

Master's Thesis



Czech  
Technical  
University  
in Prague

**F3**

Faculty of Electrical Engineering  
Department of Computer Science

## Interpretable Symmetry-Aware Deep Learning for Planning

Martin Krutský

Supervisor: Ing. Gustav Šír, Ph.D.  
Field of study: Open Informatics  
Subfield: Artificial Intelligence  
May 2024



## I. Personal and study details

Student's name: **Krutský Martin** Personal ID number: **483792**  
Faculty / Institute: **Faculty of Electrical Engineering**  
Department / Institute: **Department of Computer Science**  
Study program: **Open Informatics**  
Specialisation: **Artificial Intelligence**

## II. Master's thesis details

Master's thesis title in English:

**Interpretable Symmetry-Aware Deep Learning for Planning**

Master's thesis title in Czech:

**Interpretovatelné hluboké učení se symetriemi pro plánování**

Guidelines:

With a growing interest in addressing planning tasks with deep learning methods, ranging from simple models to Graph Neural Networks (GNNs) [1] and Transformers [2], questions of data efficiency and model trustworthiness arise. Recently, Geometric Deep Learning [3] has emerged to address the former issue by designing neural models invariant to various symmetries, while the latter has normally been tackled by interpretability methods [4] rooted in relational logic formalisms [5].

The aim of this thesis is to explore intersections of the two aspects by delving into the inner structures of the neural models to advance their trustworthy applications in planning domains.

- 1) Review common deep learning approaches to planning problems [6], focusing on symmetry-invariant architectures, such as GNNs [1].
- 2) Select suitable planning domains to showcase the principles of interpretable symmetries, such as the Rubik's cube problem [7].
- 3) Explore the algebraic and group-theoretic properties [3] of the selected domains and architectures.
- 4) Propose, implement, and evaluate appropriate symmetry-aware models for the selected problems.
- 5) Review common model-agnostic [4] and planning-related [8] approaches to explaining predictions of neural networks, and analyze your models' predictions.

Bibliography / sources:

- [1] Wu, Zonghan, et al. "A comprehensive survey on graph neural networks." IEEE transactions on neural networks and learning systems 32.1 (2020): 4-24.
- [2] Joshi, Chaitanya. "Transformers are graph neural networks." The Gradient 7 (2020).
- [3] Bronstein, Michael M., et al. "Geometric deep learning: Grids, groups, graphs, geodesics, and gauges." arXiv preprint arXiv:2104.13478 (2021).
- [4] Arrieta, Alejandro Barredo, et al. "Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI." Information fusion 58 (2020): 82-115.
- [5] Sourek, Gustav, et al. "Lifted relational neural networks: Efficient learning of latent relational structures." Journal of Artificial Intelligence Research 62 (2018): 69-100.
- [6] Almasan, Paul, et al. "Deep reinforcement learning meets graph neural networks: Exploring a routing optimization use case." Computer Communications 196 (2022): 184-194.
- [7] Agostinelli, Forest, et al. "Solving the Rubik's cube with deep reinforcement learning and search." Nature Machine Intelligence 1.8 (2019): 356-363.
- [8] Tsirtsis, Stratis, Abir De, and Manuel Rodriguez. "Counterfactual explanations in sequential decision making under uncertainty." Advances in Neural Information Processing Systems 34 (2021): 30127-30139.

Name and workplace of master's thesis supervisor:

**Ing. Gustav Šír, Ph.D. Intelligent Data Analysis FEE**

Name and workplace of second master's thesis supervisor or consultant:

Date of master's thesis assignment: **01.02.2024** Deadline for master's thesis submission: **24.05.2024**

Assignment valid until: **21.09.2025**

\_\_\_\_\_  
Ing. Gustav Šír, Ph.D.  
Supervisor's signature

\_\_\_\_\_  
Head of department's signature

\_\_\_\_\_  
prof. Mgr. Petr Páta, Ph.D.  
Dean's signature

### III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

\_\_\_\_\_  
Date of assignment receipt

\_\_\_\_\_  
Student's signature

## Acknowledgements

I am very grateful to my supervisor for guiding me through the thesis, significantly helping me to improve my academic writing, and supporting me in my research interests.

I want to thank my family for creating an environment where I could concentrate on the thesis and my friends for reinforcing my self-confidence when I needed it.

## Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, May 21, 2024

## Abstract

In many domains, including automated planning, deep learning has become the mainstream approach to solving hard problems by learning from data without the need for domain-expert input. Such an approach is, however, often inefficient in terms of data and computation resource requirements. The situation can be improved by informing the neural architecture about the symmetries of the respective domain. This approach has been formalized as the Geometric Deep Learning framework. This work contributes to the framework by studying symmetries commonly found in planning problems and implementing a well-substantiated, symmetry-invariant graph neural architecture for the Rubik's cube problem based on its group-theoretical properties. In our cost-to-goal regression experiments, this architecture not only outperforms the previous state-of-the-art architecture, both in the mean absolute error and solved rate when employed as a heuristic in search algorithms, but it also exhibits a high degree of interpretability. We show that explanations based on this intrinsic property are preferable to standard explainer algorithms for graph neural networks.

**Keywords:** geometric deep learning, planning, interpretability, symmetries, Rubik's cube, graph neural networks

**Supervisor:** Ing. Gustav Šír, Ph.D.

## Abstrakt

V mnoha doménách včetně automatizovaného plánování se hluboké učení stalo hlavním přístupem k řešení složitých problémů pomocí učení se z dat bez nutnosti zapojení doménových expertů. Tento postup je ale často neefektivní vzhledem k množství potřebných dat a výpočetních prostředků. Situaci lze zlepšit informováním neuronové architektury o symetriích příslušné domény. Tento přístup byl zformalizován do frameworku geometrického hlubokého učení. Tato práce přispívá do frameworku studiem symetrií, které se běžně vyskytují v plánovacích problémech, a implementací dobře zdůvodněné grafové neuronové architektury, invariantní vůči symetriím, pro problém Rubikovy kostky založené na jejích vlastnostech z teorie grup. V našich regresních experimentech predikce ceny k cíli tato architektura nejenže překonává předchozí nejmodernější architekturu ve střední absolutní chybě a v míře vyřešených stavů při použití v prohledávacích algoritmech, ale také vykazuje vysokou úroveň vnitřní vysvětlitelnosti. Ukážeme, že vysvětlení založená na této vlastnosti jsou vhodnější než standardní vysvětlovací algoritmy pro grafové neuronové sítě.

**Klíčová slova:** geometrické hluboké učení, plánování, vysvětlitelnost, symetrie, Rubikova kostka, grafové neuronové sítě

**Překlad názvu:** Interpretovatelné hluboké učení se symetriemi pro plánování

# Contents

<b>1 Introduction</b>	<b>1</b>		
1.1 Problem Statement	2		
1.2 Related Work	2		
1.2.1 Related Work on Deep Learning in Planning	2		
1.2.2 Related Work on Neural Interpretability	3		
1.2.3 Related Work on Symmetries in Deep Learning	4		
1.2.4 Related Work on Solving Rubik’s Cube	5		
<b>2 Theoretical Background</b>	<b>7</b>		
2.1 AI Planning	7		
2.1.1 Planning Problems	8		
2.2 Planning Problems Analysis	8		
2.2.1 Planning Representations	9		
2.2.2 Planning Symmetries	10		
2.2.3 Planning Domain Selection	11		
2.2.4 Rubik’s Cube Planning Problem	12		
2.3 Rubik’s Cube Analysis	12		
2.3.1 Problem Structure	12		
2.3.2 Group Theory	15		
2.3.3 Domain symmetries	16		
2.4 Symmetry-Aware Deep Learning	18		
2.4.1 Equivalence Classes	19		
2.4.2 Conjugacy Classes	20		
2.4.3 Other Cube Equivalence Classes	20		
2.4.4 Invariant Neural Architecture Design	21		
2.4.5 Neural Symmetry Detection	22		
2.5 Parameters of GNN Interpretability Methods	26		
<b>3 Rubik’s Cube Experiments</b>	<b>27</b>		
3.1 Data	27		
3.1.1 Color Patterns	27		
3.1.2 Reverse Generation	27		
3.1.3 Sampling	28		
3.2 Results	28		
3.2.1 Architecture Design	28		
3.2.2 Pattern Representation Ablations	29		
3.2.3 Learning Performance	31		
3.2.4 Search Performance	34		
3.2.5 Performance Comparison Tables	35		
3.2.6 Implementation Details and Hyperparameters	36		
3.3 Interpretability of SymmetryNet	38		
3.3.1 Distinguishing Different Distances	38		
3.3.2 Explaining Single Cube Move with GNNExplainer	39		
3.3.3 Architecture-Based Interpretations	40		
3.3.4 Comparison of Interpretability Techniques	41		
<b>4 Conclusion</b>	<b>43</b>		
4.1 Future Work	43		
<b>Bibliography</b>	<b>45</b>		
<b>A Other Cube Pattern Representations</b>	<b>53</b>		
A.1 Eliminated Pattern Representation	53		
A.1.1 Surface-Distance Invariant	53		
A.2 CNN-based Global Invariant Ablation	54		
A.2.1 Face Crosses	54		
<b>B Learning Performance Plots</b>	<b>55</b>		
<b>C Source Code and Resources</b>	<b>57</b>		
C.1 Acknowledgments	57		

## Figures

2.1 The minimal examples of the object (left), atom (middle), and object-atom (right) representations for the Blocksworld domain with blocks $a$ , $b$ , and $c$ . . . . .	9
2.2 Our taxonomy of symmetries found in planning problems. . . . .	10
2.3 A solved cube state in the standard color ordering, with denoted face $F_j$ views ( $F, B, D, U, R, L$ ) composed of individual facelets $f_i^{F_j}$ in a flattened “cube cross” view (below), and an outline of three rotation and reflection axes in a 3D view (above).	13
2.4 The (standard) one-hot encoding of the cube state representation . . .	14
2.5 An original (A), a recoloring-equivalent (B), and a symmetry-equivalent (C) cube states, originating through a color permutation $\pi$ and a $90^\circ$ counter-clockwise geometric rotation $\phi$ around the front face. . . . .	17
2.6 The proposed symmetry-invariant architecture (Sec. 2.4.4). . . . .	23
3.1 Test-set MAE model performances when generalizing from decreasing fractions of the exhaustive dataset ( $d_{qt} = [1 \dots 5]$ ), displayed with a 95% confidence interval. . . . .	32
3.2 The effect of symmetry-invariance demonstrated through the models’ learning convergences when generalizing from 10% of the exhaustive dataset ( $d_{qt} = [1 \dots 5]$ ), displayed with a 95% confidence interval. . . . .	33
3.3 The effect of symmetry-invariance demonstrated through the models’ learning convergences on the full $10^5$ sampled dataset ( $d_{ft} = [14 \dots 19]$ ), displayed with a 95% confidence interval. . . . .	33
3.4 Models’ learning performances when generalizing to the symmetry-equivalent states from an increasingly large portion of the sampled dataset ( $d_{ft} = [14 \dots 19]$ ), displayed with a 95% confidence interval. . . . .	34
3.5 Model comparison through the problem-solving (search) performance metrics - (i) accuracy of selecting optimal successor, (ii) solution rates, and (iii) optimal solution rates (higher is better). . . . .	35
3.6 Log-scale comparison of the models’ A* search performances as measured through the average number of expanded nodes (lower is better). . .	35
3.7 A detailed depiction of the SymmetryNet V1, SymmetryNet V2, and DeepCube models (left to right) decomposed into individual layers.	37
3.8 Explanations of cube states of distance $d = 1$ produced by GNNEExplainer. . . . .	38
3.9 Explanations of cube states of distance $d = 3$ produced by GNNEExplainer. . . . .	39
3.10 Comparison of cube states: a solved one on the left side, and one resulting by clock-wise 90 deg rotation of the right face. . . . .	39
3.11 Two explanations of a cube state with $d = 1$ : a better explanation on the left side, highlighting mostly colors of facelets twisted by the move, a worse explanation on the right side, mixing both twisted and untwisted facelets. . . . .	39
3.12 An illustrative example of the second message passing of distance-vectors, which are weighted by the distances before aggregation.	41
B.1 Models’ learning convergences when generalizing from 90% of the exhaustive dataset. . . . .	55



B.2 Models' learning convergences when generalizing from 50% of the exhaustive dataset.....	55
B.3 Models' learning convergences when generalizing from 10% of the exhaustive dataset.....	56
B.4 Models' learning convergences on the $10^3$ -sized subset of the sampled dataset. ....	56
B.5 Models' learning convergences on the full $10^5$ sampled dataset. ....	56

## Tables

2.1 The Rubik's cube state space decomposed by the goal distances .	15
2.2 Alternative equivalence classes of the Rubik's cube and their properties. ....	21
3.1 State space statistics of the 3 datasets, with (expressiveness) error and compression of the proposed ( $WL^2$ ) model design.....	29
3.2 Expressiveness errors of the different pattern invariants on the three (distance-decomposed) datasets.....	31
3.3 Compression rates of the pattern invariants on the datasets .....	32
3.4 Comparison of the models' MAE performances across different train-test ratios of the exhaustive dataset .....	36
3.5 Comparison of the models' MAE performances while increasing the (subset) size of the sampled dataset	36
3.6 Comparison of the models' performances in problem solving ..	36





# Chapter 1

## Introduction

Incorporating suitable inductive biases into machine learning models can generally improve their training efficiency, robustness, and generalization to unseen data. In deep learning, this has typically been approached by exploiting certain domain “symmetries”, i.e., transformations that retain important input properties. The most popular example of this approach has been Convolutional Neural Networks (CNNs, [62]), exploiting translation equivariance in regular grid domains, such as images. Recently, the term Geometric Deep Learning [13] has been coined for the overall paradigm, with a number of new architectures proposed to exploit the symmetries beyond the regular Euclidean domains, such as the permutation invariance naturally present in data structured in the form of sets [119] and graphs [113]. Nevertheless, explorations into less obvious symmetries beyond the standard application domains remain scarce.

One reason for that is the increasing emphasis on scaling and deepening well-known types of neural networks, such as feed-forward networks and transformers, instead of designing justified, efficient architectures. But, with the expansion of deep learning applications to critical domains, including many planning tasks, where humans are ultimately responsible for the decisions made, it is crucial for the model to be explainable [5, 85, 84] and enlarging neural models only worsens the situation. The black-box nature of deep networks prevents us from fully understanding the networks’ inner computation, and we have to resort to approximate explanations of networks’ predictions mainly through post-hoc methods [61, 71, 106]: generating visual/textual explanations, studying the impact of local changes, generating examples, simplifying the model, or assigning importance scores to inputs [5]. Lately, we have also seen a surge in the popularity of the novel field of mechanistic interpretability [21, 75], which uses some of the mentioned post-hoc methods to study neural circuits in transformer-based architectures. All the above-listed methods depend heavily on data selection, as they analyze the dependency between model input and prediction. The transparency and inherent interpretability of the model, which would be independent of the input data selection, is usually left to simple machine learning models, such as linear regression, decision trees, k-nearest neighbors, etc.

Inspired by the simple models, we aspire to understand models’ decisions by design without the need for external algorithms, concentrating on modeling and reasoning about the neural architectures and representations instead of tuning post-hoc explainers.

## 1.1 Problem Statement

This work addresses the interpretability problem with an alternative approach, achieving more transparent deep neural models via geometric deep learning (GDL) principles. It provides a blueprint for extending the GDL principles into planning domains by taking advantage of their graph-induced symmetry structures, improving the learning efficiency of the architectures. In order to implement geometric deep learning architectures, the thesis identifies the most promising categories of planning symmetries and the domains where they are applicable and puts an extra emphasis on the *Rubik's cube domain*, which is well-known for the complexity of its intriguing symmetries rooted in group theory. Furthermore, the choice of Rubik's cube conveniently connects two topics from Bronstein's GDL taxonomy of 5Gs (Grids, Groups, Graphs, Geodesics, and Gauges; [13]), groups through the underlying symmetry and graphs through the data representation.

We incorporate the group conjugacy-based symmetry between Rubik's cube states into the neural network and systematically train and evaluate the resulting architecture, both w.r.t. classical machine learning and planning metrics. We argue that such models and their decisions are more interpretable due to being constrained to produce representations adhering to human-understandable symmetries and equivalences. This type of interpretability is useful since it does not rely on (possibly biased) data selection.

## 1.2 Related Work

The work is naturally related to previous works on using deep learning to address planning problems, either within a reinforcement learning framework or in the form of learning a heuristic for search algorithms. We also review previous literature on deep learning interpretability techniques, specifically those concerned with automated planning and architectures implementing equi-/invariances, e.g., graph neural networks. We continue with the topic of symmetries in deep learning, reviewing unifying frameworks and state-of-the-art approaches. Finally, we turn to related work on the selected planning problem, the Rubik's cube (for further explanation of the selection, see Sec. 2.2.3). We discuss previous attempts at solving Rubik's cube algorithmically, both with deep learning algorithms and classical solvers.

### 1.2.1 Related Work on Deep Learning in Planning

While automated planning has been studied since the 1950s [46], the prevalence of machine learning (in particular deep learning) techniques in the field is relatively new. Previously, the optimized solvers, search algorithms, and handcrafted heuristics clearly overshadowed the performance of learning algorithms. Today, however, we see a growing interest in using (deep) neural networks for solving planning tasks, be it by learning policies [50, 39], the transition model [111], heuristics for a search algorithm [33, 104, 116], or even learning end-to-end planners directly from visual information [6, 31, 7]. Most of the mentioned works approach machine learning

algorithms as *uninformed* universal approximators, usually resorting to feed-forward networks and classical vector-based ordered representations of the input data.

There is, however, another natural representation of planning problems – graphs, suggesting that graph neural networks (GNNs) might be a more suitable choice of a learning algorithm. Indeed, GNNs are currently being employed in predicting the importance of objects in a planning state [92], in learning policies within a reinforcement learning setup [79, 3], and, most commonly, in learning approximations of heuristic functions, evaluating states with an estimation of the cost to goal [17, 15].

However, such experiments are sensitive to the selected graph representation and GNN architecture, as some combinations are not expressive enough to distinguish between different planning states [114, 43]. Thus, multiple variants of graph representations were proposed, either for GNNs directly [88, 92, 99], or by utilizing precomputed GNN-like features, e.g., the Weisfeiler-Lehman color refinement [89, 18], for classical machine learning algorithms.

### ■ 1.2.2 Related Work on Neural Interpretability

While most of the AI community agrees on the need to explain deep neural networks, the taxonomy for explainable AI (XAI) is not agreed upon universally. Arrieta et al. [5] create a taxonomic tree distinguishing transparent (interpretable) models and post-hoc explainability on the first level and further dividing post-hoc explainability into model-agnostic and model-specific methods on the second level. Saleem et al. [85] further divide techniques into global and local explanations. Kenny et al. [51], on the other hand, dislike the ambiguous term “transparent model” and instead talk about the pre-hoc and post-hoc methods, with the latter being factual, counterfactual, or semi-factual. In many classification/decision tasks, there is a growing interest in counterfactual explanations in particular [105].

The distinction between AI explainability and interpretability also differs in literature. Glanois et al. [38] define interpretability as the “intrinsic property of a model”, while explainability is “a post-hoc operation”. On the other hand, Fan et al. [32] talk about the interpretability of neural networks and include even classical post-hoc methods in their taxonomy of interpretation methods.

For the purpose of this thesis, we will not distinguish between AI interpretability and explainability and will mention only methods usable for deep learning models. This decision rules out training fully transparent models. Based on Arrieta’s taxonomy [5], in the model-agnostic post-hoc category, there are simplification methods (such as LIME [78]), feature relevance explanations (e.g., SHAP [68]), and visual explanations. For the deep-learning model-specific techniques, there are, among others, model simplification techniques (such as tree extraction methods [26]) and feature relevance methods [91]; for convolutional neural networks specifically, there is a well-known visual explanation technique based on saliency maps [93].

With respect to our work, it is also interesting to look at Fan’s taxonomy [32], which adds an alternative bracket next to the post-hoc category, the ad-hoc modeling, including interpretable representations. While Fan et al. mention regularization techniques steering the learning algorithm to explainable representations, we believe

that inductive biases in the form of symmetries are an even more powerful way to achieve interpretable yet expressive models.

