Graph Embedding for Neural Architecture Search with Input-Output Information

Anonymous Author(s)
Affiliation
Address
email

Abstract

Graph representation learning has been used in neural architecture search, for example in performance prediction. 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. 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.

1 Introduction

Representation learning has been an essential part of the Neural Architecture Search (NAS). While some NAS systems use encodings designed for the specific search algorithm, other systems learn the representation of the neural graph during the search process, for example as a part of a performance estimator. Alternatively, unsupervised pretraining is performed to extract more general features. The advantage of this approach is that the features can represent the similarities between the architectures better than the supervised variant [20].

Although the unsupervised embedding of neural networks is a promising direction in NAS, existing works focused mostly on the neural graphs. However, neural networks with a relatively different architecture may still learn a similar function of the input (as has been shown in works on network morphism [19]). In other words, we need to gather some meta-knowledge about the learned functions to improve the representation.

In our work, we adress this issue by extending an existing model for unsupervised network embedding, *arch2vec* [20], with input-output meta-information. That is, along with the architecture embedding, we learn the output features of the network on the input images.

2 Related Work

Existing works used different models for feature extraction on input architectures. In PNAS, NAO and SemiNAS [8, 12, 13], a string-based representation of the networks is passed to a LSTM model. The graph neural networks are widely used in NAS, notably as surrogate models for performance prediction [10, 18, 22]. In BONAS [17], a GCN is trained to perform the embedding through a global node, then a regressor is trained on the embeddings to predict network accuracy. The GATES model [14] simulates the information flow through the computational graph. Unsupervised pretraining has been done through graph variational autoencoders. The arch2vec model utilizes GIN layers and inner product adjacency matrix decoding [20]. SVGe is a two-sided variational autoencoder that focuses on smooth encoding and accurate reconstruction of networks [11].

Some recent works used meta-information to improve the accuracy of performance prediction or the running time. FEAR ranks the architectures according to usefulness of extracted features by freezing most weights after a short time of training [1]. In zero-shot architecture design, the network performance is estimated by the non-linearity coefficient, a scalar statistics, without training the networks [16]. Another work introduces the coefficient of variation of untrained accuracy as a powerful estimation metric for the final accuracy [3].

3 Our solution

In this section, we first shortly introduce the original arch2vec model, then we describe the labeled input-output dataset (IO-dataset), and finally we present our extended model and its training. The whole workflow is summarized in Figure 1.

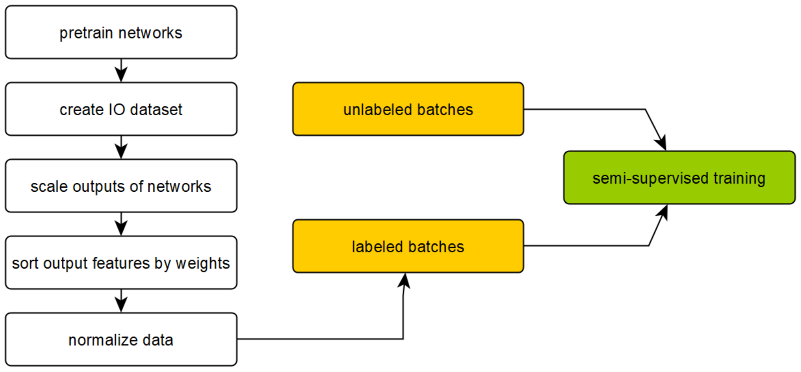

Figure 1: The training workflow of our model.

3.1 The arch2vec Model

The arch2vec is a variational graph autoencoder similar to VGAE [5] with some differences — it uses GIN layers instead of GCN, 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 1).

$$L = \mathbb{E}_{q_\phi(Z|X,A)}[\log p_\theta(X, A|Z)] - D_{KL}(q_\phi(Z|X, A) \,||\, p_\theta(Z)) \tag{1}$$

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

3.2 IO-dataset

The IO-dataset is 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.

In our work, we focus on the CIFAR-10 dataset and NAS-Bench-101 search space [6, 21]. We now describe the process of creating the dataset.

Pretraining 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. 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 [21] with some differences. Since arch2vec is implemented in PyTorch [15], we did not use the original TensorFlow implementation of NAS-Bench-101, but a PyTorch implementation (NASBench-PyTorch[1] [4]). 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). The hyperparameters along with other training details are specified in the Appendix (Section A.1).

