two unitary irreps ρ_i, ρ_j of a compact group G into a direct sum of irreps ρ_k

$$\mathrm{CG}_{ij}(\rho_i \otimes \rho_j)\mathrm{CG}_{ij}^{-1} = \bigoplus_{k \in \hat{G}} \bigoplus_{s=1}^{m_{k,ij}} \rho_k, \tag{A.8}$$

where $m_{k,ij} \in \mathbb{N}_0$ is the multiplicity of irrep ρ_k in the tensor product, and ρ_k with $k \in \hat{G}$ covers all irreps of G. The *Clebsch-Gordan coefficients* CG_{ij} directly determine which specific irreps are consistent with an equivariant mapping between the irreducible input

representation ρ_i and output representation ρ_j. Both, the coefficients and the multiplicities are fixed by the choice of the group representations ρ_i, ρ_j [174].

A.2.3 Lie Algebra

For the purpose of finding all finite-dimensional irreps of matrix Lie groups, the *Lie algebra* is often preferred, as it provides an easier path towards deriving all irreps compared to directly studying them on the Matrix Lie group [64].

Definition A.2.3. *Given a matrix Lie group G. The Lie algebra* $\mathfrak{g}$ *of G is the set of all matrices* $\mathbf{X}$ *such that*

$$e^{t\mathbf{X}} \in G \,\forall\, t \in \mathbb{R} \tag{A.9}$$

with the ***matrix exponential*** $e^{\mathbf{X}}$

$$e^{\mathbf{X}} = \sum_{m=0}^{\infty} \frac{\mathbf{X}^m}{m!}, \tag{A.10}$$

where $\mathbf{X}^m$ *is the matrix multiplied m times with itself and* $\mathbf{X}^0$ *is the identity matrix.*

The derived Lie algebra representations π can be lifted to the corresponding Lie group representation ρ by means of the *exponential map* $\exp : \mathfrak{g} \to G$. Consequently, it is customary to find all irreps using the Lie algebra and lifting them to the corresponding group [64].

Appendix B

Implementation Details

B.1 "Scaling the Invariant Integration Layer Towards Real-World Problems"

To increase the reproducibility of our results, we provide our exact HP settings. We optimized the standard Wide-ResNets using SGD and the HPs of the corresponding paper [186]. All steerable networks were optimized using Adam optimization [89]. We used exponential learning rate decay for Rotated-MNIST and SVHN, while we employed step-wise decay on CIFAR-10 and STL-10. All steerable filter weights were regularized using elastic net regularization with factor 10^{-7} (cf. [176]). For all WRNs, we additionally use ℓ_2-regularization for the learnable BatchNorm coefficients with factor 0.1. All regularization losses were then multiplied by the regularization constant.

We optimized the HPs using Bayesian Optimization with Hyperband (BOHB, [44]) on 80/20 validation splits, if it was not already predetermined by the dataset. They are shown in Tables B.1-B.4. On Rotated-MNIST we used data augmentation with random rotations following [176]. On CIFAR-10 and SVHN, we followed [186] and used random crops and flips for CIFAR-10 and no data augmentation for SVHN. On STL-10, we use random crops, flips and cutout [31] .

For the monomial architectures, we applied II per channel, and pruned $M = 50$ initial monomials to $n_m = 5$ for Rot-MNIST and $n_m = 10$ monomials for SVHN, CIFAR-10 and STL-10. We used one intermediate pruning step after 10 epochs with $n_m = 25$ and train additional 5 epochs before the final pruning step. All other invariant integration layers were implemented with constant number of channels.

B.1.1 Number of Parameters

For our invariant architectures, we keep the number of parameters constant by reducing the number of channels accordingly. A standard convolutional filter with kernel size k, c_i

Table B.1: (Table 6 in Rath & Condurache [133]) II-SFCNN HPs on Rotated-MNIST.

HP	MH-SA	Global-WS	Local-WS	MLP	G-Conv+Avg-Pool	Monomials	SFCNN
Optimizer	Adam	Adam	Adam	Adam	Adam	Adam	Adam
Batch Size	32	32	32	32	32	32	64
Epochs	100	100	100	100	100	100	100
n_{FC}	95	85	30	85	85	90	96
Learning Rate	5e-3	1e-4	1e-3	1e-4	5e-4	1e-4	1e-3
LR Decay	0.5	0.1	0.5	0.1	0.2	0.75	0.9
LR Decay Epoch	20	40	25	30	25	15	20
Reg. Constant	1e-3	0.1	1e-3	1e-3	0.01	0.15	1.
Dropout Rate	0.05	0.45	0.4	0.5	0.4	0.45	0.7
Attention Heads	1	-	-	-	-	-	-
Attention Dropout	0.	-	-	-	-	-	-

Table B.2: (Table 7 in Rath & Condurache [133]) HPs on SVHN.

HP	SFCNN	Global-WS	Local-WS	Monomials
Optimizer	Adam	Adam	Adam	Adam
Batch Size	128	128	128	64
Epochs	100	100	100	100
n_{FC}	32	64	36	85
Learning Rate	1e-3	5e-4	5e-4	5e-4
LR Decay	0.4	0.1	0.25	0.25
LR Decay Epoch	20	30	25	20
Reg. Constant	2e-3	0.25	0.2	0.05
Dropout Rate	0.55	0.7	0.5	0.7

input channels and c_o output channels has $k^2 c_i c_o$ parameters. A rotation-steerable filter has $2 n_F n_\alpha c_i c_o$ parameters with n_α rotations and n_F basis filters. In order to keep the number of parameters constant, we equate

$$k^2 c_i c_o = 2 n_F n_\alpha \tilde{c}_i \tilde{c}_o \quad \Leftrightarrow \quad \frac{\tilde{c}_i \tilde{c}_o}{c_i c_o} = \frac{2 n_F n_\alpha}{k^2} \tag{B.1}$$

We use $k = 3$, $n_\alpha = 8$, $n_F = 16$ and obtain a final ratio of $\frac{256}{9}$ by which we need to reduce $c_i c_o$. Hence, we reduce the number of channels by $\sqrt{\frac{256}{9}} = \frac{16}{3}$. For the lifting convolution, the filter is not used among all rotations, so we only need to reduce the ratio by $\sqrt{\frac{32}{9}}$.

B.2 "Invariance to Multiple Transformations at Once"

We present details about the network architectures and the HPs needed to reproduce our results. We optimized the HPs using Bayesian optimization with hyperband (BOHB) [44]. For the baselines, we used the HPs reported in the respective papers or official implementations. If no validation split was pre-defined, we used a 80-20 train-validation

Table B.3: (Table 8 in Rath & Condurache [133]) HPs on CIFAR-10.

HP	SFCNN	Local-WS	Monomials
Optimizer	Adam	Adam	Adam
Batch Size	64	64	64
Epochs	100	100	100
n_{FC}	-	90	30
Learning Rate	1e-3	5e-4	5e-4
LR Decay	0.5	0.2	0.025
LR Decay Epoch	50	50	50
Reg. Constant	0.1	5e-6	0.008
Dropout Rate	0.3	0.1	0.4

Table B.4: (Table 9 in Rath & Condurache [133]) HPs on STL-10.

HP	SFCNN	Local-WS	Monomials
Optimizer	Adam	Adam	Adam
Batch Size	96	64	32
Epochs	1000	1000	1000
n_{FC}	-	16	10
Learning Rate	5e-4	0.01	5e-4
LR Decay	0.1	0.3	0.1
LR Decay Epoch	300	300	300
Reg. Constant	1e-8	1e-9	5e-9
Dropout Rate	0.1	0.15	0.05

split for the HP optimization. All networks were trained using a single (Scaled-MNIST, SVHN) or two (CIFAR-10, STL-10) NVIDIA GTX-1080 Ti GPUs. For SVHN, CIFAR-10 and STL-10 we used WRNs as baseline architectures [186].

For the E(2)-invariant network, we reduced the size of the final trivial representation to the same size used by the Standard- and Scale-WRN. For the original implementation, the final representation is multiplied by $\sqrt{|G|}$ where $|G|$ is the order of the group the last block is equivariant to [173]. For the Scale-II WRNs, we added BatchNorm after the II layer to stabilize the gradients backpropagated through the division within the Scale-II.

For the dual- and triple-stream architectures with constant parameters, we naively divided each channel size C by $\lceil \frac{C}{\sqrt{2}} \rceil$ or $\lceil \frac{C}{\sqrt{3}} \rceil$, respectively. We did not further tune HPs for the smaller-sized streams.