**Interpretability of Graph Neural Networks.** Due to the inherent structure of their input data, GNNs have their own specific set of explainability and interpretability methods that exploit human understanding of graph structure. These methods often try to find important nodes, node features, edges, or subgraphs that impact the model’s decisions. Yuan et al. [117] divide the techniques into instance- and model-level explanations. While the first category focuses on finding importance in the input data features, the second explains general model behavior independent of the input graphs. The most widely used instance-level explanations are perturbation-based methods, e.g., GNNExplainer [115], PGExplainer [69], or GraphMask [86]. Other instance-level methods include gradient-based models (e.g., Grad-CAM [9]) and decomposition methods (e.g., GNN-LRP [87]). One of the only known techniques for model-level explanations is XGNN [118].

**Interpretability of Neural Networks in Planning.** Many planning and scheduling domains have a long tradition of human planners employing brushed-up domain knowledge and would, therefore, benefit from human-AI collaboration. However, typical deep learning techniques used for automated planning lack interpretability [16]. The interpretability efforts are often centered around model-specific techniques, e.g., for GNNs (see Sec. 1.2.2). We discuss a few exceptions. Skirzyński et al. [95] suggest the AI-Interpret algorithm that transforms learned planning policies into interpretable descriptions, allowing for collaboration with human planners and improving their strategies. Consul et al. [22] proceed by “cognitively informed” reinforcement learning methods that lead to better human-like strategies. Finally, Lyu et al. [70] employ a “Trustworthy Decision-Making” framework combining symbolic planning with learning algorithms.

### ■ 1.2.3 Related Work on Symmetries in Deep Learning

Researchers have been experimenting with different inductive biases in the architectures of neural networks for a long time. We are interested in those connected to the notion of symmetry and invariance<sup>1</sup> [10, 110]. Symmetries have been shown to be an effective way to improve sample efficiency and generalization of the model [108], being the perfect compromise between uninformed deep learning and expert feature extraction. While there exist symmetry-related inductive biases of many different kinds, they are often implemented using similar techniques, mostly parameter sharing and aggregation [59].

One particularly known class of symmetries, geometric symmetries, including rotation, reflection, translation, and many others, was unified in the Geometric Deep Learning framework [13] following the previous work on Deep Symmetry Networks [37]. However, the basis for geometrically invariant architectures was laid out much earlier, e.g., with the (re-)invention of the translation equivariant CNNs [62], or with the proposal of rotation invariant neural networks [83]. What started as an attempt to capture basic axial symmetry in Euclidean spaces was

<sup>1</sup>or equivariance, which can be thought of as a weaker form of invariance

further extended to capture a much wider range of symmetries [12], e.g., those related to groups (group equivariance [20], group-action invariance [53], etc.), or to graphs [52, 112]. Concerning the latter, GNNs are the go-to architecture praised for its invariance to graph isomorphism [8] and node permutation [73]. Perhaps surprisingly, transformers are another architecture sharing similar traits with GNNs [49]. Thus, it has been possible to show different geometric equivariances of transformer-based architectures [102, 36].

However, symmetry-aware neural architectures do not have to be only invariant to intuitively recognizable symmetries; they can also be identified based on the methods used. Using the mentioned weight sharing and a “weight symmetry” [44], recurrent neural networks and their derivatives (GRU, LSTM, etc.) are one such example [101]. The DeepSets architecture [119] is another example, achieving permutation invariance via representation aggregation.

#### ■ 1.2.4 Related Work on Solving Rubik’s Cube

For the Rubik’s cube problem, we follow up mainly on the original DeepCube system [72], later extended for Nature [2], which proposed a new variant of (approximate) policy iteration in combination with A\* search. In the spirit of deep reinforcement learning, the value function, estimating the distance from the solved state, was represented by a neural network. The architecture in use was rather conventional feed-forward network with residual layers and batch normalization. Using the same neural architecture, some improvements in the search strategy have recently been published [103]. The work was also followed by Johnson [48, 47], who used a similar approach while translating the problem into a supervised setting. This was done by starting from the solved state and considering the reverse stepwise evolution through the state space as a labeled example generator for the value function approximation, similarly to McAleer et al. [72] (and Sec. 3.1.2). The model in use was again a standard feed-forward network equipped with dropout layers. A similar work [14] also explored classic feed-forward networks as the heuristic value learner, together with several handcrafted features.

Apart from deep learning solutions, there are also well-known algorithms of varying difficulty for solving the Rubik’s cube, which gave rise to algorithmic solvers [57], highly optimized through years of development, often making use of large pattern databases [27]. These explicit approaches often use advanced computational group theory to analyze the problem space, leading to several interesting insights, including the insight of geometrically interpretable symmetry-equivalent states.

However, little attention has been devoted to the resulting insights in the deep learning-based approaches. Works studying some form of invariances in the problem include Lim et al. [65] who, however, examine the symmetries in moves instead of the states while attempting to learn the game rules, or Ferraro et al. [34] who introduce a generative model to analyze the learned symmetries merely after training.

Only a handful of works then use symmetries to improve the actual deep learning approach to the problem. The works of Corli et al. [24, 23] described an interesting theoretical approach via quantum formalism, inspired by the Ising model, introducing a Hamiltonian approach to deep reinforcement learning. Nevertheless, the work is rather theoretical, and the introduced neural model was again a standard (two-layer)

feed-forward network. A more related work of Konen [58] then explored symmetries generated (merely) by explicit cube rotation in an update of the value function, represented by a (symmetry-unaware) N-tuple neural network [67]. In particular, a subset of symmetry-equivalent (rotated) states was explicitly expanded here to jointly update the value function, yielding a 10 – 20% improvement in solving the cube.

Contrary to these existing works that use symmetries or other group-theoretic insights in the pre/post-processing stages, this thesis introduces an approach of building the respective invariants *directly* into the neural architecture, in the spirit of geometric deep learning [13]. Moreover, in contrast to the rather descriptive character of [13], it also provides a concrete and instructive blueprint for applying the principles in new domains with complex symmetry structures.



## Chapter 2

### Theoretical Background

We start by introducing AI planning and multiple classical planning problems. Then, we analyze some of their common symmetric features, before delving into one specific problem – solving the Rubik’s cube.

#### 2.1 AI Planning

Since the beginning of the field of artificial intelligence, researchers have been interested in solving games and other combinatorial problems. One approach gave birth to the field of AI planning or, more formally, automated planning and scheduling. Automated planning is a subfield of symbolic artificial intelligence that is interested in reasoning about and choosing actions in an environment by creating a strategy or policy, or using other forms of evaluation of the environment. However, many real-world environments are very complex and hard to model precisely. Researchers have, therefore, often resorted to simple game environments with well-defined rules that mimic at least parts of the real world.

Many application-oriented subfields emerged within AI planning, including planning for robotics and autonomous vehicles, task scheduling, or resource allocation. In this thesis, we will not go into application-specific techniques. Instead, we will refer to general concepts related to *classical planning*. A classical planning task is defined by a transition system.

**Definition 2.1.** *Transition system* is a tuple  $\mathcal{T} = (S, A, T, I, G)$  where  $S$  is a finite set of states,  $A$  is a finite set of actions,  $T : S \times A \times S$  is a transition relation,  $i \in S$  is the initial state, and  $G \subseteq S$  is a set of goal states.

Alternatively, the transitions can be described by a *transition function*  $\gamma : S \times A \rightarrow S$ , *s.t.*  $\gamma(s, a) = s' \iff (s, a, s') \in T$ . The transition system defines a search graph with the states represented by nodes and transitions represented by edges. Solving the planning problem then equates to finding a path from the state  $i$  to one of the goal states  $g \in G$ , which can be done by search algorithms, such as A\* [40], greedy best-first search (GBFS), Dijkstra [30], etc.

It is usually infeasible to fully enumerate the whole state space/search graph. Instead, the properties of states and consequences of actions can be tracked, allowing for a compact problem representation. Conveniently, the properties can be formalized in different formal systems based on, e.g., propositional or first-order logic.

### 2.1.1 Planning Problems

While many real-world problems could be formulated as planning tasks, there are some well-established problems in the planning community, especially due to the flagship International Planning Competition (IPC)<sup>1</sup>. This work further discusses selected problems previously featured in IPC.

**Blocksworld, Transport, Freecell, Sokoban.** We selected the following classical planning problems from the IPC domain repository<sup>2</sup> to showcase the symmetry analysis: Blocksworld, Transport, Freecell, and Sokoban. *Blocksworld* is a well-known domain containing  $N$  blocks that can sit on top of each other (creating “towers”). The goal is to rearrange the blocks to achieve the required configuration. It is one of the simplest domains, yet it can contain symmetries in the form of position invariance. *Transport* is a planning domain in which the goal is to transport packages to prescribed locations by using vehicles with a certain capacity. We are interested in transport because it involves multiple interchangeable objects which can be the base of a symmetry-invariant architecture. *Freecell* is a card game belonging to the popular family of solitaire games. Its goal is to lay out cards of the same suit into one of the goal boxes (foundations) in ascending rank, following rules that specify which card can sit on top of another card in the “cascades”. The user can use multiple “free cells”, which serve as a temporary storage. On top of other symmetries, Freecell inherently contains color- and suit-invariance symmetries. *Sokoban* is a classical grid-world game with a player pushing stones (or boxes) and the goal of getting them to specified positions. There are different geometric (directional) symmetries, such as certain rotations and reflections.

**Rubik’s Cube.** The Rubik’s cube is a popular game comprising a 3D cube with 6 distinctively colored 2D faces, each divided into a  $3 \times 3$  grid of colored squares. Facilitated by an internal pivot mechanism, each face can be independently rotated, thereby mixing up the color grids’ arrangements. The objective is to rearrange the cube so that each face contains squares of a uniform color. It was featured at IPC in 2023<sup>3</sup>. Despite its apparent simplicity, the problem exhibits a complex internal structure deeply rooted in group theory. Expanding upon this foundation, algorithms of varying difficulty have been established for manual solving by human players, along with highly optimized algorithmic solvers [57], often making use of large pattern databases [27].

## 2.2 Planning Problems Analysis

To identify symmetries and invariances that might enhance our neural architecture, we analyzed the properties of graph representations of problems from Sec. 2.1.1. Even though the representations do not necessarily rely on formal planning representation, we start by defining some key terms.

<sup>1</sup><https://www.icaps-conference.org/competitions/>

<sup>2</sup><https://github.com/AI-Planning/classical-domains>

<sup>3</sup><https://github.com/ipc2023-classical/domain-rubiks-cube>

### 2.2.1 Planning Representations

Some of the most common planning representations include Finite Domain Representation (FDR; e.g., SAS), which operates on a grounded level, and STRIPS, which operates on a lifted level. Finally, there is the Planning Domain Definition Language (PDDL), a generalization of STRIPS. Both PDDL and STRIPS are a common starting point for the creation of relational or graph representation structures on the lifted level.<sup>4</sup> The definitions below are based on Chen et al. [15].

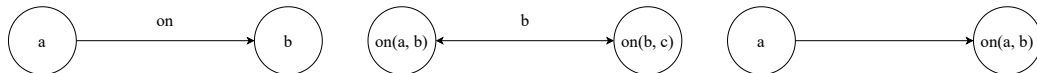
**Definition 2.2.** A *STRIPS* planning task is defined by a tuple  $\mathcal{T} = (P, A, s_0, G)$ , where  $P$  is a set of facts,  $A$  is a set of actions,  $s_0 \in P$  is an initial state, and  $G \subseteq P$  are the goal conditions.

**Definition 2.3.** An *action*  $a \in A$  in the STRIPS planning task is a tuple  $a = (pre(a), add(a), del(a))$  of preconditions, positive effects and negative (delete) effects, with  $pre(a), add(a), del(a) \subseteq P$  and  $add(a) \cap del(a) = \emptyset$ .

PDDL, compared to STRIPS representation, also allows for negative literals and (unassigned) variables in preconditions of actions and goals. Both STRIPS and PDDL can operate with objects, which means that the facts are predicates over the objects. Further, PDDL allows for (hierarchical) types of objects.

#### Graph Representation

In this analysis, we started with a PDDL-like representation. There are two fundamentally different levels on which we can create graph representations for a planning task: (i) the search-space level and (ii) the state level. In (i), the nodes of the graph are states, and the edges correspond to atoms, while in (ii), the nodes and edges correspond to properties of a single state. We continue with approach (ii), as it fits the geometric deep learning framework of invariances and equivariances better.



**Figure 2.1:** The minimal examples of the object (left), atom (middle), and object-atom (right) representations for the Blocksworld domain with blocks  $a$ ,  $b$ , and  $c$ .

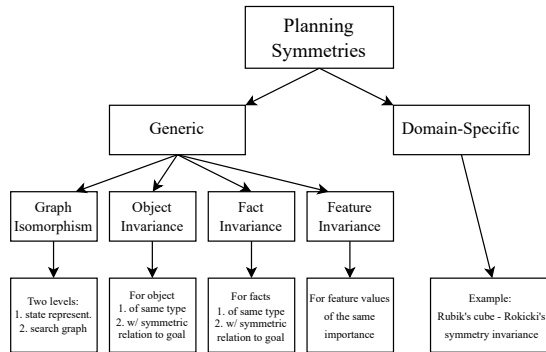
We discuss three state-level representations for which the minimal examples are depicted in Fig. 2.1. The most straightforward approach, the *object representation*, is to treat all objects (and nullary predicates/static facts) as nodes in the graph and the facts/predicates as edges between objects. For example, in the left part of Fig. 2.1, the nodes represent blocks  $a$  and  $b$ , and the edge represents the binary predicate  $on(a, b)$  in Blocksworld. Further, the object types and unary predicates (facts involving a single object) can be represented either as additional nodes or as node features. Other representations include the *atom representation*, which treats atoms – facts grounded to concrete objects – as nodes and relates atoms that share an object by an edge (in the middle part of Fig. 2.1, the edge represents that facts

<sup>4</sup>On the lifted level, it is usually easier to reason about symmetries since they are usually connected to object types, not concrete object instances.

$on(a, b)$  and  $on(b, c)$  share block  $b$ ), or the *object-atom representation*, which has nodes for both objects and atoms and relates an object with an atom by an edge if the atom includes the object (in the right part of Fig. 2.1, the edge signals that fact  $on(a, b)$  includes block  $a$ ), creating a bipartite graph [43].

## 2.2.2 Planning Symmetries

We have identified multiple categories of symmetries that are universally applicable to representations of planning states based on PDDL (generic symmetries), as well as a category of domain-specific symmetries. For each category, we give examples in the selected domains, as well as a suggestion of possible ways to incorporate the respective symmetries into the design of a graph neural network.



**Figure 2.2:** Our taxonomy of symmetries found in planning problems.

**Permutation and Graph Isomorphism Invariance.** The permutation and graph isomorphism invariance is a property that holds for most planning problems. For an example of the permutation invariance, in Blocksworld, the order of block towers is arbitrary; in Freecell, it is not necessary to distinguish the order of cascades, foundations, or free cells. Regarding graph isomorphism, the presented order of the facts in the domain/state specification and, thus, the resulting form of the graph does not matter in either Freecell, Sokoban, or Rubik’s cube. Note that the permutation and graph isomorphism invariance are not necessarily related just to *graph state representations*. These properties should also hold for the (search) *graph representation of the state space*. As for the incorporation of the mentioned invariances into a GNN, it is known that a GNN can inherently compress isomorphic and some permuted graphs into the same (invariant) representation [19]. Thus, these symmetries are automatically accounted for by most standard GNN architecture, and there is no need to manipulate the model further to achieve the invariance.

**Object and Fact Invariance.** Secondly, we want to enforce the equivalence between objects with the same “behavior”. The simplest reason for an equivalent behavior is that the objects are of the same type, e.g., stone objects in Sokoban or vehicles in Transport, and thus, two states that differ only by a permutation of  $N$  such equivalent

objects are essentially the same. A generalization of the equivalent behavior can be observed between objects that are of arbitrary type but play the same role in the search graph, i.e., they appear in states that are on a *symmetric path(s) to the goal*. However, this, in practice, requires the enumeration of a substantial part of the state space just for the detection of the equivalence. Also, this property of objects might not be transferable between instances of the same planning problems. Similarly to object invariance, we can find facts (and corresponding edges) of the same type or of the same (symmetric) relationship w.r.t. the path(s) to the goal. This symmetry only applies to graph representations where we differentiate facts by edge features. A straightforward solution to the invariance of the same-type objects/facts is to enforce the same features (feature vectors) for such objects/facts. This is, however, already part of the standard graph object representation (Sec. 2.2.1).

**Feature Invariance.** Next, there are possible invariances tied to node features, concretely permutation of their values. The idea is that the possible feature values need to be treated equally by the representation, but the corresponding nodes still need to be distinguished. For example, in Freecell, we would like for the suits of cards to be equal so that if we create a second state  $S'$  by permuting the suits of the cards in state  $S$ , i.e.  $S' = S_{\pi_{suits}}$ , the states are equivalent. A similar idea applies to Rubik's cube with the color of the stickers where, intuitively, we care more about how scattered the colors on the cube are rather than which permutation of colors is applied to the stickers. As for symmetry-incorporation ideas, we can start with a similar solution for object invariance. But, if we naively enforce nodes with the same representation, we effectively remove all information about the node feature. There are two main fixes: (i) creating an additional aggregation node for each value of the feature and connecting it by edges to the original nodes with that feature value, or (ii) adding an extra type of edge for each feature value and fully connecting the subgraph formed by the respective nodes. In both ways, we distinguish the feature values using the structure of the graph.

**Domain-Specific Invariances.** Finally, we can come up with domain-specific invariances that cannot be defined generally on PDDL-like graph representations. An example of that is Rokicki's symmetry-equivalence [80], which is further discussed in Sec. 2.3.3.

### ■ 2.2.3 Planning Domain Selection

Now that we have analyzed the taxonomy of planning symmetries, we discuss choosing a specific planning domain, considering both the symmetry analysis and the principles of geometric deep learning. From the taxonomy analysis, we have learned that some of the mentioned symmetries (e.g., graph isomorphism) proved to be trivially included in any reasonable graph representation, while others are already incorporated in standard lifted planning representations (e.g., object and fact invariance). This leaves feature invariance and domain-specific invariances as the most promising paths.

We have identified multiple candidate domains in these two invariance classes. However, also considering the 5Gs of Geometric Deep Learning (Sec. 1.1), the Rubik's

cube problem (which contains both feature invariances as well as rich domain-specific symmetries) stands out with its highly geometric structure and a connection to group theory. In the next section, we conclude the planning problems' analysis by formally defining Rubik's cube as a planning problem.

### 2.2.4 Rubik's Cube Planning Problem

Solving the Rubik's cube is defined as a planning problem by the transition system or using the STRIPS representation:

**Definition 2.4.** *Rubik's cube transition system* is a tuple  $\mathcal{T}_{RC} = (\mathcal{S}, R, T, S_{init}, S^*)$  where  $\mathcal{S}$  is the set of cube states,  $R$  is the set of rotations of cube's faces,  $T : \mathcal{S} \times R \times \mathcal{S}$  is a transition relation,  $S_{init} \in \mathcal{S}$  is the initial cube state, and  $S^* \subseteq \mathcal{S}$  is the solved/goal cube state.

**Definition 2.5.** *Rubik's cube STRIPS planning task* is defined by a tuple  $\mathcal{T} = (C, R, S_{init}, S^*)$ , where  $C$  is a set of color assignments to positions on the cube,  $R$  is the set of actions,  $S_{init} \in C$  is the initial cube state, and  $S^* \subseteq C$  is the solved cube state. An action  $r \in R$  is a tuple  $r = (pre(r), add(r), del(r))$  with  $pre(r)$  containing assignments of colors to all cube positions, and  $add(r)$  and  $del(r)$  describing color changes (which must have a one-to-one correspondence).

Other planning representations have been compiled by Muppasani et al. [74]. For the rest of this work, we zoom in on the cube problem, analyze it in detail, and provide experimental results. We start by discussing the cube-specific definitions.

## 2.3 Rubik's Cube Analysis

We adopt a largely standard domain-specific representation of the Rubik's cube problem [94], as exploited in a number of previous works (Sec. 1.2) for compatibility.

