GRASP-GCN: Graph-Shape Prioritization for Neural Architecture Search under Distribution Shifts

Sofia Casarin   SCASARIN@UNIBZ.IT
Faculty of Computer Science
Free University of University-Bolzano
Bolzano, BZ 39100, ITA

Emanuele Caruso   EMCARUSO@UNIBZ.IT
Free University of Bozen-Bolzano
Bolzano, BZ 39100, ITA

Oswald Lanz   LANZ@INF.UNIBZ.IT
Free University of Bozen-Bolzano
Bolzano, BZ 39100, ITA

Abstract

Neural Architecture Search (NAS) methods have shown to output networks that largely outperform human-designed networks. However, conventional NAS methods have mostly tackled the single dataset scenario, incuring in a large computational cost as the procedure has to be run from scratch for every new dataset. In this work, we focus on predictor-based algorithms and propose a simple and efficient way of improving their prediction performance when dealing with data distribution shifts. We exploit the Kronecker-product on the randomly wired search-space and create a small NAS benchmark composed of networks trained over four different datasets. To improve the generalization abilities, we propose GRASP-GCN, a ranking Graph Convolutional Network that takes as additional input the shape of the layers of the neural networks. GRASP-GCN is trained with the not-at-convergence accuracies, and improves the state-of-the-art of 3.3 % for Cifar-10 and increasing moreover the generalization abilities under data distribution shift.

Keywords: Distribution Shifts, Neural Architecture Search, Graph Convolutional Networks

1 Introduction

Neural Architecture Search (NAS) has drawn large research attention due to its efficacy in automatically optimizing the architecture of Deep Neural Networks (DNNs), replacing the error-prone manual design which demands high expertise. As the NAS process can be very expensive many methods were proposed to save time or computation, following two main directions: i) reducing the time required to evaluate each searched architecture proposing a weight sharing mechanism (Cai et al. (2020); Bender et al. (2018) Pham et al. (2018); Liu et al. (2019); Xie et al. (2020)), ii) using sample efficient algorithms so that only few architectures are evaluated (Zoph et al. (2018); Zela et al. (2018); Klein et al. (2016); Real et al. (2019)). Predictor-based algorithms follow the second approach, and train a proxy model that can infer the validation accuracy of DNNs directly from their network structure. During optimization, the proxy can be used to narrow down the number of architectures for which the true validation accuracy must be computed, which makes predictor-based algorithms sample efficient. Predictor-based algorithms have been proposed by Wen et al. (2019); Ning et al. (2020) and Dudziak et al. (2020). Despite the success of these kinds of approaches, only few methods (Lee et al. (2021); Huang et al. (2022)) tackle the problem of sharing or re-using the predictor knowledge on different datasets. Most conventional NAS methods are indeed task-specific, requiring repeatedly training the model from scratch for each new dataset.

Moreover, existing NAS benchmarks either i) provide architectures trained on a single dataset (Ying et al. (2019)), ii) define a benchmark across different tasks but not datasets (Duan et al. (2021)), iii) or do not provide the full training-log of architectures and define not-unique networks in their searchspace (Dong et al. (2020)). This limit the possible studies that can be done on the datasets to correctly interpret possible predictor results. In this paper, we restrict the problem to predictor-based algorithms, which given a NAS benchmarks are extremely fast to train ($\sim$10 min on GeForce GTX 1080). Therefore, a natural question arises: *Can we re-use the knowledge that a predictor has learned on one dataset and transfer it to get a more sample-efficient algorithm on another dataset?* To this aim, we study the impact of distribution shifts on the predictor performance, by analysing the ranking of the architectures trained on commonly used datasets for the task of image classification, and propose simple yet effective solutions to address the shift. Specifically, we design a randomly wired search space, that exploits the Kronecker product to impose a Resnet-like structure and create a dataset of 2000 architectures trained on Cifar-10, Cifar-100, Tiny-ImageNet and Fashion-MNIST datasets. We study the generalization abilities of the predictor when directly used on a new dataset without fine-tuning and propose to integrate the so called *vertex shapes* - the shapes each layer has given a different input size, and to adopt early stopping - training the predictor with not-converged accuracies. Our study shows which are the limitations of predictor based algorithms, and our simple approach improves of **3.7%** and **9.5%** (without and with distribution shift) with respect to the naïve approach . To summarize, our contributions are threefold:

- We propose a new way of defining search spaces, that exploit the generality of randomly wired spaces but samples neural network efficiently through the Kronecker product and a criterion based on a desired skeleton, to obtain specific categories of neural networks.

- We analyse the generalization capabilities of naïve predictor based algorithms on two different scenarios, involving different latent data but the same observed data and different latent data with different observed data.

- We propose GRASP-GCN, which integrates the shapes of the layers of the neural networks as input to predictor, and trains with the accuracies of non-specialized neural networks.

2 Related Works

Different techniques were proposed to mitigate the large computation burden of NAS. These approaches primarily target the acceleration of either the evaluation or search modules within the NAS framework. The former accelerates the evaluation of each DNN, the latter increases the sample efficiency so that fewer architectures need to be evaluated for discovering a good network. Our work falls under the second category, as the predictor can be utilized to sample architectures that most likely perform well on a given task.

2.1 Single dataset predictor-based algorithms

Many predictor-based methods, that set the baseline for following works, focus on a single dataset. Wen et al. (2019) train a regressor model on a small built dataset and select the top-K predicted architectures to train them from scratch. The proposed approach leads to a more than $20\times$ sample efficient algorithm compared to standardly used evolution ones. Wei et al. (2023) used graph neural network-based accuracy predictors and an iterative approach to estimate the accuracy of models. Wen et al. (2019) propose a Graph-based neural Architecture Encoding Scheme, *i.e.* GATES, to improve the generalization abilities of performance predictors by modeling the information flow of
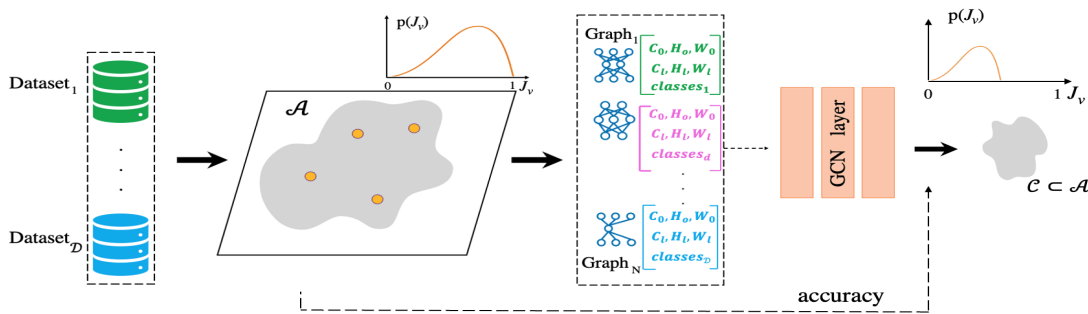
Figure 1: Architectures are sampled from the search space and trained over 4 datasets. The structure of DNNs is given as input with the shapes of the layers to a ranking GCN, which given the accuracies, ranks DNNs so that the search space is narrowed down.