We built upon the official code-bases of Sosnovik et al. ([160], `https://github.com/ISosnovik/disco`), as well as Weiler and Cesa ([173], `https://github.com/QUVA-Lab/e2cnn`), which we both ported to Tensorflow v2.3. We used the models with a constant number of parameters and inserted the II layer to replace the spatial pooling operation. Specifically for DISCO, we had to thoroughly re-create all default settings from PyTorch in Tensorflow in order to recreate the performance. This

Table B.5: (Table 6 in Rath & Condurache [134]) II-DISCO HPs on Scaled-MNIST. Parameters with ⋆ were optimized using BOHB.

HP	II-DISCO WS	II-DISCO Monomials
Batch Size	128	128
Learning Rate ⋆	5e-3	1e-3
Weight Decay ⋆	5e-7	5e-6
BatchNorm Decay ⋆	0.01	5e-4
Dropout Rate ⋆	0.1	0.7

Table B.6: (Table 7 in Rath & Condurache [134]) II-DISCO architecture on Scaled-MNIST.

Layer	Output Size	C_{in}	C_{out}	n_S	ReLU	BatchNorm	Dropout
Upsampling	56×56	1	1	-	x	x	x
Scale-LiftConv	56×56	1	32	3	✓	✓	x
MaxPool	28×28	32	32	3	x	x	x
Scale-Conv1	28×28	32	63	3	✓	✓	x
MaxPool	14×14	63	63	3	x	x	x
Scale-Conv2	14×14	63	95	3	✓	✓	x
Scale-MaxProj	14×14	95	95	1	x	x	x
Scale-II	1×1	95	95	-	x	x	x
Dense1	-	95	256	-	✓	✓	✓
Dense2	-	256	10	-	x	x	x

included variable initializers, batch norm parameters and the Adam optimizer's HPs. Additionally, we had to use a custom-written *grouped convolution* within the scale-equivariant convolution code rather than using *tf.nn.conv2d* with a filter with reduced input-channels which implicitly also calculates a grouped convolution.

B.2.1 Scaled-MNIST

On Scaled-MNIST, we used the CNN proposed by [160, 162] composed of three convolutional and two dense layers with an effective kernel size of 7 and $n_S = 4$ scales. We adapted it by using scale-II with $k = 3$ instead of the spatial pooling layer that is a combination of average and max pooling. We trained our network for 60 epochs using the Adam optimizer [89] with step-wise learning rate decay of 0.1 after 20 and 40 epochs, l_2-regularization and data augmentation, where we artificially scaled the input with $s \in [0.5, 2]$. The network details can be found in Table B.6 and the used HPs in Table B.5.

Table B.7: (Table 8 in Rath & Condurache [134]) HPs used for SVHN experiments. Parameters with ⋆ were optimized using BOHB.

HP	E(2)-II	Scale-II
Batch Size	128	128
Learning Rate ⋆	5e-3	0.2
Weight Decay ⋆	5e-3	1e-4
Dropout Rate ⋆	0.5	0.4

Table B.8: (Table 9 in Rath & Condurache [134]) Combination head HPs on SVHN.

HP	Dual	Triple
Batch Size	128	128
Learning Rate	0.01	0.01
Weight Decay	1e-4	1e-4

B.2.1.1 SVHN

On the SVHN dataset, we used a WRN16-4 with pre-activation nonlinearites as baseline architecture. The E(2)-networks are equivariant to flips and $n_r = 8$ rotations for each convolutional layer. The scale-convolutions use $n_S = 3$ scales. We used E(2)-II with $k = 3$, $n_r = 8$ angles and $n_F = 2$ flips using bilinear interpolation where necessary. We applied Scale-II with $k = 3$. We followed the training approach by [186]. We used an SGD optimizer with Momentum 0.9, trained for 160 epochs and reduced the learning rate by 0.1 after 80 and 120 epochs. We did not apply any data augmentation. The detailed architecture of the E(2)-II-WRN16-4 is shown in Table B.9, the architecture of the Scale-II-WRN16-4 in Table B.10. The used HPs are shown in Table B.7.

For the classification head combining several invariant streams, we froze all trained convolutional and II layers and combined the individual streams via a learnable mapping and a weighted sum. We then trained those layer as well as the final dense layers. We used the same training settings as above, but divided the number of training epochs as well as all epoch-dependent HPs by 4. The HPs of the refinement training are listed in Table B.8.

B.2.2 CIFAR-10

On CIFAR-10, we used a WRN28-10 as the baseline network. The E(2)-networks are equivariant to flips and $n_r = 8$ rotations for the first residual block, $n_r = 4$ rotations for the second and third residual blocks. The scale-convolutions use $n_S = 3$ scales. We

Table B.9: (Table 10 in Rath & Condurache [134]) E(2)-II-WRN16-4 architecture for SVHN. Each equivariant residual block consists of two consecutive convolution layers (e.g., E(2)-Conv1.1 and E(2)-Conv1.2) and a shortcut connection with a 1x1-convolution whenever the output size changes.

Layer	Output Size	C_{in}	C_{out}	n_r	n_F	ReLU	BN	Dropout
E(2)-LiftConv	32×32	3	4	8	2	✓	✓	x
E(2)-Conv1.1	32×32	4	16	8	2	✓	✓	x
E(2)-Conv1.2	32×32	16	16	8	2	✓	✓	✓
E(2)-Conv1.3	32×32	16	16	8	2	✓	✓	x
E(2)-Conv1.4	32×32	16	16	8	2	✓	✓	✓
E(2)-Conv2.1	16×16	16	32	8	2	✓	✓	x
E(2)-Conv2.2	16×16	32	32	8	2	✓	✓	✓
E(2)-Conv2.3	16×16	32	32	8	2	✓	✓	x
E(2)-Conv2.4	16×16	32	32	8	2	✓	✓	✓
E(2)-Conv3.1	8×8	32	64	8	2	✓	✓	x
E(2)-Conv3.2	8×8	64	64	8	2	✓	✓	✓
E(2)-Conv3.3	8×8	64	64	8	2	✓	✓	x
E(2)-Conv3.4	8×8	64	256	1	1	✓	✓	✓
E(2)-II	-	256	256	-	-	x	x	x
Dense	-	256	10	-	-	x	x	x

used E(2)-II with $k = 3$, $n_r = 4$ angles and $n_F = 2$ flips. We applied Scale-II with $k = 3$. We again followed the training approach by [186]. We used an SGD optimizer with Momentum 0.9, trained for 200 epochs and reduced the learning rate by 0.2 after 60, 120 and 160 epochs. We used data augmentation with random pads-and-crops as well as random flips. The detailed architecture of the E(2)-II-WRN28-10 is shown in Table B.13, the architecture of the Scale-II-WRN28-10 in Table B.14. The HPs can be found in Table B.11. We used the same approach for the combination heads as for SVHN where we divided all epochs by 4. The HPs of the re-training are shown in Table B.12.

B.2.3 STL-10

On STL-10, we used a WRN16-8 as the baseline network. The E(2)-networks are equivariant to flips as well as $n_r = 8$ rotations for the first residual block, $n_r = 4$ rotations for the second and $n_r = 1$ rotations for the third residual block. The scale-convolutions use $n_S = 3$ scales. We used E(2)-II with $k = 3$, $n_r = 1$ angles and $n_F = 2$ flips and Scale-II with $k = 3$. We used an SGD optimizer with Nesterov Momentum 0.9, trained for 1000 epochs and reduced the learning rate by 0.2 after 300, 400, 600 and 800 epochs. We used data augmentation with random pads-and-crops, flips and Cutout [31]. The detailed architecture of the E(2)-II-WRN16-8 is shown in Table B.17, the architecture of the Scale-

Table B.10: (Table 11 in Rath & Condurache [134]) Scale-II-WRN16-4 architecture for SVHN. Each equivariant residual block consists of two consecutive convolution layers (e.g., Scale-Conv1.1 and Scale-Conv1.2) and a shortcut connection with a 1x1-convolution whenever the output size changes.

Layer	Output Size	C_{in}	C_{out}	n_S	ReLU	BN	Dropout
Scale-LiftConv	32×32	3	16	3	✓	✓	x
Scale-Conv1.1	32×32	16	64	3	✓	✓	x
Scale-Conv1.2	32×32	64	64	3	✓	✓	✓
Scale-Conv1.3	32×32	64	64	3	✓	✓	x
Scale-Conv1.4	32×32	64	64	3	✓	✓	✓
Scale-Conv2.1	16×16	64	128	3	✓	✓	x
Scale-Conv2.2	16×16	128	128	3	✓	✓	✓
Scale-Conv2.3	16×16	128	128	3	✓	✓	x
Scale-Conv2.4	16×16	128	128	3	✓	✓	✓
Scale-Conv3.1	8×8	128	256	3	✓	✓	x
Scale-Conv3.2	8×8	256	256	3	✓	✓	✓
Scale-Conv3.3	8×8	256	256	3	✓	✓	x
Scale-Conv3.4	8×8	256	256	3	✓	✓	✓
Scale-MaxProj	8×8	256	256	1	x	x	x
Scale-II	1×1	256	256	-	x	✓	x
Dense	-	256	10	-	x	x	x

Table B.11: (Table 12 in Rath & Condurache [134]) HPs on CIFAR-10. Parameters with * were optimized using BOHB.