Dataset Construction The IO-dataset is created by passing *input* images from the CIFAR-10 validation set through the pretrained networks. As the *outputs*, we use the features preceding the last dense layer (the output of the global average pooling layer), which is a vector of size 512. The predictions of the sampled training networks are the labeled train set, while the validation networks create an *unseen networks* validation set. Additionaly, the CIFAR-10 test set is used to create a second validation set — predictions of training networks on *unseen images*. More details about the dataset splits are listed in Section A.2.

Preprocessing The preprocessing of networks is the same as in the arch2vec implementation, and the *input* images from CIFAR-10 do not have to be modified for the training. However, there are two main problems with the *output* data that need to be solved.

The first problem is that there is no ordering defined on the features, in other words, the output of the global average pooling of two different networks may be the same, but permuted. The weights of the last dense layer determine the *feature importance* of the global average pooling outputs for a specific class. We use them in the following way: for an input image, we choose the dense weights corresponding to the image class, and we also append 1 to the vector to represent the bias. Then, we sort the feature vector according to the weights. Lastly, we multiply the result and the weight vector. This preprocessing ensures better comparability among the networks. It does not alleviate all shuffling errors — for example, two networks may still extract the same feature with a slightly different importance, e.g. as feature number 2 in the first network and feature number 3 in the second one — nevertheless, two very similar networks should still have a similar response to the same image.

The second issue is that the networks may produce outputs with a different scale. Since the activation of the last layer is softmax, the final output is normalized and it does not matter whether the features before applying softmax are large or small. However, in our use-case we need to make the *outputs* of networks comparable. As so, we fit a scaler for every network separately and normalize the features before sorting them. There are multiple ways to do so, e.g. scaling each feature separately or scaling all features by a global value. In the preliminary experiments, we normalize the dataset using the mean and standard deviation (scalars computed from all outputs). Lastly, the final sorted dataset is normalized once more along the features, so that all features contribute equally to the loss.

3.3 The Extended Model

In this section, we describe how we have extended the arch2vec model to learn from the IO-dataset. Figure 2 shows the overall structure of the model.

We use the original arch2vec VAE to encode the neural architecture into the latent matrix Z. The matrix is passed through the decoder, producing the first model output. In the next step, we flatten the matrix and pass it through a dense layer to create a vector representation of the input network. The second part of the model processes the input image using a sequence of 3×3 convolutional layers[2] with batch normalization and ReLU activation, followed by a global average pooling, obtaining the vector representation of the image. Finally, the network and image vectors are concatenated and passed through dense layers as to predict the output features — the second model output.

Since our dataset contains both labeled and unlabeled data, our model is trained in a semi-supervised manner. As there is more labeled than unlabeled data, we interchangeably train on 300 labeled

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

[2]We use stride 2 at the end of every convolutional block to decrease the spatial dimensions.

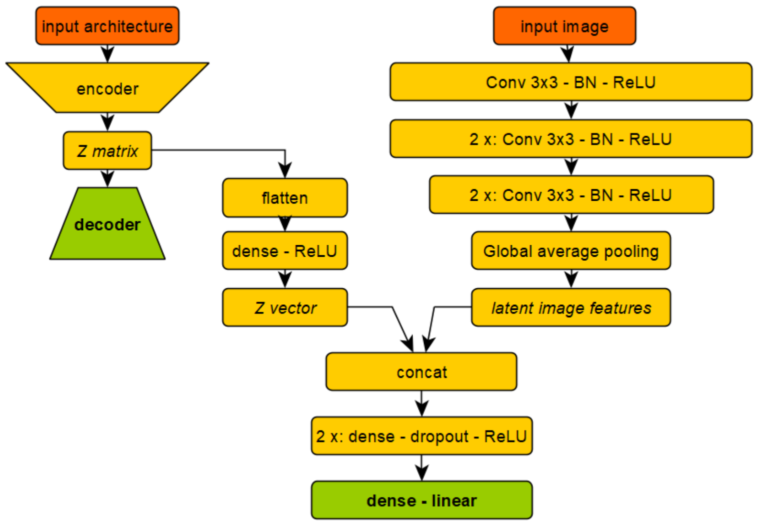

Figure 2: The architecture of the our model.

batches followed by 200 unlabeled batches. We use the same model and training hyperparameters as in the original arch2vec. 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. The full list of training hyperparameters is listed in Table 3 in the Appendix.