the actual data processing of the architecture as the attributes of the input nodes. Dudziak et al. (2020) propose an efficient hardware-aware NAS method enabled by an accurate performance predictor based on Graph Convolutional Network (GCN). The authors show that the sample efficiency of predictor based NAS can be improved by considering binary relations of models and an iterative data selection strategy. Similarly to BRP-NAS, we employ a binary ranking GCN, but we extend the focus on multiple datasets and employ as labels the validation accuracy of non-specialized networks to improve the generalization abilities.

2.2 Transferable predictor-based algorithms

Relevant approaches to ours tackling the generalization problem across multiple dataset are MetaD2A by Lee et al. (2021), and Arch-Graph by Huang et al. (2022). MetaD2A stochastically generates graphs from a dataset via a cross-modal latent space that is learned via amortized meta-learning. From the encoding of the dataset, obtained through a permutation invariant encoder set, a graph is decoded. A meta-predictor estimates and selects the best architecture for a given dataset. Instead of using an encoder set, we propose a much simpler and more general solution that does not lack the possibility to adapt it to video-datasets due to the encoder-set. We provide as additional input to the GCN the "vertex shapes", which are strictly related to the shapes of the data. In Arch-Graph the generalization problem is addressed as *task* generalization, rather than dataset generalization. The method predicts task-specific optimal architectures with respect to given task embeddings, by leveraging correlations across multiple tasks through their embeddings as a part of the predictor's input for fast adaptation. Despite being sample efficient across many tasks, the method requires predictor-tuning on the new task/dataset. With respect to previous approaches, our work tackles the problem from the point of view of the distribution shift in the dataset, proposing a simpler yet more general method that accounts the specialization (or overfitting) of neural networks over datasets, the shape characteristics of the data and their effect on networks, and that does not require fine-tuning the predictor.

3 Methods

Our goal is to obtain a predictor that generalizes well across different datasets without need to re-train or finetune. To this end, we sample architectures from our search-space and train them over 4 image classification datasets (3.1) and propose GRASP-GCN (3.2), which trains a ranking predictor with an additional input consisting in the vertex shapes.

3.1 Search-space definition

NAS is formalized as a bi-level optimization problem:

$$\mathcal{A}^* = \operatorname{argmin}_{\mathcal{A}} J_v(\mathcal{A}, \mathbf{w}^*)$$
$$\text{s.t.} \quad \mathbf{w}^* = \operatorname{argmin}_{\mathbf{w}} J_t(\mathcal{A}, \mathbf{w}), \tag{3.1}$$

where $\mathcal{A}$ describes the architecture, $\mathbf{w}$ are the trainable weights of the considered DNN, and J_v and J_t are the validation and training loss, respectively. In our work we define $\mathcal{A} = (\mathbf{A}, \mathbf{X})$, with $\mathbf{A}^{N \times N}$ and $\mathbf{X}^{N \times D}$ encoding the connections in the graph and the types of layers, respectively. The dimensions of the matrices are related to the N number of nodes (layers) and the D number of input features, *i.e.* the number of layer types allowed. We sample $\mathcal{A}$ from our search space composed of all feed-forward convolutional networks, belonging to the randomly-wired search space Xie et al. (2019) and exploit the Kronecker product, a trick that allows us balancing flexibility and efficiency, avoiding the sampling of big random matrices. We generate resnet-like architectures miming evolutionary sampling. Two random matrices $\mathbf{R}_1^{4 \times 4}$ and $\mathbf{R}_2^{4 \times 4}$ are indeed sampled from the search space and multiplied with two *skeleton matrices* $\mathbf{K}_1^{4 \times 4}$ and $\mathbf{K}_2^{4 \times 4}$:

$$\mathbf{A} = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{K_1} \otimes \begin{bmatrix} & & R_1 & \end{bmatrix} + \underbrace{\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}}_{K_2} \otimes \begin{bmatrix} & & R_2 & \end{bmatrix} = \begin{bmatrix} R_1 & R_2 & & \\ & R_1 & R_2 & \\ & & R_1 & R_2 \\ & & & R_1 \end{bmatrix} \tag{3.2}$$

In Eq. 3.2, $\mathbf{K_1}$ which is multiplied by $\mathbf{R_1}$ defines the feed-forward structure, while $\mathbf{K_2}$, with the off-diagonal values, defines the shortcuts. The *input* and the *output* layers are finally added to the generated matrix $\mathbf{A}^{16 \times 16}$, leading to a maximum dimension of 18×18. As shown, the Kronecker product $\otimes$ allows repeating $\mathbf{R}_1$ and $\mathbf{R}_2$ structures, limiting the randomly wired search space in a meaningful way. Moreover, it generates easily scalable networks, a key advantage as proven by Zoph et al. (2018), by stacking multiple blocks. This design choice was intentional to focus on analyzing the true impact of the dataset itself. Moreover, it does not represent a limitation as the only requirement to properly train a GCN predictor is to have both very-well and very badly-performing DNNs. Table 1 summarizes our candidate layers constituting our $\mathbf{X}^{18 \times 9}$.

Layer	stem conv3×3	conv3×3	conv3×3×d	conv3×3×h	conv3×3s2	conv3×3s2×d	conv3×3s2×h
Channels	64	**Same**	**Doubles**	**Halves**	**Same**	**Doubles**	**Halves**
Stride	-	-	-	-	2	2	2

Table 1: Layers in the search space. Same / Double / Halves refer to the channels of the parent nodes.

