Graph Embedding for Neural Architecture Search with Input-Output Information

Gabriela Suchopárová[1,2] **Roman Neruda**[1]

[1]The Czech Academy of Sciences, Institute of Computer Science
[2]Charles University in Prague, Faculty of Mathematics and Physics

Abstract Graph representation learning has been widely used in neural architecture search as a part of performance prediction models. Existing works focused mostly on neural graph similarity without considering functionally similar networks with different architectures. In this work, we adress this issue by using meta-information of input images and output features of a particular neural network. We extended the arch2vec model, a graph variational autoencoder for neural architecture search, to learn from this novel kind of data in a semi-supervised manner. We demonstrate our approach on the NAS-Bench-101 search space and the CIFAR-10 dataset, and compare our model with the original arch2vec on a REINFORCE search task and a performance prediction task. We also present a semi-supervised accuracy predictor, and we discuss the advantages of both variants. The results are competitive with the original model and show improved performance.

1 Introduction

Representation learning has been an essential part of the Neural Architecture Search (NAS) — while some NAS systems use hand-designed encodings, other systems learn the representation of neural graphs during the search or as a part of an unsupervised pretraining. These works focused on architectural similarity, however, neural networks with different architectures may still learn a similar function of the input (Wei et al. (2016)). As so, including some meta-knowledge might lead to improved representations of the search space.

In our work, we adress this problem by extending an existing model for unsupervised network embedding, *arch2vec* (Yan et al. (2020)), with input-output meta-information (IO data). That is, we learn the representation of a network and its output features on the input images. We construct a novel IO dataset, and present a semi-supervised model that learns from this type of data. We also compare the model with a variant that performs semi-supervised accuracy prediction. Both models are evaluated on two NAS tasks — REINFORCE search and performance prediction on NAS-Bench-101.

2 Related Work

Existing works used different encodings for performance prediction and NAS optimization, such as a string-based representation in PNAS, NAO and SemiNAS (Liu et al. (2018); Luo et al. (2018, 2020)), or path-based encoding in BANANAS (White et al. (2021)). Other systems used graph neural networks to encode the architectures, for example as a performance predictor in BONAS (Shi et al. (2020)) or GATES (Ning et al. (2020)), or in unsupervised embedding as in SVGe (Lukasik et al. (2021)) and arch2vec (Yan et al. (2020)). We describe the arch2vec in more detail in the Appendix (Section B). Graph neural networks have also been used as a part of surrogate benchmarks (Zela et al. (2022)).

Some recent works used meta-information to improve the accuracy or running time of performance prediction. FEAR ranks the architectures according to usefulness of extracted features

(Dey et al. (2021)). Other work estimate the performance using scalar statistics (Philipp (2021); Gracheva (2021)). As for meta-features related to network outputs, there exist metrics that measure neural representation similarity, such as centered kernel alignment (CKA, Kornblith et al. (2019)). However, to the authors' knowledge, these have not been used in NAS yet.

3 The IO Dataset

In our work, we extend the arch2vec model, which learns from a dataset of architectures of a search space. For our design, we introduce the IO-dataset, a set of triplets (net, in, out), where *net* is a network architecture, *in* is the *input* image and *out* is the *output* obtained through passing *in* to *net*. The *out* data can be any intermediate output of the network, e.g. feature maps or vector features. We use the CIFAR-10 dataset and NAS-Bench-101 search space to create the dataset (Krizhevsky (2009); Ying et al. (2019)). To reduce the training cost, we sampled 608 networks for the training IO dataset and 77 for the validation set, and the networks were trained on the CIFAR-10 train set (training details are in the Appendix (Section C).

The IO-dataset is created by passing *input* images from the CIFAR-10 validation set through these networks. As the *outputs*, we use the features preceding the last dense layer (the output of the global average pooling — a vector of size 512). We use only the top 10 most important features.

An issue with the dataset is that there is no ordering defined on the *output* features, meaning that the output of the global average pooling of two different networks may be the same, but permuted. We propose the following preprocessing — the weights of the last dense layer determine the *feature importance* of the global average pooling outputs for a target class. For an input image, we choose the dense weights corresponding to the correct target image class. Then, we sort the feature vector according to the weights, and we multiply the result and the weight vector. This preprocessing ensures better comparability among the networks. It does not alleviate all shuffling errors — two networks may extract the same feature with a slightly different importance, e.g. as the second and third most important feature respectively. However, two very similar networks should still have a similar response to the same image.