### 2.3.1 Problem Structure

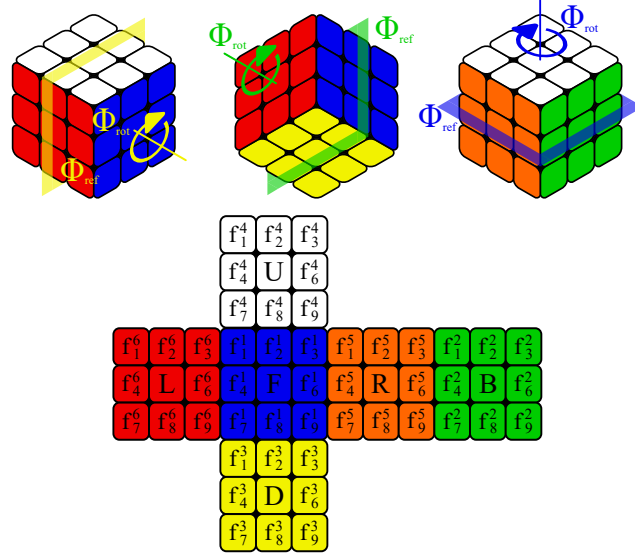
The standard cube consists of 27 smaller cubes called *cubies*. The surfaces of cubies visible to the outside, colored in one of  $\mathcal{C} = (\underline{blue}, \underline{green}, \underline{yellow}, \underline{white}, \underline{orange}, \underline{red})$  colors, are referred to as *facelets*  $f_i \in \mathcal{F}$ . There are 6 cubies with only one (central) facelet, 12 cubies with two (edge) facelets, 8 cubies with three (corner) facelets, and 1 face-less cubie hidden in the center of the cube. Each of the cube's *faces*  $F_1, \dots, F_6$  is then formed by nine facelets  $f_1^{F_i}, \dots, f_9^{F_i} \in \mathcal{F}$  arranged on a planar  $3 \times 3$  grid.

### Cube States

With the cube structure introduced, we can define the configuration *state* of the Rubik's cube problem.

**Definition 2.6.** A cube state  $S : \mathcal{F} \rightarrow \mathcal{C}$  is defined by assigning facelets with colors, given some reference ordering.

The facelet ordering generally depends on the cube's *orientation*  $o : \{F_i\} \longleftrightarrow \mathcal{C}$ , determining the (color) ordering of the faces. Note that, while solving the puzzle,



**Figure 2.3:** A solved cube state in the standard color ordering, with denoted face  $F_j$  views ( $F, B, D, U, R, L$ ) composed of individual facelets  $f_i^{F_j}$  in a flattened “cube cross” view (below), and an outline of three rotation and reflection axes in a 3D view (above).

the color of the *central* facelet  $f_5^{F_i}$  on each face  $F_i$  never changes.<sup>5</sup> This allows to define a *fixed ordering* of the faces based on the color of their respective central facelets, e.g., as

$$F_1 \mapsto \underline{b}, F_2 \mapsto \underline{g}, F_3 \mapsto \underline{y}, F_4 \mapsto \underline{w}, F_5 \mapsto \underline{o}, F_6 \mapsto \underline{r}$$

corresponding to a *canonical orientation* (rotation) of the cube. Such canonical orientation of the *goal/solved state*  $S^*$ , where all the facelets  $f_i$  within each face  $F_j$  are of the same color  $C_j$  as  $\forall i, j : f_i^{F_j} \mapsto C_j$ , is shown in Fig. 2.3.

## State Representations

Consequently, a state can be represented simply as an ordered list of facelet colors  $S : (f_1^{F_1}, \dots, f_9^{F_1}, \dots, f_9^{F_6}) \mapsto (C_i^1, \dots, C_j^{54})$ . If we represent the colors with categorical (one-hot) vectors, the whole cube state can then be seen as a sparse matrix defined by an indicator value

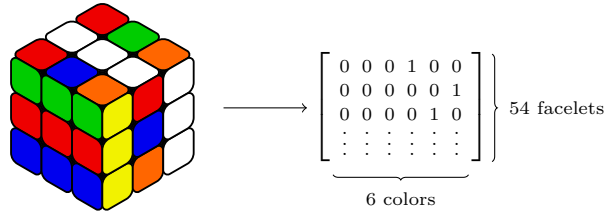
$$X_{i,j} = \begin{cases} 1 & \text{iff the } i\text{-th facelet is of } j\text{-th color,} \\ 0 & \text{otherwise,} \end{cases} \quad (2.1)$$

as shown in Figure 2.4. Note that this representation, as used in DeepCube [2], is oblivious to any symmetries.<sup>6</sup>

<sup>5</sup>Note also that there are no two cubies with the same combination of facelet colors on their surfaces and, furthermore, the corner facelets can never become edge facelets and vice versa.

<sup>6</sup>Given the insight about the immutable color of the central facelets, we could reduce the representation to 48 movable facelets. Even further reduction could be achieved as the cube does not have 48 degrees of freedom due to the fixed color ordering, and inherent invariants such as permutation parity of cubie orientations [100]. Nevertheless, to align with [2], we choose to implement the symmetry-invariance on top of this simple representation.





**Figure 2.4:** The (standard) one-hot encoding of the cube state representation

## ■ Move Metrics

A *move* (turn) is a change from one state to another  $g : S_1 \mapsto S_2$  that happens by rotating 9 cubies, forming one of the cube’s faces  $F_i$ , along its central (perpendicular) axis (Fig. 2.3). There are two types of “move metrics” known as (i) *face-turn* and (ii) *quarter-turn*. While face-turns allow any of the three sensible face rotations ( $90^\circ$ ,  $180^\circ$ , and  $270^\circ$ ) within one move, the quarter-turns only allow rotations of  $90^\circ$  clockwise and  $90^\circ$  anti-clockwise. In the quarter-turn metric, a move can be represented by specifying the face  $F_i$  and the direction of the rotation – either clockwise, denoted simply as  $F_i$ , or counterclockwise, denoted as  $F_i'$ . Given a canonical orientation of the cube, these faces can also be referenced relative to the cube orientation as *front, back, down, up, right, left*, denoted as  $F, B, D, U, R, L$ , or  $F', B', D', U', R', L'$ , respectively (Fig. 2.3). In the face-turn metric, additional moves of  $F2, B2, D2, U2, R2, L2$  denote the turn of the respective face by  $180^\circ$  and do not need their counterclockwise counterparts, as they would represent the same moves. Depending on the metric, there can thus be 12 or 18 different moves. Importantly, the two metrics result in two optimization problems that need to be solved in separation, since the optimal solution in the face-turn metric is not trivially translatable to the optimal solution in the quarter-turn metric and vice versa (see Sec. 2.3.1 below). However, our approach to symmetry-invariance in the domain is independent of the metric.

## ■ State Distribution

With the notion of state distance and move metrics, two state distributions – one for the face-turn ( $d_{ft}$ ) and one for the quarter-turn ( $d_{qt}$ ) metric, arise by partitioning the state space into sets of states with an equal optimal goal distance  $d(S, S^*)$ . Precise counts are known for  $d_{ft} \leq 15$  and  $d_{qt} \leq 18$ , respectively, while for the remaining distances, approximations or lower bounds are known, as reported in Tab. 2.1, based on the sources of [81, 82].<sup>7</sup>

## ■ State Generators

A *generator*  $G \in \mathcal{G}$  is a sequence of moves  $g$ , represented as a space-delimited sequence of the corresponding symbols (e.g.,  $FU'FFD'B$ ). The number of moves in a generator is its *length*. Each cube state  $S$  can then be represented by a generator

<sup>7</sup><https://cube20.org/> and <https://cube20.org/qtm/>



$d(S, S^*)$	$ S $ in $d_{ft}$	$ S $ in $d_{gt}$
0	1	1
1	18	12
2	243	114
3	3,240	1,068
4	43,239	10,011
5	574,908	93,840
6	7,618,438	878,880
7	100,803,036	8,221,632
8	1,332,343,288	76,843,595
9	17,596,479,795	717,789,576
10	232,248,063,316	6,701,836,858
11	3,063,288,809,012	62,549,615,248
12	40,374,425,656,248	583,570,100,997
13	531,653,418,284,628	5,442,351,625,028
14	6,989,320,578,825,358	50,729,620,202,582
15	91,365,146,187,124,313	472,495,678,811,004
16	$\approx 1,100,000,000,000,000,000$	4,393,570,406,220,123
17	$\approx 12,000,000,000,000,000,000$	40,648,181,519,827,392
18	$\approx 29,000,000,000,000,000,000$	368,071,526,203,620,348
19	$\approx 1,500,000,000,000,000,000$	$\approx 3,000,000,000,000,000,000$
20	$\approx 490,000,000$	$\approx 14,000,000,000,000,000,000$
21	0	$\approx 19,000,000,000,000,000,000$
22	0	$\approx 7,000,000,000,000,000,000$
23	0	$\approx 24,000,000,000,000,000,000$
24	0	$\approx 150,000$
25	0	$\geq 36$
26	0	$\geq 3$

**Table 2.1:** The Rubik's cube state space decomposed by the goal distances

$G$ , corresponding to a sequence of moves applied to the solved state  $S^*$ . The length of the shortest generator<sup>8</sup> then corresponds to the *distance* of the state  $d(S, S^*)$  from the goal.

### 2.3.2 Group Theory

With the definition of state generators, we may now view the Rubik's cube state space from a group-theoretical perspective.

#### Rubik's Cube Group

The set of all possible generators  $\mathcal{G}$ , equipped with the (non-commutative) binary operation of composition, forms the *Rubik's group*:

$$\mathcal{RG} = (\mathcal{G}, \circ), \quad \circ : \mathcal{G} \times \mathcal{G} \rightarrow \mathcal{G}$$

with the identity being a “no-action” generator, and the inverse generator  $G^{-1}$  constructed by reversing the order of moves and switching their direction (counter-

<sup>8</sup>An infinite number of generators leading to a single state exists because, e.g., inserting pairs of mutually inverse moves does not influence the state generated by the sequence.

/clockwise). The group has infinitely many finite *group generators*, the simplest one being the set of the 12 elementary moves.<sup>9</sup> It has been found that in the quarter-turn metric, the diameter of the Rubik’s cube group is 26, i.e., any valid Rubik’s cube can be solved with up to 26 90° face rotations [81], and 20 moves for the face-turn metric, respectively [82]. Together with a given ordering of the face colors, such as the canonical one displayed in Fig. 2.3, the Rubik’s group forms a *Rubik’s universum*.

**Definition 2.7.** A Rubik’s universum is the set of all cube states  $\mathcal{S}$  closed under the valid generators  $G$ , given a fixed color ordering  $\mathcal{C}$  of the faces in the solved state.

Since the colors of the central facelets remain static, there is no valid (non-destructive) way to transition between distinct universa, each representing a unique spatial relationship between the colors. A single *standard universum* is then usually obeyed by cube manufacturers (Fig. 2.3).<sup>10</sup> Assuming the standard universum and substituting generators with their corresponding cube states, the formalization of Rubik’s cube is often simplified into  $\mathcal{G} = \mathcal{S}$  (as, e.g., in [81]).

## ■ Cayley Graphs

For the purpose of searching through the state space, the cube universum is then commonly represented by a *Cayley graph*, in which nodes represent the cube states and directed edges represent the valid moves. Particularly, for each node  $S_i \in \mathcal{S}$  in the Cayley graph, an outgoing edge  $S_i \xrightarrow{g} S_j$  is assumed for each move  $g$  from the selected generating set of the group. Thus, with the elementary choice of moves in the quarter-turn metric, we have a regular graph  $\forall S \in \mathcal{S} : \delta^+(S) = 12$ . The problem of solving the Rubik’s cube can then be formalized as finding the (shortest) path between  $S$  and  $S^*$  in the Cayley graph.

### ■ 2.3.3 Domain symmetries

The Rubik’s cube involves a number of intriguing *symmetries*.

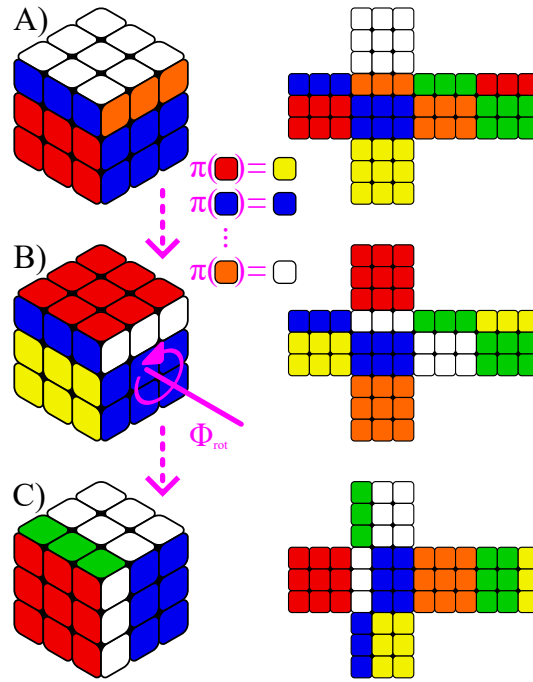
**Definition 2.8.** Symmetry of an object  $S$  is a transformation  $\phi : \mathcal{S} \rightarrow \mathcal{S}$  that preserves some target property  $d$  of  $S \in \mathcal{S}$ .

For instance, it is immediately obvious that arbitrarily rotating the whole cube does not change any property of its corresponding state (as explored in some previous works in Sec. 1.2). However, note that this trivial symmetry is already captured by establishing the canonical orientation (Sec. 2.3.1), mapping all such rotations onto a single representation (Fig. 2.3). The unique symmetries of this domain, which we target in this work, are more intriguing in that they combine such, normally studied, geometric transformations with discrete coloring permutations. Starting with the latter, following the work of Rokicki [80], we first define the notion of *recoloring-equivalence*:

**Definition 2.9.** We call two cube states  $S_1, S_2$  *recoloring-equivalent*  $S_1 \stackrel{\pi}{\equiv} S_2$  iff there is an injective mapping  $\pi : \mathcal{C} \rightarrow \mathcal{C}$  between the colors  $\mathcal{C}$ , such that  $S_1 \circ \pi = S_2$ .

<sup>9</sup>There are even smaller group generators, e.g.,  $\{F, B, L, R, U\}$ .

<sup>10</sup>There are actually two common universa with different color neighborhoods, the Japanese and the Western color schemes, but this choice has no effect as long as it is consistent.



**Figure 2.5:** An original (A), a recoloring-equivalent (B), and a symmetry-equivalent (C) cube states, originating through a color permutation  $\pi$  and a  $90^\circ$  counter-clockwise geometric rotation  $\phi$  around the front face.

That is, when states  $S_i : \mathcal{F} \rightarrow \mathcal{C}$ , possibly from different universa, are being equivalent up to a certain permutation  $\pi$  of the colors  $\mathcal{C}$ , as exemplified in Fig. 2.5 (A-B). With that, again following [80], we may finally define the domain *symmetries* we exploit in this work.

**Definition 2.10.** We call two cube states  $S_1, S_2$  *symmetry-equivalent*  $S_1 \stackrel{\phi}{\underset{\pi}{\equiv}} S_2$  iff there is a geometric transformation  $\phi$  and a recoloring  $\pi$ , such that  $\phi \circ S_1 \circ \pi = S_2$ .

That is,  $S_1$  and  $S_2$  are symmetry-equivalent if after applying  $\phi$  to  $S_1$  it becomes recoloring-equivalent to  $S_2$ , as exemplified in Fig. 2.5 (A-C). The geometric transformations  $\phi$  in our scope are then *rotation* and *reflection* (Fig. 2.3). See Sec. 2.4.3 for a more detailed explanation of the (alternative) symmetries present in the Rubik's cube problem.

### ■ Symmetric Sets

Combinations of these two types of transformations then give rise to a *symmetry-transformation group* with 48 unique members (including identity), which we form through a minimal sufficient base of 3 rotations, with axes going perpendicularly through the centers of the  $F$ ,  $L$ , and  $U$  faces, and a single reflection that cuts the cube in half (out of 3 such reflections), e.g., along the  $F$  face (Fig. 2.3). When applied to the state space  $\mathcal{S}$ , these induce subsets of symmetry-equivalent states  $\{\mathcal{S}^j \mid \forall S_1, S_2 \in \mathcal{S}^j : S_1 \stackrel{\phi}{\underset{\pi}{\equiv}} S_2\}$ , consisting of up to 48 distinct elements, as also shown in [42]. We further confirm (Sec. 3.2) that an overwhelming majority have exactly

48 members, with only a few subsets being smaller due to self-symmetries in the states, i.e., cases where more than one of the 48 symmetries acts as identity.<sup>11</sup> These then shrink the corresponding sets of symmetry-equivalent states into 24, 16, 12, 8, or 6 members, respectively, depending on the number of the state’s self-symmetries.

## Generalized Rubik’s Cube Group

The 48 transformations forming the symmetry-transformation group can then be used to generalize the definition of the Rubik’s cube group (Sec. 2.3.2) by introducing the set of *generalized generators*  $\mathcal{G}_+$ , formed through combinations of the cube generators and the respective symmetry transformations.

$$\mathcal{RG}_+ = (\mathcal{G}_+, \circ), \quad \circ : \mathcal{G}_+ \times \mathcal{G}_+ \rightarrow \mathcal{G}_+$$

The *generalized Rubik’s group* is then formed analogically, with the (non-commutative) binary operation of composition, the “no-action” generator serving as identity, and the inverse generator  $G^{-1}$  constructed by:

1. reversing the order of the moves and the transformations;
2. flipping the direction (counter-/clockwise) of the moves and inverting the symmetry transformations.

The group generator is then the union of the set of the elementary moves, generating the Rubik’s cube group, together with the minimal sufficient basis of the symmetry-transformation group. The generalized group then describes all possible cube universa in all color orderings, with both the Rubik’s cube group and the symmetry-transformation group being its subgroups.

## 2.4 Symmetry-Aware Deep Learning

Each Rubik’s cube universum has an astronomically large configuration space  $|\mathcal{S}| \approx 4.3 \times 10^{19}$  with a single corresponding goal  $S^*$ . The complexity of this task, proven to be NP-complete [29], makes it necessary to resort to heuristic approaches, and the quality of the heuristic value function  $f : \mathcal{S} \rightarrow \mathbb{R}_0^+$ , estimating the distance  $d(S, S^*)$  of the current state  $S$  from the goal state  $S^*$  in the Cayley graph, is thus of crucial importance for both reinforcement learning and search-based techniques (Sec. 1.2), where the task of estimating the distance by a parameterized model  $f_\theta$  is cast as a regression problem:

$$\min_{\theta \in \Theta} \sum_{S \in \mathcal{S}} |f_\theta(S) - d(S, S^*)|$$

Praising their universal approximation capability [28], classic feed-forward neural networks have previously been used in place of  $f_\theta$  (Sec. 1.2), ignoring the structural properties of the domain. As discussed, we aim to improve upon that front by exploiting the problem symmetries (Sec. 2.3.3) in the form of *invariance* of the model  $f_\theta$ .

<sup>11</sup>These cube states with highly regular color patterns have been referred to as “symmetric states” in [80]. However, these are not to be confused with the symmetries exploited in this work.

**Definition 2.11.** Function  $f$  is an invariant of  $S \in \mathcal{S}$  with respect to a transformation  $\phi : \mathcal{S} \rightarrow \mathcal{S}$  iff  $f(\phi(S)) = f(S)$ .

The crucial property of the symmetries introduced in Sec. 2.3.3 is that they naturally preserve the inner state configuration and thus the distance  $d(S, S^*)$  from the goal state [80], making them suitable to enhance  $f_\theta : \mathcal{S} \rightarrow \mathbb{R}_0^+$ . Following the analysis from Sec. 2.3.2, we see that, in the quarter-turn metric, each Rubik's universum  $\mathcal{S}$  can be partitioned into 26 subsets  $(\mathcal{S}_1, \dots, \mathcal{S}_{26})$  of *distance-equivalent* states<sup>12</sup> w.r.t.  $d(S, S^*)$ . Thus, theoretically, an ideal maximally-invariant model would induce exactly these 26 subsets as  $f_\theta^*(S) = i \iff d(S, S^*) = i$ . However, this is clearly impossible due to the computational complexity of the problem [29]. Instead, our aim is a more modest compression of the problem complexity by decomposing the  $(\mathcal{S}_1, \dots, \mathcal{S}_{26})$  further into their subsets induced by the symmetry-transformation group (Sec. 2.3.3), giving rise to distance-equivalent symmetry classes:

$$\{\mathcal{S}_i^j \mid \forall S \in \mathcal{S}_i^j : d(S, S^*) = i \wedge \forall S_1, S_2 \in \mathcal{S}_i^j : S_1 \stackrel{\phi}{\equiv} S_2\}$$

### 2.4.1 Equivalence Classes

To create these classes, we need to turn the declarative description of the cube symmetries from Def. 2.10 into a constructive procedure. Since naively examining all the recolorings  $\pi$  (Def. 2.9) would be computationally infeasible, we design an efficient procedure described in Alg. 1. Note that the 2<sup>nd</sup> step essentially performs a certain color permutation, hence the states are recoloring-equivalent (Def. 2.9) after the 3<sup>rd</sup> step, and the geometric transformation from step 4 then forms a symmetry-equivalent state, matching the Def. 2.10.<sup>13</sup> Finally, we further simplify the described transformation of  $\phi^{-1} \circ G \circ \phi \circ S^*$  into  $G' \circ S^*$  by  $\phi$ -transforming the generator  $G$  itself, instead of the cubes, resulting in Alg. 2. Indeed, for each of the 48 transformations  $\phi$ , a mapping  $\phi^G$  between the *moves* can be found to match the purpose, e.g., for the clockwise rotation along the vertical axis of the cube (Fig. 2.3), it is  $\phi_i^G = \{F \mapsto L, B \mapsto R, L \mapsto B, R \mapsto F, U \mapsto U, D \mapsto D\}$ .

