Neural Predictor for Neural Architecture Search

Patara Trirat
School of Computing, KAIST
Daejeon, South Korea
patara.t@kaist.ac.kr

Guntitat Sawadwuthikul
School of Computing, KAIST
Daejeon, South Korea
guntitats@kaist.ac.kr

***Abstract*—Since the emerging of Neural Architecture Search (NAS), various recent studies have attempted to make the NAS algorithm faster as possible because it requires high computational costs. In this project, we select Neural Predictor to replicate due to its simplicity and effectiveness. Therefore, this project aims to reproduce and replicate the Neural Predictor. We implement Neural Predictor with Tensorflow 2 and conduct several experiments by following similar settings in the original paper. Our experiments exhibit both similar and conflicting results. Additionally, we make the Neural Predictor work with NAS-Bench-NLP, which is a novel NAS benchmark dataset with our extension. By extensively training on various additional image datasets, we also provide new trained (architecture, validation accuracy) pairs in this project.**

***Index Terms*—Neural Architecture Search, Neural Predictor, Image Classification, Deep Learning**

I. Introduction

Neural architecture search (NAS) has been emerging in deep learning, aiming to find the best architecture from a given architectural space according to the performance, which may refer to accuracy, inference time, number of FLOPs, number of parameters, and other indices [1], [2]. Many studies utilize NAS to find best-performing neural network which results in outstanding performance, outperforming tailored neural networks [1] [3]–[5]. However, the computational cost can be extremely high, training and evaluating each network individually.

A novel methodology *Neural Predictor* proposed by Wen et al [6]. tackles the training cost problem efficiently. Using basic machine learning toolbox, the neural predictor classifies the quality of an architecture without the necessity to train an arbitrary model from scratch. Results show that this methodology significantly reduces the number of models need to be trained and thus the computation time.

In this paper, our contribution includes some replications of the experiments from the original paper [6]. We apply the methodology to other search spaces and benchmark datasets to test the robustness of the predictor which reveals the durability of the predictor across a variety of datasets. Detailed contributions include:

- The neural predictor is re-implemented using Tensorflow 2.5 along with other search spaces: NAS-Bench-101, ProxylessNAS, and NAS-Bench-NLP.
- Experiments under NAS-Bench-101 section of the original paper are reproduced but in a smaller scale. The results show high correlation with the origin.
- For ProxylessNAS search space, the neural predictor is tested in other three datasets other than ImageNet, including CIFAR-100 [7], Caltech-101 [8], and Oxford-IIIT Pet [9] which turns out to be satisfying only in some cases.
- Another search space, NAS-Bench-NLP [10] is added to the experiment for evaluation of the neural predictor. The performance of the predictor is mediocre due to the fixed set of hyperparameters.

[1]According to ImageNet benchmark dataset, the proposed network from the cited studies rank in the first place at the time of publishing.

II. Related Work

Several methods have been proposed to reduce the search cost and maintain the efficiency, in terms of the quality of the output model, of the search algorithm.

A. Sampling-based Techniques

Sampling-based NAS works iteratively to find a better candidate until there is no potential candidates for the best network. Leveraging the policy gradient for NAS, reinforcement learning [11] can be utilized to sample network and evaluate it after training it for a few epochs. The policy gradient sampler is later replaced by evolutionary algorithm which helps discover new models without human aid [12], [13]. Alternatively, the iterative sampling can also be improved using Gaussian process with Bayesian optimization as seen in [14], [15].

This technique can be computationally expensive since all sampled models should be trained from scratch for at least a few epochs. It cannot be parallelized when a network should be evaluated before a new network is methodologically sampled. In addition, tuning the hyperparameters can be burdensome. One needs to readjust the training settings and retrain from the beginning.

B. Weight and Parameter Sharing

Without the need to train the model from scratch, weight and parameter sharing proposes joint training of many networks or trains the super network at once and in the end extract the subnetwork as the output model [16]–[18]. Bi-level optimization is another technique that is commonly used in parameter sharing [19].