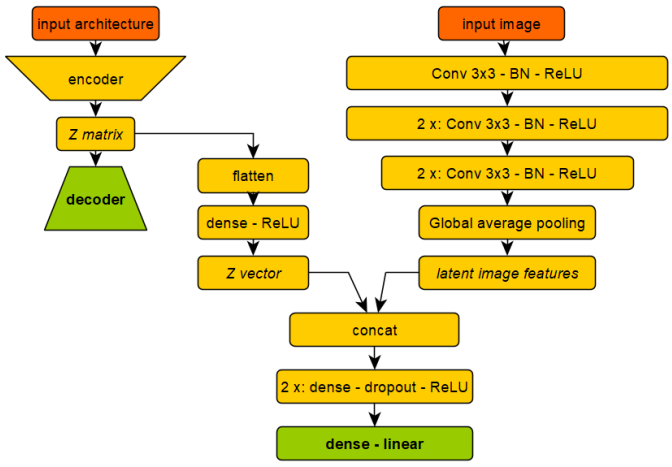

Figure 1: The architecture of our model.

4 The Extended Model

Now we describe how we extended the arch2vec model to learn from the IO dataset. Figure 1 shows the overall structure of the model. We will refer to our model as the *IO model*. We use the

original arch2vec VAE to encode the neural architecture into the latent matrix Z. The model has two outputs — one is the graph reconstructed from the latent matrix, the other are the predicted *output* features. The second output is present only in the labeled data. Our model is trained in a semi-supervised manner on both labeled and unlabeled data. Full training details are described in Section F in the Appendix. For comparison purposes, we also introduce a second model — a semi-supervised accuracy predictor. Its architecture is described in the Appendix (Section E). We will refer to the model as the *Accuracy model*.

5 Output Features Analysis

To gain insight into the IO dataset, we analyzed the output features and their relation to network test accuracy. Figures 2 and 3 visualize the features for a single network and of all networks on a single image respectively. We can see that there is a substantial variance between network outputs in both cases.

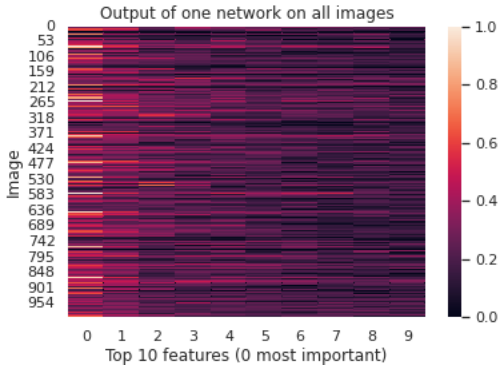

Figure 2: Output features of a single network on all train images.

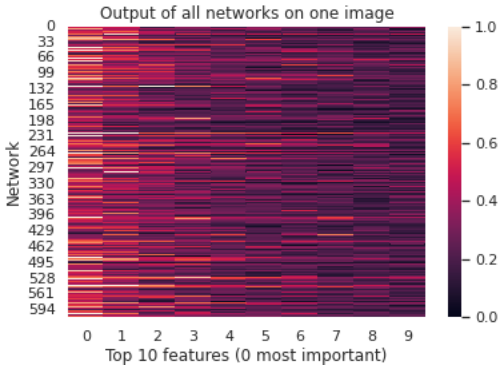

Figure 3: Output features of all networks on a single image.

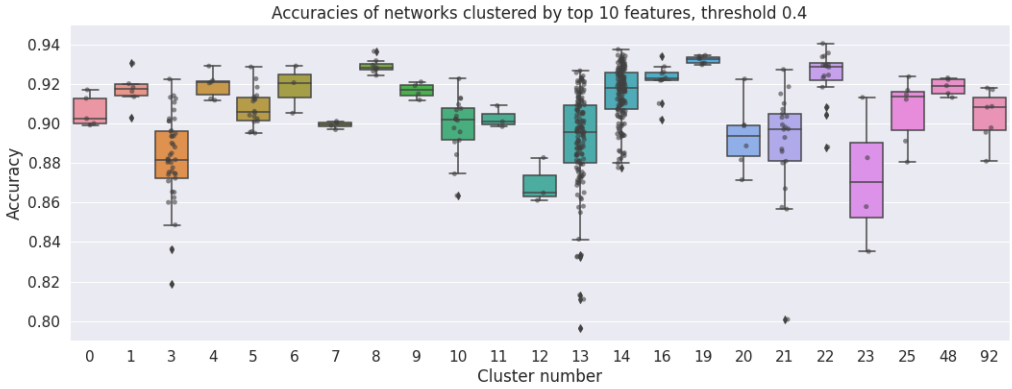

Figure 4: Networks clustered by output features, clusters with size larger than 2 are shown.