HP	E(2)-II	Scale-II
Batch Size	96	128
Learning Rate *	5e-3	0.1
Weight Decay *	5e-3	5e-4
Dropout Rate *	0.1	0.2

II-WRN16-8 in Table B.18. We adapted the official implementation of the Scale-networks to use a stride of 2 instead of 1 in the initial residual block – as done by [31, 173]. The used HPs can be found in Table B.15. We used the same approach for the combination heads as for SVHN and CIFAR-10, but this time divided all epochs by 10. The HPs of the re-training can be found in Table B.16.

Table B.12: (Table 13 in Rath & Condurache [134]) Combination head HPs on CIFAR-10.

HP	Dual	Triple
Batch Size	128	128
Learning Rate	0.01	0.01
Weight Decay	1e-3	1e-3

Table B.13: (Table 14 in Rath & Condurache [134]) E(2)-II-WRN28-10 architecture for CIFAR-10. Each equivariant residual block consists of two consecutive convolution layers (e.g., E(2)-Conv1.1 and E(2)-Conv1.2) and a shortcut connection with a 1x1-convolution whenever the output size changes.

Layer	Output Size	C_{in}	C_{out}	n_r	n_F	ReLU	BN	Dropout
E(2)-LiftConv	32×32	3	4	8	2	✓	✓	x
E(2)-Conv1.1	32×32	4	40	8	2	✓	✓	x
E(2)-Conv1.2	32×32	40	40	8	2	✓	✓	✓
E(2)-Conv1.3	32×32	40	40	8	2	✓	✓	x
E(2)-Conv1.4	32×32	40	40	8	2	✓	✓	✓
E(2)-Conv1.5	32×32	40	40	8	2	✓	✓	x
E(2)-Conv1.6	32×32	40	40	8	2	✓	✓	✓
E(2)-Conv1.7	32×32	40	40	8	2	✓	✓	x
E(2)-Conv1.8	32×32	40	40	8	2	✓	✓	✓
E(2)-Conv2.1	16×16	40	113	4	2	✓	✓	x
E(2)-Conv2.2	16×16	113	113	4	2	✓	✓	✓
E(2)-Conv2.3	16×16	113	113	4	2	✓	✓	x
E(2)-Conv2.4	16×16	113	113	4	2	✓	✓	✓
E(2)-Conv2.5	16×16	113	113	4	2	✓	✓	x
E(2)-Conv2.6	16×16	113	113	4	2	✓	✓	✓
E(2)-Conv2.7	16×16	113	113	4	2	✓	✓	x
E(2)-Conv2.8	16×16	113	113	4	2	✓	✓	✓
E(2)-Conv3.1	8×8	113	226	4	2	✓	✓	x
E(2)-Conv3.2	8×8	226	226	4	2	✓	✓	✓
E(2)-Conv3.3	8×8	226	226	4	2	✓	✓	x
E(2)-Conv3.4	8×8	226	226	4	2	✓	✓	✓
E(2)-Conv3.5	8×8	226	226	4	2	✓	✓	x
E(2)-Conv3.6	8×8	226	226	4	2	✓	✓	✓
E(2)-Conv3.7	8×8	226	226	4	2	✓	✓	x
E(2)-Conv3.8	8×8	226	640	1	1	✓	✓	✓
E(2)-II	-	640	640	-	-	x	x	x
Dense	-	640	10	-	-	x	x	x

Table B.14: (Table 15 in Rath & Condurache [134]) Scale-II-WRN28-10 architecture for CIFAR-10. Each equivariant residual block consists of two consecutive convolution layers (e.g., Scale-Conv1.1 and Scale-Conv1.2) and a shortcut connection with a 1x1-convolution whenever the output size changes.

Layer	Output Size	C_{in}	C_{out}	n_S	ReLU	BN	Dropout
Scale-LiftConv	32×32	3	16	3	✓	✓	x
Scale-Conv1.1	32×32	16	160	3	✓	✓	x
Scale-Conv1.2	32×32	160	160	3	✓	✓	✓
Scale-Conv1.3	32×32	160	160	3	✓	✓	x
Scale-Conv1.4	32×32	160	160	3	✓	✓	✓
Scale-Conv1.5	32×32	160	160	3	✓	✓	x
Scale-Conv1.6	32×32	160	160	3	✓	✓	✓
Scale-Conv1.7	32×32	160	160	3	✓	✓	x
Scale-Conv1.8	32×32	160	160	3	✓	✓	✓
Scale-Conv2.1	16×16	160	320	3	✓	✓	x
Scale-Conv2.2	16×16	320	320	3	✓	✓	✓
Scale-Conv2.3	16×16	320	320	3	✓	✓	x
Scale-Conv2.4	16×16	320	320	3	✓	✓	✓
Scale-Conv2.5	16×16	320	320	3	✓	✓	x
Scale-Conv2.6	16×16	320	320	3	✓	✓	✓
Scale-Conv2.7	16×16	320	320	3	✓	✓	x
Scale-Conv2.8	16×16	320	320	3	✓	✓	✓
Scale-Conv3.1	8×8	320	640	3	✓	✓	x
Scale-Conv3.2	8×8	640	640	3	✓	✓	✓
Scale-Conv3.3	8×8	640	640	3	✓	✓	x
Scale-Conv3.4	8×8	640	640	3	✓	✓	✓
Scale-Conv3.5	8×8	640	640	3	✓	✓	x
Scale-Conv3.6	8×8	640	640	3	✓	✓	✓
Scale-Conv3.7	8×8	640	640	3	✓	✓	x
Scale-Conv3.8	8×8	640	640	3	✓	✓	✓
Scale-MaxProj	8×8	640	640	1	x	x	x
Scale-II	1×1	640	640	-	x	✓	x
Dense	-	640	10	-	x	x	x

Table B.15: (Table 16 in Rath & Condurache [134]) HPs on STL-10. Parameters with * were optimized using BOHB.

HP	E(2)-II	Scale-II
Batch Size	96	96
Learning Rate *	2e-3	0.05
Weight Decay *	1e-2	1e-3
Dropout Rate *	0.1	0.3

Table B.16: (Table17 in Rath & Condurache [134]) Combination head HPs on STL-10.

HP	Dual	Triple
Batch Size	128	128
Learning Rate	0.01	0.01
Weight Decay	1e-4	1e-4

Table B.17: (Table 18 in Rath & Condurache [134]) E(2)-II-WRN16-8 architecture for STL-10. Each equivariant residual block consists of two consecutive convolution layers (e.g., E(2)-Conv1.1 and E(2)-Conv1.2) and a shortcut connection with a 1x1-convolution whenever the output size changes.

Layer	Output Size	C_{in}	C_{out}	n_r	n_F	ReLU	BN	Dropout
E(2)-LiftConv	96×96	3	4	8	2	✓	✓	x
E(2)-Conv1.1	48×48	4	32	8	2	✓	✓	x
E(2)-Conv1.2	48×48	32	32	8	2	✓	✓	✓
E(2)-Conv1.3	48×48	32	32	8	2	✓	✓	x
E(2)-Conv1.4	48×48	32	32	8	2	✓	✓	✓
E(2)-Conv2.1	24×24	32	90	4	2	✓	✓	x
E(2)-Conv2.2	24×24	90	90	4	2	✓	✓	✓
E(2)-Conv2.3	24×24	90	90	4	2	✓	✓	x
E(2)-Conv2.4	24×24	90	90	4	2	✓	✓	✓
E(2)-Conv3.1	12×12	90	362	1	2	✓	✓	x
E(2)-Conv3.2	12×12	362	362	1	2	✓	✓	✓
E(2)-Conv3.3	12×12	362	362	1	2	✓	✓	x
E(2)-Conv3.4	12×12	362	512	1	1	✓	✓	✓
E(2)-II	-	512	512	-	-	x	x	x
Dense	-	512	10	-	-	x	x	x

Table B.18: (Table 19 in Rath & Condurache [134]) Scale-II-WRN16-8 architecture for STL-10. Each equivariant residual block consists of two consecutive convolution layers (e.g., Scale-Conv1.1 and Scale-Conv1.2) and a shortcut connection with a 1x1-convolution whenever the output size changes.