---

#### Algorithm 1 Construction of Equivalence Classes $\mathcal{S}^j$

---

Given a cube state  $S$  and its generator  $G$ , its symmetry-equivalent states  $S'_{1..48} \in \mathcal{S}^j$  can be constructed as follows:

1. Start with the solved  $S^*$  in the canonical orientation.
  2. Apply one of the 48 geometric transformations  $\phi_i$  (Sec. 2.3.3), resulting in  $S_i^{*'}$ .
  3. Apply the state's  $S$  generator  $G$  to the state  $S_i^{*'}$ .
  4. Apply inverse  $\phi_i^{-1}$  of the transformation from step 2.
- 

<sup>12</sup>There is, of course, also the 27<sup>th</sup> trivial subset  $\mathcal{S}_0$  containing the solved state with  $d(S^*, S^*) = 0$ , which we omit for simplicity.

<sup>13</sup>The usage of a shared generator here hints towards equivalent states sharing the same sets of facet patterns (disregarding the color), further elaborated in Sec. 2.4.4

**Algorithm 2** Simplified Equivalence Class  $\mathcal{S}^j$  Construction

Given  $S$  and  $G$ , the symmetry-equivalent states  $S'_{1..48} \in \mathcal{S}^j$  can also be constructed as follows:

1. Apply  $\phi_i^G$  to generator  $G$ :  $\phi_i^G(G) = G'$ .
2. Apply the new generator  $G'$  to the solved state  $S^*$ .

### 2.4.2 Conjugacy Classes

The analysis from the constructive algorithms (Alg. 1, Alg. 2) then allows to further formalize the chosen equivalence classes in group-theoretic terms.

**Definition 2.12.** A group element  $T' \in \mathcal{G}$  is said to be *conjugate* to element  $T \in \mathcal{G}$  if there exists an element  $X \in \mathcal{G}$  such that  $T' = X^{-1} \circ T \circ X$

It was stated that a cube state  $S = G \circ S^*$ , acquired by applying the generator  $G$  on the solved state  $S^*$ , has a symmetry-equivalent state representable as  $S' = \phi^{-1} \circ G \circ \phi \circ S^* = G' \circ S^*$ . Thus, from Def. 2.12 it follows that the elements of the generalized Rubik's cube group (Sec. 2.3.3), generalized generators  $G$ ,  $G' = \phi^{-1} \circ G \circ \phi \in \mathcal{G}$ , are *conjugates* and thus the sought-after equivalence classes can be seen as *conjugacy classes*.

Consequently, from the perspective of geometric deep learning [13], we may classify the problem of symmetry-invariant learning in the Rubik's group as that of *conjugation invariance* [66]. As such, the symmetries extend beyond the typically studied spatial invariances, and their incorporation into neural architecture design is worth further attention, as it has not yet been attempted by previous works, e.g., [66]), that instead resorted to a precomputed representation via a global coordinate system.

### 2.4.3 Other Cube Equivalence Classes

Apart from the reported symmetry-equivalence (Sec. 2.4.1), there are a number of other symmetries of the cube states that might inspire alternative (inferior) geometric deep learning models. Generally, the equivalence class can comprise multiple types of symmetric transformations  $\mathcal{T}$ , with each such transformation generating at most  $n$  equivalent states. However, states can also be (self-)symmetric w.r.t.  $\mathcal{T}$  (Sec.2.3.3), resulting in smaller equivalence classes. In addition,  $\mathcal{T}$  can map the state to a different universum than  $\mathcal{S}$  (Def. 2.7), or may not preserve the state's *distance*  $d(S, S^*)$ . Lastly, some of the transformations are trivially reversible by the canonical representation (Sec. 2.3). We describe a number of such alternative symmetry-equivalences w.r.t. these properties in Tab. 2.2.

Firstly, there are simple *geometric* cube symmetries involving 24 cube rotations and 9 reflections along the respective planes (Fig. 2.3), which do not involve any moves. However, the rotations do not change the cube state and are thus nullified by the selected canonical representation (Sec. 2.3.1). Further, the reflections, even though they preserve the property of the state's distance, map cube states into different universa (Def. 2.7). Nevertheless, these can be extended to the actual

Equivalence Class	Transformation $\mathcal{T}$	Upper Bound	$\forall S : S \in \mathcal{S}$	Preserves $d(S, S^*)$	Reversible
Geometric equivalence	Cube rotation	$6 \cdot 4 = 24$	Yes	Yes	Yes
	Plane reflection	$3 + 6 = 9$	No	Yes	No
Symmetry-equivalence	Rotation only	24	Yes	Yes	No
	Rotation+reflection	24	Yes	Yes	No
Inverse-equivalence	Generator's inverse	2	Yes	Yes	No
Color permutations	Bijjective mapping	$6! = 720$	No	Yes	Some

**Table 2.2:** Alternative equivalence classes of the Rubik’s cube and their properties.

*symmetry* equivalences described in Def. 2.10. Thirdly, there is an *inverse* equivalence, inspired by [80]. Starting with a position  $S$  and its generator  $G$ , we arrive at the inverse-equivalent position  $S'$  by inverting the order of the moves in the generator and replacing clockwise moves with anti-clockwise moves, and vice versa. However, this requires knowledge of the actual cube generator, for which it is not possible to incorporate this equivalence into a learning model designed to do so. Finally, the overall generic class of all the color *permutations* actually covers the geometric symmetries, too, including the symmetry-equivalence and inverse-equivalence classes (if we allow their combination with the geometric symmetries). However, the cube universum (Sec. 2.3.2) is not closed under all such color permutations, hence this class cannot be used in the model design either.

#### 2.4.4 Invariant Neural Architecture Design

Ultimately, should we be able to endow  $f_\theta$  with invariance w.r.t. the symmetry-equivalence, we would effectively alleviate the (data) complexity of the problem of learning  $d(S, S^*)$  by a factor of up to 48 (Sec. 2.3.3). A straightforward universal solution to design such an architecture would be to internally expand the corresponding symmetry class  $\mathcal{S}^j$  for each input state and aggregate the respective representations of all its members  $\{S_1, \dots, S_{48}\}$  within the computation graph of the model  $f_\theta$  (see Sec. A.2 for a related CNN-based model). Nevertheless, this naive solution is rather expensive, and the group expansion, described in Alg. 1, is not easy to vectorize.<sup>14</sup> Instead, building on the insights from Sec. 2.3, we seek to design a more efficient, compact architecture satisfying the same invariance properties from the ground up.

#### Recoloring Invariance

The core insight can be drawn from the recoloring equivalence (Def. 2.9), realizing that no permutation of the cube colors plays any role in distinguishing the equivalence classes. Turning around the definition of symmetry-equivalence (Def. 2.10), we know that some recolorings  $\pi$ , matched with an appropriate transformation  $\phi$ , produce equivalent states. While other such transformations do not produce these, it is merely because they lead to different cube universa (Def. 2.7),<sup>15</sup> where they form

<sup>14</sup>Moreover, although it follows a design similar to some of the geometric deep learning models [13] and related work (Sec. 1.2), it is arguably close in spirit to mere data augmentation.

<sup>15</sup>It might not be trivial to see why all other combinations of recoloring and rotations/reflection lead to different universa, but it is simply because the 48 recolorings exhaust the set of color



their own equivalence classes, while still retaining the inner state structure and distance to their respective goal states. Consequently, our architecture should treat all the individual colors exactly the same, i.e., remain *recoloring invariant*.

### ■ Color Patterns

Removing the information about specific colors, we are only left with their *patterns*.

**Definition 2.13.** A *color pattern*  $P$  is the set of relative positions of facelets  $\{f_1, \dots, f_9\}$  of a single color  $C$ , disregarding their absolute coordinates, orientations, and color name.

Note that, following Def. 2.10, each color pattern is an *invariant* of a cube state w.r.t. its symmetric transformations. That is, using a fixed cube orientation, a pattern  $P$  can be transformed in one of the  $\phi_1.. \phi_{48}$  ways, followed by recoloring  $\pi$  w.r.t. the final position of the middle facelet. As these transformations do not change the relative positions of the facelets, all the symmetry-equivalent states  $S' \in \mathcal{S}^j$  necessarily share the same set of color patterns. Consequently, we know that the architecture should necessarily represent the states of a cube  $S \in \mathcal{S}$  as an invariant composition of such color-independent  $\phi$ -invariant pattern representations  $\{P_1, \dots, P_6\}$ , the parameterization of which should be shared across all the 6 patterns.

### ■ Invariant Pattern Representation

We may essentially view each color pattern as a set of 9 3D points lying on a cube surface. Following that view, we choose to represent these points directly as a distance-weighted spatial *graph*, in the spirit of geometric deep learning. This is a convenient format upon which the existing stack of Graph Neural Networks (GNNs) can be exploited. Nevertheless, as we disregard both the positions and colors of the facelets, we are left with no features associated with the nodes. We address this by seeding the nodes with a modified run of the Weisfeiler-Lehman’s (WL) color refinement [109], underlying the GNNs [114]. Particularly, each node (facelet) collects distances to all its neighbors into a set  $\{d_1, \dots, d_6\}$ , similarly to Li et al. [64]. To turn these into the required feature vectors while avoiding any undesired ordering of the nodes, we sort them w.r.t. their *values*, resulting in  $\vec{x}_i = (d_1, \dots, d_6)$ . For further possible WL iterations, the respective neighbor vectors  $\{\vec{x}_1, \dots, \vec{x}_6\}$  can then again be concatenated in a lexicographic order, or simply aggregated in a standard GNN fashion, with which the architecture may then continue. The proposed concept is outlined in Fig. 2.6.

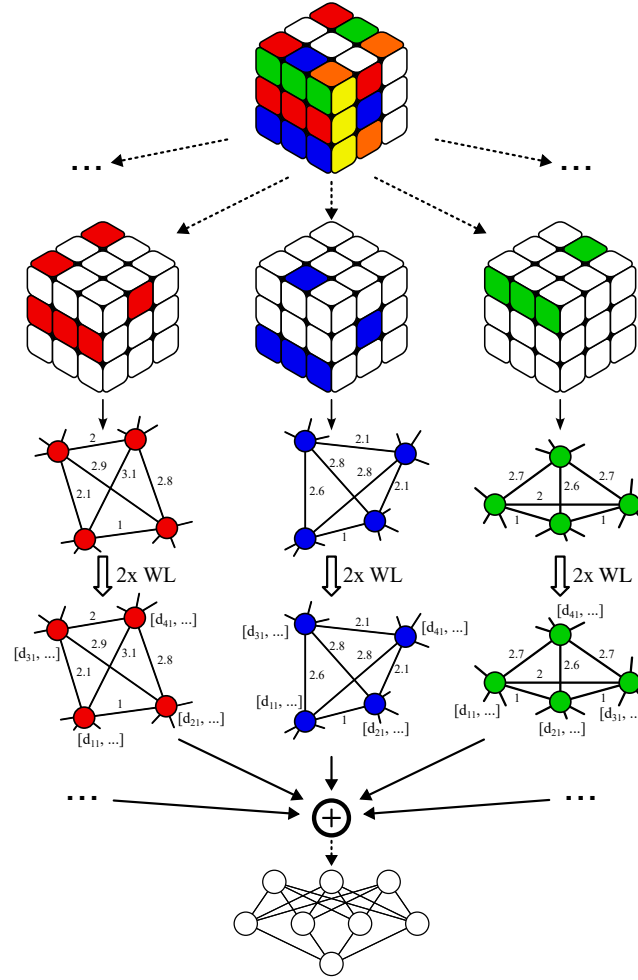
### ■ 2.4.5 Neural Symmetry Detection

In search for the specific design choices for the final symmetry-invariant neural architecture  $f_\theta$ , based on the outlined building blocks, there are generally two competing objectives: (i) *compression* and (ii) *expressiveness*. Compression refers to the decrease of the problem complexity via collapsing the state space into the

---

permutations maintaining the same color neighborhood, which is a defining factor for the cube universa.





**Figure 2.6:** The proposed symmetry-invariant architecture (Sec. 2.4.4).

symmetry-equivalence classes  $\mathcal{S}^j$ , while expressiveness refers to the ability to distinguish the states from the different distance-equivalence classes  $\mathcal{S}_{d=i}$  (Sec. 2.4.1). For instance, a standard feed-forward architecture trivially satisfies the expressiveness criterion but yields no compression. On the other hand, invariant architectures that compress the problem space may not be sufficiently expressive. In the case of Rubik’s cube, introducing any undesired facelet or color ordering in the architecture would hurt the invariance-induced compression, while compressing too much would hurt expressiveness and thus the capacity of the model  $f_\theta$  to learn  $d(S, S^*)$ .

### Empirical Procedure

To effectively navigate the specific design choices, we propose the following *universal* neural symmetry-detection routine, operating on a given set of labeled samples  $\mathcal{S}' \subseteq \mathcal{S}$ . Firstly, we organize the set into distance-equivalence classes  $\mathcal{S}'_i$ , and further into symmetry-equivalence classes  $\mathcal{S}'_i^{j'}$ , e.g., following the Alg. 2. Secondly, given a neural model  $f_\theta$  with randomly initialized parameters  $\theta$ , we also group the  $\mathcal{S}'$  into new *value-equivalence* classes  $\mathcal{V}_k$  induced by their empirical model-evaluations

as  $\mathcal{V}_k = \{S \mid f_\theta(S) = k\}$ .<sup>16</sup> Thirdly, we check intersections between the  $\mathcal{V}_k$ 's and  $\mathcal{S}_i^{j'}$ 's. If the model  $f_\theta$  perfectly captures the domain symmetries  $\phi$ , the sets should coincide, i.e., all the samples  $S$  within each given symmetry class  $\mathcal{S}_i^{j'}$  should map to an identical value  $k = f_\theta(S)$  (compression), while no two samples from different  $\mathcal{S}_i^{j'}$  should (expressiveness).

We note that it may happen that evaluations of two samples  $f_\theta(S_1), f_\theta(S_2)$  coincide by chance rather than genuine symmetry-invariance of  $f_\theta$ .<sup>17</sup> We effectively mitigate the probability of such inadvertent value clashes by an ‘‘amplification trick’’ normally used in the design of randomized algorithms [96]. Specifically, we repeatedly sample  $\theta$  to yield a list of predictions  $(f_{\theta_1}(S_i), \dots, f_{\theta_{rep}}(S_i))$  for each  $S_i \in \mathcal{S}'$ . By hashing the resulting value lists, the whole procedure can be performed in linear amortized time, w.r.t.  $f_\theta, |\mathcal{S}'|$  and  $rep$ , using hashmaps for both the  $\mathcal{S}_i^{j'}$  and  $\mathcal{V}_k$  partitionings of  $\mathcal{S}'$ .

The generic routine, described in Alg. 3, is then applicable to *any* symmetry-aware deep learning architecture, independent of the domain. Different domains might, however, require different approaches to the *partitioning* of the samples into the equivalence classes. There are generally three options to do so, either (i) a full expansion of each class  $\mathcal{S}_i^{j'}$  from a single sample  $S \in \mathcal{S}_i^{j'}$  (as in Alg. 2 for the Rubik’s cube domain), (ii) augmentation of the samples  $S$  through a given symmetry transformation  $\phi$  (e.g., in domains where the full expansion would be expensive), or (iii) by directly matching equivalent states  $f_\theta(S_1) = f_\theta(S_2)$  within a dataset (if an equivalence certificate is easier to compute than the expansion).

Additionally, the requirement for a perfect overlap of the empirically-induced  $\mathcal{V}_k$  and theoretically-induced  $\mathcal{S}_i^{j'}$  decompositions of  $\mathcal{S}'$  may be relaxed in two ways. Firstly, it is generally not necessary for evaluations of states from two distinct symmetry classes  $S_1 \in \mathcal{S}_i^1, S_2 \in \mathcal{S}_i^2$  to be different, i.e.,  $f_\theta(S_1) = f_\theta(S_2)$  is allowed as long as they are from the same distance class  $\mathcal{S}_i$  (necessary expressiveness). Such a model  $f_\theta$  might lead to even higher compression than that induced by the domain symmetries. Secondly, on the contrary, even if the domain symmetries are only partially captured, i.e., not all  $S \in \mathcal{S}_i^{j'}$  map to the same  $f_\theta(S)$ , the model  $f_\theta$  might still be beneficial, as long as it retains at least some compression.

## ■ Invariance Error Metrics

Following the outlined reasoning, we further define a fine-grained error measure for the candidate models  $f_\theta$  failing the strict tests from Alg 3. This can happen either because (i) there is more than one hash value  $h_S$  in some  $\mathcal{V}_i^j$  of the equivalence classes  $\mathcal{S}_i^{j'}$  (insufficient compression ratio;  $r < 48$  for the Rubik’s cube), or because (ii) different classes  $\mathcal{S}_i^{j'}$  share the same (hash) value  $h_S$  (insufficient expressiveness).

<sup>16</sup>In practical terms, this means collecting and hashing output activations of a forward pass through the representation model.

<sup>17</sup>Generally, the probability of this is rather small if  $f$  is continuous in  $\theta$ . For that purpose, non-injective functions, such as *ReLU*, may be replaced with more suitable ones, such as *tanh*.

<sup>18</sup>Before hashing, it is advisable to magnify the activations by several orders of magnitude and round the result to an integer. This is to avoid possible problems with numeric instability of real/floating point numbers in computers.

---

**Algorithm 3** Neural Symmetry-Detection Routine

---

```

1:  $\{\mathcal{S}_i^{j'}\} = \text{partition}(\mathcal{S}')$ 
2: Initialize  $f_{\theta_1}, \dots, f_{\theta_{rep}}$ 
3:  $\mathcal{V} \leftarrow \{\}$ 
4: for  $\mathcal{S}'_i \in \mathcal{S}'$  do
5:   for  $\mathcal{S}_i^{j'} \in \mathcal{S}'_i$  do
6:      $\mathcal{V}_i^j \leftarrow \{\}$ 
7:     for  $S \in \mathcal{S}_i^{j'}$  do
8:       for  $q = 1, \dots, rep$  do
9:          $k_q \leftarrow f_{\theta_q}(S)$ 
10:      end for
11:       $h_S \leftarrow \text{hash}((k_1, \dots, k_{rep}))$ 18
12:       $\mathcal{V}_i^j \leftarrow \mathcal{V}_i^j \cup \{h_S\}$ 
13:    end for
14:     $\mathcal{V} \leftarrow \mathcal{V} \cup \{\mathcal{V}_i^j\}$ 
15:  end for
16: end for
17: for  $\mathcal{V}_k \in \mathcal{V}$  do
18:   Check  $|\mathcal{V}_k| = 1$  (compression)
19: end for
20: for  $\mathcal{V}_k, \mathcal{V}_l \in \mathcal{V}, k \neq l$  do
21:   Check  $\mathcal{V}_k \cap \mathcal{V}_l = \{\}$  (expressiveness)
22: end for

```

---

Note that it only makes sense to report the compression ratio  $r = \frac{|\{h_S | S \in \mathcal{S}_i^{j'}\}|}{|\mathcal{S}_i^{j'}|}$  within the equivalence classes  $\mathcal{S}_i^{j'}$  if these are correctly recognized, i.e., there is no error of the second type. Thus, to properly report the number of hash value clashes of the states  $\mathcal{S}'$  across the equivalence classes  $\mathcal{S}_i^{j'}$ , taking into account their possibly varying sizes,<sup>19</sup> we base the metric on the number of *incorrectly matched state pairs*.

**Definition 2.14.** States  $S$  and  $S'$  are *incorrectly matched*, denoted as  $S ! S'$ , if they share the same evaluations  $h_S = h_{S'}$  but belong to different symmetry equivalence classes  $\mathcal{S}_i^{j'}$ .