Next, we analyzed the relation between the output features and network test accuracy. For pairs of networks, we computed the mean squared error (MSE) of outputs given the same image, and used the mean of the errors as pairwise network distance. Then, we performed agglomerative clustering with complete linkage and distance threshold 0.4. The results are depicted in Figure 4. We can see that there is some relation between feature similarity and test accuracy — for instance, none of the worst performing networks are in a cluster with the best performing chosen networks,

and there is not a cluster that would stretch across the whole range of accuraccies completely. This indicates that along with architectural similarity, output features similarity may also correlate with the actual performance.

6 REINFORCE and Performance Prediction on NAS-Bench-101

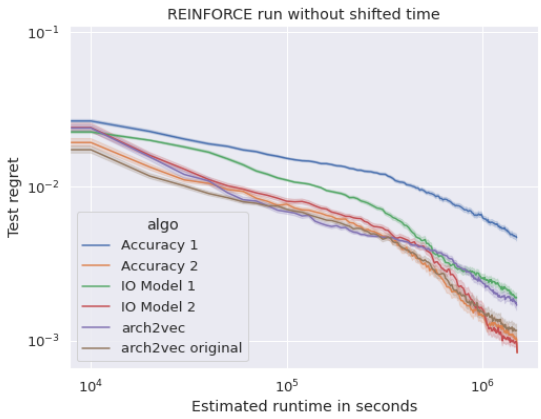

Figure 5: Reinforce search — 100 runs on NAS-Bench-101.

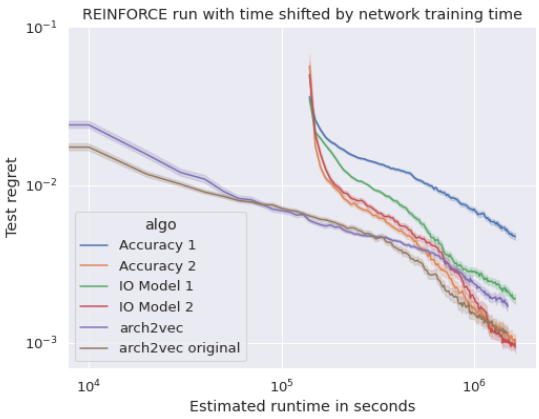

Figure 6: Reinforce search — 100 runs on NAS-Bench-101, time shifted by training time.

We ran the REINFORCE search on NAS-Bench-101 and CIFAR-10 dataset, and compared the results of the IO model, Accuracy model and arch2vec. We used embeddings extracted after 10 epochs of training. Training of the IO and Accuracy model is described in the Appendix in Section F, and details about the REINFORCE experiment are in Section G. We report the test regret, i.e. the difference between the maximal accuracy of the search space and the accuracy of the best performing network found during the search. Model 1 and model 2 denote different embedding methods also described in the Appendix. We can see that both Accuracy and IO models perform the same as the original arch2vec when the same embedding method is used (model 1). The arch2vec trained jointly with the IO model and the other embedding method (model 2) performs worse.

Figure 7: Pearson's r across different sample sizes.

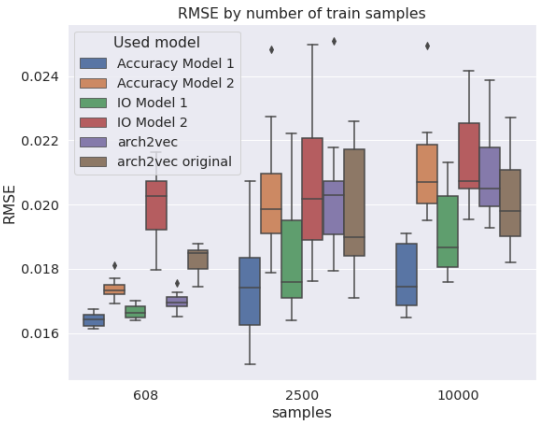

Figure 8: RMSE across different sample sizes.

To compare the models, we repeated the performance prediction experiment from the original arch2vec paper (Yan et al. (2020)). The experiment is described in Section H in the Appendix. We used a random forest in place of the gaussian process used in arch2vec due to reproducibility reasons. Figures 7 and 8 depict the distributions of Pearson's r and RMSE respectively across the 10 seeds [1]. We can see that the arch2vec is consistent across all sample sizes, while the Accuracy and IO model increase with sample size, surpassing the arch2vec models for some sample sizes. The second embedding method (flatten) performed better than the first method, and the Accuracy model yields overall better results than the IO model. Also, the original arch2vec method does not yield good results for smaller sample sizes, which is a suprising result.