3.2 Ranking GCN

Graph Convolutional Networks are DNN architectures that extract multi-scaled localized spatial features to extract highly expressive representations of graphs, dealing with the difficulty of "localized convolution" filters in non-Euclidean domains. GCNs performs a convolution looking for essential vertices and edges with the goal of learning the features of the graph. It takes as input: (i) a feature description X_i for every node i summarized in a feature matrix $\mathbf{X}^{N \times D}$ where N is the number of nodes, D the number of input features; (ii) a representative description of the graph structure summarized in the adjacency matrix $\mathbf{A}^{N \times N}$. For the classification task, the GCN produces a node-level output $\mathbf{H}^{N \times F}$, where F is the number of output features per node. The GCN outputs $h(\mathbf{A}, \mathbf{X})$ which is the concatenation of each $\mathbf{H}^l$ layer mapping done, as described in Eq. 3.3:
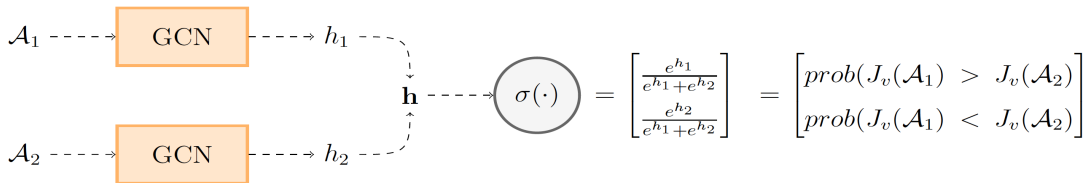
Figure 2: Working principles of a GCN used as a ranking network.

$$h(\mathbf{A}, \mathbf{X}) = (H^L \circ H^{L-1} \circ \cdots \circ H^l \cdots \circ H^1)(\mathbf{A}, \mathbf{X}) \tag{3.3}$$

where $l = 1, 2, .. $ L, is the number of layers in the GCN, and $\mathbf{H}^l$ is given by the propagation rule (eq. 3.4) defined by in Kipf and Welling (2017):

$$H^{(l+1)} = \sigma(\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} H^{(l)} W^{(l)}) \tag{3.4}$$

with $\tilde{A} = A + I$, (I identity matrix), $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$ is the degree matrix, $W^{(l)}$ is the layer-specific trainable weight matrix, σ is the ReLU activation function, $H^{(l)} \in \mathbb{R}_{N \times D}$ is the matrix of activations in the $l^{(th)}$ layer, and $H^{(l+1)}$ is the output to the next layer. Graph-level outputs are modeled by introducing some form of pooling operation. In our work we exploit a GCN with 3 layers and a classification head, and following BRP-NAS we exploit a Ranking GCN, which learns a binary relation that focuses on the prediction of ranking. Indeed, as previously observed by Dudziak et al. (2020) *i)* accuracy prediction is not necessarily required to produce faithful estimates (in the absolute sense) as long as the predicted accuracy preserves the ranking of the models; *ii)* any antisymmetric, transitive and convex binary relation produces a linear ordering of its domain, implying that NAS could be solved by learning binary relations, where $O(n^2)$ training samples can be used from <u>n</u> measurements. Given the architecture's search space, a ranking network predicts *how likely* any network in the search space reaches a higher accuracy than the current best. Fig. 2 shows how a GCN can be used as a ranking network: two architectures $\mathcal{A}_1$ and $\mathcal{A}_2$ are fed to the GCN. The GCN outputs two graph encodings h_1 and h_2, which are then concatenated into the vector $\mathbf{h}$. A softmax function $\sigma()$ is applied, obtaining the ranking probabilities which are then compared with the target $\mathbf{t}$. Given for example the tuple $(acc_{i,\mathcal{A}_j}, acc_{i,\mathcal{A}_k})$ where $acc_{i,\mathcal{A}_j}$ is the accuracy of architecture $\mathcal{A}_j$ and $acc_{i,\mathcal{A}_k}$ is the accuracy of $\mathcal{A}_k$, both over dataset$_i$, if $acc_{i,\mathcal{A}_j} > acc_{i,\mathcal{A}_k}$, then the target vector takes as values $\mathbf{t} = [1 \quad 0]$. Our goal is to maximize the log probability that $\mathcal{A}_j$ is better than $\mathcal{A}_k$, and therefore the predictor is trained with the loss in Eq. 3.5:

$$J = -\mathbf{t}^T \cdot ln(\sigma(\mathbf{y})). \tag{3.5}$$

As $\mathcal{A}_{j,k}$ gives the information about the structure of the graphs, and $\mathcal{X}$ defines the layers in the DNN architecture, given a DNN architecture that is characterized by a bottleneck, the predictor won't capture the implications of a severe reduction of the feature maps if provided with these inputs only. We therefore add the dimensions characterizing each DNN layer. This info shortly called *"vertex shapes"* and is concatenated to $\mathcal{X}$ that has now dimensions $18{\times}12$. We normalize the vertex shapes with respect to the maximum dimensions in our dataset and encode them as float numbers. One hot encoding was discarded as it implicitly looses any ordering and distance knowledge, *i.e.* we don't know anymore if (3,32,32) is closer to (3,64,64) or to (4096,3,3).

4 Experiments

We investigate the statistics of our small NAS dataset composed of (architectures, accuracy) pairs to provide ground-truth values for the predictor (Sec. 4.1). We show the performance of our

	F-MNIST	C10	C100	Tiny
F-MNIST	0	0.0273	0.0461	0.0516
C10	0.1781	0	0.1032	0.1293
C100	0.4559	0.2653	0	0.1743
Tiny	0.5157	0.3456	0.1817	0

Table 2: 1-NDCG@2092
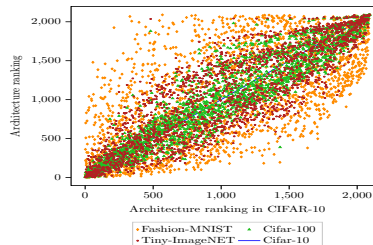
(a)

Figure 3: Left: the columns display the training dataset, rows the validation dataset Color bar: (lowest ▬▬▬▬ highest). Right: the ranking of each architecture on one of the four datasets, sorted by the ranking in Cifar-10.

predictor under distribution shift (Sec. 4.2) and further validate the usage of the vertex shapes as additional input and of not converged accuracies through ablations (Sec. 4.3). Our performance are reported with three different measures: The validation accuracy, the precision@10, and the Normalized Discounted Cumulative Gain (NDCG). We refer the reader to appendices A and C for insights on the measures and for implementation details.