For each cube state  $S \in \mathcal{S}^k$  and a corresponding set of  $n$  symmetry equivalence classes sharing the same hash value  $\{\mathcal{S}^1, \dots, \mathcal{S}^n \mid \forall j, \forall S' \in \mathcal{S}^j : S' \rightarrow h_{S^j}\}$ , we may define the set  $\mathcal{F}^S = \{S' \mid S ! S'\}$  of all the states<sup>20</sup> from the remaining ( $j \neq k$ ) symmetry classes as  $\mathcal{F}^S = \bigcup_{\substack{j=1 \\ j \neq k}}^n \mathcal{S}^j$ . The total number of incorrectly matched state

---

<sup>19</sup>One could, e.g., report the total number of classes with a non-unique hash value, however, it would not be clear if they are grouped in pairs, triples, quadruples, etc., which determines the maximal number of classes distinguishable by the learner.

<sup>20</sup>or, possibly, patterns for the *pattern* dataset (Sec. 3.1.1)

pairs for all the states from all the classes is then

$$|\mathcal{F}| = \sum_S \sum_k |\mathcal{S}^k| \cdot \sum_{j \neq k} |\mathcal{S}^j|,$$

$$s.t. \forall S \in \mathcal{S}, \forall S \in \mathcal{S}^k, S' \in \mathcal{S}^j, \forall \theta : f_\theta(S) = f_\theta(S')$$

## 2.5 Parameters of GNN Interpretability Methods

In Sec. 2.4.4, we have outlined our target neural architecture as a graph neural network. Therefore, we return to GNN interpretability methods and discuss the related methodology and design choices w.r.t. the parameters of the selected post-hoc<sup>21</sup> methods.

Following up on Yuan’s taxonomy [117] described in Sec. 1.2.2, Amara et al. [4] created a systematic evaluation framework of GNN interpretability techniques looking at different possible objectives of the user. The user should first decide the **focus** of explanation – whether they are more interested in discovering the *phenomena* existing in the data or the *model’s* behavior<sup>22</sup>. The second aspect is the **mask nature** – whether the edge mask  $M_E$  should be discrete (*hard*),  $M_E \in \{0, 1\}^{M \times M}$ , or continuous (*soft*),  $M_E \in [0, 1]^{M \times M}$ . Finally, there is the aspect of **mask transformation/aggregation** – the explanation algorithm should output only the important masks, and the filter can be set, e.g., by specifying the top-k number of masks or a threshold on the importance value. While the last two aspects are a matter of parameter tuning, the focus of the explanation is a fundamental question that should be decided based on the purpose of the explanation.

In Sec. 3.3, we compare the transparency of the interpretations based on the existing techniques [117], concretely the GNNExplainer [115] and Parameterized Explainer (PGExplainer) [69], with our own intuitive explanations based on the understanding of the symmetry-aware architecture. Even though there exist metrics for evaluating explanations specifically modified for graphs, such as *fidelity* [117] or *unfaithfulness* [1], they are defined for classification tasks, not regression, which is our task at hand. Thus, in the comparison, we stress the usefulness of the produced explanations. Especially, we would like the explanation methods to be able to *distinguish representation of inputs with different ground-truth* in a way that is understandable for humans.

<sup>21</sup>All methods mentioned in Sec. 1.2.2 belong to the category of post-hoc explainability methods.

<sup>22</sup>Model focus of an explanation is not to be confused with the model-level explanation (e.g., the XGNN method in Sec. 1.2.2). The explanation with model focus is still based on dependency between concrete data and predictions, while model-level methods are independent of data input.

## Chapter 3

### Rubik's Cube Experiments

With the experiments, we aim to validate the design of the introduced symmetry-invariant neural architecture blueprint by answering the following questions:

- Q1 What is the simplest, sufficiently expressive model design of the color patterns and their combination?
- Q2 What is the real state-space compression factor of the resulting architecture?
- Q3 How does it compare to related work in terms of data efficiency and training generalization?
- Q4 How does it compare as a heuristic estimator in terms of searching for (optimal) solutions?

#### 3.1 Data

The Rubik's cube has a fully observable state space, but with 43,252,003,274,489,856,000 unique configurations, it is impossible to analyze in full, for which we resort to data generation and sampling techniques.

##### 3.1.1 Color Patterns

Firstly, we generated an artificial, exhaustive dataset, specifically created to navigate the design of suitable color-pattern representations, addressing the Q1. The dataset is a collection of *all* valid single-color facelet configurations, restricted only by the physical cube properties (Sec. 2.3.2). Specifically, we restrict the configurations to consist of exactly 4 corner facelets, 4 edge facelets, and 1 middle facelet, yielding  $\binom{24}{4} \cdot \binom{24}{4} \cdot 6 = 677,471,256$  configurations that were labeled by their symmetry classes (Sec. 2.4.4), which we further filter out if there are multiple facelets of the same color on the same cubie, resulting in  $\binom{8}{4} \cdot 3^4 \cdot \binom{12}{4} \cdot 2^4 \cdot 6 = 269,438,400$  configurations.

##### 3.1.2 Reverse Generation

To obtain a suitable dataset for analyzing the whole cube configurations, we, similarly to the original DeepCube [2] experiments, generated the states by progressing from

the solved configuration  $S^*$  in a reverse manner, through which we exhaustively generated a dataset of all the Rubik’s cube states with  $d(S, S^*) \leq 5$ . We used the quarter-turn metric  $d_{qt}$  (Sec. 2.3.1) to align with the choice of [2]. With the exponential growth of the state space, this results in a dataset with  $\approx 10^5$  unique states. In contrast to [2] that sampled also from greater depths, this allows to retain the exact ground truth optimal distance  $d(S, S^*)$  information by keeping a closed list of visited states, which is necessary for our exact analysis.

### 3.1.3 Sampling

Finally, to address the lack of distance-labeled configurations from greater depths, which cannot be reached using the closed list technique, one may resort to sampling in combination with (optimal) solvers. While running the solvers is relatively expensive, fortunately, there is a freely available dataset of (another)  $10^5$  solved positions due to [56]. This dataset is available in the face-turn metric  $d_{ft}$  (Sec. 2.3.1). We further expanded the dataset using the generation Alg. 1, yielding 4,800,000 labeled states, spanning the distances of  $d(S, S^*) = [14, \dots, 19]$ .<sup>1</sup>

## 3.2 Results

In this section, we report the results of the model design choices (Q1, Q2), learning performance (Q3), and problem-solving performance (Q4) of the proposed neural architecture outlined in Sec. 2.4.4. Ablations are then reported in Sec. 3.2.2.

### 3.2.1 Architecture Design

We tested several specific configurations of the proposed architecture (Fig. 2.6) w.r.t. the choices of (i) distance metric, (ii) graph connectivity, and (iii) aggregation operators, and evaluated the resulting model’s expressiveness. Using the symmetry detection technique (Sec. 2.4.5) on the exhaustive color-pattern dataset (Sec. 3.1.1), we established that a simple choice of (i) Euclidean distance weighting of a (ii) fully connected graph of the facelets  $f_i \in P$  (Sec. 2.4.4) is sufficient, if combined with (a minimum of)  $k = 2$  iterations of the WL-based message-passing updates ( $WL^k$ ) as:

$$\vec{x}_i^{(k)} = \vec{x}_i^{(k-1)} + \sum_{\substack{j=1 \\ j \neq i}}^6 \|f_i - f_j\|_2 \cdot \vec{x}_j^{(k-1)}$$

Similarly, for the (iii) aggregation operator, the simple choice of *sum* was found sufficiently expressive, in accordance with the findings in related geometric architectures [119, 114], despite the existence of more expressive standalone operators (such as the lexicographic sorting, Sec. 2.4.4). This also applied to the pattern-level aggregation, where a direct summation of the resulting representations again proved

<sup>1</sup>The limited span is due to the heavily uneven distribution of the underlying state space, with an absolute majority of states residing around these depths. Also, none of the states in the set were self-symmetric. Thus, the expansion factor was exactly 48.

Dataset	$ \mathcal{S} $	$\#S^j$	$\overline{ S^j }$	Err(WL <sup>2</sup> )	$\overline{ \text{WL}^2 }$
$d_{qt} = 1$	12	1	12.00	0	12.00
$d_{qt} = 2$	114	5	22.80	0	22.80
$d_{qt} = 3$	1,068	25	42.72	0	42.72
$d_{qt} = 4$	10,011	219	45.71	0	45.71
$d_{qt} = 5$	93,840	1,978	47.44	0	47.44
$d_{ft} = 14$	864	18	48.00	0	48.00
$d_{ft} = 15$	9,456	197	48.00	0	48.00
$d_{ft} = 16$	130,080	2,710	48.00	0	48.00
$d_{ft} = 17$	1,280,304	26,673	48.00	0	48.00
$d_{ft} = 18$	3,220,752	67,099	48.00	0	48.00
$d_{ft} = 19$	158,544	3,303	48.00	0	48.00
patterns	269,438,400	5,617,306	47.9658	0	47.9658

**Table 3.1:** State space statistics of the 3 datasets, with (expressiveness) error and compression of the proposed (WL<sup>2</sup>) model design.

sufficiently expressive, concluding the search for the simplest invariant architecture delineated in Q1.

Subsequently, using the same technique (Sec. 2.4.5) on the exhaustive and sampled datasets, we established that the architecture indeed follows the exact symmetry decomposition induced by the domain symmetries (Sec. 2.3.3), as the partitioning into the model’s symmetry-equivalence classes matched the theoretical distribution *exactly*. Consequently, apart from the few low-depth subsets of the state space that contain a lot of self-symmetric configurations, the model is indeed able to compress the data space by the theoretical (upper bound) factor of 48 (Sec. 2.3.3), answering the Q2.

The results are reported in Tab. 3.1, decomposing the cube state space into the respective depths  $d(\mathcal{S}, S^*)$  of the *exhaustive* (quarter-turn,  $d_{qt} = [1..5]$ ) and *sampled* (face-turn,  $d_{ft} = [14..19]$ ) datasets, extended with the color-pattern (**patterns**) dataset (Sec. 3.1). We report the statistics of the number of states ( $|\mathcal{S}|$ ), number of symmetry-equivalence classes ( $\#S^j$ ), and their average sizes ( $\overline{|S^j|}$ ). Finally, we report the expressiveness errors of the proposed WL<sup>2</sup> design, as measured by the number of incorrectly matched state pairs (Def. 2.14), and its respective compression rates (Sec. 2.4.5), corresponding to the average sizes of the induced value-equivalence classes. As outlined, the proposed architecture achieves perfect compression with zero (expressiveness) error. See Sec. 3.2.2 for comparison with other (inferior) model designs for further substantiation of the proposed architecture.

### ■ 3.2.2 Pattern Representation Ablations

Apart from the proposed WL-based pattern invariant design (Sec. 2.4.4), we also tested a number of different  $\phi$ -invariant features, namely (i) the scalar representing the volume of the convex hull of the pattern points, (ii) the sorted collection of distances to the (fixed) middle facelet of the (color) pattern, and (iii) the sorted and

aggregated 2D collection of all pairwise distances of the pattern points. Invariants (ii) and (iii) were considered in variants with different distance metrics (Euclidean, Manhattan, and another related (custom) distance, allowing merely surface traversal across the cube). Recall that a correct pattern invariant must be consistent with the definition of the color pattern (Def. 2.13), disregarding absolute positions and orientations of the pattern points. While the (i) volume invariant fulfills this requirement in a straightforward manner, the (ii) invariant requires sorting the distance collection so as not to introduce any arbitrary orientation and ordering of the pattern points. This issue was further amplified in invariant (iii), where a vector of distances for each pattern point is computed. Thus, the vectors had to be either aggregated (e.g., summed) into a single vector before sorting, as in invariant (ii), or the whole 2D collection had to be sorted lexicographically. The described transformations then ensured compliance of all three invariants with the color invariance (Sec. 2.4.4). The motivation behind these invariants is that they (intuitively) naturally reflect the distance  $d(S, S^*)$ , e.g., the pattern volume in the solved state is 0, while the volume grows as the facelets become more scattered across the cube. However, all these invariants eventually proved of insufficient expressiveness, as reported in the following subsection.

### ■ Tested Pattern Invariants

Recall that each color pattern can be viewed as 9 3-dimensional points (Sec. 2.4.4). Adopting that view, a convex hull can be computed while extracting its *volume* (Vol) as an invariant. This is a computationally expensive operation, however, it can be precomputed and cached, amortizing its computational costs over multiple experiments. Hashing each color pattern by volume, the *expressiveness test* (Alg. 3) from the neural symmetry-detection routine failed. The median number of classes to which each hash corresponded was 6,366.5, with a mean of 18,238.01 classes. There were  $2.91 \cdot 10^{14}$  incorrectly matched patterns.

Expanding of the insufficient expressiveness of Vol, several *distance invariants* were considered. The first version (L2<sub>mid</sub>) computed a single vector of Euclidean distances w.r.t. the middle facelet, proceeding with sorting and hashing of the result. This invariant also failed the expressiveness test. The median number of classes to which each hash corresponded was 576, and the mean was 2,292.78 classes. There were  $6.97 \cdot 10^{13}$  incorrectly matched patterns.

The second version (L2<sub>N×N</sub>) produced a (Euclidean) distance matrix  $9 \times 9$ , with the  $i^{th}$  row/column representing the distances of all facelets to the  $i^{th}$  facelet. The rows of the matrix were sorted first. After that, the matrix was either (a) sorted lexicographically or (b) summed along the column dimension, and the result was hashed. However, the expressiveness test still failed in both cases. For both sum and sort variants, the median number of classes to which each hash corresponded was 1, the mean was 1.00001 classes, and there were 97,344 incorrectly matched patterns.

The third invariant version (L1<sub>N×N</sub>) adjusted the second version, exchanging the Euclidean metric of distance for the Manhattan distance. Hashing the distance matrix similarly with sum (a) and sort (b) variants, both experiments still failed the expressiveness test. The median number of classes to which each hash corresponded was 5 for sum and 1 for lexicographic sort, and the mean was 19.56 classes for sum



and 1.78 for sort. There were  $1.92 \cdot 10^{12}$  incorrectly matched patterns for sum and  $9.99 \cdot 10^9$  incorrectly matched patterns for sort.

Besides the pattern dataset, the described pattern invariant ablations were also tested w.r.t. recognizing the overall cube states. Naturally, none of these invariants was successful in that either, nevertheless, the most promising Euclidean distance invariants ( $L2_{N \times N}$ ) remained very close to perfect on both the exhaustive and sampled datasets,<sup>2</sup> reinforcing the concept that most of the model expressiveness lies in correctly recognizing the patterns themselves.

Similarly to Tab. 3.1 with the main model ( $WL^2$ ) results, we then report the expressiveness errors and compression rates for the described ablation invariants ( $Vol$ ,  $L2_{mid}$ ,  $L2_{N \times N}^+$ ,  $L2_{N \times N}^{sort}$ ,  $L1_{N \times N}^+$ ,  $L1_{N \times N}^{sort}$ ) in Tab. 3.2, and Tab. 3.3, respectively.

Dataset	Err(Vol)	Err( $L2_{mid}$ )	Err( $L2_{N \times N}^+$ )	Err( $L2_{N \times N}^{sort}$ )	Err( $L1_{N \times N}^+$ )	Err( $L1_{N \times N}^{sort}$ )
$d_{qt} = 1$	0	0	0	0	0	0
$d_{qt} = 2$	1,152	0	0	0	0	0
$d_{qt} = 3$	16,128	6,912	0	0	0	0
$d_{qt} = 4$	331,776	0	0	0	0	0
$d_{qt} = 5$	336,384	6,912	0	0	0	0
$d_{ft} = 14$	2,484,864	0	0	0	0	0
$d_{ft} = 15$	24,810,624	0	0	0	0	0
$d_{ft} = 16$	345,217,536	0	0	0	0	0
$d_{ft} = 17$	2,614,694,400	3,456	0	0	3,456	0
$d_{ft} = 18$	2,950,297,344	2,304	0	0	4,608	0
$d_{ft} = 19$	426,272,256	1,152	0	0	1,152	0
patterns	2.9183E+14	6.9748E+13	97,334	97,334	1.9198E+12	9.9933E+9

**Table 3.2:** Expressiveness errors of the different pattern invariants on the three (distance-decomposed) datasets.

### 3.2.3 Learning Performance

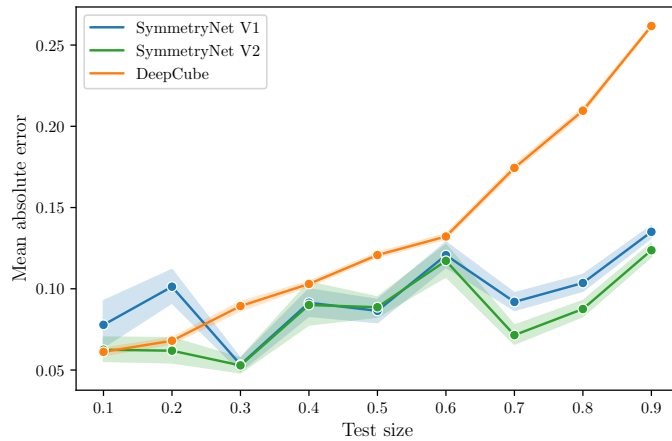
Proceeding with the architecture from 3.2.1, we explored how the improved data efficiency, induced by the symmetry-invariance of the model, translates into its learning performance. Knowing that the aggregated color-pattern representation resulting from the message-passing ( $WL$ ) steps is sufficiently expressive, one can turn the model  $f_\theta$  into a universal approximator of  $d(S, S^*)$  by simply adding a sufficient number (min=2) of fully-connected (FC) layers, as outlined in Fig. 2.6.

To properly evaluate the effect of symmetry-invariance w.r.t. the state-of-the-art, we align these layers with the DeepCube architecture [2], consisting of two such FC layers, followed by additional 2-layer (FC) blocks with residual connections [41]. The resulting symmetry-invariant architecture, referred to as **SymmetryNet**, then contains the same number of parameters as **DeepCube**, with all their hyperparameters and training protocols being completely aligned, too. The last choice for the **SymmetryNet** architecture is the message-passing scheme. Particularly, we considered (i) the simplest possible, non-parametric, message-passing leading to **SymmetryNet V1**, and

<sup>2</sup>The only error made by this invariant was mistaking 144 equivalence classes from a distance  $d_{qt} = 6$ , which would be allowed under the relaxed version of the symmetry-detection routine.

Dataset	$ \text{Vol} $	$ \text{L2}_{\text{mid}} $	$ \text{L2}_{\text{N}\times\text{N}}^+ $	$ \text{L2}_{\text{N}\times\text{N}}^{\text{sort}} $	$ \text{L1}_{\text{N}\times\text{N}}^+ $	$ \text{L1}_{\text{N}\times\text{N}}^{\text{sort}} $
$d_{qt} = 1$	12.00	12.00	12.00	12.00	12.00	12.00
$d_{qt} = 2$	28.50	28.50	22.80	22.80	22.80	22.80
$d_{qt} = 3$	42.72	80.09	42.72	42.72	42.72	42.72
$d_{qt} = 4$	49.32	94.41	45.71	45.71	45.71	45.71
$d_{qt} = 5$	85.70	96.15	47.44	47.44	47.44	47.44
$d_{ft} = 14$	57.60	48.00	48.00	48.00	48.00	48.00
$d_{ft} = 15$	50.84	48.00	48.00	48.00	48.00	48.00
$d_{ft} = 16$	123.65	48.00	48.00	48.00	48.00	48.00
$d_{ft} = 17$	649.57	48.00	48.00	48.00	48.005	48.00
$d_{ft} = 18$	1,427.01	48.002	48.00	48.00	48.003	48.00
$d_{ft} = 19$	142.58	48.00	48.00	48.00	48.00	48.00
patterns	874,800	109974.8571	47.9662	47.9662	938.3390	85.3277

**Table 3.3:** Compression rates of the pattern invariants on the datasets

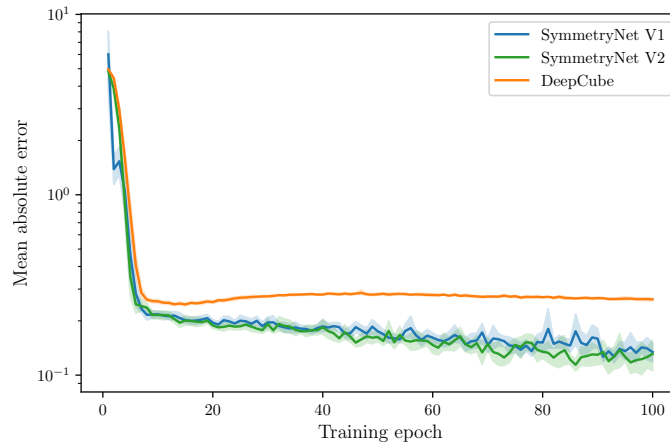


**Figure 3.1:** Test-set MAE model performances when generalizing from decreasing fractions of the exhaustive dataset ( $d_{qt} = [1 \dots 5]$ ), displayed with a 95% confidence interval.

