Distribution-Based Invariant Deep Networks for Learning Meta-Features

Gwendoline De Bie[1] **Herilalaina Rakotoarison**[2] **Gabriel Peyré**[1] **Michele Sebag**[2]

[1]ENS, PSL University, Paris, France
[2]TAU, LISN-CNRS–INRIA, Université Paris-Saclay, Orsay, France

Abstract Recent advances in deep learning from probability distributions successfully achieve classification or regression from distribution samples, thus invariant under permutation of the samples. The first contribution of the paper is the Dida distributional architecture, extending the state of the art to achieve invariance under permutation of the features, too. The Dida properties of universal approximation, and robustness with respect to bounded transformations of the input distribution, are established. The second contribution is to empirically demonstrate the merits of the Dida architecture on two tasks defined at the dataset level. The first task consists of predicting whether any two dataset patches are extracted from the same initial dataset. The second task consists of predicting whether a hyper-parameter configuration dominates another configuration, in terms of the learning performance of a fixed learning algorithm on a dataset extracted from the OpenML benchmarking suite. On both tasks, Dida outperforms the state of the art as well as models based on hand-crafted meta-features. The penultimate layer neurons can thus be viewed as *learned* meta-features, defining an accurate and computationally affordable description of datasets.

1 Introduction

Deep networks architectures, initially devised for structured data such as images and speech, have been extended to enforce some invariance or equivariance properties (Shawe-Taylor, 1993) for more complex data representations.[1] The merit of invariant or equivariant neural architectures is twofold. On the one hand, they inherit the universal approximation properties of neural nets (Cybenko, 1989; Leshno et al., 1993). On the other hand, the fact that these architectures comply with the invariances attached to the considered data representation yields more robust and more general models (through constraining the neural weights and/or reducing the number of weights, as examplified by convolutional networks). For instance, when considering point clouds (Qi et al., 2017) or probability distributions (Bie et al., 2019), the network output is required to be invariant with respect to permutations of the input points.

Related works. Invariance or equivariance properties are relevant to a wide range of applications. In the sequence-to-sequence framework, one might want to relax the sequence order (Vinyals et al., 2016). When modelling dynamic cell processes, one might want to follow the cell evolution at a macroscopic level, in terms of distributions as opposed to, a set of individual cell trajectories (Hashimoto et al., 2016). In computer vision, one might want to handle a set of pixels, as opposed to a voxellized representation, for the sake of a better scalability in terms of data dimensionality and computational resources (Bie et al., 2019).

On the theoretical side, neural architectures enforcing invariance or equivariance properties have been pioneered by (Hartford et al., 2018). Characterizations of invariance or equivariance under group actions have been proposed in the finite (Ravanbakhsh et al., 2017) or infinite case

[1]Function $f : X \mapsto Y$ is said to be invariant under operator σ defined on domain X iff $f(\sigma(x)) = f(x)$ for all x in X. Function $f : X \mapsto X$ is said to be equivariant iff $f(\sigma(x)) = \sigma(f(x))$ for all x in X.

(Kondor and Trivedi, 2018). (Maron et al., 2018; Keriven and Peyré, 2019) have proposed a general characterization of linear layers enforcing invariance or equivariance properties with respect to the whole permutation group on the feature set. The universal approximation properties of such architectures have been established in the case of sets (Zaheer et al., 2017), point clouds (Qi et al., 2017), discrete measures (Bie et al., 2019), invariant (Maron et al., 2019) and equivariant (Keriven and Peyré, 2019) graph neural networks. (Maron et al., 2020) presents a neural architecture invariant w.r.t. the ordering of points and their features, handling point clouds.

Motivations. This paper aims to build representations of datasets through *learned meta-features*. Meta-features, meant to represent a dataset as a vector of characteristics, have been mentioned in the ML literature for over 40 years, in relation with several key ML challenges: a) learning a performance model, predicting *a priori* the performance of an algorithm (and the hyper-parameters thereof) on a dataset (Rice, 1976; Hutter et al., 2019); b) learning a generic model able of quick adaptation to new tasks, e.g. one-shot or few-shot learning, through the so-called meta-learning approach (Finn et al., 2017; Baz et al., 2021); c) hyper-parameter transfer learning (Perrone et al., 2018).