7 Conclusion

In this work, we presented a novel approach to neural graph embedding. It is an extension of the graph variational autoencoder arch2vec in that it learns from input-output meta-information. We analyzed the output features and their relation to test accuracy of networks. We compared our model with a semi-supervised accuracy predictor and the arch2vec on two NAS tasks — REINFORCE-based search, and performance prediction with random forest. On the performance prediction task, we outperformed the arch2vec on larger sample sizes, while in the REINFORCE tasks, we matched the original results.

The followup work will focus on several directions. In case of the IO-dataset, an important question is if the results improve with more labeled networks or images. Another topic is the comparison of output features between networks, exploring different strategies such as the usage of metrics like CKA or utilizing feature maps instead of feature vectors. The work should be extended to other common search spaces like NAS-Bench-201 or DARTS (Dong and Yang (2020); Liu et al. (2019)), and transfer learning capabilities should be explored. Overall, we have shown that although the original arch2vec outperformed the surrogate model trained in a supervised manner during the NAS search task, the semi-supervised approaches may bring an additional improvement to the results.

8 Limitations and Broader Impact Statement

The limitations of our work are mainly in the small number of labeled networks. This may explain the validation loss results, which could improve and be more stable with a larger train set. The instability between seeds also seemed to affect the results of the NAS tasks. Another possibility is to increase the training time of the IO model.

The results of the Accuracy model in performance prediction were better that the results of the IO model. However, it is important to note that for every IO batch, the Accuracy model receives the accuracy label of the network that produced the output, leading to a larger supervision signal. As so, the results should be analyzed across different numbers of epochs. Extensions to other search spaces (NAS-Bench-201 or DARTS) might provide more insight as well.

Our research falls into the machine learning techniques area. The main impact on NAS research is the introduction of semi-supervised embedding methods that utilize meta-information. This class of methods may help to reduce the resource needs by accurately estimating the performance of networks, avoiding the training. This has a positive influence on the environmental aspect of NAS. However, the overall cost remains relatively high.

Like all NAS and AutoML methods, the societal implications are the accessibility of better models to wider application areas outside the machine learning community. The performance prediction models in particular make machine learning more available to users and researchers with less computational power than leading companies in the field.

[1] We provide the same results summarized in Tables 6 and 7 in the Appendix.

9 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 author's and review guidelines and ensured that your paper conforms to them? `https://automl.cc/ethics-accessibility/` [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
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]
   (b) Did you include the raw results of running the given instructions on the given code and data? [Yes]
   (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? [Yes]
   (d) Did you ensure sufficient code quality such that your code can be safely executed and the code is properly documented? [Yes]
   (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]
   (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] Comparisons with more methods and on more search spaces will be the subject of future work. This work is the introduction of the general concept.
   (g) Did you run ablation studies to assess the impact of different components of your approach? [Yes] More extensive evaluation will also be a part of future works.
   (h) Did you use the same evaluation protocol for the methods being compared? [Yes]
   (i) Did you compare performance over time? [Yes] In REINFORCE, we accounted for the training time of the models. The evaluation across the epochs should be explored in future extensions.
   (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? [Yes] We used the tabular benchmark NAS-Bench-101.

(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] We included the training time of the models (main paper) as well as the resources (appendix).

(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)? [N/A] Most of the hyperparameters were kept the same as the original arch2vec. Hyperparameter selection of our model was based on preliminary results which were a part of the author's thesis (anonymized).

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? [Yes] In the supplementary materials.

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

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [Yes] arch2vec is openly available

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] Applies to CIFAR-10 and NAS-Bench-101.

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. Gabriela Suchopárová was supported by Charles University Grant Agency project no. 246322. This research was (partially) supported by SVV project number 260 575.

Computational resources were supplied by the project "e-Infrastruktura CZ" (e-INFRA CZ LM2018140) supported by the Ministry of Education, Youth and Sports of the Czech Republic.

References

Dey, D., Shah, S., and Bubeck, S. (2021). Ranking architectures by feature extraction capabilities. In *8th ICML Workshop on Automated Machine Learning (AutoML)*.

Dong, X. and Yang, Y. (2020). NAS-Bench-201: Extending the scope of reproducible neural architecture search. In *International Conference on Learning Representations (ICLR)*.

Gracheva, E. (2021). Trainless model performance estimation based on random weights initialisations for neural architecture search. *Array*, 12:100082.