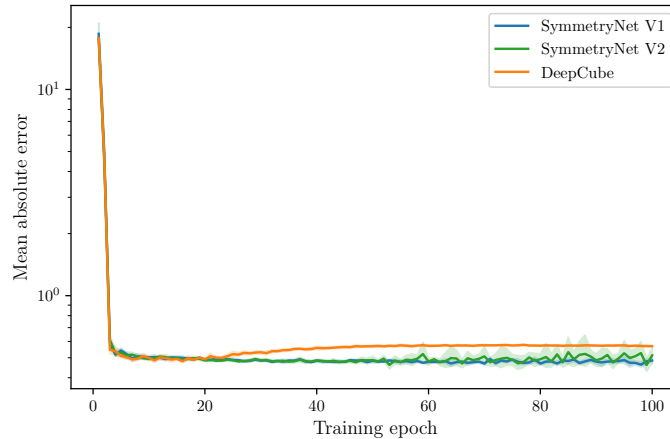
(ii) the generalized graph convolutional layer [63] leading to SymmetryNet V2, while still retaining the exact same number of learnable parameters. Implementation details of all the architectures are then described in Sec. 3.2.6.

First, we report the learning performances of the three models through mean absolute error (MAE) against the optimal  $d(S, S^*)$ . Starting with the exhaustive dataset (Sec. 3.1.2), Fig. 3.1 depicts the (converged) MAE test-set performances for varying train-test split ratios, while Fig. 3.2 displays the progress of the performance when generalizing from 10% of the data, corresponding to the most realistic setting from the given splits.<sup>3</sup> For the full  $10^5$ -sized sampled dataset (Sec. 3.1.3), the models’

<sup>3</sup>The real train-set ratio is even (much) lower during the actual search, given the vast size of the state space.



**Figure 3.2:** The effect of symmetry-invariance demonstrated through the models’ learning convergences when generalizing from 10% of the exhaustive dataset ( $d_{qt} = [1 \dots 5]$ ), displayed with a 95% confidence interval.

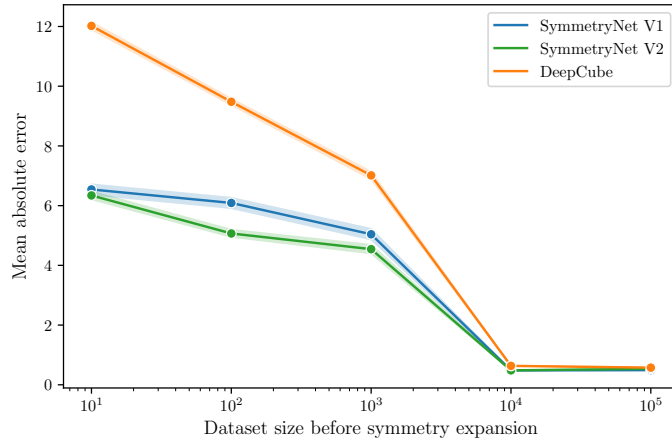


**Figure 3.3:** The effect of symmetry-invariance demonstrated through the models’ learning convergences on the full  $10^5$  sampled dataset ( $d_{ft} = [14 \dots 19]$ ), displayed with a 95% confidence interval.

test-set MAE convergence is then depicted in Fig. 3.3, with Fig. 3.4 showing the models’ converged test performances while increasing the train split of the sampled dataset. The results for both datasets are further summarized in detail in Sec. 3.2.5.

As can be observed, the **SymmetryNet** models provide substantially better generalization<sup>4</sup> than the symmetry-unaware **DeepCube**, which is much more prone to overfitting early in training (Fig. 3.2), resulting in generally worse test errors. This advantage of **SymmetryNet** models then grows with the decreasing size of the labeled fraction of the datasets (Fig. 3.1, Fig. 3.4). This is of particular importance due to the vast size of the actual state space of which only a tiny fraction can ever be

<sup>4</sup>Furthermore, performance-wise, **SymmetryNet V2** is slightly preferable over **SymmetryNet V1** for some of the tasks. The advantage is, however, balanced by the V2 being more computationally expensive.



**Figure 3.4:** Models’ learning performances when generalizing to the symmetry-equivalent states from an increasingly large portion of the sampled dataset ( $d_{ft} = [14 \dots 19]$ ), displayed with a 95% confidence interval.

used for training during the search. Finally, despite not being very discernible from Fig. 3.4 (see Sec. 3.2.5 for detail), even on the full  $10^5$  established dataset from [56] both the SymmetryNet architectures end up with a significant performance advantage over DeepCube. In accordance with the theoretical assumptions (Sec. 2.4), we can thus confirm the SymmetryNet to yield better data efficiency and generalization, answering the Q3.

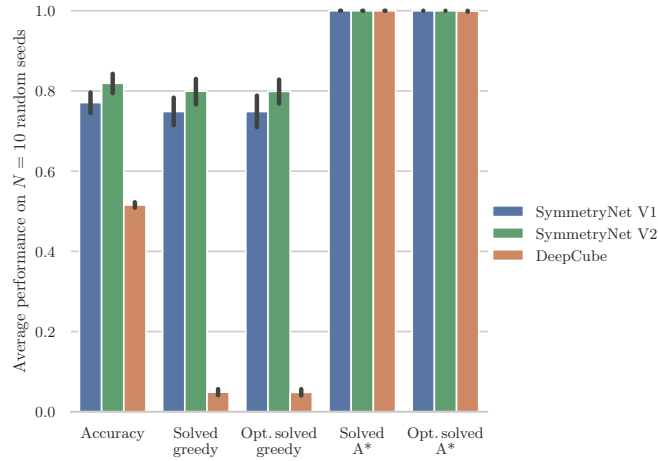
### 3.2.4 Search Performance

Finally, to directly compare the proposed SymmetryNet architectures with DeepCube, we used the trained models as heuristic function estimators during the actual problem-solving. Particularly, we considered two types of search – (i) greedily selecting the next state following the smallest heuristic value in each iteration, and the A\* search, as also used in [2].<sup>5</sup> For a more detailed analysis, we also computed an *accuracy metric*, measuring the ratio of states in which an optimal successor has correctly been selected. In [2], the data and labels were bootstrapped starting from the solved state, and continuing towards larger distances while updating the heuristic model regularly with a small number of samples. This is analogical to the exhaustive dataset (Sec. 3.1.2) with a correspondingly small train-test ratio, which we set to the (generous) 10%.<sup>6</sup>

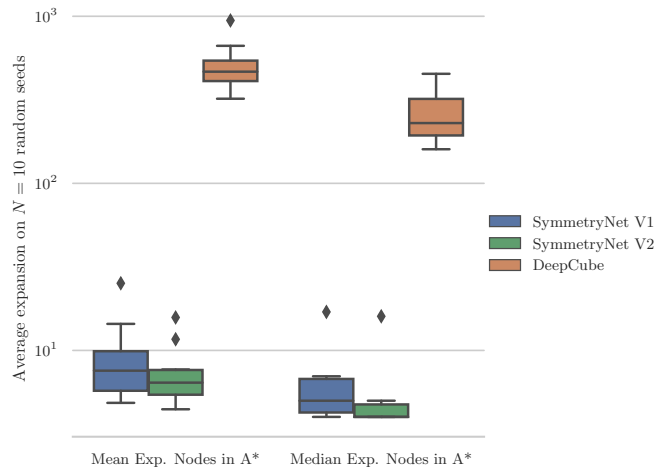
We report the accuracy, and the ratio of solved and optimally solved states in Fig. 3.5. Since, given the relatively generous search space budget for the given problem size, all the models are able to successfully solve the given instances with A\*, we additionally report the mean and median numbers of expanded nodes in Fig. 3.6. As can be observed from the figures, similarly to the MAE performances,

<sup>5</sup>For practical reasons, we restricted the search to paths  $l(Path) \leq 20$ , which is a very generous bound for the dataset.

<sup>6</sup>This is, again, more than generous, considering that merely a tiny fraction of the actual state space can ever be visited.



**Figure 3.5:** Model comparison through the problem-solving (search) performance metrics - (i) accuracy of selecting optimal successor, (ii) solution rates, and (iii) optimal solution rates (higher is better).



**Figure 3.6:** Log-scale comparison of the models' A\* search performances as measured through the average number of expanded nodes (lower is better).

the SymmetryNet architectures are clearly superior during problem-solving too, as measured through all the accuracy, the solution rates of the greedy search, and the size of the expanded search space during the A\* search.

### 3.2.5 Performance Comparison Tables

Accompanying the mean absolute error (MAE) performance graphs from Sec. 3.2, we further report the MAE comparisons with standard deviations over  $N = 10$  random initializations for the exhaustive and the sampled datasets in Tab. 3.4 and Tab. 3.5, respectively.

For the average search performances and expansion rates with their standard deviations, see Tab. 3.6.

Exhaustive		MAE $\pm$ std. dev.	
Test Size	DeepCube	SymmetryNet V1	SymmetryNet V2
0.1	<b>0.0612 <math>\pm</math> 0.0119</b>	0.0778 $\pm$ 0.0694	0.0625 $\pm$ 0.0376
0.2	0.0680 $\pm$ 0.0098	0.1013 $\pm$ 0.0555	<b>0.0619 <math>\pm</math> 0.0378</b>
0.3	0.0894 $\pm$ 0.0148	0.0533 $\pm$ 0.0203	<b>0.0529 <math>\pm</math> 0.0230</b>
0.4	0.1030 $\pm$ 0.0072	0.0914 $\pm$ 0.0421	<b>0.0900 <math>\pm</math> 0.0651</b>
0.5	0.1208 $\pm$ 0.0080	<b>0.0864 <math>\pm</math> 0.0347</b>	0.0887 $\pm$ 0.0315
0.6	0.1322 $\pm$ 0.0086	0.1207 $\pm$ 0.0396	<b>0.1172 <math>\pm</math> 0.0506</b>
0.7	0.1744 $\pm$ 0.0098	0.0919 $\pm$ 0.0284	<b>0.0715 <math>\pm</math> 0.0303</b>
0.8	0.2096 $\pm$ 0.0089	0.1036 $\pm$ 0.0267	<b>0.0876 <math>\pm</math> 0.0247</b>
0.9	0.2617 $\pm$ 0.0044	0.1351 $\pm$ 0.0189	<b>0.1237 <math>\pm</math> 0.0222</b>

**Table 3.4:** Comparison of the models’ MAE performances across different train-test ratios of the exhaustive dataset

Sampled		MAE $\pm$ std. dev.	
Dataset Size	DeepCube	SymmetryNet V1	SymmetryNet V2
10	12.0195 $\pm$ 0.6078	6.5421 $\pm$ 0.8793	<b>6.3418 <math>\pm</math> 0.6854</b>
100	9.4784 $\pm$ 0.5460	6.0905 $\pm$ 0.9017	<b>5.0683 <math>\pm</math> 0.6419</b>
1,000	7.0128 $\pm$ 0.6516	5.0410 $\pm$ 0.9732	<b>4.5430 <math>\pm</math> 0.7243</b>
10,000	0.6340 $\pm$ 0.0164	<b>0.4834 <math>\pm</math> 0.0342</b>	0.4839 $\pm$ 0.0304
100,000	0.5745 $\pm$ 0.0104	<b>0.4913 <math>\pm</math> 0.0208</b>	0.5355 $\pm$ 0.0934

**Table 3.5:** Comparison of the models’ MAE performances while increasing the (subset) size of the sampled dataset

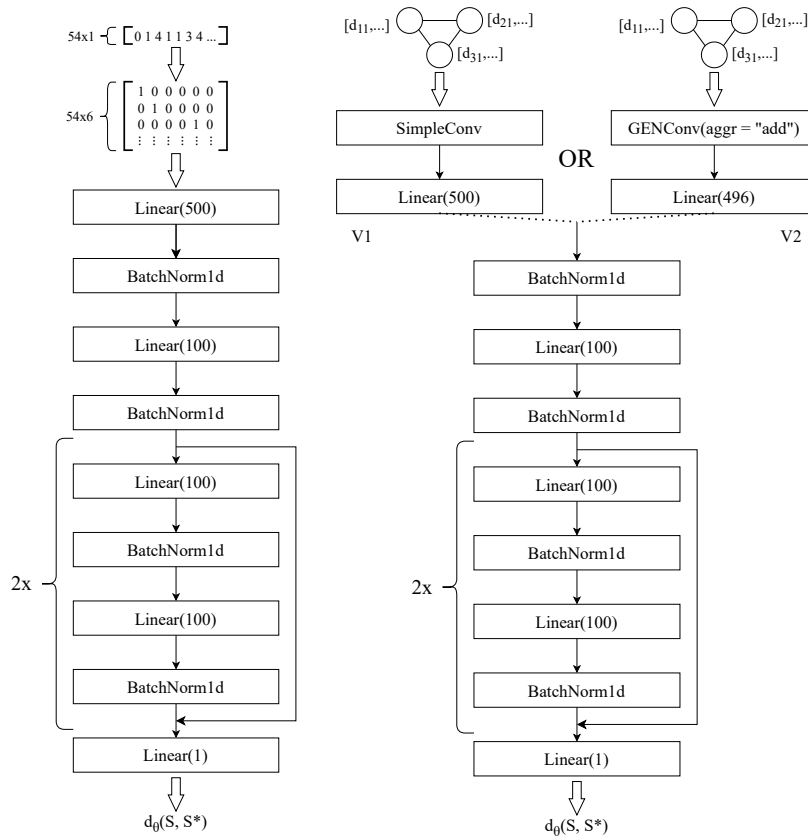
Metric	DeepCube	SymmetryNet V1	SymmetryNet V2
Accuracy	0.5157 $\pm$ 0.0119	0.7710 $\pm$ 0.0415	<b>0.8191 <math>\pm</math> 0.0381</b>
Solved <sup>gr</sup>	0.0489 $\pm$ 0.0125	0.7487 $\pm$ 0.0606	<b>0.7995 <math>\pm</math> 0.0494</b>
Opt. solved <sup>gr</sup>	0.0484 $\pm$ 0.0127	0.7487 $\pm$ 0.0606	<b>0.7990 <math>\pm</math> 0.0498</b>
Solved <sup>A*</sup>	<b>1.0000 <math>\pm</math> 0.0000</b>	<b>1.0000 <math>\pm</math> 0.0000</b>	<b>1.0000 <math>\pm</math> 0.0000</b>
Opt. solved <sup>A*</sup>	0.9992 $\pm$ 0.0023	<b>1.0000 <math>\pm</math> 0.0001</b>	<b>1.0000 <math>\pm</math> 0.0001</b>
Mean exp. <sup>A*</sup>	512.2757 $\pm$ 170.1668	9.5995 $\pm$ 5.8721	<b>7.6210 <math>\pm</math> 3.3123</b>
Median exp. <sup>A*</sup>	268.5500 $\pm$ 102.3094	6.4000 $\pm$ 3.6932	<b>5.4000 <math>\pm</math> 3.5553</b>

**Table 3.6:** Comparison of the models’ performances in problem solving

### 3.2.6 Implementation Details and Hyperparameters

The conceptual structure of our proposed model design is described in Sec. 3.2.3. In Fig. 3.7, we depict the concrete implementation details.

While the DeepCube architecture assumed a one-hot-encoded representation of the cube’s state (Fig. 2.4) as an input, the SymmetryNet architectures are based on the described graph-based representation followed by the (custom) message-passing



**Figure 3.7:** A detailed depiction of the SymmetryNet V1, SymmetryNet V2, and DeepCube models (left to right) decomposed into individual layers.

procedure (Sec. 2.4.4).<sup>7</sup> From there, however, both the SymmetryNet and DeepCube architectures continue directly as scaled-down versions of the original DeepCube architecture from [2]. Particularly, we used 2 residual blocks instead of 4, with 10 times fewer neurons in total. For implementation, we used the PyTorch [77] and PyTorch Geometric [35] frameworks. All the networks were trained for 100 epochs using the Adam optimizer [54] with a learning rate  $lr = 0.001$ , and a batch size of 1024.

### ■ Considered Graph Convolutional Layers

Apart from the message passing layer (`SimpleConv`) and generalized graph convolutional layer (`GENConv`; [63]), other graph convolutional layers were considered but yielded worse performance. Notably, we tried out the following state-of-the-art graph neural layers: the graph convolutional operator (`GCNConv`; [55]), the principal neighborhood aggregation graph convolutional operator (`PNACConv`; [25]), the second version of the graph attention operator (`GATv2Conv`; [11]), and the graph transformer

<sup>7</sup>Technically, `SimpleConv` corresponds to PyTorch Geometric’s implementation of the WL message-passing without any learnable parameters, and `GENConv` is the PyTorch Geometric’s implementation of the generalized graph convolutional layer from [63]

operator (`TransformerConv`; [90]).

These layers were tried out and eliminated since they consistently exhibited a behavior which made successful training difficult: albeit, when inserted into the architecture described by Fig. 2.6, they formally passed the expressiveness test, the resulting activations had a very low variance between different equivalence classes (regardless of their distance from goal), which increased the difficulty of learning the equivalence-class concepts. The `SimpleConv` and `GENConv` did not pose such problems.

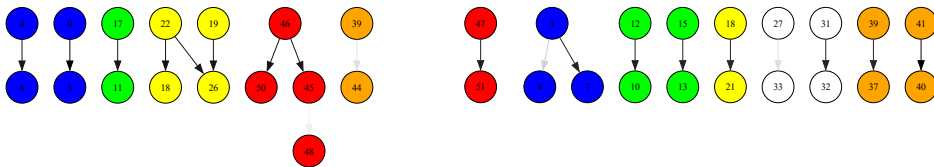
### 3.3 Interpretability of SymmetryNet

Based on our literature review (Sec. 1.2.2), we concluded that none of the planning-specific methods match our task, i.e., none of them explains a graph-based neural network evaluating a planning state. On the other hand, multiple GNN-specific interpretability methods are relevant. We tested two of them, `GNNExplainer` and `PGExplainer`, which are both implemented in `PyTorch Geometric`<sup>8</sup> and applied them on the trained `SymmetryNet`, concretely `SymmetryNet V1`.

As per the taxonomy from Sec. 2.5, we tested both phenomenon- and model-based approaches. Furthermore, we used the non-converted soft mask and tried out different top-k values for the mask transformation threshold strategy before deciding on  $k = 10$ . For both explainers, we produced explanations in the form of edge importance scores/mask matrices. With `GNNExplainer`, it is further possible to calculate node-related scores, concretely feature importance scores. The explanations were computed on sample cube states from the exhaustive dataset, and for each explanation, we produced a visualization of the respective important edges and nodes they are connecting. We report a selected subset of those visualizations.

#### 3.3.1 Distinguishing Different Distances

Comparing explanations for sample cubes of distance from goal  $d = 1$  and  $d = 3$  for different random seeds proved the instability and inconclusiveness of the GNN explainability methods. For illustration, we include visualizations of phenomenon-based explanations of `GNNExplainer` for two cubes of  $d = 1$  in Fig. 3.8<sup>9</sup> and visualizations of two cubes of  $d = 3$  in Fig. 3.9.

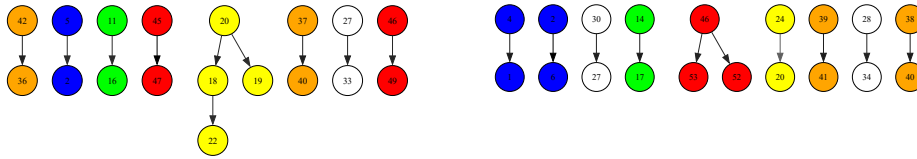


**Figure 3.8:** Explanations of cube states of distance  $d = 1$  produced by `GNNExplainer`.

<sup>8</sup>See <https://pytorch-geometric.readthedocs.io/en/latest/modules/explain.html>

<sup>9</sup>Note that all cubes of distance 1 share the exact same pattern.