A large number of meta-features have been manually designed along the years (Muñoz et al., 2018; Rivolli et al., 2022), ranging from sufficient statistics to the so-called *landmarks* (Pfahringer et al., 2000), computing the performance of (fast) ML algorithms on the considered dataset. The challenge is the following: on the one hand meta-features should capture the joint distribution underlying the dataset in order to help tackling tasks a), b) and c); on the other hand, the meta-features should be sufficiently fast to compute to make sense using them (compared to tackling the above tasks using brute force). How to *learn meta-features* has been first investigated by (Jomaa et al., 2021) to our best knowledge. The authors build the Dataset2Vec representation by tackling a supervised learning problem at the dataset level: specifically, given two dataset patches, that is, two subsets of examples, described by two (different) subsets of features, Dataset2Vec is trained to predict whether those patches are extracted from the same initial dataset. In the same line of approach, Meskhi et al. (2021) have proposed to predict algorithm performances, then consider as meta-features the representations extracted at the last hidden layer.

Contributions. In order to learn meta-features, this paper proposes a new *distribution-based invariant deep architecture* (Dida), which is independent of the dimension d of the distribution support. The merits of the Dida architecture are experimentally demonstrated on two tasks defined at the dataset level, significantly outperforming state of art architectures (Maron et al., 2020; Jomaa et al., 2021; Muñoz et al., 2018) on these tasks (Section 3).

The novelty of the approach is to handle continuous and discrete probability distributions on $\mathbb{R}^d$, extending state of art approaches dealing with point clouds (Maron et al., 2020; Jomaa et al., 2021). This extension yields more general approximation results (Appendix **??**).

Notations.. $[\![1;m]\!]$ denotes the set of integers $\{1,\dots m\}$. Distributions, including discrete distributions (datasets) are noted in bold font. Vectors are noted in italic, with $x[k]$ denoting the k-th coordinate of vector x.

2 Distribution-Based Invariant Networks for Meta-Feature Learning

This section describes the core of the proposed Dida architecture, specifically the mechanism of mapping a point distribution onto another one subject to sample and feature permutation invariance, referred to as *invariant layer*. The Lipchitzness and universal approximation properties of Dida , which guarantee both its robustness and expressiveness, are discussed in Appendix **??**.

2.1 Distribution-Based Invariant Layers

The building block of the proposed architecture, the invariant layer meant to satisfy the feature and label invariance requirements, is defined as follows, taking inspiration from Bie et al. (2019).

Definition 1. *(Distribution-based invariant layers) Let an interaction functional $\varphi : \mathbb{R}^d \times \mathbb{R}^d \longrightarrow \mathbb{R}^r$ be G-invariant:*

$$\forall \sigma, z_1, z_2 \in G \times \mathbb{R}^d \times \mathbb{R}^d, \quad \varphi(z_1, z_2) = \varphi(\sigma(z_1), \sigma(z_2)).$$

The distribution-based invariant layer f_φ is defined as $f_\varphi : \mathbf{z} = (z_i)_{i \in [\![1;n]\!]} \in Z(\mathbb{R}^d) \mapsto f_\varphi(\mathbf{z}) \in Z(\mathbb{R}^r)$ with

$$f_\varphi(\mathbf{z}) \overset{\text{def.}}{=} \left(\frac{1}{n} \sum_{j=1}^{n} \varphi(z_1, z_j), \ldots, \frac{1}{n} \sum_{j=1}^{n} \varphi(z_n, z_j) \right) \tag{1}$$

By construction, f_φ is G-invariant if φ is G-invariant. The construction of f_φ is extended to the general case of possibly continuous probability distributions by replacing sums with integrals (Appendix ??).

It is important that f_φ invariant layers (in particular the first layer of the neural architecture) can handle datasets of arbitrary number of features d_X and number of multi-labels d_Y. An original approach is to define φ as follows. Let $z = (x, y)$ and $z' = (x', y')$ be two samples in $\mathbb{R}^{d_X} \times \mathbb{R}^{d_Y}$. Considering two functions (to be learned) $u : \mathbb{R}^4 \mapsto \mathbb{R}^t$ and $v : \mathbb{R}^t \mapsto \mathbb{R}^r$, then φ is obtained by applying v on the sum of $u(x[k], x'[k], y[\ell], y'[\ell])$ for k ranging in $[\![1; d_X]\!]$ and ℓ in $[\![1; d_Y]\!]$:

$$\varphi(z, z') = v \left(\sum_{k=1}^{d_X} \sum_{\ell=1}^{d_Y} u(x[k], x'[k], y[\ell], y'[\ell]) \right) \tag{2}$$

By construction φ is invariant to both feature and label permutations; this invariance property is instrumental to a good empirical performance (Section 3). Note that (after learning u and v, implementation details in Appendix ??) f_φ can map a n-size dataset $\mathbf{z}$ onto an n-size $f_\varphi(\mathbf{z})$ dataset for any arbitrary n. The overall complexity of f_φ is thus $\mathcal{O}(n^2.d_X.d_Y)$.