Layer	Output Size	C_{in}	C_{out}	n_S	ReLU	BN	Dropout
Scale-LiftConv	96×96	3	32	3	✓	✓	x
Scale-Conv1.1	48×48	32	128	3	✓	✓	x
Scale-Conv1.2	48×48	128	128	3	✓	✓	✓
Scale-Conv1.3	48×48	128	128	3	✓	✓	x
Scale-Conv1.4	48×48	128	128	3	✓	✓	✓
Scale-Conv2.1	24×24	128	256	3	✓	✓	x
Scale-Conv2.2	24×24	256	256	3	✓	✓	✓
Scale-Conv2.3	24×24	256	256	3	✓	✓	x
Scale-Conv2.4	24×24	256	256	3	✓	✓	✓
Scale-Conv3.1	12×12	256	512	3	✓	✓	x
Scale-Conv3.2	12×12	512	512	3	✓	✓	✓
Scale-Conv3.3	12×12	512	512	3	✓	✓	x
Scale-Conv3.4	12×12	512	512	3	✓	✓	✓
Scale-MaxProj	12×12	512	512	1	x	x	x
Scale-II	1×1	512	512	-	x	✓	x
Dense	-	512	10	-	x	x	x

Abbreviations & Symbols

Abbreviations

II Invariant Integration 6

irrep irreducible representation 35

LLM Large Language Model 1

LSE Least Square Error 8

MH-SA multi-head self-attention 16

ML Machine Learning 1

MLP Multi-Layer Perceptron 5

MTE Mean Test Error 69

PTN Polar Transformer Network 54

ReLU Rectified Linear Unit 78

SA self-attention 6

SGD Stochastic Gradient Descent 16

SES-CNN Scale-Equivariant Steerable Filter CNN 84

SFCNN Rotation-Equivariant Steerable Filter CNN 69

STN Spatial Transformer Network 54

SVM Support Vector Machine 63

TE Test Error 93

VAE Variational Autoencoder 42

VIT Vision Transformer 6

WRN Wide-ResNet 72

Symbols

$*$	(Translation) convolution
$\star$	(Translation) correlation
$\star_G$	Group convolution
$\rtimes$	Semi-Direct matrix product
$\bigoplus$	Direct sum of group representations
$\odot$	Hadamard product
$\otimes$	Tensor product
$\subset$	Subset or Subgroup
$\circ$	Group operation
$\frac{\partial f(x)}{\partial x}$	Partial derivative of $f(x)$ w.r.t. x
$\mathbb{1}$	Identity transformation
$a, \mathbf{a}$	Output of a linear matrix multiplication, called activation $a \in \mathbb{R}, \mathbf{a} \in \mathbb{R}^m$
$\mathbf{A}$	Attention scores
$A_G\{\cdot\}$	Global average pooling
$A[f](\mathbf{x})$	Group average based on polynomials $f(\mathbf{x})$
$A[m](\mathbf{x})$	Group average based on monomials $m(\mathbf{x})$
$A[\text{MLP}](\mathbf{x})$	Group average based on a MLP
$A[\text{SA}](\mathbf{x})$	Group average based on SA
$A[\text{WS}](\mathbf{x})$	Group average based on a WS
$A_T[f](\mathbf{x})$	Translation-invariant group average
$A_{G_s}[f](\mathbf{x})$	Scale-invariant group average
b_i	Monomial exponents
b, $\mathbf{b}$	Bias of a DNN layer
B	Batch Size
C	Number of classes to predict for a classification task
$\mathbb{C}$	Set of complex numbers

C_i	Number of input channels of a specific DNN layer
C_l	Number of output channels of DNN layer l
C_o	Number of output channels of a specific DNN layer
d_u, d_v	Monomial distance
$\mathrm{d}\mu$	Haar measure
D	Number of dimensions of a tensor
$\mathcal{D}$	Dataset used to train a DNN consisting of input-output-pairs $\mathcal{D} = \{\mathbf{X}, \mathbf{Y}\}$
$e^{\mathbf{X}}$	Matrix exponential
$E[x]$	Expected value
$f^h(g)$	Transformation-steerable filter
$\mathcal{F}$	Feature or output space of a DNN $\mathcal{F} \in \mathbb{R}^n$
g	Group element $g \in G$
G	Group consisting of a set G and a group operation $\circ$
$\lvert G \rvert$	Group order
G_x	Stabilizer subgroup
$\mathcal{G}$	Graph
H	Height of a 2D input
$H(x)$	Heaviside step function
$\mathbf{I}$	Identity Matrix
k	Kernel size of a convolutional filter
K	Monomial order
$\mathbf{K}$	Key matrix used to compute attention
$\mathcal{L}$	Loss function
L	Number of DNN layers
$m(\mathbf{x})$	Monomial of order K
M	Number of monomials
$\max_k$	Maximum within neighbourhood of extent k
n_α	Number of angles used for G-Convs and II
n_m	Number of selected monomials
n_s	Number of scales used for G-Convs
$\mathbf{O}$	Transformation matrix covering rotations & flips
$\mathcal{O}()$	Big O notation used to describe the computational complexity

$O(x)$	Orbit of a group
$\mathbf{P}$	Permutation matrix
$\mathbf{P}_{x_j - x_i}$	Positional encoding between pixel i and j used for 2D visual attention
$\mathbf{Q}$	Query matrix used to compute attention
$\mathbb{R}$	Set of real numbers
$\mathbb{R}^+$	Set of all positive real numbers
$\mathbf{R}$	Rotation Matrix
$\mathcal{R}$	Set of Rotations
s	Scale factor
s_j	Pruning score
t	Time step
T	Number of tokens used in an attention layer
$\mathfrak{T}$	Group action
$\mathcal{T}$	Left group action
$\tilde{\mathcal{T}}$	Right group action
$\mathcal{T}_{\mathbf{A}}$	Left group action of $\mathrm{GL}(n, \mathbb{R})$
$\mathcal{T}_f$	2D Flip operator, left group action of the flip group
$\mathcal{T}_g$	Left group action for a specific element $g \in G$
$\mathcal{T}_s$	Scale operator, left semigroup action of the scale semigroup
$\mathcal{T}_\phi$	2D Rotation operator, left group action of SO(2)
$\mathcal{T}_t$	Translation operator, left group action of T(2)
$\mathcal{T}_{t,\mathbf{O}}$	Left group action of E(2) covering 2D translations, rotation and flips
$\mathbf{V}$	Value matrix used to compute attention
$\mathcal{V}$	Vector space
W	Width of a 2D input
$\mathbf{w}$	Learnable weight vector $\mathbf{w} \in \mathbb{R}^n$
$\mathbf{W}$	Learnable weight matrix $\mathbf{W} \in \mathbb{R}^{n \times m}$
$x, \mathbf{x}, \mathbf{X}$	Input scalar, x vector $\mathbf{x}$ or tensor $\mathbf{X}$
X	Set of elements
$x_{\min}$	Minimum value of an input x
$y, \mathbf{y}, \mathbf{Y}$	Output or target scalar y, vector $\mathbf{y}$ or tensor $\mathbf{Y}$
$\hat{\mathbf{y}}$	Predictions of a DNN

β	Smoothing factor of the exponential moving average
Δ	Invariance error
ε	Small constant to prevent division by 0
η	Learning rate of gradient descent
μ	Mean
∇	Nabla operator returning all partial derivatives as a vector
ν	Exponential moving average of all gradients, called momentum
ϕ	Orientation angle
π	Lie algebra representation of a group G on a vector space $\mathcal{V}$
ψ	Learnable convolutional filter
ρ	Group representation of a group G on a vector space $\mathcal{V}$
τ	Number of training epochs
θ	Parameters of a DNN
σ	Standard deviation
$\sigma(x)$	Non-linear activation function

Groups & Semigroups

Aff(n)	Affine group including all invertible, affine transformations
D_n	Dihedral symmetry group
E(n)	n-dimensional Euclidean group including rotations, flips and translations
G	Group
G_S	Scale-translation semigroup $G_S = T(n) \rtimes T$
GL(n, $\mathbb{R}$)	General Linear group consisting of all real-valued, invertible $n \times n$ matrices
O(n)	n-dimensional Orthogonal group including rotations and flips
p	Permutation group
S	Scale semigroup
SE(n)	n-dimensional Special Euclidean group including rotations and translations
SO(n)	n-dimensional Special Orthogonal group including rotations
SU($n, \mathbb{C}$)	Special Unitary group consisting of all complex-valued, unitary $n \times n$ matrices
T(n)	n-dimensional Translation group
T_4	Tetrahedral symmetry group
V	Klein's four group

Lists of Figures & Tables

List of Figures

List of Tables

Bibliography

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] D. Agrawal and J. Ostrowski. A classification of g-invariant shallow neural networks. In S. K. andS. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022.

[3] V. Andrearczyk, J. Fageot, V. Oreiller, X. Montet, and A. Depeursinge. Local rotation invariance in 3d cnns. *Medical Image Anal.*, 65:101756, 2020.