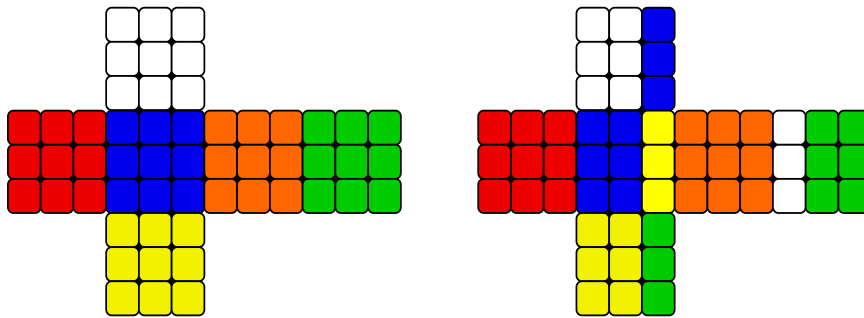




**Figure 3.9:** Explanations of cube states of distance  $d = 3$  produced by GNNExplainer.

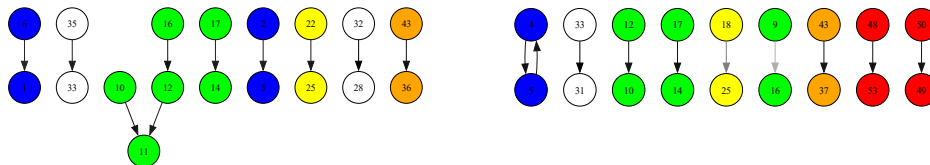
It is clear that there are no discernible differences that would effectively distinguish the data of different distances, even though the trained network can do so. Even more, the graphics show that the explainer sometimes finds bigger differences between cubes from the same distance than between distances 1 and 3. Similar results were obtained with other explainer variants, i.e., with PGExplainer and model-based explanations.

### 3.3.2 Explaining Single Cube Move with GNNExplainer



**Figure 3.10:** Comparison of cube states: a solved one on the left side, and one resulting by clock-wise 90 deg rotation of the right face.

Since the previous explanations were inconclusive, we degress to a simpler setting. We compare two cube states with the minimal difference of a single move, i.e., the solved cube state  $S_1$  and cube state  $S_2$  of distance  $d = 1$ , achieved by rotation of the right face (i.e., applying the generator  $G = R$ ). The 2D projections of both cubes are depicted in Fig. 3.10.



**Figure 3.11:** Two explanations of a cube state with  $d = 1$ : a better explanation on the left side, highlighting mostly colors of facelets twisted by the move, a worse explanation on the right side, mixing both twisted and untwisted facelets.

One would expect that, unlike in the solved state, in the state of distance  $d = 1$ ,

the explanations would highlight either the 4 colors with twisted facelets (blue, green, yellow, and white) or, on the contrary, the two colors with all facelets in their original place (orange and red). We observed that for distance  $d = 1$ , the explainer was able to clearly highlight either orange and red or the remaining 4 colors. The remaining explanations were, however, mixed and not clearly distinguishable from their  $d = 0$  counterparts. For an example of a better and a worse explanation of  $d = 1$  cube state, see Fig. 3.11.

### 3.3.3 Architecture-Based Interpretations

Finally, we offer our own interpretations based on the architecture design as an alternative to the standard GNN explainers which proved to be not so reliable for the task. Starting from the last task of explaining a single cube move, we look at the input representations (equivalent to the first iteration of the Weisfeiler-Lehman color refinement algorithm) and at the hidden representations (equivalent to the second iteration of the WL algorithm) of the graph convolutional layer, and consider two transformations: (i) lexicographic sorting, and (ii) arithmetic mean.

For the first transformation, we get a matrix input and matrix output per each cube state. Then, inspired by the top-k threshold (Sec. 2.5, if we select the first  $k = 10$  rows of the matrix (after sorting) corresponding to the solved state  $S_1$ , we get:

$$R_{in}(S_1) = \begin{bmatrix} 0 & 1 & 1 & 1 & 2 & 2 & 2 & 2 & 2 \\ 0 & 1 & 1 & 1 & 2 & 2 & 2 & 2 & 2 \\ 0 & 1 & 1 & 1 & 2 & 2 & 2 & 2 & 2 \\ 0 & 1 & 1 & 1 & 2 & 2 & 2 & 2 & 2 \\ 0 & 1 & 1 & 1 & 2 & 2 & 2 & 2 & 2 \\ 0 & 1 & 1 & 1 & 2 & 2 & 2 & 2 & 2 \\ 0 & 1 & 1 & 1 & 2 & 2 & 2 & 2 & 2 \\ 0 & 1 & 1 & 1 & 2 & 2 & 2 & 2 & 2 \\ 0 & 1 & 1 & 1 & 2 & 2 & 2 & 2 & 2 \\ 0 & 1 & 1 & 1 & 2 & 2 & 2 & 2 & 2 \end{bmatrix} \quad R_{hid}(S_1) = \begin{bmatrix} 0 & 14.71 & 14.71 & 14.71 & 21.54 & 21.54 & 28.02 & 28.02 & 28.02 \\ 0 & 14.71 & 14.71 & 14.71 & 21.54 & 21.54 & 28.02 & 28.02 & 28.02 \\ 0 & 14.71 & 14.71 & 14.71 & 21.54 & 21.54 & 28.02 & 28.02 & 28.02 \\ 0 & 14.71 & 14.71 & 14.71 & 21.54 & 21.54 & 28.02 & 28.02 & 28.02 \\ 0 & 14.71 & 14.71 & 14.71 & 21.54 & 21.54 & 28.02 & 28.02 & 28.02 \\ 0 & 14.71 & 14.71 & 14.71 & 21.54 & 21.54 & 28.02 & 28.02 & 28.02 \\ 0 & 14.71 & 14.71 & 14.71 & 21.54 & 21.54 & 28.02 & 28.02 & 28.02 \\ 0 & 14.71 & 14.71 & 14.71 & 21.54 & 21.54 & 28.02 & 28.02 & 28.02 \\ 0 & 14.71 & 14.71 & 14.71 & 21.54 & 21.54 & 28.02 & 28.02 & 28.02 \\ 0 & 14.71 & 14.71 & 14.71 & 21.54 & 21.54 & 28.02 & 28.02 & 28.02 \end{bmatrix}$$

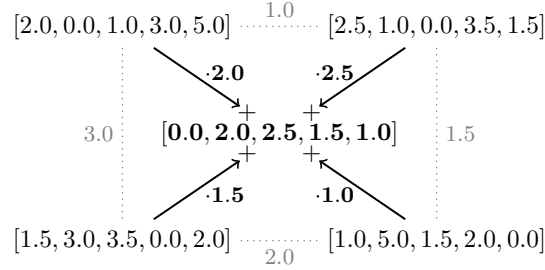
with arithmetic means of  $\overline{R_{in}(S_1)} = 1.29$  and  $\overline{R_{hid}(S_1)} = 17.08$ . While for state  $S_2$  in distance  $d = 1$ , we get:

$$R_{in}(S_2) = \begin{bmatrix} 0 & 1 & 2 & 2 & 3 & 3 & 3 & 3 & 4 \\ 0 & 1 & 2 & 2 & 3 & 3 & 3 & 3 & 4 \\ 0 & 1 & 2 & 2 & 3 & 3 & 3 & 3 & 4 \\ 0 & 1 & 2 & 2 & 3 & 3 & 3 & 3 & 4 \\ 0 & 1 & 1 & 1 & 2 & 2 & 3 & 3 & 4 \\ 0 & 1 & 1 & 1 & 2 & 2 & 3 & 3 & 4 \\ 0 & 1 & 1 & 1 & 2 & 2 & 3 & 3 & 4 \\ 0 & 1 & 1 & 1 & 2 & 2 & 3 & 3 & 4 \\ 0 & 1 & 1 & 1 & 2 & 2 & 3 & 3 & 3 \\ 0 & 1 & 1 & 1 & 2 & 2 & 3 & 3 & 3 \end{bmatrix} \quad R_{hid}(S_2) = \begin{bmatrix} 0 & 23.33 & 23.33 & 23.33 & 37.31 & 37.31 & 47.65 & 55.4 & 71.32 \\ 0 & 23.33 & 23.33 & 23.33 & 37.31 & 37.31 & 47.65 & 55.4 & 71.32 \\ 0 & 23.33 & 23.33 & 23.33 & 37.31 & 37.31 & 47.65 & 55.4 & 71.32 \\ 0 & 23.33 & 23.33 & 23.33 & 37.31 & 37.31 & 47.65 & 55.4 & 71.32 \\ 0 & 18.49 & 22.55 & 22.55 & 36.39 & 36.39 & 39.62 & 45.57 & 57.29 \\ 0 & 18.49 & 22.55 & 22.55 & 36.39 & 36.39 & 39.62 & 45.57 & 57.29 \\ 0 & 18.49 & 22.55 & 22.55 & 36.39 & 36.39 & 39.62 & 45.57 & 57.29 \\ 0 & 18.49 & 22.55 & 22.55 & 36.39 & 36.39 & 39.62 & 45.57 & 57.29 \\ 0 & 18.3 & 19.3 & 19.3 & 30.47 & 30.47 & 38.79 & 44.22 & 57.56 \\ 0 & 18.3 & 19.3 & 19.3 & 30.47 & 30.47 & 38.79 & 44.22 & 57.56 \end{bmatrix}$$

with arithmetic means of  $\overline{R_{in}(S_2)} = 1.50$  and  $\overline{R_{hid}(S_2)} = 24.29$ .

We can see from both the top-k matrix as well as from the arithmetic means of both input and hidden representations that the scalar values in the respective matrices and arithmetic means have increased (except for the first matrix column, which represents the 0 distances of the self-loops). Indeed, that is an expected outcome, as scrambling the cube generally increases the average distances between facelets of the same color. This stable deterministic (heuristic-like) interpretation

method reliably differentiates the two states, as we have designed the message-passing architecture to collect distance measurements between the facelets/nodes of the graph, see Fig. 3.12.



**Figure 3.12:** An illustrative example of the second message passing of distance-vectors, which are weighted by the distances before aggregation.


Similarly, for the task of distinguishing cube states of different distances, the method also succeeds, yielding  $\overline{R_{in}(S_{d_1})} = 1.50$  and  $\overline{R_{hid}(S_{d_1})} = 24.29$  for distance  $d = 1$ , and  $\overline{R_{in}(S_{d_3})} = 1.72 \pm 0.08$  and  $\overline{R_{hid}(S_{d_3})} = 32.77 \pm 2.60$  for distance  $d = 3$ . Note that for distance  $d = 3$ , we also report the standard deviation, as (unlike with  $d = 1$ ) different sets of color patterns with different representations arise for greater distances.<sup>10</sup>

### 3.3.4 Comparison of Interpretability Techniques

We have seen some of the standard explanation methods for GNNs, as well as custom interpretations based on the knowledge of the architecture. It is not surprising that the latter was more successful in distinguishing and reasoning over cube states from different cost to goal. We believe that this holds true for most applications, i.e., if possible, it is preferable to utilize symmetry-related knowledge or other inductive biases during the architecture design, not only to improve the effectiveness but also to boost the interpretability of the whole model.

<sup>10</sup>We present only aggregated explanations; matrices can be generated in the `explainer.ipynb` Jupyter notebook.





## Chapter 4

### Conclusion

In the thesis, we researched symmetry-aware deep learning models for solving planning tasks, mostly based on graph neural networks and the Weisfeiler-Lehman style of feature extraction. We discussed approaches toward the interpretability of such architectures. We have laid out different model-agnostic and model-specific techniques applicable either to neural networks in general, as well as graph neural networks, specifically. We have concluded that our approach, interpretability through built-in symmetry-based inductive biases, is rarely used yet promising. We then followed the discussion with an analysis of multiple classical planning domains and the symmetric properties of their graph representations.

Next, we focused on one specific planning problem, the Rubik’s cube. For the Rubik’s cube problem, we introduced a well-substantiated neural architecture that exploits the symmetries in the Rubik’s cube group, following a thorough analysis of the problem. To efficiently navigate the space of specific model design choices, we also introduced a simple but effective model-symmetry detection procedure that can also be applied universally to all neural models in any discrete domain. In the spirit of geometric deep learning, we then demonstrated superior efficiency of the resulting architecture in both learning generalization and problem-solving over the previously used state-of-the-art neural model. We argued that this process might then serve as a blueprint for (geometric) deep learning researchers aiming to expand to other new intriguing domains and that the resulting architectures are inherently more transparent and interpretable.

Finally, we employed some of the described interpretability methods for graph neural networks on the learned networks and compared their conclusions with our own interpretations, concluding the desirability of inherent interpretability of deep learning architecture, as opposed to mere post-hoc explanations.



#### 4.1 Future Work

One of the promising paths of further research is generalizing the outlined symmetry-based invariant architecture design (stemming from the analysis in Sec. 2.2) to all standard planning tasks. However, there is always a trade-off between the generality of the method and the possible compression that can be achieved by the symmetry-aware architecture – domain-specific knowledge is usually needed to achieve greater compression rates.





## Bibliography

- [1] Chirag Agarwal et al. “Evaluating explainability for graph neural networks”. In: *Scientific Data* 10.1 (2023), p. 144.
- [2] Forest Agostinelli et al. “Solving the Rubik’s cube with deep reinforcement learning and search”. In: *Nature Machine Intelligence* 1.8 (2019), pp. 356–363.
- [3] Paul Almasan et al. “Deep reinforcement learning meets graph neural networks: Exploring a routing optimization use case”. In: *Computer Communications* 196 (2022), pp. 184–194.
- [4] Kenza Amara et al. “Graphframex: Towards systematic evaluation of explainability methods for graph neural networks”. In: *arXiv preprint arXiv:2206.09677* (2022).
- [5] Alejandro Barredo Arrieta et al. “Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI”. In: *Information fusion* 58 (2020), pp. 82–115.
- [6] Masataro Asai and Alex Fukunaga. “Classical planning in deep latent space: Bridging the subsymbolic-symbolic boundary”. In: *Proceedings of the aaai conference on artificial intelligence*. Vol. 32. 1. 2018.
- [7] Masataro Asai et al. “Classical planning in deep latent space”. In: *Journal of Artificial Intelligence Research* 74 (2022), pp. 1599–1686.
- [8] Waiss Azizian and Marc Lelarge. “Expressive power of invariant and equivariant graph neural networks”. In: *arXiv preprint arXiv:2006.15646* (2020).
- [9] Federico Baldassarre and Hossein Azizpour. “Explainability techniques for graph convolutional networks”. In: *arXiv preprint arXiv:1905.13686* (2019).
- [10] Etienne Barnard and David Casasent. “Invariance and neural nets”. In: *IEEE Transactions on neural networks* 2.5 (1991), pp. 498–508.
- [11] Shaked Brody, Uri Alon, and Eran Yahav. “How attentive are graph attention networks?” In: *arXiv preprint arXiv:2105.14491* (2021).
- [12] Michael M Bronstein et al. “Geometric deep learning: going beyond euclidean data”. In: *IEEE Signal Processing Magazine* 34.4 (2017), pp. 18–42.
- [13] Michael M Bronstein et al. “Geometric deep learning: Grids, groups, graphs, geodesics, and gauges”. In: *arXiv preprint arXiv:2104.13478* (2021).

- [14] Robert Brunetto and Otakar Trunda. “Deep Heuristic-learning in the Rubik’s Cube Domain: An Experimental Evaluation.” In: *ITAT*. 2017, pp. 57–64.
- [15] Dillon Z Chen, Sylvie Thiébaux, and Felipe Trevizan. “Learning Domain-Independent Heuristics for Grounded and Lifted Planning”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 38. 18. 2024, pp. 20078–20086.
- [16] Dillon Z Chen, Felipe Trevizan, and Sylvie Thiébaux. “Return to Tradition: Learning Reliable Heuristics with Classical Machine Learning”. In: *arXiv preprint arXiv:2403.16508* (2024).
- [17] Dillon Ze Chen, Sylvie Thiébaux, and Felipe Trevizan. “GOOSE: Learning domain-independent heuristics”. In: *NeurIPS 2023 Workshop on Generalization in Planning*. 2023.
- [18] Dillon Ze Chen, Felipe Trevizan, and Sylvie Thiébaux. “Graph Neural Networks and Graph Kernels For Learning Heuristics: Is there a difference?” In: *NeurIPS 2023 Workshop on Generalization in Planning*. 2023.
- [19] Zhengdao Chen et al. “On the equivalence between graph isomorphism testing and function approximation with gnns”. In: *Advances in neural information processing systems* 32 (2019).
- [20] Taco Cohen and Max Welling. “Group equivariant convolutional networks”. In: *International conference on machine learning*. PMLR. 2016, pp. 2990–2999.
- [21] Arthur Conmy et al. “Towards automated circuit discovery for mechanistic interpretability”. In: *Advances in Neural Information Processing Systems* 36 (2023), pp. 16318–16352.
- [22] Saksham Consul et al. “Improving human decision-making by discovering efficient strategies for hierarchical planning”. In: *Computational Brain & Behavior* 5.2 (2022), pp. 185–216.
- [23] Sebastiano Corli et al. “Casting Rubik’s Group into a Unitary Representation for Reinforcement Learning”. In: vol. 2533. IOP Publishing, 2023, p. 012006.
- [24] Sebastiano Corli et al. “Solving Rubik’s cube via quantum mechanics and deep reinforcement learning”. In: *Journal of Physics A: Mathematical and Theoretical* 54.42 (2021), p. 425302.
- [25] Gabriele Corso et al. “Principal neighbourhood aggregation for graph nets”. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 13260–13271.
- [26] Mark William Craven. *Extracting comprehensible models from trained neural networks*. The University of Wisconsin-Madison, 1996.
- [27] Joseph C Culberson and Jonathan Schaeffer. “Pattern databases”. In: *Computational Intelligence* 14.3 (1998), pp. 318–334.
- [28] George Cybenko. “Approximation by superpositions of a sigmoidal function”. In: *Mathematics of control, signals and systems* 2 (1989), pp. 303–314.



- [29] Erik D Demaine, Sarah Eisenstat, and Mikhail Rudoy. “Solving the Rubik’s Cube Optimally is NP-complete”. In: *arXiv preprint arXiv:1706.06708* (2017).
- [30] Edsger W Dijkstra. “A note on two problems in connexion with graphs”. In: *Numerische mathematik* 1.1 (1959), pp. 269–271.
- [31] Danny Driess et al. “Deep visual heuristics: Learning feasibility of mixed-integer programs for manipulation planning”. In: *2020 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2020, pp. 9563–9569.
- [32] Feng-Lei Fan et al. “On interpretability of artificial neural networks: A survey”. In: *IEEE Transactions on Radiation and Plasma Medical Sciences* 5.6 (2021), pp. 741–760.
- [33] Patrick Ferber, Malte Helmert, and Jörg Hoffmann. “Neural network heuristics for classical planning: A study of hyperparameter space”. In: *ECAI 2020*. IOS Press, 2020, pp. 2346–2353.
- [34] Stefano Ferraro et al. “Symmetry and complexity in object-centric deep active inference models”. In: *Interface Focus* 13.3 (2023), p. 20220077.
- [35] Matthias Fey and Jan Eric Lenssen. “Fast graph representation learning with PyTorch Geometric”. In: *arXiv preprint arXiv:1903.02428* (2019).
- [36] Fabian Fuchs et al. “Se (3)-transformers: 3d roto-translation equivariant attention networks”. In: *Advances in neural information processing systems* 33 (2020), pp. 1970–1981.
- [37] Robert Gens and Pedro M Domingos. “Deep symmetry networks”. In: *Advances in neural information processing systems* 27 (2014).
- [38] Claire Glanois et al. “A survey on interpretable reinforcement learning”. In: *Machine Learning* (2024), pp. 1–44.
- [39] Edward Groshev et al. “Learning generalized reactive policies using deep neural networks”. In: *Proceedings of the International Conference on Automated Planning and Scheduling*. Vol. 28. 2018, pp. 408–416.
- [40] Peter E Hart, Nils J Nilsson, and Bertram Raphael. “A formal basis for the heuristic determination of minimum cost paths”. In: *IEEE transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107.
- [41] Kaiming He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.
- [42] D Hoey. *The real size of cube space*. [http://www.math.rwth-aachen.de/~Martin.Schoenert/Cube-Lovers/Dan\\_Hoey\\_The\\_real\\_size\\_of\\_cube\\_space.html](http://www.math.rwth-aachen.de/~Martin.Schoenert/Cube-Lovers/Dan_Hoey_The_real_size_of_cube_space.html). Accessed: 2023-08-09. 1994.
- [43] Rostislav Horcik and Gustav Šír. “Expressiveness of Graph Neural Networks in Planning Domains”. In: *34th International Conference on Automated Planning and Scheduling*. 2024.
- [44] Shell Xu Hu, Sergey Zagoruyko, and Nikos Komodakis. “Exploring weight symmetry in deep neural networks”. In: *Computer Vision and Image Understanding* 187 (2019), p. 102786.