As said, f_φ is based on interaction functionals $\varphi(z_i, z_j)$. This original architecture is rooted in theoretical and algorithmic motivations. On the one hand, interaction functionals are crucial components to reach universal approximation results (see Appendix ??, Theorem ??). On the other hand, the use of local interactions allows to create more expressive architectures; the benefit of these architectures is illustrated in the experiments (Section 3).

2.2 Learning from distributions

Dida distributional neural architecture, defined on point distributions, maps a multi-labelled dataset $\mathbf{z} \in \mathrm{Z}(\mathbb{R}^d)$ onto a real-valued vector noted $\mathcal{F}_\zeta(\mathbf{z})$, with

$$\mathcal{F}_\zeta(\mathbf{z}) \overset{\text{def.}}{=} f_{\varphi_m} \circ \ldots \circ f_{\varphi_{o+1}} \circ f_{\varphi_o} \circ \ldots \circ f_{\varphi_1}(\mathbf{z}) \in \mathbb{R}^{d_{m+1}} \tag{3}$$

where ζ are the trainable parameters of the architecture (below). This architecture inherits from Lipschitzness of the interaction functional φ, which guarantees its robustness to input perturbation, as well as universal approximation abilities denoting its expressiveness. Both results are detailed in the general multi-labelled case, in Appendix ??. For simplicity, only the single label case ($d_Y = 1$) is considered in the following.

The first invariant layer is defined from φ_1, mapping pairs of vectors in $\mathbb{R}^d$ ($d_1 = d$) onto $\mathbb{R}^{d_2}$; it is possibly followed by other invariant layers (the impact of using 1 *vs* 2 invariant layers is experimentally studied in Section 3). The last o-th invariant layer is followed by a first non-invariant one, defined from some φ_{o+1} only depending on its second argument; it is possibly followed by other standard layers. The functions defined from the neural nodes on the penultimate layer are referred to as meta-features.

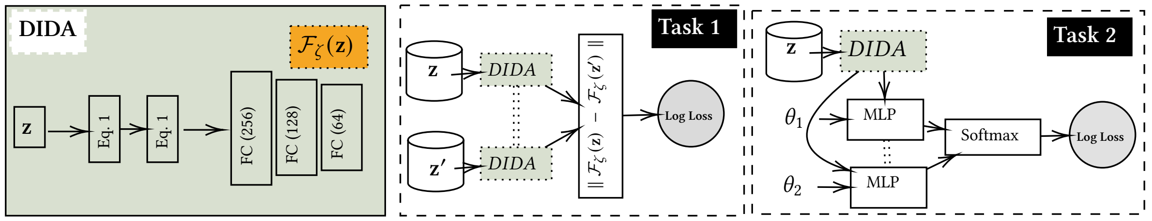

Figure 1: (Left) The Dida architecture (FC for fully connected layer). (Middle) Task 1: Learning meta-features for patch identification using a Siamese architecture (Appendix ??). (Right) Task 2: learning meta-features for ranking hyper-parameter configurations θ_1 and θ_2 (Appendix ??).

3 Experimental Validation

All source codes (Dida and baselines) are publicly available at https://github.com/herilalaina/dida-metafeatures for the sake of reproducibility.

3.1 Experimental setting

Two evaluation tasks are considered:
$\longrightarrow$ **Task 1** is a patch identification problem inspired from (Jomaa et al., 2021) aiming to identify if two dataset patches are extracted from a same dataset.
$\longrightarrow$ **Task 2** aims to rank hyper-parameter configurations for a fixed supervised learning algorithm, according to their performance on the considered dataset.

Dida is compared to three baselines (detailed in Appendix ??): three DSS (Maron et al., 2020) variants (linear invariant layers, non-linear invariant layers, and equivariant + invariant layers); Dataset2Vec (Jomaa et al., 2021); and a function of 43 hand-crafted meta-features.

Three benchmarks are used: TOY and UCI, taken from (Jomaa et al., 2021), and OpenML CC-18 (Bischl et al., 2019). The selection and selection of patches are detailed in Appendix ??.