[4] A. Bardes, J. Ponce, and Y. LeCun. Vicreg: Variance-invariance-covariance regularization for self-supervised learning. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022.

[5] S. L. Batzner, T. E. Smidt, L. Sun, J. P. Mailoa, M. Kornbluth, N. Molinari, and B. Kozinsky. Se(3)-equivariant graph neural networks for data-efficient and accurate interatomic potentials. *CoRR*, abs/2101.03164, 2021.

[6] A. Behboodi, G. Cesa, and T. S. Cohen. A pac-bayesian generalization bound for equivariant networks. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022.

[7] E. J. Bekkers. B-spline cnns on lie groups. In *8th International Conference on*

Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020. OpenReview.net, 2020.

[8] E. J. Bekkers, M. W. Lafarge, M. Veta, K. A. J. Eppenhof, J. P. W. Pluim, and R. Duits. Roto-translation covariant convolutional networks for medical image analysis. In *Medical Image Computing and Computer Assisted Intervention - MICCAI 2018 - 21st International Conference, Granada, Spain, September 16-20, 2018, Proceedings, Part I*, pages 440–448, 2018.

[9] G. W. Benton, M. Finzi, P. Izmailov, and A. G. Wilson. Learning invariances in neural networks from training data. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[10] J. Brandstetter, R. Hesselink, E. van der Pol, E. J. Bekkers, and M. Welling. Geometric and physical quantities improve E(3) equivariant message passing. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022.

[11] J. Bruna and S. Mallat. Invariant scattering convolution networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1872–1886, Aug 2013.

[12] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. Spectral networks and locally connected networks on graphs. In Y. Bengio and Y. LeCun, editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

[13] G. Cesa, L. Lang, and M. Weiler. A program to build e(n)-equivariant steerable cnns. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022.

[14] L. Chen, P. Wu, K. Chitta, B. Jaeger, A. Geiger, and H. Li. End-to-end autonomous driving: Challenges and frontiers. *CoRR*, abs/2306.16927, 2023.

[15] A. Coates, A. Y. Ng, and H. Lee. An analysis of single-layer networks in unsupervised feature learning. In G. J. Gordon, D. B. Dunson, and M. Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence*

and Statistics, AISTATS 2011, Fort Lauderdale, USA, April 11-13, 2011, volume 15 of *JMLR Proceedings*, pages 215–223. JMLR.org, 2011.

[16] T. Cohen, M. Weiler, B. Kicanaoglu, and M. Welling. Gauge equivariant convolutional networks and the icosahedral CNN. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, pages 1321–1330, 2019.

[17] T. Cohen and M. Welling. Group equivariant convolutional networks. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pages 2990–2999, 2016.

[18] T. S. Cohen, M. Geiger, J. Köhler, and M. Welling. Spherical cnns. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, 2018.

[19] T. S. Cohen, M. Geiger, and M. Weiler. A general theory of equivariant cnns on homogeneous spaces. In H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pages 9142–9153, 2019.

[20] T. S. Cohen and M. Welling. Steerable cnns. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[21] A. P. Condurache and A. Mertins. Sparse representations and invariant sequence-feature extraction for event detection. *VISAPP 2012 - Proceedings of the International Conference on Computer Vision Theory and Applications*, 1, Jan 2012.

[22] B. Coors, A. Condurache, A. Mertins, and A. Geiger. Learning transformation invariant representations with weak supervision. In *International Conference on Computer Vision Theory and Applications*, 2018.

[23] B. Coors, A. P. Condurache, and A. Geiger. Nova: Learning to see in novel viewpoints and domains. In *2019 International Conference on 3D Vision, 3DV 2019, Québec City, QC, Canada, September 16-19, 2019*, pages 116–125. IEEE, 2019.

[24] F. Cotter and N. G. Kingsbury. Visualizing and improving scattering networks. In *27th IEEE International Workshop on Machine Learning for Signal Processing, MLSP 2017, Tokyo, Japan, September 25-28, 2017*, pages 1–6, 2017.

[25] F. Cotter and N. G. Kingsbury. A learnable scatternet: Locally invariant convolutional layers. In *2019 IEEE International Conference on Image Processing, ICIP 2019, Taipei, Taiwan, September 22-25, 2019*, pages 350–354, 2019.

[26] J. Dai, H. Qi, Y. Xiong, Y. Li, G. Zhang, H. Hu, and Y. Wei. Deformable convolutional networks. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pages 764–773. IEEE Computer Society, 2017.

[27] P. de Haan, M. Weiler, T. Cohen, and M. Welling. Gauge equivariant mesh cnns: Anisotropic convolutions on geometric graphs. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

[28] M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In D. D. Lee, M. Sugiyama, U. von Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 3837–3845, 2016.

[29] M. Defferrard, M. Milani, F. Gusset, and N. Perraudin. Deepsphere: a graph-based spherical CNN. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[30] N. Dehmamy, R. Walters, Y. Liu, D. Wang, and R. Yu. Automatic symmetry discovery with lie algebra convolutional network. In M. Ranzato, A. Beygelzimer, Y. N. Dauphin, P. Liang, and J. W. Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 2503–2515, 2021.

[31] T. Devries and G. W. Taylor. Improved regularization of convolutional neural networks with cutout. *CoRR*, abs/1708.04552, 2017.

[32] N. Dey, A. Chen, and S. Ghafurian. Group equivariant generative adversarial

networks. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

[33] N. Diaconu and D. E. Worrall. Affine self convolution. *CoRR*, abs/1911.07704, 2019.

[34] N. Diaconu and D. E. Worrall. Learning to convolve: A generalized weight-tying approach. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, pages 1586–1595, 2019.

[35] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

[36] D. S. Dummit and R. M. Foote. *Abstract Algebra*. Wiley, 3 edition, 2003.

[37] B. Elesedy and S. Zaidi. Provably strict generalisation benefit for equivariant models. In M. Meila and T. Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 2959–2969. PMLR, 2021.

[38] C. Esteves. Theoretical aspects of group equivariant neural networks. *CoRR*, abs/2004.05154, 2020.

[39] C. Esteves, C. Allen-Blanchette, A. Makadia, and K. Daniilidis. Learning SO(3) equivariant representations with spherical cnns. In *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part XIII*, pages 54–70, 2018.

[40] C. Esteves, C. Allen-Blanchette, X. Zhou, and K. Daniilidis. Polar transformer networks. In *International Conference on Learning Representations*, 2018.

[41] C. Esteves, K. Daniilidis, and A. Makadia. Labeling panoramas with spherical hourglass networks. *CoRR*, abs/1809.02123, 2018.

[42] C. Esteves, A. Makadia, and K. Daniilidis. Spin-weighted spherical cnns. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Advances*

in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual, 2020.

[43] C. Esteves, Y. Xu, C. Allen-Blanchette, and K. Daniilidis. Equivariant multi-view networks. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, pages 1568–1577. IEEE, 2019.

[44] S. Falkner, A. Klein, and F. Hutter. BOHB: Robust and efficient hyperparameter optimization at scale. In *Proceedings of the 35th International Conference on Machine Learning*, pages 1436–1445, 2018.

[45] I. Feige. Invariant-equivariant representation learning for multi-class data. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, pages 1882–1891, 2019.

[46] D. Feng, C. Haase-Schütz, L. Rosenbaum, H. Hertlein, C. Gläser, F. Timm, W. Wiesbeck, and K. Dietmayer. Deep multi-modal object detection and semantic segmentation for autonomous driving: Datasets, methods, and challenges. *IEEE Trans. Intell. Transp. Syst.*, 22(3):1341–1360, 2021.

[47] M. Finzi, G. Benton, and A. G. Wilson. Residual pathway priors for soft equivariance constraints. In M. Ranzato, A. Beygelzimer, Y. N. Dauphin, P. Liang, and J. W. Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 30037–30049, 2021.

[48] M. Finzi, S. Stanton, P. Izmailov, and A. G. Wilson. Generalizing convolutional neural networks for equivariance to lie groups on arbitrary continuous data. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 3165–3176. PMLR, 2020.

[49] M. Finzi, M. Welling, and A. G. Wilson. A practical method for constructing equivariant multilayer perceptrons for arbitrary matrix groups. In M. Meila and T. Zhang, editors, *Proceedings of the 38th International Conference on Machine*

Learning, ICML 2021, 18-24 July 2021, Virtual Event, volume 139 of *Proceedings of Machine Learning Research*, pages 3318–3328. PMLR, 2021.

[50] D. Franzen and M. Wand. General nonlinearities in so(2)-equivariant cnns. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 9086–9098. Curran Associates, Inc., 2021.