- [45] Xiaowei Hu et al. “Direction-aware spatial context features for shadow detection”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018.
- [46] Sergio Jiménez et al. “A review of machine learning for automated planning”. In: *The Knowledge Engineering Review* 27.4 (2012), pp. 433–467.
- [47] Colin G Johnson. “Solving the Rubik’s cube with stepwise deep learning”. In: *Expert Systems* 38.3 (2021), e12665.
- [48] Colin G Johnson. “Stepwise evolutionary learning using deep learned guidance functions”. In: *International Conference on Innovative Techniques and Applications of Artificial Intelligence*. Springer. 2019, pp. 50–62.
- [49] Chaitanya Joshi. “Transformers are Graph Neural Networks”. In: *The Gradient* (2020).
- [50] Peter Karkus, David Hsu, and Wee Sun Lee. “Qmdp-net: Deep learning for planning under partial observability”. In: *Advances in neural information processing systems* 30 (2017).
- [51] Eoin M Kenny et al. “Post-hoc explanation options for XAI in deep learning: The Insight centre for data analytics perspective”. In: *Pattern Recognition. ICPR International Workshops and Challenges: Virtual Event, January 10–15, 2021, Proceedings, Part III*. Springer. 2021, pp. 20–34.
- [52] Nicolas Keriven and Gabriel Peyré. “Universal invariant and equivariant graph neural networks”. In: *Advances in Neural Information Processing Systems* 32 (2019).
- [53] Piotr Kicki, Piotr Skrzypczyński, and Mete Ozay. “A new approach to design symmetry invariant neural networks”. In: *2021 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2021, pp. 1–8.
- [54] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).
- [55] Thomas N. Kipf and Max 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.
- [56] Herbert Kociemba. *Cube explorer*. <http://kociemba.org/cube.htm>. Accessed: 2023-08-09. 2018.
- [57] Herbert Kociemba. *Two-phase algorithm details*. <http://kociemba.org/math/imptwophase.htm>. Accessed: 2023-08-09. 2014.
- [58] Wolfgang Konen. “Towards Learning Rubik’s Cube with N-tuple-based Reinforcement Learning”. In: *arXiv preprint arXiv:2301.12167* (2023).
- [59] Martin Krutský. “Základy symetrií v hlubokém učení”. B.S. thesis. České vysoké učení technické v Praze. Vypočetní a informační centrum., 2021.
- [60] Dmitry Laptev et al. “Ti-pooling: transformation-invariant pooling for feature learning in convolutional neural networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 289–297.

- [61] Thibault Laugel et al. “The dangers of post-hoc interpretability: Unjustified counterfactual explanations”. In: *arXiv preprint arXiv:1907.09294* (2019).
- [62] Yann LeCun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.
- [63] Guohao Li et al. “Deepergcn: All you need to train deeper gcns”. In: *arXiv preprint arXiv:2006.07739* (2020).
- [64] Zian Li et al. “Is Distance Matrix Enough for Geometric Deep Learning?”. In: *Advances in Neural Information Processing Systems* 36 (2024).
- [65] Sangho Lim, Eun-Gyeol Oh, and Hongseok Yang. “Learning symmetric rules with SATNet”. In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 13251–13262.
- [66] Aaron Yi Rui Low, Subhroshekhar Ghosh, and Yong Sheng Soh. “Conjugation Invariant Learning with Neural Networks”. In: (2021).
- [67] Simon M Lucas. “Learning to play Othello with n-tuple systems”. In: *Australian Journal of Intelligent Information Processing* 4 (2008), pp. 1–20.
- [68] Scott M Lundberg and Su-In Lee. “A unified approach to interpreting model predictions”. In: *Advances in neural information processing systems* 30 (2017).
- [69] Dongsheng Luo et al. “Parameterized explainer for graph neural network”. In: *Advances in neural information processing systems* 33 (2020), pp. 19620–19631.
- [70] Daoming Lyu et al. “Tdm: trustworthy decision-making via interpretability enhancement”. In: *IEEE Transactions on Emerging Topics in Computational Intelligence* 6.3 (2021), pp. 450–461.
- [71] Andreas Madsen, Siva Reddy, and Sarath Chandar. “Post-hoc interpretability for neural nlp: A survey”. In: *ACM Computing Surveys* 55.8 (2022), pp. 1–42.
- [72] Stephen McAleer et al. “Solving the rubik’s cube with approximate policy iteration”. In: *International Conference on Learning Representations*. 2018.
- [73] Peter Meltzer, Marcelo Daniel Gutierrez Mallea, and Peter J Bentley. “Pinet: A permutation invariant graph neural network for graph classification”. In: *arXiv preprint arXiv:1905.03046* (2019).
- [74] Bharath Muppasani et al. “On Solving the Rubik’s Cube with Domain-Independent Planners Using Standard Representations”. In: *arXiv preprint arXiv:2307.13552* (2023).
- [75] Neel Nanda et al. “Progress measures for grokking via mechanistic interpretability”. In: *arXiv preprint arXiv:2301.05217* (2023).
- [76] Xingang Pan et al. “Spatial as deep: Spatial cnn for traffic scene understanding”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. 2018.

- [77] Adam Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035. URL: <http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf>.
- [78] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. “" Why should i trust you?" Explaining the predictions of any classifier”. In: *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 2016, pp. 1135–1144.
- [79] Or Rivlin, Tamir Hazan, and Erez Karpas. “Generalized planning with deep reinforcement learning”. In: *arXiv preprint arXiv:2005.02305* (2020).
- [80] Tomas Rokicki. “Solving All 164,604,041,664 Symmetric Positions of the Rubik’s Cube in the Quarter Turn Metric”. In: *Gathering 4 Gardner* (2014).
- [81] Tomas Rokicki. “Towards God’s number for Rubik’s cube in the quarter-turn metric”. In: *The College Mathematics Journal* 45.4 (2014).
- [82] Tomas Rokicki et al. “The diameter of the rubik’s cube group is twenty”. In: *siam REVIEW* 56.4 (2014), pp. 645–670.
- [83] Henry A Rowley, Shumeet Baluja, and Takeo Kanade. “Rotation invariant neural network-based face detection”. In: *Proceedings. 1998 IEEE computer society conference on computer vision and pattern recognition (Cat. No. 98CB36231)*. IEEE. 1998, pp. 38–44.
- [84] Waddah Saeed and Christian Omlin. “Explainable AI (XAI): A systematic meta-survey of current challenges and future opportunities”. In: *Knowledge-Based Systems* 263 (2023), p. 110273.
- [85] Rabia Saleem et al. “Explaining deep neural networks: A survey on the global interpretation methods”. In: *Neurocomput.* 513.C (2022), pp. 165–180. ISSN: 0925-2312. DOI: 10.1016/j.neucom.2022.09.129. URL: <https://doi.org/10.1016/j.neucom.2022.09.129>.
- [86] Michael Sejr Schlichtkrull, Nicola De Cao, and Ivan Titov. “Interpreting graph neural networks for NLP with differentiable edge masking”. In: *arXiv preprint arXiv:2010.00577* (2020).
- [87] Thomas Schnake et al. “Higher-order explanations of graph neural networks via relevant walks”. In: *IEEE transactions on pattern analysis and machine intelligence* 44.11 (2021), pp. 7581–7596.
- [88] William Shen, Felipe Trevizan, and Sylvie Thiébaux. “Learning domain-independent planning heuristics with hypergraph networks”. In: *Proceedings of the International Conference on Automated Planning and Scheduling*. Vol. 30. 2020, pp. 574–584.
- [89] Nino Shervashidze et al. “Weisfeiler-lehman graph kernels.” In: *Journal of Machine Learning Research* 12.9 (2011).
- [90] Yunsheng Shi et al. “Masked label prediction: Unified message passing model for semi-supervised classification”. In: *arXiv preprint arXiv:2009.03509* (2020).

- [91] Avanti Shrikumar et al. “Not just a black box: Learning important features through propagating activation differences”. In: *arXiv preprint arXiv:1605.01713* (2016).
- [92] Tom Silver et al. “Planning with learned object importance in large problem instances using graph neural networks”. In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 35. 13. 2021, pp. 11962–11971.
- [93] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. “Deep inside convolutional networks: Visualising image classification models and saliency maps”. In: *arXiv preprint arXiv:1312.6034* (2013).
- [94] David Singmaster. *Notes on Rubik’s magic cube*. Enslow Pub Incorporated, 1981.
- [95] Julian Skirzyński, Frederic Becker, and Falk Lieder. “Automatic discovery of interpretable planning strategies”. In: *Machine Learning* 110 (2021), pp. 2641–2683.
- [96] Gustav Sourek, Filip Zelezny, and Ondrej Kuzelka. “Lossless Compression of Structured Convolutional Models via Lifting”. In: *International Conference on Learning Representations*. 2020.
- [97] Gustav Šourek, Filip Železný, and Ondřej Kuželka. “Beyond graph neural networks with lifted relational neural networks”. In: *Machine Learning* 110.7 (2021), pp. 1695–1738.
- [98] Gustav Šourek et al. “Lifted relational neural networks: Efficient learning of latent relational structures”. In: *Journal of Artificial Intelligence Research* 62 (2018), pp. 69–100.
- [99] Simon Ståhlberg, Blai Bonet, and Hector Geffner. “Learning general optimal policies with graph neural networks: Expressive power, transparency, and limits”. In: *Proceedings of the International Conference on Automated Planning and Scheduling*. Vol. 32. 2022, pp. 629–637.
- [100] Timothy Sun. “Commutators in the Rubik’s Cube Group”. In: *The American Mathematical Monthly* 131.1 (2024), pp. 3–19.
- [101] Shobhita Sundaram et al. “Recurrent connections facilitate symmetry perception in deep networks”. In: *Scientific Reports* 12.1 (2022), p. 20931.
- [102] Kai Sheng Tai, Peter Bailis, and Gregory Valiant. “Equivariant transformer networks”. In: *International Conference on Machine Learning*. PMLR. 2019, pp. 6086–6095.
- [103] Kyo Takano. “Self-Supervision is All You Need for Solving Rubik’s Cube”. In: *arXiv preprint arXiv:2106.03157* (2021).
- [104] Otakar Trunda and Roman Barták. “Deep Learning of Heuristics for Domain-independent Planning.” In: *ICAART (2)*. 2020, pp. 79–88.
- [105] Stratis Tsirtsis, Abir De, and Manuel Rodriguez. “Counterfactual explanations in sequential decision making under uncertainty”. In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 30127–30139.



## Appendix A

### Other Cube Pattern Representations

We mention other cube pattern representations that we experimented with but did not make it to the final metric computation and comparison.

#### A.1 Eliminated Pattern Representation

Apart from the six mentioned ablated invariants, other color pattern invariants were considered and eliminated after partial results from the symmetry-detection routine. The most notable of them is the surface-distance invariant.

##### A.1.1 Surface-Distance Invariant

Similarly to distance invariant (ii) described in Sec. 2.4.4, the surface-distance invariant has used distances from the middle facelet. However, the choice of distance metric was different: the shortest Manhattanian path on the cube’s surface. The rationale of this metric was to approximately mimic a heuristic of the goal distance function that is given by the shortest surface path  $P_{surf}^*$  of facelets from their goal destination,  $f = \frac{\ell(P_{surf}^*)}{c}$ , where  $c \in \mathbb{R}_+$  is a constant accounting for a different scale of the distance compared to move metrics and simultaneous movement of multiple facelets in one move.

---

**Algorithm 4** Surface-Distance Algorithm

---

Given indices of a color pattern facelets:

1. Prepare 6 2D cube projections, each with a different center face and different orientation.
  2. For each pattern facelet, calculate distances to the middle facelet on all 6 2D projections.
  3. The *surface distance* is the minimum of the 6 distances.
- 

The calculation of the heuristic is, however, challenging. We devised a solution described in Alg. 4, in which the first step can be amortized, and the second step can be cached.



## ■ A.2 CNN-based Global Invariant Ablation

Besides the proposed decomposition of the cube into the color pattern representations, one might attempt to approach the problem more directly with existing (geometric) deep learning architectures. Indeed, given the regular structure of the problem, using convolutional neural networks (CNNs) with 2D or 3D filters seems rather natural. For the purpose, we designed custom 2D convolutions over the flattened *cube cross* representations (Fig. 2.3), capturing the characteristic (surface) structure of the cube (Sec. 2.3.1). Following the requirement of color invariance (Sec. 2.4.4), we utilized the same convolutional filter across each color dimension in the one-hot-encoded state representation (Fig. 2.4), using a kernel size of  $3 \times 3$ .<sup>1</sup>

Firstly, we tested a simple convolution with a stride of 3, essentially multiplying each face by the filter. This resulted in an embedding vector for each cube’s face in each color. These embeddings were then summed over the color dimension to force color invariance, resulting in a single embedding vector for each face. These were further concatenated and inputted into a feedforward layer. Naturally, though, this representation is not invariant w.r.t. rotations and reflections. To address that, we further computed face embeddings for all the rotations and reflections of each face, and summed over the respective dimension, too. This follows the spirit of previous (custom) neural architectures addressing the problem of directionality of patterns with either convolutional or recurrent networks, such as ReNet [107], spatial RNNs [45] and CNNs [76], as well as the CNNs with transformation-invariant pooling [60]. Nevertheless, both our architectures remained dependent on the cube’s orientation, introduced by the (ad-hoc) ordering of face embeddings during the concatenation. The requirement of preserving the structural face-neighborhood relations while aggregating them in an order-invariant manner then led to the concept of “face cross” CNN filters.

### ■ A.2.1 Face Crosses

While regular CNNs assume a single top-down, left-right sliding of the filter, the structure of the cube (Fig. 2.3) states suggests the need for a custom motion in multiple directions. Moreover, multiple cross-views from different cube orientations should be considered to account for all the grid neighborhoods, for which we designed a custom CNN-based model producing embeddings for all the cube crosses centered in each of the 6 faces containing its 4 neighboring faces. This meant 6 crosses, each with 4 possible orientations, hence 24 embedding vectors in total. We tested several ablations of this CNN-based design, such as directly convolving the filters over the one-hot-encoded representation of the cross, or firstly reproducing the cross from the computed face-color embedding, followed by convolving over the cross. None of these ablations, however, achieved sufficient expressiveness/compression of the target symmetry-equivalence classes, proving the CNN approach inappropriate, in contrast to the new, more thoroughly substantiated, concept of the color patterns, described in the main body of the paper.

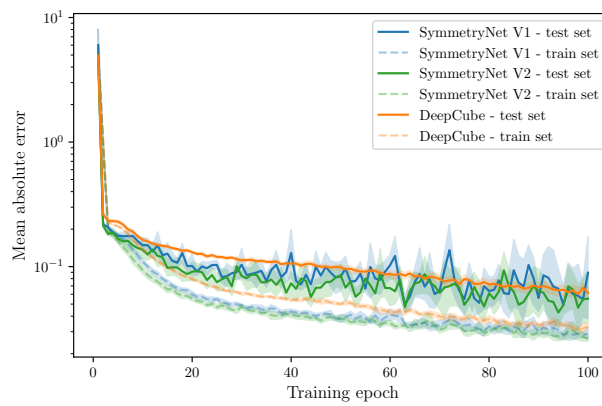
<sup>1</sup>CNNs require rectangular inputs. Thus, we convolved along 3 possible major beams (of size  $3 \times 9$ ) of the cube’s cross.



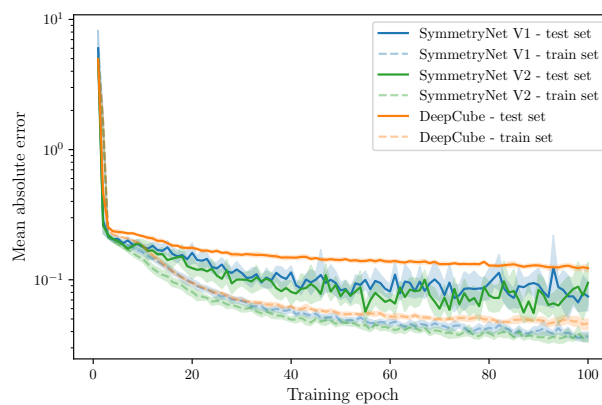
## Appendix B

### Learning Performance Plots

Below, we present further learning performance plots from the Rubik's cube problem. Unlike in Sec. 3.2.3, here we also include the metric on the training set.

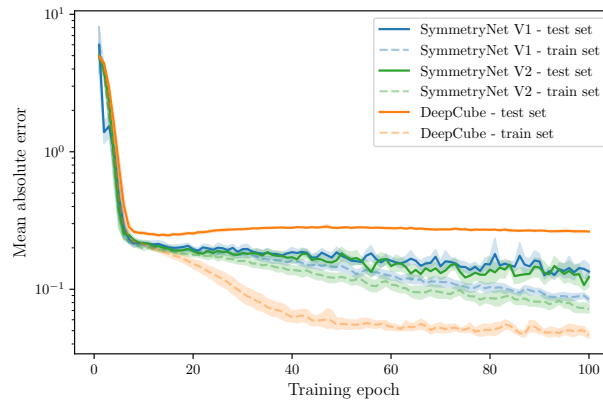


**Figure B.1:** Models' learning convergences when generalizing from 90% of the exhaustive dataset.

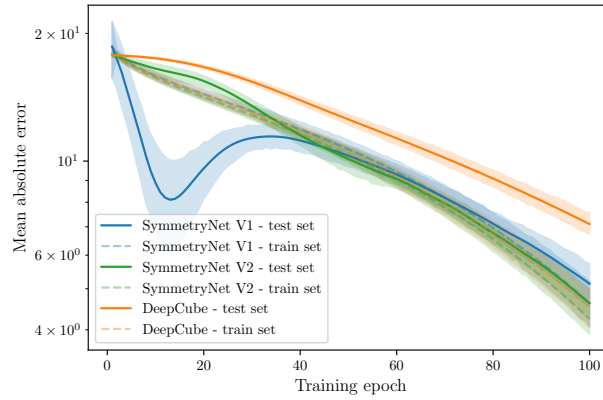


**Figure B.2:** Models' learning convergences when generalizing from 50% of the exhaustive dataset.

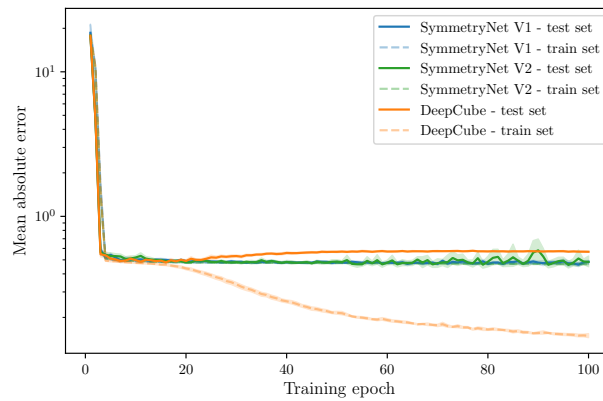
B. Learning Performance Plots



**Figure B.3:** Models' learning convergences when generalizing from 10% of the exhaustive dataset.



**Figure B.4:** Models' learning convergences on the  $10^3$ -sized subset of the sampled dataset.



**Figure B.5:** Models' learning convergences on the full  $10^5$  sampled dataset.

## Appendix C

### Source Code and Resources

The code used for the Rubik's cube experiments can be found at <https://github.com/martin-krutsky/rubik-dl-symmetries>. Refer to the Jupyter notebooks at the root level for instructions on how to run the data generation, training, and evaluation.

The project is structured in the following way:

- cube data structures are implemented in folder `classes/`
- data generation and symmetry-equivalence compression is implemented in folder `generate/`, Python script `utils/compressions.py`, and Jupyter notebooks `generate.ipynb` and `weisfeiler-lehman_compressions.ipynb`
- PyTorch models and training runners are defined in `pytorch_classes/`
- all scripts run on the cluster are in folder `scripts/`
- analysis of the training and search results can be found in `analyzeResults.ipynb`, `summarize_accuracies.ipynb`, and `summarize_search.ipynb` Jupyter notebooks
- explainability experiments can be found in Jupyter notebook `explainer.ipynb`

Some additional generated plots can be found separately at Google Drive: <https://drive.google.com/drive/u/1/folders/14ez10zEoX2d5CWgkDuFBUCjBHScYstKC>.

#### C.1 Acknowledgments

We acknowledge the support from the European Union's Horizon Europe Research and Innovation program under the grant agreement TUPLES No 101070149.