4.1 Dataset

The 4 datasets were chose to investigate a variety of conditions for the transferability of the predictor, and to extend and be partially comparable with previous works that involve three datasets, as in NATS-Bench Dong et al. (2020). Two different scenarios were defined: i) transferability when different latent data is involved *but* the observed data is **the same**, ii) transferability when different latent data is involved *and* the observed data is **not the same**. Hence, we chose (a) Cifar-10 as a baseline dataset, (b) Cifar-100 as it is composed of Cifar-10 images, labeled differently, (c) Fashion-MIST as it is composed of black and white images (different latent data), belonging to completely different categories with respect to Cifar-10, and (d) Tiny-ImageNET to consider a more complex dataset with a larger number of classes. 2000 *unique* architectures were sampled from the search space and trained over the aforementioned datasets. Models trained over different input distributions are not guaranteed to perform in the same way. Do they "overfit" the dataset specializing during training? Can early stopping be applied to reduce the training time it takes to get the true validation accuracy of the architectures?

Distribution Shift Tab. 2, to be read column-wise, highlights that the order induced by compelx datasets generalizes better for simpler datasets than vice-versa. This result is complemented by Fig. 2a, that displays the ranking stability of each architecture on one of the four datasets. The plot was obtained by sorting the architectures trained on Cifar-10 in descending order with respect to their accuracy on the dataset, and inducing the same ID-order on the architectures trained on different datasets. In this way, it is possible to observe what rank has been assigned to the architectures trained over different datasets, having as baseline Cifar-10 rank. We can observe that (i) Fashion-MNIST architectures cause a higher variability in terms of architectures rank ii) the plot start tighter, increase variance as we move towards bigger ranks, and gets tighter as we approach the "worst" architectures. We can deduce that, not only the worst architectures are such since the very beginning of training, (Fig. 4b), but as it could be naturally expected, some architectures simply do not have enough capacity to solve any classification task and perform badly independently on the input distribution they are provided, further validating the early stopping method we propose in Sec. 4.2.

Network Specialization We compared through NDCG the ranking induced by the validation

	F-MNIST	C10	C100	Tiny
F-MNIST	85.1 ± 0.7	76.4 ± 0.5	68.2 ± 0.6	68.4 ± 0.8
C10	74.2 ± 0.6	87.9 ± 0.4	84.9 ± 0.4	83.4 ± 0.7
C100	70.8 ± 0.6	85.0 ± 0.1	87.0 ± 0.4	85.6 ± 0.3
Tiny	72.3 ± 0.3	83.6 ± 0.9	85.3 ± 0.2	87.9 ± 0.2

	F-MNIST	C10	C100	Tiny
FMNIST	100 ± 0	93 ± 2	94 ± 2	96 ± 1
C10	80 ± 4	99 ± 1	97 ± 2	95 ± 3
C100	36 ± 3	{60 ± 2	65 ± 1	63 ± 1
Tiny	41 ± 2	61 ± 3	57 ± 3	69 ± 2

(a) Average validation accuracy of the predictor.(%)   (b) Average Precision@10 (×100) of the predictor.

Table 3: For each element of the table, the mean and the std of four runs are given. The columns are the training datasets, the rows are the validation datasets.

accuracy v_acc$_i$ with $i = 1, \dots, 120$ with v_acc$_{120}$ at the end of training. Fig. 4a highlights a possible correlation between the change of rank and the epoch where the learning rate is dropped (we refer to Appendix C for details on hyper-parameters). Moreover, focusing on the NDCG@10, that considers only the top-10, the plot reaches zero way before the end of the training. Therefore, we formulated the hypothesis that the difference between NDCG@2092 and the NDCG@10 could be caused by the average performing networks, which are more strongly influenced by the hyper-parameters. We checked the evolution of ranking during training (Fig. 4b) by counting how many times the architectures change the relevance value (Eq. A.2 Appendix). We observe that top-performing DNNs stop changing rank at earlier epochs, average performing ones have the peak shifted towards the end of training, and DNNs not able to solve the task are such since the early epochs.

(a) Cifar-10 ranking correlation.   (b) Cifar-10 Ranking evolution.   (c) Validation accuracy of the GCN.
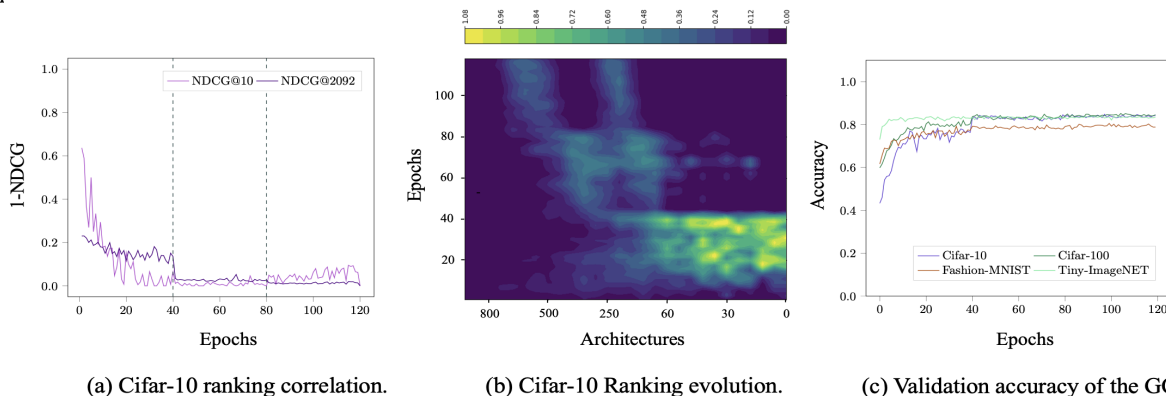
Figure 4: Cifar-10 results. 1-NDCG plot (a) showing the ranking correlation. The lower the better. The dashed lines highlight the epoch where the learning rate is dropped. Ranking evolution during training (b) derivative plot. The x-axis shows the architectures in a descent ordered w.r.t their accuracy. The y-axis carries the training epochs. The heatmap displays large numbr of rank changes (yellow) to no changes (blue). (c) Validation accuracy the GCN can obtain when trained with the previous rankings.