Training setups. The same Dida architectures are used for both tasks, involving 1 or 2 invariant layers followed by 3 fully connected (FC) layers (Figure 1, left). All experiments run on 1 NVIDIA-Tesla-V100-SXM2 GPU with 32GB memory, using Adam optimizer with base learning rate 10^{-3} and batch size 32. For all considered architectures, meta-features $\mathcal{F}_\zeta(\mathbf{z})$ consist of the output of the penultimate layer, with ζ denoting the trained parameters.

3.2 Results

Task 1. Table 1 reports the empirical results on TOY and UCI datasets. On TOY, Dida with 2 invariant layers, referred to as 2L-Dida behaves on a par with Dataset2Vec and DSS. On UCI, the task appears to be more difficult, which is explained from the higher and more diverse number of features in the datasets. The fact that 2L-Dida significantly outperforms all other approaches is explained from the interaction functional structure (Eqs. 1, 2), expected to better grasp contrasts among examples. Dida with 1 invariant layer (1L-Dida) is much behind 2L-Dida; with a significantly lesser number of parameters than 2L-Dida, the 1L-Dida architecture might lack representational power. A fourth baseline, No-FInv-DSS (Zaheer et al., 2017) only differs from DSS as it is *not* feature permutation invariant; this additional baseline is used to assess the impact of this invariance property. The fact that No-FInv-DSS lags behind all DSS variants, all with similar number of parameters, confirms the importance of this invariance property. Note also that No-FInv-DSS is outperformed by 1L-Dida, while the latter involves significantly less parameters.

Task 2. The comparative performances are displayed in Table 2, reporting their ranking accuracy. 2L-Dida (respectively 1L-Dida) significantly outperforms all baseline approaches except in the *Alg* = LR case (resp., in the *Alg* = *k*-NN case). A higher performance gap is observed for the k-NN case, which is explained as this algorithm mostly exploits the local geometry of the examples.

Method	# params	TOY	UCI
Hand-crafted	53,312	77.05 %± 1.63	58.36 %± 2.64
No-FInv-DSS (no inv. in features)	1,297,692	90.49 %± 1.73	64.69 %± 4.89
Dataset2Vec (reported from Jomaa et al. (2021))	-	96.19 %± 0.28	88.20 %± 1.67
Dataset2Vec (our implementation)	257,088	97.90 %± 1.87	77.05 %± 3.49
DSS layers (Linear aggregation)	1,338,684	89.32 %± 1.85	76.23 %± 1.84
DSS layers (Non-linear aggregation)	1,338,684	96.24 %± 2.04	83.97 %± 2.89
DSS layers (Equivariant+invariant)	1,338,692	96.26 %± 1.40	82.94 %± 3.36
Dida (1 invariant layer)	323,028	91.37 %± 1.39	81.03 %± 3.23
Dida (2 invariant layers)	1,389,089	97.20 % ± 0.10	**89.70** % ± **1.89**

Table 1: Comparative performances (average and std of accuracy over 10 runs) on Task 1 of Dida, No-FInv-DSS, Dataset2Vec, DSS and functions of hand-crafted meta-features.

Method	SGD	SVM	LR	k-NN
Hand-crafted	71.18 %± 0.41	75.39 %± 0.29	86.41 %± 0.419	65.44 %± 0.73
Dataset2Vec (our implementation)	74.43 %± 0.90	81.75 %± 1.85	89.18 %± 0.45	72.90 %± 1.13
DSS (Linear aggregation)	73.46 %± 1.44	82.91 %± 0.22	87.93 %± 0.58	70.07 %± 2.82
DSS (Equivariant+Invariant)	73.54 %± 0.26	81.29 %± 1.65	87.65 %± 0.03	68.55 %± 2.84
DSS (Non-linear aggregation)	74.13 %± 1.01	83.38 %± 0.37	87.92 %± 0.27	73.07 %± 0.77
DIDA (1 invariant layer)	77.31 %± 0.16	84.05 %± 0.71	**90.16** %± **0.17**	74.41 %± 0.93
DIDA (2 invariant layers)	**78.41** %± **0.41**	**84.14** %± **0.02**	89.77 %± 0.50	**78.91** %± **0.54**

Table 2: Comparative ranking performances (average and std over 3 runs) of Dida, Dataset2Vec, DSS and functions of hand-crafted meta-features.