[51] W. T. Freeman and E. H. Adelson. The design and use of steerable filters. *IEEE Trans. Pattern Anal. Mach. Intell.*, 13(9):891–906, 1991.

[52] F. Fuchs, D. E. Worrall, V. Fischer, and M. Welling. Se(3)-transformers: 3d roto-translation equivariant attention networks. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[53] F. B. Fuchs, E. Wagstaff, J. Dauparas, and I. Posner. Iterative se(3)-transformers. In F. Nielsen and F. Barbaresco, editors, *Geometric Science of Information - 5th International Conference, GSI 2021, Paris, France, July 21-23, 2021, Proceedings*, volume 12829 of *Lecture Notes in Computer Science*, pages 585–595. Springer, 2021.

[54] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36:193–202, 1980.

[55] J. A. D. Gardner, B. Egger, and W. Smith. Rotation-equivariant conditional spherical neural fields for learning a natural illumination prior. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022.

[56] S. Gauthier, B. Thérien, L. Alsène-Racicot, M. Chaudhary, I. Rish, E. Belilovsky, M. Eickenberg, and G. Wolf. Parametric scattering networks. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2022, New Orleans, LA, USA, June 18-24, 2022*, pages 5739–5748. IEEE, 2022.

[57] M. Geiger and T. Smidt. e3nn: Euclidean neural networks. *CoRR*, abs/2207.09453, 2022.

[58] R. Gens and P. M. Domingos. Deep symmetry networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 2537–2545, 2014.

[59] R. Ghosh and A. K. Gupta. Scale steerable filters for locally scale-invariant convolutional neural networks. *CoRR*, abs/1906.03861, 2019.

[60] I. J. Goodfellow, Y. Bengio, and A. C. Courville. *Deep Learning*. Adaptive computation and machine learning. MIT Press, 2016.

[61] T. Gramss. Word recognition with the feature finding neural network (ffnn). In *Neural Networks for Signal Processing Proceedings of the 1991 IEEE Workshop*, pages 289–298, 1991.

[62] N. Gruver, M. A. Finzi, M. Goldblum, and A. G. Wilson. The lie derivative for measuring learned equivariance. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023.

[63] J. Guo, K. Han, H. Wu, Y. Tang, X. Chen, Y. Wang, and C. Xu. CMT: convolutional neural networks meet vision transformers. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2022, New Orleans, LA, USA, June 18-24, 2022*, pages 12165–12175. IEEE, 2022.

[64] B. C. Hall. *Lie Groups, Lie Algebras, and Representations*. Springer International Publishing, 2nd edition, 2015.

[65] S. Han, J. Pool, J. Tran, and W. J. Dally. Learning both weights and connections for efficient neural network. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 1135–1143, 2015.

[66] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR*

2016, Las Vegas, NV, USA, June 27-30, 2016, pages 770–778. IEEE Computer Society, 2016.

[67] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In B. Leibe, J. Matas, N. Sebe, and M. Welling, editors, *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part IV*, volume 9908 of *Lecture Notes in Computer Science*, pages 630–645. Springer, 2016.

[68] L. He, Y. Chen, Z. Shen, Y. Dong, Y. Wang, and Z. Lin. Efficient equivariant network. In M. Ranzato, A. Beygelzimer, Y. N. Dauphin, P. Liang, and J. W. Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 5290–5302, 2021.

[69] L. He, Y. Dong, Y. Wang, D. Tao, and Z. Lin. Gauge equivariant transformer. In M. Ranzato, A. Beygelzimer, Y. N. Dauphin, P. Liang, and J. W. Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 27331–27343, 2021.

[70] J. F. Henriques and A. Vedaldi. Warped convolutions: Efficient invariance to spatial transformations. In D. Precup and Y. W. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 1461–1469. PMLR, 2017.

[71] J. Hestness, S. Narang, N. Ardalani, G. F. Diamos, H. Jun, H. Kianinejad, M. M. A. Patwary, Y. Yang, and Y. Zhou. Deep learning scaling is predictable, empirically. *CoRR*, abs/1712.00409, 2017.

[72] G. E. Hinton, A. Krizhevsky, and S. D. Wang. Transforming auto-encoders. In T. Honkela, W. Duch, M. A. Girolami, and S. Kaski, editors, *Artificial Neural Networks and Machine Learning - ICANN 2011 - 21st International Conference on Artificial Neural Networks, Espoo, Finland, June 14-17, 2011, Proceedings, Part I*, volume 6791 of *Lecture Notes in Computer Science*, pages 44–51. Springer, 2011.

[73] G. E. Hinton, S. Sabour, and N. Frosst. Matrix capsules with EM routing. In *6th*

International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings, 2018.

[74] P. Holderrieth, M. Hutchinson, and Y. W. Teh. Equivariant learning of stochastic fields: Gaussian processes and steerable conditional neural processes. In M. Meila and T. Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 4297–4307. PMLR, 2021.

[75] E. Hoogeboom, J. W. T. Peters, T. S. Cohen, and M. Welling. Hexaconv. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

[76] M. Horie, N. Morita, T. Hishinuma, Y. Ihara, and N. Mitsume. Isometric transformation invariant and equivariant graph convolutional networks. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

[77] Y. Hu, J. Yang, L. Chen, K. Li, C. Sima, X. Zhu, S. Chai, S. Du, T. Lin, W. Wang, L. Lu, X. Jia, Q. Liu, J. Dai, Y. Qiao, and H. Li. Planning-oriented autonomous driving. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2023, Vancouver, BC, Canada, June 17-24, 2023*, pages 17853–17862. IEEE, 2023.

[78] A. Hurwitz. über die erzeugung der invarianten durch integration. *Nachrichten von der Gesellschaft der Wissenschaften zu Göttingen, Mathematisch-Physikalische Klasse*, 1897:71–2, 1897.

[79] M. Hutchinson, C. L. Lan, S. Zaidi, E. Dupont, Y. W. Teh, and H. Kim. Lietransformer: Equivariant self-attention for lie groups. In M. Meila and T. Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 4533–4543. PMLR, 2021.

[80] A. Immer, T. F. A. van der Ouderaa, G. Rätsch, V. Fortuin, and M. van der Wilk. Invariance learning in deep neural networks with differentiable laplace approximations. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and

A. Oh, editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022.

[81] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In F. R. Bach and D. M. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 448–456. JMLR.org, 2015.

[82] M. M. Issakkimuthu and K. V. Subrahmanyam. So(2)-equivariance in neural networks using tensor nonlinearity. In *30th British Machine Vision Conference 2019, BMVC 2019, Cardiff, UK, September 9-12, 2019*, page 86. BMVA Press, 2019.

[83] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu. Spatial transformer networks. In *Advances in Neural Information Processing Systems 28*, pages 2017–2025. Curran Associates, Inc., 2015.

[84] Z. Ji, N. Lee, R. Frieske, T. Yu, D. Su, Y. Xu, E. Ishii, Y. Bang, A. Madotto, and P. Fung. Survey of hallucination in natural language generation. *ACM Comput. Surv.*, 55(12):248:1–248:38, 2023.

[85] C. M. Jiang, J. Huang, K. Kashinath, Prabhat, P. Marcus, and M. Nießner. Spherical cnns on unstructured grids. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[86] J. M. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Zídek, A. Potapenko, A. Bridgland, C. Meyer, S. A. A. Kohl, A. Ballard, A. Cowie, B. Romera-Paredes, S. Nikolov, R. Jain, J. Adler, T. Back, S. Petersen, D. A. Reiman, E. Clancy, M. Zielinski, M. Steinegger, M. Pacholska, T. Berghammer, S. Bodenstein, D. Silver, O. Vinyals, A. W. Senior, K. Kavukcuoglu, P. Kohli, and D. Hassabis. Highly accurate protein structure prediction with alphafold. *Nature*, 596:583 – 589, 2021.

[87] A. Kanazawa, A. Sharma, and D. W. Jacobs. Locally scale-invariant convolutional neural networks. *CoRR*, abs/1412.5104, 2014.

[88] M. Kawano, W. Kumagai, A. Sannai, Y. Iwasawa, and Y. Matsuo. Group equivariant conditional neural processes. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

[89] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In Y. Bengio and Y. LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[90] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[91] R. Kondor. N-body networks: a covariant hierarchical neural network architecture for learning atomic potentials. *CoRR*, abs/1803.01588, 2018.

[92] R. Kondor, Z. Lin, and S. Trivedi. Clebsch-gordan nets: a fully fourier space spherical convolutional neural network. In S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada*, pages 10138–10147, 2018.

[93] R. Kondor and S. Trivedi. On the generalization of equivariance and convolution in neural networks to the action of compact groups. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pages 2752–2760, 2018.