Hong, R. (2013). Nasbench-pytorch. `https://github.com/romulus0914/NASBench-PyTorch`.

Kingma, D. P. and Welling, M. (2014). Auto-Encoding Variational Bayes. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*.

Kipf, T. N. and Welling, M. (2016). Variational graph auto-encoders. *NIPS Workshop on Bayesian Deep Learning*.

Kipf, T. N. and Welling, M. (2017). Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*.

Kornblith, S., Norouzi, M., Lee, H., and Hinton, G. (2019). Similarity of neural network representations revisited. In *International Conference on Machine Learning*, pages 3519–3529. PMLR.

Krizhevsky, A. (2009). Learning multiple layers of features from tiny images. Technical report.

Liu, C., Zoph, B., Neumann, M., Shlens, J., Hua, W., Li, L.-J., Fei-Fei, L., Yuille, A., Huang, J., and Murphy, K. (2018). Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)*.

Liu, H., Simonyan, K., and Yang, Y. (2019). Darts: Differentiable architecture search.

Lukasik, J., Friede, D., Zela, A., Hutter, F., and Keuper, M. (2021). Smooth variational graph embeddings for efficient neural architecture search. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8.

Luo, R., Tan, X., Wang, R., Qin, T., Chen, E., and Liu, T.-Y. (2020). Semi-supervised neural architecture search. *ArXiv*, abs/2002.10389.

Luo, R., Tian, F., Qin, T., and Liu, T.-Y. (2018). Neural architecture optimization. In *NeurIPS*.

Ning, X., Zheng, Y., Zhao, T., Wang, Y., and Yang, H. (2020). A generic graph-based neural architecture encoding scheme for predictor-based nas. In Vedaldi, A., Bischof, H., Brox, T., and Frahm, J.-M., editors, *Computer Vision – ECCV 2020*, pages 189–204, Cham. Springer International Publishing.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

Philipp, G. (2021). The nonlinearity coefficient - A practical guide to neural architecture design. *CoRR*, abs/2105.12210.

Shi, H., Pi, R., Xu, H., Li, Z., Kwok, J., and Zhang, T. (2020). Bridging the gap between sample-based and one-shot neural architecture search with bonas. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1808–1819. Curran Associates, Inc.

Wei, T., Wang, C., Rui, Y., and Chen, C. W. (2016). Network morphism. *CoRR*, abs/1603.01670.

White, C., Neiswanger, W., and Savani, Y. (2021). Bananas: Bayesian optimization with neural architectures for neural architecture search. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(12):10293–10301.

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., and Yu, P. S. (2021). A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. (2019). How powerful are graph neural networks? In *International Conference on Learning Representations*.

Yan, S., Zheng, Y., Ao, W., Zeng, X., and Zhang, M. (2020). Does unsupervised architecture representation learning help neural architecture search? In *NeurIPS*.

Ying, C., Klein, A., Christiansen, E., Real, E., Murphy, K., and Hutter, F. (2019). NAS-bench-101: Towards reproducible neural architecture search. In Chaudhuri, K. and Salakhutdinov, R., editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 7105–7114. PMLR.

Zela, A., Siems, J., Zimmer, L., Lukasik, J., Keuper, M., and Hutter, F. (2022). Surrogate nas benchmarks: Going beyond the limited search spaces of tabular nas benchmarks. In *International Conference on Learning Representations (ICLR)*.

A Graph Neural Networks

The graph neural networks have been used in performance prediction for neural architecture search due to their ability to process neural graphs. One of the commonly used architectures is the Graph Convolutional Network (GCN) (Kipf and Welling (2017)).

Let us denote the node features as $X \in R^{n \times k}$ (n is the number of nodes, k is the dimension of features), the matrix of node degrees as D (D_{ii} is the number of outgoing edges from node i), and the adjacency matrix as A (A_{ij} is 0 if there is an edge from node i to node j, otherwise 1). Moreover, let the filter $g_\theta = \Theta_{ij}$ be a matrix of learnable parameters. Then, the convolution is defined as follows:

$$X = X *_G g_\theta = f(\bar{A}X\Theta), \tag{1}$$

where $\bar{A} = I_n + D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$ and $f(\cdot)$ is the activation function.