4.2 GRASP-GCN

We use the validation accuracy and the Precision@10 measures to evaluate the performance of GRASP-GCN. Tab 3a, Tab. 3b show the results obtained for each training dataset (columns) on each of the four validation datasets (rows). The values in the table are above 80 % when Fashion-MNIST is not involved, which suggests that the predictor trained over the datasets involved in the lower part of the table, e.g. Cifar-10, Cifar-100, Tiny- ImageNET, can transfer knowledge over one of these datasets. On the other hand, when Fashion-MNIST is involved, either as the training or

	F-MNIST	C10	C100	Tiny
F-MNIST	84.1 | 82.2 | 79.8	74.4 | 71.1 | 68.2	68.2 | 64.8 | 68.2	68.4 | 63.5 | 65.5
C10	74.2 | 73.8 | 73.2	87.9 | 84.1 | 86.2	84.9 | 81.1 | 80.8	83.4 | 81.1 | 80.7
C100	70.8 | 68.5 | 70.7	85.0 | 80.1 | 79.1	87.0 | 82.1 | 84.3	85.6 | 82.4 | 85.0
Tiny	72.3 | 66.4 | 70.1	83.6 | 79.2 | 83.2	85.3 | 80.1 | 83.4	87.9 | 84.0 | 86.6

Table 4: Comparison between (black) ours, (blue) BRP-NAS, (red), MetaD2A.

the validation dataset, the performance drops drastically. Finally, Tab. 4 shows that GRASP-GCN surpasses all other methods when trained and evaluated over Cifar-10 and when directly applied to new datasets.

4.3 Ablation

	F-MNIST	C10	C100	Tiny
F-MNIST	84.1 | 79.1 | 83.8	74.4 | 67.7 | 74.7	68.2 | 65.8 | 69.2	68.4 | 61.7 | 65.9
C10	74.2 | 71.9 | 73.4	87.9 | 84.2 | 86.6	84.9 | 82.3 | 84.0	83.4 | 80.1 | 83.4
C100	70.8 | 68.9 | 71.2	85.0 | 82.5 | 84.3	87.0 | 84.4 | 86.5	85.6 | 82.7 | 84.8
Tiny	72.3 | 62.5 | 70.9	83.6 | 81.1 | 83.6	85.3 | 83.0 | 85.0	87.9 | 84.0 | 87.3

Table 5: Accuracy (%) with vertex shapes and early stopping (black values), without vertex shapes (purple), without early stopping (blue).

Tab. 5 highlights how the predictor significantly improves the performance when trained and validated over the same dataset (diagonal values) improving the baseline of more than 3 %. If we focus on transferability, looking out of the diagonal, we have even larger improvements, with gaps exceeding 9%. We refer to Appendix D for Precision@10 comparisons. Finally, Fig. 4c ablates on the early-stopping mechanism we propose to employ in the NAS procedure. We can observe that if we use as training set the accuracies the architectures have after the first drop of the learning rate (epoch 40) the performance of the predictor is not affected. This is further validated by Tab. 5, which compares our best results with and without early stopping.

5 Conclusions

In our work, we face the problem of analyzing the transferability of a predictor under data distribution shift. For this reason, we created our small dataset composed of architectures trained on four datasets. Our ranking analysis on the trained networks showed an association between the drop of the learning rate and the epoch where architectures stop changing their ranking during training, highlighting that top-performing architectures keep their ranking since early training epochs, while the worst ones are such since the early epochs. We moreover spotted that the ranking induced by complex datasets generalizes better than those induced by simple datasets. Given this, we improve the naive predictor training by including the vertex shapes as input, and employing an early stopping procedure. Our method surpasses state-of-the art performance on predictor-based algorithms addressed by distribution shifts.

Limitations and future works Possible future works could include enlarging the datasets by considering new tasks such as object detection and new modalities such as videos. Another interesting direction could be studying how the method improves with the inclusion of gradients information as input to the predictor.

References

Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. Understanding and simplifying one-shot architecture search, 10–15 Jul 2018. URL `http://proceedings.mlr.press/v80/bender18a.html`.

Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment, 2020.

Xuanyi Dong, Lu Liu, Katarzyna Musial, and Bogdan Gabrys. Nats-bench: Benchmarking nas algorithms for architecture topology and size, 08 2020.

Yawen Duan, Xin Chen, Hang Xu, Zewei Chen, Xiaodan Liang, Tong Zhang, and Zhenguo Li. Transnas-bench-101: Improving transferability and generalizability of cross-task neural architecture search. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 5251–5260, 2021.

Łukasz Dudziak, Thomas Chau, Mohamed Abdelfattah, Royson Lee, Hyeji Kim, and Nicholas Lane. Brp-nas: Prediction-based nas using gcns, 07 2020.

Minbin Huang, Zhijian Huang, Changlin Li, Xin Chen, Hang Xu, Zhenguo Li, and Xiaodan Liang. Arch-graph: Acyclic architecture relation predictor for task-transferable neural architecture search, 2022.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2017.

Aaron Klein, Stefan Falkner, Simon Bartels, Philipp Hennig, and Frank Hutter. Fast bayesian optimization of machine learning hyperparameters on large datasets, 05 2016.

Hayeon Lee, Eunyoung Hyung, and Sung Ju Hwang. Rapid neural architecture search by learning to generate graphs from datasets, 2021.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search, 2019.

Xuefei Ning, Yin Zheng, Tianchen Zhao, Yu Wang, and Huazhong Yang. A generic graph-based neural architecture encoding scheme for predictor-based nas, 2020.

Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. Efficient neural architecture search via parameter sharing, 2018.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc Le. Regularized evolution for image classifier architecture search. Proceedings of the AAAI Conference on Artificial Intelligence, 33, 02 2019. doi: 10.1609/aaai.v33i01.33014780.

Chen Wei, Chuang Niu, Yiping Tang, Yue Wang, Haihong Hu, and Jimin Liang. Npenas: Neural predictor guided evolution for neural architecture search. IEEE Transactions on Neural Networks and Learning Systems, 34(11):8441–8455, 2023. doi: 10.1109/TNNLS.2022.3151160.

Wei Wen, Hanxiao Liu, Hai Li, Yiran Chen, Gabriel Bender, and Pieter-Jan Kindermans. Neural predictor for neural architecture search, 2019.

Saining Xie, Alexander Kirillov, Ross Girshick, and Kaiming He. Exploring randomly wired neural networks for image recognition, 2019.

Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. Snas: Stochastic neural architecture search, 2020.

Chris Ying, Aaron Klein, Esteban Real, Eric Christiansen, Kevin Murphy, and Frank Hutter. Nas-bench-101: Towards reproducible neural architecture search, 02 2019.

Arber Zela, Aaron Klein, Stefan Falkner, and Frank Hutter. Towards automated deep learning: Efficient joint neural architecture and hyperparameter search, 07 2018.