4 Experiments

We verified our approach on the following tasks — we analyzed the labeled validation loss and compared it with a baseline, and we compared the extended model with arch2vec. Then, we evaluated our model on two NAS task — REINFORCE search and performance prediction.

4.1 Training

We trained the model from Section 3.3 on the IO-dataset (Section 3.2) for 30 epochs. Figures 3 and 4 show the validation loss mean and bootstrapped 95% confidence interval during the training. For every dataset, we also compute the *baseline* — we take the mean vector of the *output* features in the dataset, and we compute the corresponding L1 loss between the examples and the mean. We report the mean of the losses and its 95% confidence interval. While the network generalizes well on unseen images, the unseen networks are challenging — although the loss is below the baseline, it does not improve during the training. Additional details of the training are reported in the Appendix (Section A.3). Looking at the reconstruction accuracy of operations and adjacency matrix (Figures 5, 6), the results are practically the same as in case of the original arch2vec model.

4.2 REINFORCE on NAS-Bench-101

In this experiment, we ran the REINFORCE search on NAS-Bench-101 and CIFAR-10 dataset, and compared the results of our model and arch2vec. We used two sets of latent features for each model — features extracted after 10 epochs, and after 30 epochs. We ran the REINFORCE search 500 times for the four feature sets, 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).

The results are depicted in Figures 7 and 8, we report the *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. The overall performance of the models was similar, with only small differences at the end

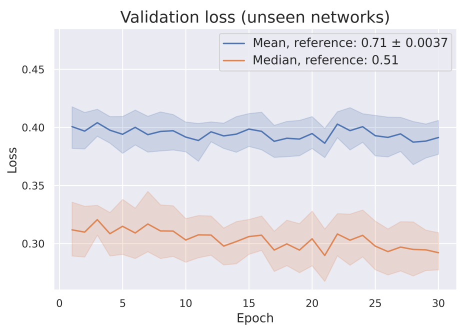

Figure 3: Unseen networks validation loss.

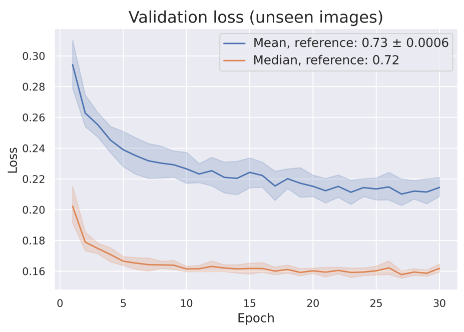

Figure 4: Unseen images validation loss.

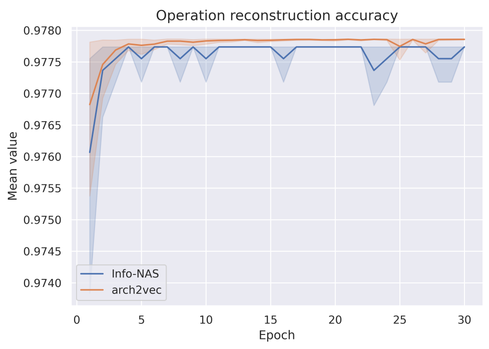

Figure 5: Operations reconstruction accuracy (validation networks).

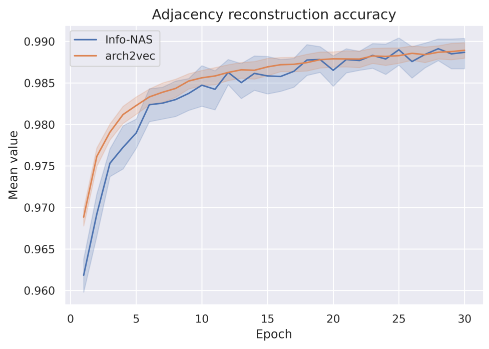

Figure 6: Adjacency matrix reconstruction accuracy (validation networks).

of the search. Arch2vec performed better in case of the validation regret, but the test regret was the same for both arch2vec runs and the 30 epoch features of our model.

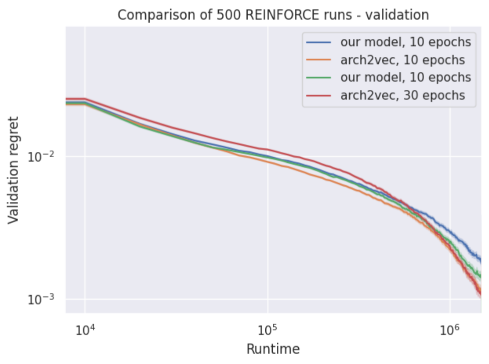

Figure 7: Validation accuracy during the search using REINFORCE.