While the GCN has been successfully applied in many applications (e.g. in BONAS (Shi et al. (2020)), Xu et al. (2019) have shown that it is unable to distinguish graphs based on the embeddings the network learns. To alleviate this issue, the authors proposed the Graph Isomorphism Network (GIN). In this network, we have a stack of convolutional layers, where the node features after applying the k-th layer are computed as follows:

$$h_v^{(k)} = MLP((1 + \epsilon^{(k)})h_v^{(k-1)} + \sum_{u \in N(v)} h_u^{(k-1)}) \tag{2}$$

In this equation, $h_v^{(k)}$ are the features of node v after applying k convolutions, $N(v)$ are neighbors of node v, MLP is a multilayer perceptron, and ϵ is a learnable parameter.

Similar to the variational autoencoder composed from convolutional neural networks (Kingma and Welling (2014)), the Variational Graph Autoencoder (VGAE) was proposed (Kipf and Welling (2016)). The encoder part consists of GCN layers, and instead of a latent vector we have a latent matrix (dimensions are $n \times l$, n is the number of nodes and l is a hyperparameter). The decoder is simply the inner product between the rows of the latent matrix, only the adjacency matrix is decoded — the inner product of i-th and j-th rows is A_{ij}. Compared to VGAE, the authors of arch2vec (Yan et al. (2020)) have used the GIN layers due to their good embedding properties, and they decode node features as well.

More information on graph neural networks can be found in the survey by Wu et al. (2021).

B The arch2vec Model

The arch2vec is a variational graph autoencoder similar to VGAE (Kipf and Welling (2016)) with some differences — it uses Graph Isomorphism Network (GIN) layers (Xu et al. (2019)) instead of Graph Convolutional Network (GCN) layers (Kipf and Welling (2017)), and unlike VGAE , its output is not only the decoded adjacency matrix, but the features as well. The features $\hat{X}$ are decoded using a dense layer with softmax activation, and the adjacency matrix $\hat{A}$ through inner product of latent vectors. Since $\hat{X}$ and $\hat{A}$ are reconstructed independently, it holds:

$$P(\hat{A}, \hat{X}|Z) = p(\hat{X}|Z) \cdot P(\hat{A}|Z).$$

The optimised loss is the variational lower bound (Equation 3).

$$L = \mathbb{E}_{q_\phi(Z|X,A)}[\log p_\theta(X, A|Z)] - D_{KL}(q_\phi(Z|X, A) \,||\, p_\theta(Z)) \tag{3}$$

During the training, the input matrix A is augmented to $\bar{A} = A + A^T$ to allow information flow in both directions.

C Pretraining Details

We sampled 608 training networks and 77 validation networks from the train and test set respectively, using the same split of the search space as in arch2vec. Figure 9 shows the distribution of test losses of the chosen networks — the majority of all NAS-Bench-101 networks has an accuracy higher than 0.9, so the distribution is skewed. From the CIFAR-10 dataset, we split off a validation set of size 1 000, and pretrained the networks on the rest of the train set.

The training was done according to the NAS-Bench-101 paper (Ying et al. (2019)) with some differences. Since arch2vec is implemented in PyTorch (Paszke et al. (2019)), we did not use the original TensorFlow implementation of NAS-Bench-101, but a PyTorch implementation (NASBench-PyTorch[2] (Hong (2013)). We used the same augmentation techniques and most of the hyperparameters, although we had to alter some settings due to resource limits (batch size 128, 12 training epochs, batch normalization and learning rate set to default values).

We performed the training on a cluster with the following resources per training process:

- 16 GB GPU (nVidia Tesla T4)
- 4 core CPU (Intel(R) Xeon(R) Gold 5218 CPU @ 2.30GHz, total 16 cores)
- 16 GB RAM

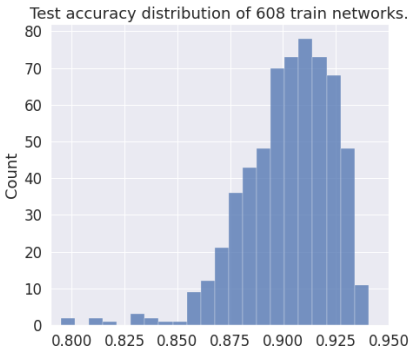

Figure 9: Accuracy distribution of the 608 networks selected for the labeled train set.

Table 1 summarizes the hyperparameters used for network pretraining on the CIFAR-10 dataset.