[94] A. R. Kosiorek, S. Sabour, Y. W. Teh, and G. E. Hinton. Stacked capsule autoencoders. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pages 15486–15496, 2019.

[95] A. Krizhevsky. Learning multiple layers of features from tiny images,. Technical report, 2009.

[96] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep

convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[97] H. Kvinge, T. Emerson, G. Jorgenson, S. Vasquez, T. Doster, and J. Lew. In what ways are deep neural networks invariant and how should we measure this? In A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho, editors, *Advances in Neural Information Processing Systems*, 2022.

[98] L. Lang and M. Weiler. A wigner-eckart theorem for group equivariant convolution kernels. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

[99] D. Laptev, N. Savinov, J. M. Buhmann, and M. Pollefeys. TI-POOLING: transformation-invariant pooling for feature learning in convolutional neural networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 289–297, 2016.

[100] H. Larochelle, D. Erhan, A. C. Courville, J. Bergstra, and Y. Bengio. An empirical evaluation of deep architectures on problems with many factors of variation. In *Machine Learning, Proceedings of the Twenty-Fourth International Conference (ICML 2007), Corvallis, Oregon, USA, June 20-24, 2007*, pages 473–480, 2007.

[101] Q. V. Le, J. Ngiam, Z. Chen, D. J. hao Chia, P. W. Koh, and A. Y. Ng. Tiled convolutional neural networks. In J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23: 24th Annual Conference on Neural Information Processing Systems 2010. Proceedings of a meeting held 6-9 December 2010, Vancouver, British Columbia, Canada*, pages 1279–1287. Curran Associates, Inc., 2010.

[102] Y. LeCun. A path towards autonomous machine intelligence, June 2022.

[103] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Handwritten digit recognition with a back-propagation network. In D. Touretzky, editor, *Advances in Neural Information Processing Systems (NIPS 1989)*, volume 2, Denver, CO, 1990. Morgan Kaufman.

[104] N. Lee, T. Ajanthan, and P. H. S. Torr. Snip: single-shot network pruning based on connection sensitivity. In *7th International Conference on Learning Repre-*

sentations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019. OpenReview.net, 2019.

[105] J. E. Lenssen, M. Fey, and P. Libuschewski. Group equivariant capsule networks. In S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada*, pages 8858–8867, 2018.

[106] Y. Liu, Z. Shen, Z. Lin, S. Peng, H. Bao, and X. Zhou. GIFT: learning transformation-invariant dense visual descriptors via group cnns. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pages 6990–7001, 2019.

[107] J. Lust and A. P. Condurache. A survey on assessing the generalization envelope of deep neural networks at inference time for image classification. *CoRR*, abs/2008.09381, 2020.

[108] S. Mallat. Group invariant scattering. *Communications on Pure and Applied Mathematics*, 65, Oct 2012.

[109] D. Marcos, B. Kellenberger, S. Lobry, and D. Tuia. Scale equivariance in cnns with vector fields. *CoRR*, abs/1807.11783, 2018.

[110] D. Marcos, M. Volpi, N. Komodakis, and D. Tuia. Rotation equivariant vector field networks. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pages 5058–5067. IEEE Computer Society, 2017.

[111] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[112] T. Miyato, M. Koyama, and K. Fukumizu. Unsupervised learning of equivariant structure from sequences. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022.

[113] A. Moskalev, A. Sepliarskaia, I. Sosnovik, and A. W. M. Smeulders. Liegg: Studying learned lie group generators. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022.

[114] F. Müller and A. Mertins. Invariant-integration method for robust feature extraction in speaker-independent speech recognition. In *INTERSPEECH 2009, 10th Annual Conference of the International Speech Communication Association, Brighton, United Kingdom, September 6-10, 2009*, pages 2975–2978, 2009.

[115] F. Müller and A. Mertins. Invariant integration features combined with speaker-adaptation methods. In *INTERSPEECH 2010, 11th Annual Conference of the International Speech Communication Association, Makuhari, Chiba, Japan, September 26-30, 2010*, pages 2622–2625, 2010.

[116] F. Müller and A. Mertins. Contextual invariant-integration features for improved speaker-independent speech recognition. *Speech Communication*, 53(6):830–841, 2011.

[117] K. P. Murphy. *Probabilistic Machine Learning: An introduction*. MIT Press, 2022.

[118] A. Nasiri and T. Bepler. Unsupervised object representation learning using translation and rotation group equivariant VAE. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022.

[119] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng. Reading digits in natural images with unsupervised feature learning. *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.

[120] E. Noether. Der endlichkeitssatz der invarianten endlicher gruppen. *Mathematische Annalen*, 77:89–92, 1916.

[121] OpenAI. GPT-4 technical report. *CoRR*, abs/2303.08774, 2023.

[122] E. Oyallon, E. Belilovsky, and S. Zagoruyko. Scaling the scattering transform: Deep hybrid networks. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 5619–5628, Oct 2017.

[123] E. Oyallon, E. Belilovsky, S. Zagoruyko, and M. Valko. Compressing the input for cnns with the first-order scattering transform. In *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part IX*, pages 305–320, 2018.

[124] E. Oyallon and S. Mallat. Deep roto-translation scattering for object classification. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2865–2873, 2015.

[125] E. Oyallon, S. Mallat, and L. Sifre. Generic deep networks with wavelet scattering. In Y. Bengio and Y. LeCun, editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Workshop Track Proceedings*, 2014.

[126] E. Oyallon, S. Zagoruyko, G. Huang, N. Komodakis, S. Lacoste-Julien, M. Blaschko, and E. Belilovsky. Scattering Networks for Hybrid Representation Learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, page 11, Sept. 2018.

[127] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Z. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 8024–8035, 2019.

[128] D. A. Patterson, J. Gonzalez, U. Hölzle, Q. V. Le, C. Liang, L. Munguia, D. Rothchild, D. R. So, M. Texier, and J. Dean. The carbon footprint of machine learning training will plateau, then shrink. *Computer*, 55(7):18–28, 2022.

[129] N. Perraudin, M. Defferrard, T. Kacprzak, and R. Sgier. Deepsphere: Efficient

spherical convolutional neural network with healpix sampling for cosmological applications. *Astronomy and Computing*, 27:130 – 146, 2019.

[130] PixabayUser3194556. Puppy, nature, dog. https://pixabay.com/photos/puppy-dog-pet-collar-dog-collar-1903313/, 2016.

[131] O. Puny, M. Atzmon, E. J. Smith, I. Misra, A. Grover, H. Ben-Hamu, and Y. Lipman. Frame averaging for invariant and equivariant network design. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022.

[132] M. Rath and A. P. Condurache. Invariant integration in deep convolutional feature space. *28th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, ESANN 2020, Bruges, Belgium, October 2-4, 2020*, pages 103–108, 2020.

[133] M. Rath and A. P. Condurache. Improving the sample-complexity of deep classification networks with invariant integration. *Proceedings of the 17th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP 2022)*, 2022.

[134] M. Rath and A. P. Condurache. Deep neural networks with efficient guaranteed invariances. *26th International Conference on Artificial Intelligence and Statistics, AISTATS 2023, Valencia, Spain, April 25-27, 2023*, 2023.

[135] M. Rath and A. P. Condurache. Boosting deep neural networks with geometrical prior knowledge: a survey. *Artif. Intell. Rev.*, 57(4):95, 2024.

[136] S. Ravanbakhsh, J. G. Schneider, and B. Póczos. Equivariance through parameter-sharing. In D. Precup and Y. W. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 2892–2901. PMLR, 2017.

[137] S. Ren, K. He, R. B. Girshick, and J. Sun. Faster R-CNN: towards real-time object detection with region proposal networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 91–99, 2015.

[138] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer. High-resolution image synthesis with latent diffusion models. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2022, New Orleans, LA, USA, June 18-24, 2022*, pages 10674–10685. IEEE, 2022.

[139] D. W. Romero, E. J. Bekkers, J. M. Tomczak, and M. Hoogendoorn. Wavelet networks: Scale equivariant learning from raw waveforms. *CoRR*, abs/2006.05259, 2020.

[140] D. W. Romero and J. Cordonnier. Group equivariant stand-alone self-attention for vision. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

[141] D. W. Romero and M. Hoogendoorn. Co-attentive equivariant neural networks: Focusing equivariance on transformations co-occurring in data. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[142] D. W. Romero and S. Lohit. Learning partial equivariances from data. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022.

[143] C. Rommel, T. Moreau, and A. Gramfort. Deep invariant networks with differentiable augmentation layers. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022.

[144] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[145] J. J. Rotman. *An Introduction to the Theory of Groups*. Graduate Texts in Mathematics 148. Springer-Verlag New York, 4 edition, 1995.