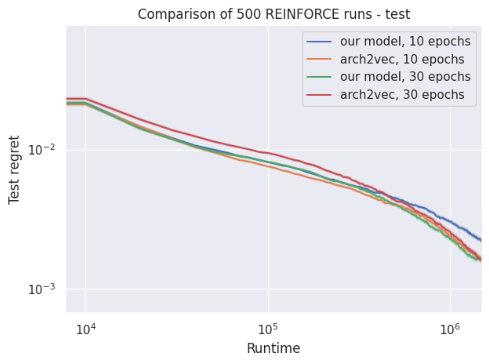

Figure 8: Test accuracy during the search using REINFORCE.

Although our approach did not bring any significant improvement to the search process, the results are not worse, given that the model had to learn two tasks at the same time. Also, since NAS-Bench-101 and CIFAR-10 are relatively easy benchmark datasets compared to other existing search spaces or datasets, the followup work should explore more challenging cases (like DARTS or ImageNet [9, 7]).

4.3 Performance Prediction on NAS-Bench-101

To compare the models, we repeated the performance prediction experiment from the original arch2vec paper [20]. 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, RMSE (0.018 ± 0.001) and Pearson's correlation coefficient r (0.67 ± 0.02), for the test accuracy prediction task.

We used features extracted using our model, the arch2vec trained concurrently with our model, and the arch2vec trained using the settings from the original paper (only 8 epochs). Additionally, we tried different train set sizes. Unfortunately, we were not able to reproduce the results, since the authors did not report the exact hyperparameter settings for the gaussian process. As so, we used a random forest, since it yielded the best results from other common regressors.

Figures 9 and 10 depict the distributions of RMSE and Pearson's r respectively across the 10 seeds. We see that both our model and arch2vec trained on the same batches yield better results on the smaller train set sizes, possibly due to the longer training time. Our model had a better median statistics than the arch2vec model on sample sizes 600 and 1 000, but overall, the results are similar.

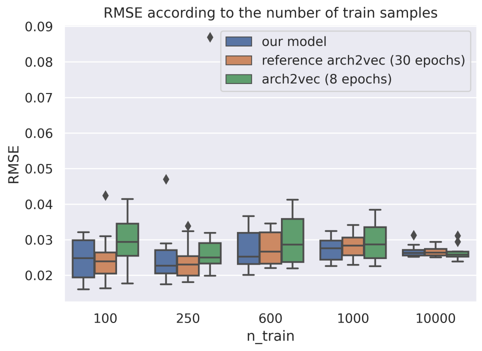

Figure 9: RMSE distribution across 10 runs for different sample sizes.

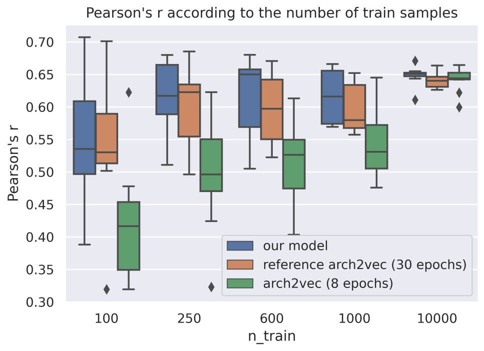

Figure 10: Pearson's r distribution across 10 runs for different sample sizes.

5 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 showed that the validation loss is better than a baseline, and we compared our model with the original arch2vec on two NAS tasks — REINFORCE-based search, and performance prediction with random forest.

The followup work will focus on multiple directions. In case of the IO-dataset, an important question is if the results improve with more labeled networks or images. During the training, other batch sampling strategies should be explored, and the problem of the unbalanced datasets should be adressed. An interesting research direction would be a generative approach for semi-supervised regression. Finally, the work will be extended to other common search spaces like NAS-Bench-201 or DARTS [2, 9].

References

[1] Debadeepta Dey, Shital Shah, and Sebastien Bubeck. Ranking architectures by feature extraction capabilities. In *8th ICML Workshop on Automated Machine Learning (AutoML)*, 2021. URL `https://openreview.net/forum?id=zOIHb2AUhE`.