Table 1: Pretraining hyperparameters for NAS-Bench-101 networks — our settings.

hyperparameters	
batch size	128
initial convolution filters	128
validation size	1000
num_workers	4
num_epochs	12
gradient clipping norm	5
optimizer	SGD
initial learning rate	0.025
final learning rate	0.0
momentum	0.9
weight_decay	10^{-4}
learning rate schedule	cosine annealing

D Labeled Datasets

The train set was created by passing training CIFAR-10 images (or seen images) through training networks. Furthermore, there are two validation datasets — one with seen images and unseen (validation) networks, the second with unseen images and seen (training) networks. The latter is created from a fraction of the test set — we selected 2 000 random images from the CIFAR-10 test

[2]Used with kind permission of the author, Romulus Hong.

set (denoted test_2k), and evaluated train networks on it. We used the results of 0.1 of training networks (61 networks) to create the aforementioned validation set, and the remaining networks to create the test set. The second test set are then unseen networks evaluated on unseen images from test_2k. Table 2 lists the datasets along with the number of examples.

Table 2: Division into train/validation/test datasets according to source dataset types.

labeled	networks	CIFAR-10	dataset size
train	train	validation	608 000
validation	validation	validation	77 000
validation	0.1 train	test_2k	122 000
test	0.9 train	test_2k	1 094 000
test	validation	test_2k	154 000

E Accuracy Model

In this Section, we describe the Accuracy model. Its architecture is similar to the IO model (Figure 1), but instead of processing images, it only passes the flattened embedding through the series of dense layers. In other words, looking at Figure 1, the layers from *input image* to *concat* are excluded, and the *Z vector* is connected directly to the two *dense – dropout – ReLU* layers. The Accuracy model is trained on the same batches as the IO model, but it predicts test accuracy of networks instead of their output features.

F Training Details

As there is more labeled than unlabeled data, we interchangeably train on 300 labeled batches followed by 200 unlabeled batches. For unlabeled batches, we optimize the original loss — variational lower bound with gaussian prior. The loss for labeled batches is the sum of the unlabeled loss and L1 loss between the predicted and original *output* features.

Table 3: Default model hyperparameters

hyperparameter	
latent dimension	16
adjacency activation	sigmoid
operations activations	softmax
reconstruction loss	binary crossentropy
dropout	0.3
GIN MLP layers	2
GIN MLP features	128
GIN iterations per layer	5
batch size	32
epochs	10
labeled loss	MSE
seeds	1, 2, 3
optimizer	Adam
learning rate	10^{-3}
betas	[0.9, 0.999]
eps	10^{-8}

Table 3 summarizes the training hyperparameters used. Most of them are the same as in arch2vec, only the number of epochs is larger (originally 8 epochs). The preprocessing of neural

architectures is the same as in the arch2vec implementation (Section B). The training was performed using the following resources:

- 8 GB GPU (nVidia GeForce RTX 2070)
- 16 core CPU (Intel(R) Core(TM) i9-9900K CPU @ 3.60GHz)
- 32 GB RAM

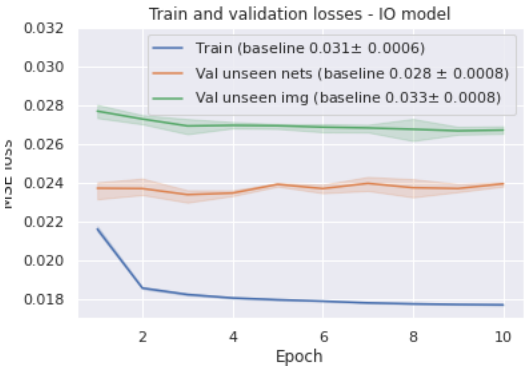

Figure 10: Train and validation losses (unseen networks and unseen images) of the IO model.

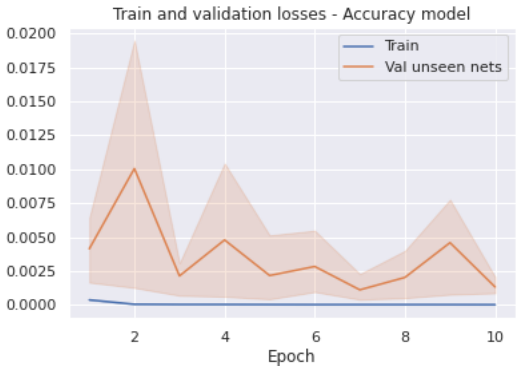

Figure 11: Train and validation loss of the Accuracy model.

We trained the models from Section 4 on the IO-dataset for 10 epochs on 3 different seeds. Figures 10 and 11 show the train and validation losses for both models. In case of the IO model, we also compute the *baseline* — we take the mean vector of the *output* features in the dataset, and we compute the corresponding MSE loss between the examples and the mean. We report the mean of the losses and its 95% confidence interval. While the IO model seems to improve on unseen images during the training, the unseen networks are challenging — although the loss is below the baseline, it does not improve during the training. The validation loss for the Accuracy model slightly decreases, but fluctuates. After the training, the IO model was evaluated on the two test sets, the results are below the baseline and summarized in Table 5. Table 4 summarizes the main metrics for both models — validity, uniqueness and reconstruction accuraccies — which however do not differ from the original arch2vec.