4 Conclusion

The contribution of the paper is the Dida architecture, able to learn from discrete and continuous distributions on $\mathbb{R}^d$, invariant w.r.t. feature ordering, agnostic w.r.t. the size and dimension d of the considered distribution sample (with d less than some upper bound D). The merits of Dida are empirically and comparatively demonstrated on two tasks defined at the dataset level. Task 2 in particular constitutes a first step toward performance modelling Rice (1976), as the learned (algorithm-dependent) meta-features support an efficient ranking of the configurations for the current dataset. On the considered tasks, they improve on the considered baselines namely, Dataset2Vec, DSS and meta-features manually defined in the last two decades (Muñoz et al., 2018). Besides, this Dida architecture also enjoys universal approximation and robustness properties.

For further work, an initial perspective is to investigate the relationships between two datasets, and estimate *a priori* the chances of a successful domain adaptation (Alvarez-Melis and Fusi, 2021).

Limitations and Broader Impact Statement. A major limitation of Dida is on handling real datasets which may include missing values, categorical variables and outliers. An another challenge is that learning meta-features for AutoML tasks (e.g. recommending hyper-parameters or initializing optimization algorithms) requires sufficiently many datasets: quite a few of our early attempts failed due to current ML benchmarks being not sufficiently representative.

Dida requires an extensive compute resources (e.g. circa 4 hours on Task 1.TOY) to be effective. Nevertheless, the approach opens key perspective for AutoML in overcoming the need for domain experts, especially, when it comes to describing and comparing datasets.

5 Reproducibility Checklist

1. For all authors...

   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]

   (b) Did you describe the limitations of your work? [Yes]

   (c) Did you discuss any potential negative societal impacts of your work? [Yes]

   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]

2. If you are including theoretical results...

   (a) Did you state the full set of assumptions of all theoretical results? [Yes]

   (b) Did you include complete proofs of all theoretical results? [Yes]

3. If you ran experiments...

   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results, including all requirements (e.g., `requirements.txt` with explicit version), an instructive `README` with installation, and execution commands (either in the supplemental material or as a URL)? [Yes] They are available at https://anonymous.4open.science/r/dida-metafeatures-5FD5/.

   (b) Did you include the raw results of running the given instructions on the given code and data? [Yes] They are summarized in Tables 1 and 2.

   (c) Did you include scripts and commands that can be used to generate the figures and tables in your paper based on the raw results of the code, data, and instructions given? [N/A]

   (d) Did you ensure sufficient code quality such that your code can be safely executed and the code is properly documented? [No]

   (e) Did you specify all the training details (e.g., data splits, pre-processing, search spaces, fixed hyperparameter settings, and how they were chosen)? [Yes] They are described in Appendix **??**.

   (f) Did you ensure that you compared different methods (including your own) exactly on the same benchmarks, including the same datasets, search space, code for training and hyperparameters for that code? [Yes]

   (g) Did you run ablation studies to assess the impact of different components of your approach? [Yes]

   (h) Did you use the same evaluation protocol for the methods being compared? [Yes]

   (i) Did you compare performance over time? [No]

   (j) Did you perform multiple runs of your experiments and report random seeds? [Yes]

   (k) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]

   (l) Did you use tabular or surrogate benchmarks for in-depth evaluations? [N/A]

   (m) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]

(n) Did you report how you tuned hyperparameters, and what time and resources this required (if they were not automatically tuned by your AutoML method, e.g. in a NAS approach; and also hyperparameters of your own method)? [No]Dida and DSS are manually tuned.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets…

   (a) If your work uses existing assets, did you cite the creators? [Yes]

   (b) Did you mention the license of the assets? [No]

   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]

   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]

   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects…

   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

Acknowledgements. The authors have many people to thank!

References

Alvarez-Melis, D. and Fusi, N. (2021). Dataset Dynamics via Gradient Flows in Probability Space. In *Proceedings of the 38th International Conference on Machine Learning*, pages 219–230. PMLR.

Baz, A. E., Guyon, I., Liu, Z., Rijn, J. N. v., Treguer, S., and Vanschoren, J. (2021). Advances in MetaDL: AAAI 2021 Challenge and Workshop. In *AAAI Workshop on Meta-Learning and MetaDL Challenge*, pages 1–16. PMLR.

Bie, G. D., Peyré, G., and Cuturi, M. (2019). Stochastic Deep Networks. In *Proceedings of the 36th International Conference on Machine Learning*, pages 1556–1565. PMLR.