[2] Xuanyi Dong and Yi Yang. NAS-Bench-201: Extending the scope of reproducible neural architecture search. In *International Conference on Learning Representations (ICLR)*, 2020. URL `https://openreview.net/forum?id=HJxyZkBKDr`.

[3] Ekaterina Gracheva. Trainless model performance estimation based on random weights initialisations for neural architecture search. *Array*, 12:100082, 2021. ISSN 2590-0056. doi: https://doi.org/10.1016/j.array.2021.100082. URL `https://www.sciencedirect.com/science/article/pii/S2590005621000308`.

[4] Romulus Hong. Nasbench-pytorch. `https://github.com/romulus0914/NASBench-PyTorch`, 2013.

[5] Thomas Kipf and M. Welling. Variational graph auto-encoders. *ArXiv*, abs/1611.07308, 2016.

[6] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[7] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, page 1097–1105, Red Hook, NY, USA, 2012. Curran Associates Inc.

[8] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

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

[10] Jovita Lukasik, David Friede, Heiner Stuckenschmidt, and Margret Keuper. Neural architecture performance prediction using graph neural networks. In Zeynep Akata, Andreas Geiger, and Torsten Sattler, editors, *Pattern Recognition*, pages 188–201, Cham, 2021. Springer International Publishing. ISBN 978-3-030-71278-5.

[11] Jovita Lukasik, David Friede, Arber Zela, Frank Hutter, and Margret Keuper. Smooth variational graph embeddings for efficient neural architecture search. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2021. doi: 10.1109/IJCNN52387.2021.9534092.

[12] Renqian Luo, Fei Tian, Tao Qin, and Tie-Yan Liu. Neural architecture optimization. In *NeurIPS*, 2018.

[13] Renqian Luo, Xu Tan, Rui Wang, Tao Qin, Enhong Chen, and Tie-Yan Liu. Semi-supervised neural architecture search. *ArXiv*, abs/2002.10389, 2020.

[14] Xuefei Ning, Yin Zheng, Tianchen Zhao, Yu Wang, and Huazhong Yang. A generic graph-based neural architecture encoding scheme for predictor-based nas. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Computer Vision – ECCV 2020*, pages 189–204, Cham, 2020. Springer International Publishing. ISBN 978-3-030-58601-0.

[15] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

[16] George Philipp. The nonlinearity coefficient - A practical guide to neural architecture design. *CoRR*, abs/2105.12210, 2021. URL `https://arxiv.org/abs/2105.12210`.

[17] Han Shi, Renjie Pi, Hang Xu, Zhenguo Li, James Kwok, and Tong Zhang. Bridging the gap between sample-based and one-shot neural architecture search with bonas. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1808–1819. Curran Associates, Inc., 2020. URL `https://proceedings.neurips.cc/paper/2020/file/13d4635deccc230c944e4ff6e03404b5-Paper.pdf`.

[18] Yehui Tang, Yunhe Wang, Yixing Xu, Hanting Chen, Boxin Shi, Chao Xu, Chunjing Xu, Qi Tian, and Chang Xu. A semi-supervised assessor of neural architectures. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1807–1816, 2020. doi: 10.1109/CVPR42600.2020.00188.

[19] Tao Wei, Changhu Wang, Yong Rui, and Chang Wen Chen. Network morphism. *CoRR*, abs/1603.01670, 2016. URL `http://arxiv.org/abs/1603.01670`.

[20] Shen Yan, Yu Zheng, Wei Ao, Xiao Zeng, and Mi Zhang. Does unsupervised architecture representation learning help neural architecture search? In *NeurIPS*, 2020.

[21] Chris Ying, Aaron Klein, Eric Christiansen, Esteban Real, Kevin Murphy, and Frank Hutter. NAS-bench-101: Towards reproducible neural architecture search. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 7105–7114. PMLR, 09–15 Jun 2019. URL `https://proceedings.mlr.press/v97/ying19a.html`.

[22] Muhan Zhang, Shali Jiang, Zhicheng Cui, Roman Garnett, and Yixin Chen. D-vae: A variational autoencoder for directed acyclic graphs. pages 1586–1598, 2019.

A Appendix

A.1 Pretraining Details

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

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

A.2 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 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

A.3 Training Details

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).