Table 4: Metrics after epoch 10 (validity, uniqueness and reconstruction accuracies).

model	validity	uniqueness	operators accuracy	adjacency accuracy	running time
arch2vec	0.467	0.994	0.978	0.986	-*
Accuracy Model	0.466	0.993	0.978	0.983	3544 s
IO Model	0.474	0.993	0.978	0.985	9310 s*

*IO Model and arch2vec were trained together and ran for 9310 seconds total.

Table 5: MSE loss of the IO model on the labeled test sets.

test set	loss mean	loss std	**baseline**	0.95 ci
unseen nets, unseen images	0.0279	0.0002	0.0319	0.0001
seen nets, unseen images (0.9)*	0.0299	0.0002	0.0361	0.0001

*The other 0.1 of this test set was used as the validation set.

G REINFORCE Experiment Details

For REINFORCE and performance prediction experiments, network embeddings have to be extracted using the trained models. For IO and Accuracy models, the embeddings are extracted in two different ways — the first method is the same as in the original arch2vec (Yan et al. (2020)), namely the mean of the latent node representations (denoted model 1), the second is the output of the flatten layer (see Figure 1) just before applying ReLU (denoted model 2). In case of arch2vec, we evaluate the arch2vec trained on the same batches as the IO model, as well as 'arch2vec original' trained as in the original paper (for 8 epochs).

We ran the REINFORCE search 100 times for all models, and we used the same run settings as in the original arch2vec paper — estimated 10^6 seconds limit per one run (via querying NAS–Bench-101). Additionally, we report results with time shifted by the training time of the 608 networks used for IO and Accuracy model training (Figure 6) — to each timepoint, the sum of network training times (approximately $1.3 \cdot 10^5$ seconds) is added. This way, we account for the pretraining time that is not necessary in case of arch2vec.

H Performance Prediction Experiment Details

In this section, we descibe the performance prediction experiment from the original arch2vec paper, and how we modified it. The authors trained a gaussian process regressor on 250 randomly chosen latent features for 10 different seeds, and then predicted the performance of the other architectures. The evaluation was done only for networks with accuracy larger than 0.8. The authors reported two metrics, Pearson's correlation coefficient r (0.67 ± 0.02) and RMSE (0.018 ± 0.001), for the test accuracy prediction task.

Table 6: Pearson's r across different sample sizes.

	608	2 500	10 000
Accuracy Model 1	0.669 ± 0.004	**0.724 ± 0.012**	**0.758 ± 0.005**
Accuracy Model 2	0.683 ± 0.004	0.679 ± 0.015	0.686 ± 0.010
IO Model 1	0.686 ± 0.005	**0.713 ± 0.011**	0.732 ± 0.009
IO Model 2	0.653 ± 0.011	0.685 ± 0.017	0.690 ± 0.008
arch2vec	**0.699 ± 0.005**	0.688 ± 0.011	0.701 ± 0.008
arch2vec original	0.615 ± 0.011	0.660 ± 0.015	0.688 ± 0.009

We used the same embeddings as in the previous experiment, and we tried different train set sizes. Unfortunately, we were not able to reproduce the results, since the authors did not report the exact settings for the gaussian process. As so, we used a random forest (with 200 estimators and max features 4), since it yielded the best results from other common regressors.

Table 7: RMSE across different sample sizes.

	608	2 500	10 000
Accuracy Model 1	**0.0164 ± 0.0001**	**0.0174 ± 0.0010**	**0.0177 ± 0.0006**
Accuracy Model 2	0.0174 ± 0.0002	0.0203 ± 0.0013	0.0211 ± 0.0010
IO Model 1	**0.0167 ± 0.0001**	0.0184 ± 0.0012	0.0191 ± 0.0008
IO Model 2	0.0201 ± 0.0007	0.0208 ± 0.0016	0.0214 ± 0.0009
arch2vec	0.0170 ± 0.0002	0.0204 ± 0.0013	0.0209 ± 0.0009
arch2vec original	0.0183 ± 0.0003	0.0197 ± 0.0012	0.0201 ± 0.0009

Tables 6 and 7 summarize the results of the performance prediction experiment (same results as in Figures 7 and 8). The Accuracy model is significantly better than the other models in most of the cases, but the IO model significantly surpasses the arch2vec in Pearson's r on larger sample sizes as well.