[146] S. Sabour, N. Frosst, and G. E. Hinton. Dynamic routing between capsules. In *Advances in Neural Information Processing Systems 30: Annual Conference on*

Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA, pages 3856–3866, 2017.

[147] V. G. Satorras, E. Hoogeboom, and M. Welling. E(n) equivariant graph neural networks. In M. Meila and T. Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 9323–9332. PMLR, 2021.

[148] H. Schulz-Mirbach. On the existence of complete invariant feature spaces in pattern recognition. In *Pattern Recognition: Eleventh International Conference 1992*, pages 178 – 182, Aug 1992.

[149] H. Schulz-Mirbach. Algorithms for the construction of invariant features. In *Tagungsband Mustererkennung 1994 (16. DAGM Symposium), Reihe Informatik Xpress, Nr.5*, pages 324–332, 1994.

[150] H. Schulz-Mirbach. Invariant features for gray scale images. In *Mustererkennung 1995, 17. DAGM-Symposium, Bielefeld, 13.-15. September 1995, Proceedings*, pages 1–14, 1995.

[151] M. Shakerinava, A. K. Mondal, and S. Ravanbakhsh. Structuring representations using group invariants. In A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho, editors, *Advances in Neural Information Processing Systems*, 2022.

[152] M. Shakerinava and S. Ravanbakhsh. Equivariant networks for pixelized spheres. In M. Meila and T. Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 9477–9488. PMLR, 2021.

[153] S. Shalev-Shwartz and S. Ben-David. *Understanding Machine Learning - From Theory to Algorithms*. Cambridge University Press, 2014.

[154] D. Shanmugam, D. W. Blalock, G. Balakrishnan, and J. V. Guttag. Better aggregation in test-time augmentation. In *2021 IEEE/CVF International Conference on Computer Vision, ICCV 2021, Montreal, QC, Canada, October 10-17, 2021*, pages 1194–1203. IEEE, 2021.

[155] P. Shaw, J. Uszkoreit, and A. Vaswani. Self-attention with relative position representations. In M. A. Walker, H. Ji, and A. 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.

[156] C. Shorten and T. M. Khoshgoftaar. A survey on image data augmentation for deep learning. *J. Big Data*, 6:60, 2019.

[157] L. Sifre and S. Mallat. Rotation, scaling and deformation invariant scattering for texture discrimination. In *2013 IEEE Conference on Computer Vision and Pattern Recognition, Portland, OR, USA, June 23-28, 2013*, pages 1233–1240, 2013.

[158] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. P. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of go with deep neural networks and tree search. *Nat.*, 529(7587):484–489, 2016.

[159] K. Sohn and H. Lee. Learning invariant representations with local transformations. In *Proceedings of the 29th International Conference on Machine Learning, ICML 2012, Edinburgh, Scotland, UK, June 26 - July 1, 2012*. icml.cc / Omnipress, 2012.

[160] I. Sosnovik, A. Moskalev, and A. W. M. Smeulders. DISCO: accurate discrete scale convolutions. In *32nd British Machine Vision Conference 2021, BMVC 2021, Online, November 22-25, 2021*, page 334. BMVA Press, 2021.

[161] I. Sosnovik, A. Moskalev, and A. W. M. Smeulders. Scale equivariance improves siamese tracking. In *IEEE Winter Conference on Applications of Computer Vision, WACV 2021, Waikoloa, HI, USA, January 3-8, 2021*, pages 2764–2773. IEEE, 2021.

[162] I. Sosnovik, M. Szmaja, and A. W. M. Smeulders. Scale-equivariant steerable networks. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[163] R. Spezialetti, F. Stella, M. Marcon, L. Silva, S. Salti, and L. D. Stefano. Learning to orient surfaces by self-supervised spherical cnns. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Advances in Neural Information*

Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual, 2020.

[164] T. Swartz. Cat, kitten, pet. https://pixabay.com/photos/cat-kitten-pet-kitty-young-cat-551554/, 2014.

[165] K. S. Tai, P. Bailis, and G. Valiant. Equivariant transformer networks. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, pages 6086–6095, 2019.

[166] N. Thomas, T. Smidt, S. M. Kearnes, L. Yang, L. Li, K. Kohlhoff, and P. Riley. Tensor field networks: Rotation- and translation-equivariant neural networks for 3d point clouds. *CoRR*, abs/1802.08219, 2018.

[167] T. F. A. van der Ouderaa, D. W. Romero, and M. van der Wilk. Relaxing equivariance constraints with non-stationary continuous filters. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022.

[168] T. F. A. van der Ouderaa and M. van der Wilk. Learning invariant weights in neural networks. In J. Cussens and K. Zhang, editors, *Uncertainty in Artificial Intelligence, Proceedings of the Thirty-Eighth Conference on Uncertainty in Artificial Intelligence, UAI 2022, 1-5 August 2022, Eindhoven, The Netherlands*, volume 180 of *Proceedings of Machine Learning Research*, pages 1992–2001. PMLR, 2022.

[169] M. van der Wilk, M. Bauer, S. T. John, and J. Hensman. Learning invariances using the marginal likelihood. In S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 9960–9970, 2018.

[170] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on*

Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA, pages 5998–6008, 2017.

[171] S. R. Venkataraman, S. Balasubramanian, and R. R. Sarma. Building deep equivariant capsule networks. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[172] R. Walters, J. Li, and R. Yu. Trajectory prediction using equivariant continuous convolution. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

[173] M. Weiler and G. Cesa. General e(2)-equivariant steerable cnns. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pages 14334–14345, 2019.

[174] M. Weiler, P. Forré, E. Verlinde, and M. Welling. *Equivariant and Coordinate Independent Convolutional Networks*. 2023.

[175] M. Weiler, M. Geiger, M. Welling, W. Boomsma, and T. Cohen. 3d steerable cnns: Learning rotationally equivariant features in volumetric data. In S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada*, pages 10402–10413, 2018.

[176] M. Weiler, F. A. Hamprecht, and M. Storath. Learning steerable filters for rotation equivariant cnns. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 849–858, 2018.

[177] P. Wimmer, J. Mehnert, and A. P. Condurache. Dimensionality reduced training by pruning and freezing parts of a deep neural network: a survey. *Artif. Intell. Rev.*, 56(12):14257–14295, 2023.

[178] M. Winkels and T. S. Cohen. Pulmonary nodule detection in CT scans with equivariant cnns. *Medical Image Anal.*, 55:15–26, 2019.

[179] R. Winter, M. Bertolini, T. Le, F. Noé, and D. Clevert. Unsupervised learning of group invariant and equivariant representations. In S. Koyejo, S. Mohamed,

A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022.

[180] D. E. Worrall and G. J. Brostow. Cubenet: Equivariance to 3d rotation and translation. In *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part V*, pages 585–602, 2018.

[181] D. E. Worrall, S. J. Garbin, D. Turmukhambetov, and G. J. Brostow. Harmonic networks: Deep translation and rotation equivariance. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 7168–7177, 2017.

[182] D. E. Worrall and M. Welling. Deep scale-spaces: Equivariance over scale. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pages 7364–7376, 2019.

[183] J. Xu, H. Kim, T. Rainforth, and Y. W. Teh. Group equivariant subsampling. In M. Ranzato, A. Beygelzimer, Y. N. Dauphin, P. Liang, and J. W. Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 5934–5946, 2021.

[184] Y. Xu, T. Xiao, J. Zhang, K. Yang, and Z. Zhang. Scale-invariant convolutional neural networks. *CoRR*, abs/1411.6369, 2014.

[185] F. Yang, Z. Wang, and C. Heinze-Deml. Invariance-inducing regularization using worst-case transformations suffices to boost accuracy and spatial robustness. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pages 14757–14768, 2019.

[186] S. Zagoruyko and N. Komodakis. Wide residual networks. In R. C. Wilson, E. R. Hancock, and W. A. P. Smith, editors, *Proceedings of the British Machine Vision Conference 2016, BMVC 2016, York, UK, September 19-22, 2016*. BMVA Press, 2016.

[187] J. Zarka, L. Thiry, T. Angles, and S. Mallat. Deep network classification by scattering and homotopy dictionary learning. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[188] Zhou and Chellappa. Computation of optical flow using a neural network. In *IEEE 1988 International Conference on Neural Networks*, pages 71–78 vol.2, 1988.

[189] A. Zhou, T. Knowles, and C. Finn. Meta-learning symmetries by reparameterization. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

[190] W. Zhu, Q. Qiu, A. R. Calderbank, G. Sapiro, and X. Cheng. Scale-equivariant neural networks with decomposed convolutional filters. *CoRR*, abs/1909.11193, 2019.

[191] X. Zhu, C. Xu, and D. Tao. Commutative lie group VAE for disentanglement learning. In M. Meila and T. Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 12924–12934. PMLR, 2021.

Index