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc Le. Learning transferable architectures for scalable image recognition, 06 2018.

Appendix A. Ranking measures

Several measures are commonly used to assess the quality of a ranking. We focus on: 1) NDCG@k, 2) Precision@k and 3) Kendall's τ.

NDCG   DCG (Discounted Cumulative Gain) is founded on the idea that when assessing search results, highly relevant documents ranked lower should receive a larger penalty than less relevant documents wrongly ranked. This is because the graded relevance value decreases logarithmically as the position of the result worsens. This measure applies very well to predictor-based algorithms since our focus is on allowing the predictor to find the best-forming architectures in such a way that they are placed on the top of the list, while we don't care about how bad networks are ranked. Eq. A.1 exactly shows how highly relevant objects that appear lower in the ranked list are penalized by reducing the graded relevance value logarithmically proportional to the position of the result:

$$DCG_k = \sum_{i=1}^{k} \frac{2^{rel_i} - 1}{log_2(i+1)}. \tag{A.1}$$

Starting from A.1, the NDCG can be easily obtained normalizing w.r.t the Ideal Discounted Cumulative Gain (IDCG), as shown in Eq. A.2:

$$NDCG_k = \frac{DCG_k}{IDCG_k} \in [0,1], \qquad IDCG_k = \sum_{i=1}^{|REL_k|} \frac{2^{rel_i} - 1}{log_2(i+1)} \tag{A.2}$$

where rel_i is the relevance value assigned to each object and REL_k is the list of relevant objects ordered by their relevance up to position k. In a perfect ranking algorithm, the DCG_p will be the same as the $IDCG_p$ producing an NDCG of 1. We used the NDCG in two different variants: the NDCG@2092 was used to get a big picture of the general behavior all architectures have, while the NDCG@10 was used to focus on the ranking quality of the top-10 performing architectures, which are those we are interested in.

Precision@k   It is defined as the proportion of recommended items in the top-k set that are relevant. It is mathematically defined as:

$$Precision@k = \frac{\#of\ items\ @k\ that\ are\ relevant}{k} \in [0,1], \tag{A.3}$$

where again, as it can be observed, the concept of relevance is involved.

Kendall's τ   The Kendall rank correlation coefficient is used to measure the ordinal association between two measured quantities. It's range is in $[-1,1]$, with Kendall's $\tau = 0$ that indicates absence of correlation. Let $(x_1, x_2, \ldots x_n)$ and $(y_1, y_2, \ldots y_n)$ be a set of observations of the joint variables X and Y, such that all the values of (x_i) and (y_i) are unique (ties are neglected for simplicity). Any pair of observations (x_i, y_i) and (x_j, y_j), where $i < j$ are said to be <u>concordant</u> if the sort of order (x_i, x_j) and (y_i, y_j) agrees: that is, if either both $x_i > x_j$ and $y_i > y_j$ holds if both $x_i < x_j$ and $y_i < y_j$; otherwise, they are said to be <u>discordant</u>. Eq. A.4 defines, based on this, the coefficient:

$$\tau = \frac{(\#of\ concordant\ pairs) - (\#of\ discordant\ pairs)}{\binom{n}{2}} \in [-1,1], \tag{A.4}$$

Appendix B. More details on the dataset

The ranking correlation of the architectures trained on the four datasets introduced in 4.1 is complemented by the Kendall'sτ correlation, reported in Tab. 6. From Tab. 6 it emerges a strong correlation among Cifar-100 and Tiny-ImageNET dataset, among Cifar-10 and Cifar-100 dataset, and, with a smaller value, among Cifar-10 and Tiny-ImageNET datasets. Fashion-MNIST on the other hand, when related to these three datasets, appears out of the choir. This is extremely interesting, as it confirms previous results, and shows also that the similarities between the image datasets might be transferred to the architectures trained over those input datasets. These values are consistent with those presented in Dong et al. (2020).

	F-MNIST	C10	C100	Tiny
F-MNIST	1.00	0.44	0.35	0.23
C10	0.44	1.00	0.81	0.70
C100	0.35	0.81	1.00	0.85
Tiny	0.23	0.70	0.85	1.00

Table 6: Kendall's τ correlation values. Every dataset is sorted with respect to the validation accuracy so that the best performing architecture is in the 1st position and the worst performing in the last one. The architectures are then assigned a rank going from 1, . . . 2000. The table is symmetric, as the definition of Kendall's τ implies.

Ranking evolution   Figures 5 complement the results previously introduced in Sec. 4.1 for Cifar-10 (Fig. 4a). The plots indicate how 1) the ordering of the architectures trained over Fashion-MNIST seems to be the same since the very first epochs, as the 1-NDCG starts from values very close to 0. 2) The correlation between the drop of the learning rate and the change in the ranking verifies also for Cifar-100 and Tiny-ImageNET.
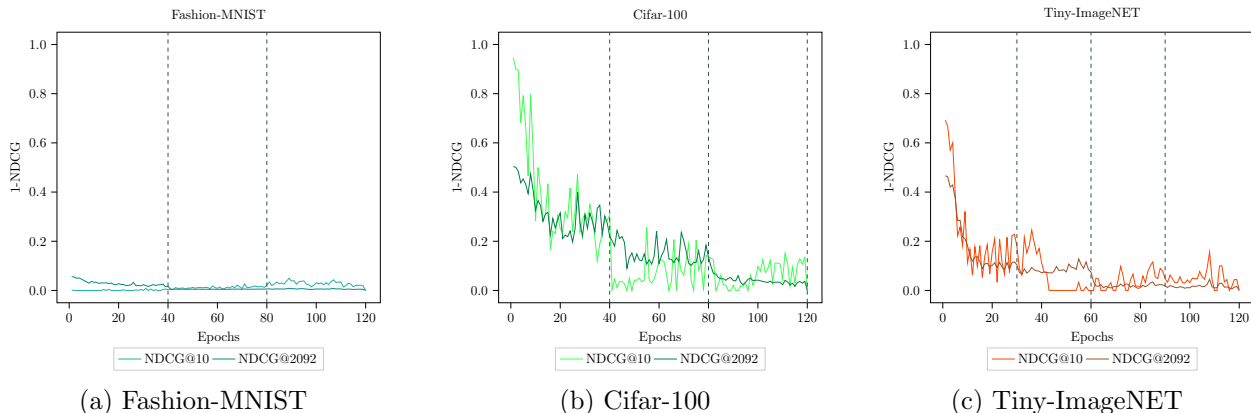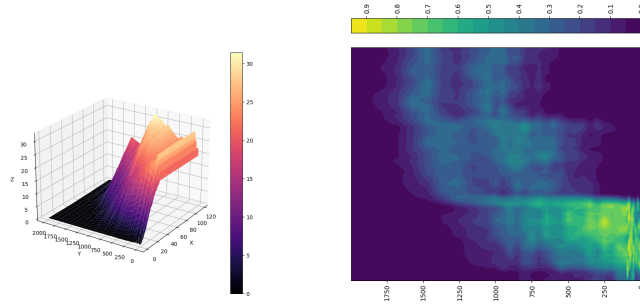