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	30
labeled loss	L1
optimizer	Adam
learning rate	10^{-3}
betas	[0.9, 0.999]
eps	10^{-8}

We also show losses for the train set (Figure 11) and the two test sets (Figures 12, 13). Although they are still below the baseline, the results are worse than in case of the validation losses, meaning that combination of unseen images and networks is challenging for the model.

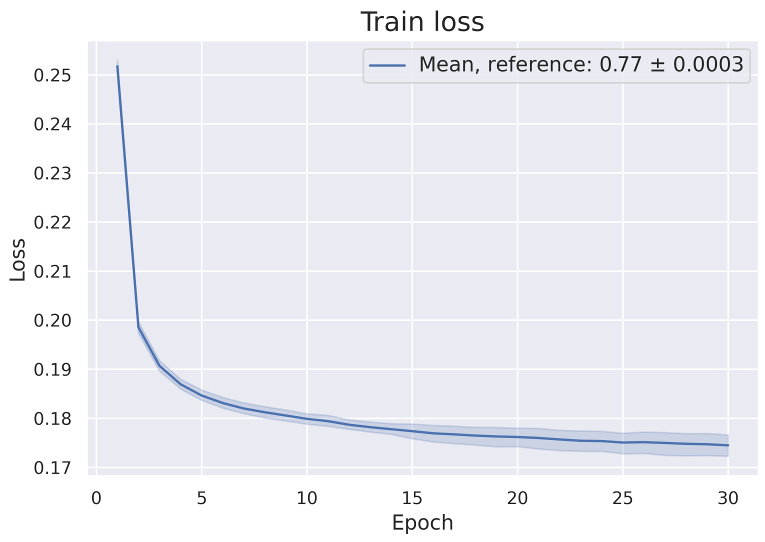

Figure 11: Train labeled loss.

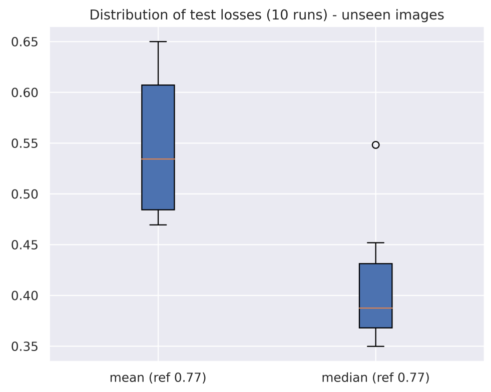

Figure 12: Test loss — seen networks, unseen images

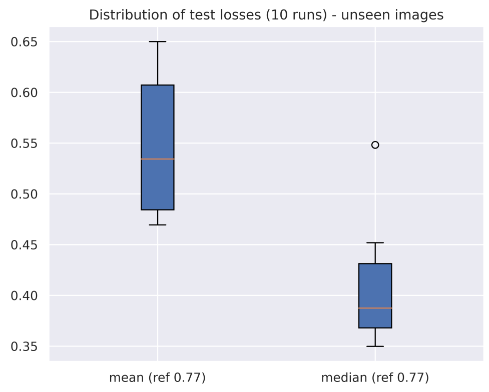

Figure 13: Test loss — unseen networks, unseen images.

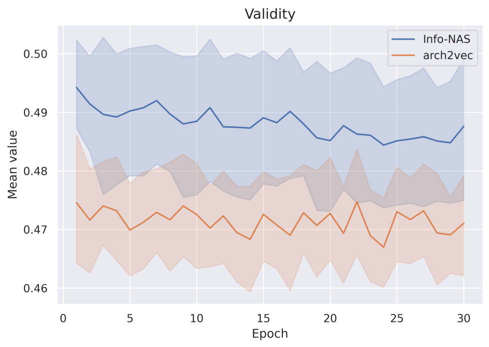

Figure 14: Validity during the training.

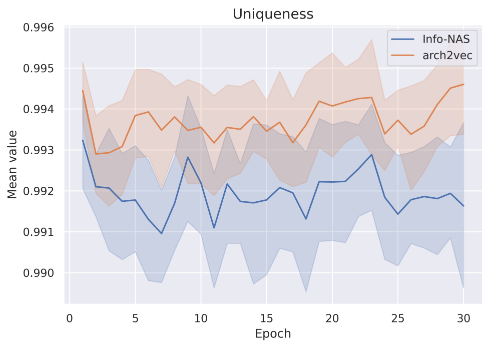

Figure 15: Uniqueness during the training.