Bischl, B., Casalicchio, G., Feurer, M., Hutter, F., Lang, M., Mantovani, R. G., van Rijn, J. N., and Vanschoren, J. (2019). OpenML Benchmarking Suites. *arXiv:1708.03731 [cs, stat]*. arXiv: 1708.03731.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314.

Finn, C., Abbeel, P., and Levine, S. (2017). Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. *arXiv:1703.03400 [cs]*. arXiv: 1703.03400.

Hartford, J., Graham, D., Leyton-Brown, K., and Ravanbakhsh, S. (2018). Deep Models of Interactions Across Sets. In *Proceedings of the 35th International Conference on Machine Learning*, pages 1909–1918. PMLR.

Hashimoto, T., Gifford, D., and Jaakkola, T. (2016). Learning Population-Level Diffusions with Generative RNNs. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 2417–2426. PMLR.

Hutter, F., Kotthoff, L., and Vanschoren, J., editors (2019). *Automated Machine Learning: Methods, Systems, Challenges*. The Springer Series on Challenges in Machine Learning. Springer International Publishing, Cham.

Jomaa, H. S., Schmidt-Thieme, L., and Grabocka, J. (2021). Dataset2Vec: learning dataset meta-features. *Data Mining and Knowledge Discovery*, 35(3):964–985.

Keriven, N. and Peyré, G. (2019). Universal Invariant and Equivariant Graph Neural Networks. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Kondor, R. and Trivedi, S. (2018). On the Generalization of Equivariance and Convolution in Neural Networks to the Action of Compact Groups. In Dy, J. G. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 2752–2760. PMLR.

Leshno, M., Lin, V. Y., Pinkus, A., and Schocken, S. (1993). Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, 6(6):861–867.

Maron, H., Ben-Hamu, H., Shamir, N., and Lipman, Y. (2018). Invariant and Equivariant Graph Networks.

Maron, H., Fetaya, E., Segol, N., and Lipman, Y. (2019). On the Universality of Invariant Networks. In *Proceedings of the 36th International Conference on Machine Learning*, pages 4363–4371. PMLR.

Maron, H., Litany, O., Chechik, G., and Fetaya, E. (2020). On Learning Sets of Symmetric Elements. In *Proceedings of the 37th International Conference on Machine Learning*, pages 6734–6744. PMLR.

Meskhi, M. M., Rivolli, A., Mantovani, R. G., and Vilalta, R. (2021). Learning Abstract Task Representations. In *AAAI Workshop on Meta-Learning and MetaDL Challenge*, pages 127–137. PMLR.

Muñoz, M. A., Villanova, L., Baatar, D., and Smith-Miles, K. (2018). Instance spaces for machine learning classification. *Machine Learning*, 107(1):109–147.

Perrone, V., Jenatton, R., Seeger, M. W., and Archambeau, C. (2018). Scalable Hyperparameter Transfer Learning. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

Pfahringer, B., Bensusan, H., and Giraud-Carrier, C. G. (2000). Meta-Learning by Landmarking Various Learning Algorithms. In Langley, P., editor, *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000), Stanford University, Stanford, CA, USA, June 29 - July 2, 2000*, pages 743–750. Morgan Kaufmann.

Qi, C. R., Su, H., Mo, K., and Guibas, L. J. (2017). PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. pages 652–660.

Ravanbakhsh, S., Schneider, J., and Póczos, B. (2017). Equivariance Through Parameter-Sharing. In *Proceedings of the 34th International Conference on Machine Learning*, pages 2892–2901. PMLR.

Rice, J. R. (1976). The Algorithm Selection Problem. In Rubinoff, M. and Yovits, M. C., editors, *Advances in Computers*, volume 15, pages 65–118. Elsevier.

Rivolli, A., Garcia, L. P. F., Soares, C., Vanschoren, J., and de Carvalho, A. C. P. L. F. (2022). Meta-features for meta-learning. *Knowledge-Based Systems*, 240:108101.

Shawe-Taylor, J. (1993). Symmetries and discriminability in feedforward network architectures. *IEEE Transactions on Neural Networks*, 4(5):816–826.

Vinyals, O., Bengio, S., and Kudlur, M. (2016). Order Matters: Sequence to sequence for sets. In Bengio, Y. and LeCun, Y., editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*.

Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R. R., and Smola, A. J. (2017). Deep Sets. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.