(a) Fashion-MNIST   (b) Cifar-100   (c) Tiny-ImageNET

Figure 5: 1-NDCG plot showing the ranking correlation among the sorting the architectures have at epoch i w.r.t the sorting of the architectures at epoch 120. Every figure displays the results for considering one dataset at a time. The dashed lines over each plot highlight the epoch where the learning rate is dropped. Every plot displays both the NDCG@10 (light color) and the NDCG@2092 (dark color).

Fig. 6 gives more details on the behaviour of the networks during training, confirming Cifar-10 results (Sec. 4.1 of the main paper). This can be used to speed-up the NAS pipeline, as we can early-discard failing architectures by checking how they perform with respect to the current best trained ones, further motivating the early-stopping procedure introduced in Sec. 4.2.

(a) Fashion-MNIST
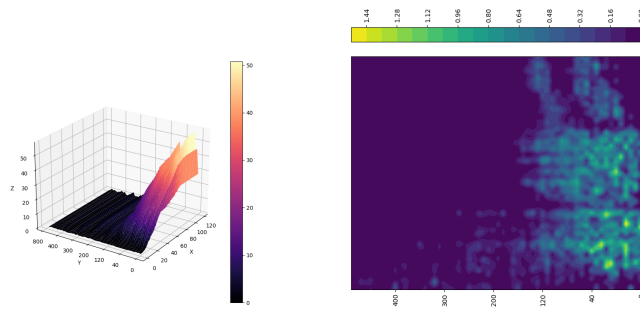
(b) Cifar-100
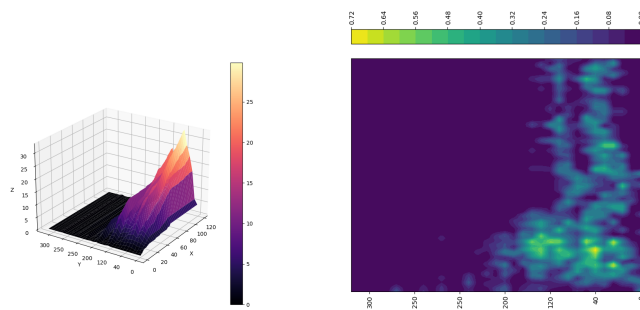
(c) Tiny-ImageNET

Figure 6: Plot of the changes of the relevance values of the architectures during training. (Left) Cumulative weighted count of the number of times an architecture changes relevance value. On the y-axis, the architectures are ordered from the best forming to the worst performing, on the x-axis the epochs are displayed. (Right) Derivative plot: the horizontal and the vertical axis represent respectively the y-axis and the x-axis of the left plot.

Appendix C. Implementation details

Our hyper-parameters are summarized in Tab. 7 which displays those used for training the architectures sampled from our search-space and in Tab 8 for training our GRASP-GCN. All hyper-parameters were optimized using Optuna framework[1]. We moreover provide the pseudo-code for the creation of our NAS dataset (1, and for the functioning of predictor-based algorithms within the NAS framework (2.

Dataset	Learning rate	Weight decay	Drop lr	Optimizer	Batch-size
F-MNIST	0.100	0.0002	of 10 at epoch 40, 80	SGD	128
C10	0.097	0.0006	of 10 at epoch 40, 80	SGD	128
C100	0.065	0.0015	of 5 at epoch 40, 80, 100	SGD	128
Tiny	0.012	0.0011	of 10 at epoch 30, 60, 90	SGD	64

Table 7: List of hyper-parameters derived from Optuna optimization for ResNet-18. The columns show the learning rate (lr), the weight decay (wd), the drop of the learning rate (drop lr), the optimizer, and the size of the batches.

Units per layer	Learning rate	Weight decay	Optimizer
265	0.019041	0.001126	Adagrad

Table 8: List of hyper-parameters derived from Optuna optimization for GRASP-GCN.

Algorithm 1 Dataset creation

1: **if** $Cifar10$ **then**
2:     **for** $iteration = 1, \ldots, 2000$ **do**
3:         sample $\mathcal{A} = \mathbf{A}, \mathbf{X}$
4:         train $\mathcal{A}$
5:         save values in hash($\mathcal{A}$)
6:     **end for**
7: **end if**
8: **if** FashionMNIST or Cifar100 or TinyImageNET **then**
9:     **for** $iteration = 1, \ldots, 2000$ **do**
10:         random sample $\mathcal{A}$ from Cifar10 dataframe
11:         new directory = hash($\mathcal{A}$)
12:         **while** new directory exist **do**
13:             random sample $\mathcal{A}$ from Cifar10 dataframe
14:             new directory = hash($\mathcal{A}$)
15:         **end while**
16:         train $\mathcal{A}$
17:         save values in new directory
18:     **end for**
19: **end if**

---

1. https://optuna.readthedocs.io/en/stable/

	F-MNIST	C10	C100	Tiny
F-MNIST	1.00 | 1.00 | 1.00	0.93 | 0.90 | 0.93	0.94 | 0.88 | 0.95	0.96 | 0.90 | 0.95
C10	0.80 | 0.78 | 0.82	0.99 | 0.98 | 0.98	0.97 | 0.80 | 0.95	0.95 | 0.90 | 0.91
C100	0.36 | 0.35 | 0.41	0.60 | 0.45 | 0.41	0.65 | 0.45 | 0.50	0.63 | 0.40 | 0.60
Tiny	0.41 | 0.10 | 0.50	0.61 | 0.40 | 0.58	0.57 | 0.43 | 0.55	0.69 | 0.50 | 0.64

Table 9: Average Precision@10 of the predictor with vertex shapes and early stopping (black values), without vertex shapes (purple), without early stopping (blue). The columns are the training datasets, the rows are the validation datasets.

Algorithm 2 Predictor-based neural architecture search

1: $\mathcal{S}$ = Architecture search space
2: $f(\mathcal{A}, \theta) : \mathcal{A} \rightarrow \mathbb{R}$: Predictor that outputs the predicted performance given the architecture
3: $N^{(k)}$: Number of architectures to sample in the k-th iteration
4: $k = 1$
5: $\tilde{\mathcal{S}} = \varnothing$
6: **while** $k \leq$ MAX_ITER **do**
7:     Sample a subset of architectures $\mathcal{C}^{(k)} = \{\mathcal{A}_j^{(k)}\}_{j=1,\ldots,N^{(k)}}$ from $\mathcal{S}$ utilizing $f(\mathcal{A}, \theta)$
8:     Evaluate architectures in $\mathcal{S}^{(k)}$, get $\tilde{\mathcal{C}}^{(k)} = \{\mathcal{A}_j^{(k)}, y_j^{(k)}\}_{j=1,\ldots,N^{(k)}}$ (y is the performance)
9:     $\mathcal{S} = \mathcal{S} - \mathcal{C}$
10:     $\tilde{\mathcal{S}} = \tilde{\mathcal{S}} \cup \tilde{\mathcal{C}}$
11:     Optimizing $f(\mathcal{A}, \theta)$ using the ground-truth architecture evaluation data $\tilde{\mathcal{S}}$
12:     $k \leftarrow k + 1$
13: **end while**
14: Output $\mathcal{A}_{j^*} \in \tilde{\mathcal{S}}$ with best corresponding y^*; Or, $\mathcal{A}^* = \text{argmax}_{\tilde{\mathcal{S}}} f(\mathcal{A}, \theta)$

Appendix D. Early stopping and vertex shapes

More ablations on the usage of early stopping and vertex shapes in GRASP-GCN. Tab. 9 presents complements the results introduced in Tab. 5 reporting the Precision@10. We can observe that results are coherent with the performance of GRASP-GCN in terms of accuracy, further proving the benefits of employing the vertex shapes as additional input. Moreover, we show how in most of the cases early-stopping either improves the performance or leaves them unchanged, meaning that we can benefit in terms of training-speed.

(a) Cifar-10

(b) Fashion-MNIST
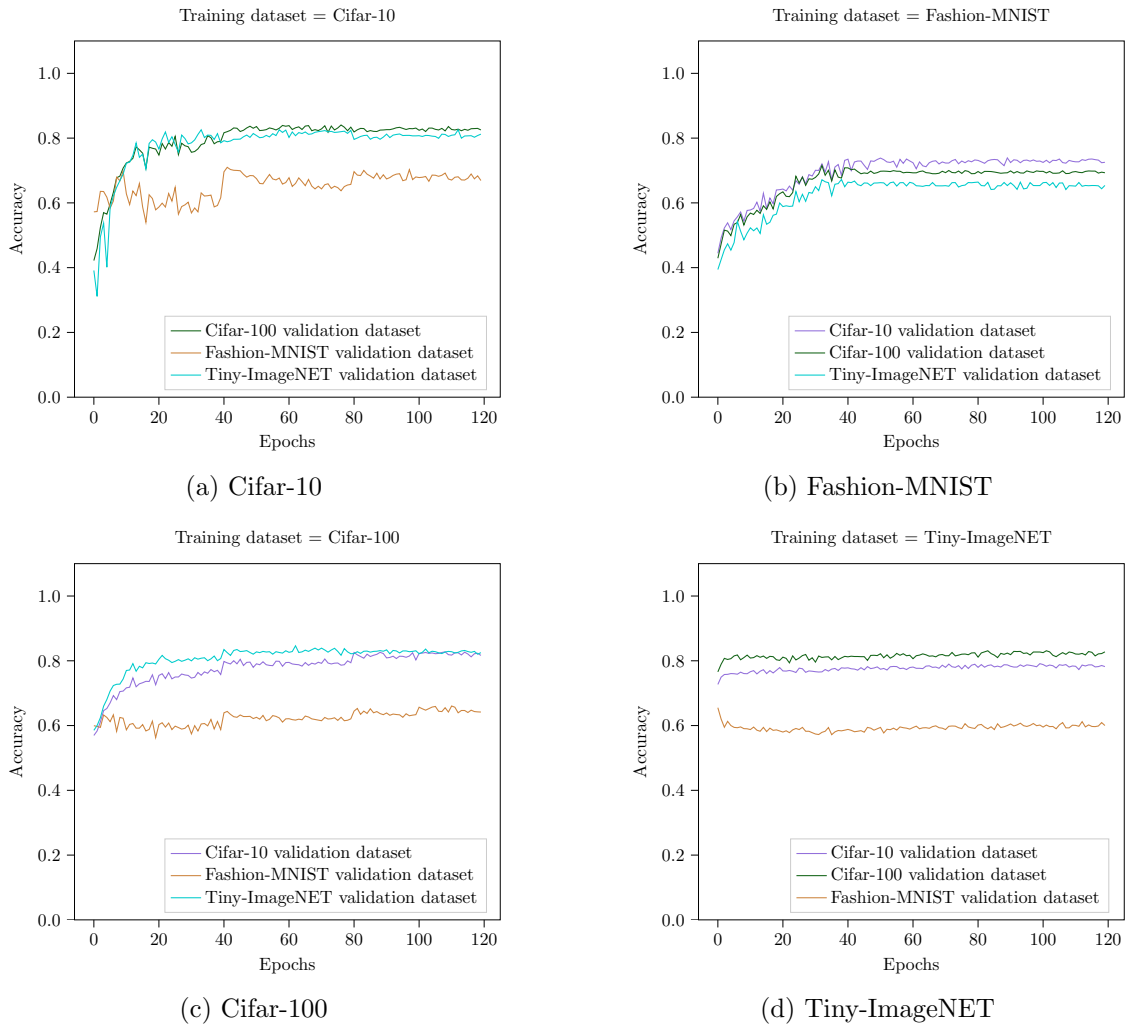
(c) Cifar-100

(d) Tiny-ImageNET

Figure 7: Validation accuracy of the GCN when trained with the previous sortings to predict the ranking of networks under distribution shift. Every point is obtained training the predictor with a training dataset composed by the sorting induced by the validation accuracy the architectures, trained over the dataset reported on the caption, have at epoch i, and with a validation dataset composed by the sorting induced by the validation accuracy the architectures, trained over the other three datasets, have at epoch 120.