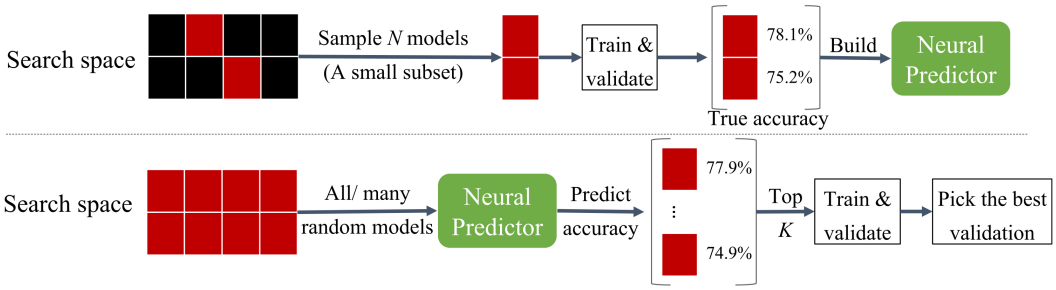

Fig. 1: Building (top) and applying (bottom) the Neural Predictor.

An approach grows and transforms a network from one to another, utilizing the pre-existing weights trained in the previous network [20]. No intensive training has to be performed which prevents traning from scratch. This way the trained weights can be reused as the network is growing. Another approach proposes a super network that encloses subnetwork candidates inside and trains the candidates altogether [21].

With this technique, NAS becomes more efficient as the training time is saved via weight reuse. However, these algorithms still have issues during hyper parameter tuning that requires the search to start over. Parallel computation is also not supported.

C. Predictive-based Methods

Instead of training and evaluating the network, performances can be estimated based on its low-fidelity performance that requires cheaper cost [1]. The network itself can be represented as numerical representations as well to aid in training the predictor [22].

There are several prediction-based approaches that rule out low-potential candidates including using a surrogate function to predict without training [4], referring from the learning curve [23], using continuous optimization to derive the best architecture from a given network [24], finding intrinsic similarities of high-potential networks [25], or even using the idea of super network and zeroing out unnecessary operations [16].

As an extension to weight sharing, another proposed method suggests to predict the weights based on the structure of the network instead of training the weights and reuse them [17].

The original paper [6] suggests a prediction-based method that solves all shortcomings aforementioned – a *Neural Predictor* – as shown in Fig. 1. This method is **sample efficient**, requiring a very small number of networks that need to be trained and also utilize those training results to train the predictor. To train these networks, due to its **parallelizability**, the tasks can be separated to multiple computing cores which immensely reduces the training time depending on the available resources. The predictor is then used to predict the quality of all candidates so that only well-performed networks will be further validated to obtain the best one. Also, this approach is **hyperparameter tuning-friendly**, to tune the neural predictor can be done at very low cost since the size of the regression model is very small.

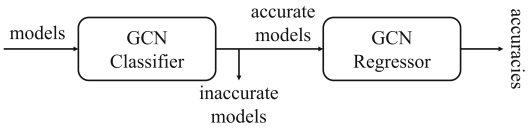

Fig. 2: Two-stage Predictor is a cascade of a classifier and a regressor. The classifier filters out inaccurate models and the regressor predicts accuracies of accurate models.

III. Neural Predictor

Naive but efficient, the Neural Predictor consists of three main steps as shown in Fig. 1. We construct the predictor based on the training results of N models and use it to estimate all networks. Only K networks with the highest accuracy will be used to implicitly train and evaluate to obtain one best network. The three steps of the predictor are:

- **Build a predictor**: N network are trained from scratch to obtain the validation accuracy from the training. The regression model is then trained on the small dataset of the network representation and the accuracy pairs.
- **Quality prediction**: All networks will be encoded to numerical representations and input to the predictor. Predicted accuracy will be ranked in order to obtain best K networks.
- **Final validation**: Final K networks are trained in a traditional fashion to search for the best one.

Considering only heavy computations such as training networks from scratch, the time complexity is $O(N + K)$ which varies from dataset to dataset as discussed in Section IV. Fortunately, these trainings can be parallelized if there are resources available. An only step that does not involve parallel computation is to train the predictor which is already a light-weighted task.

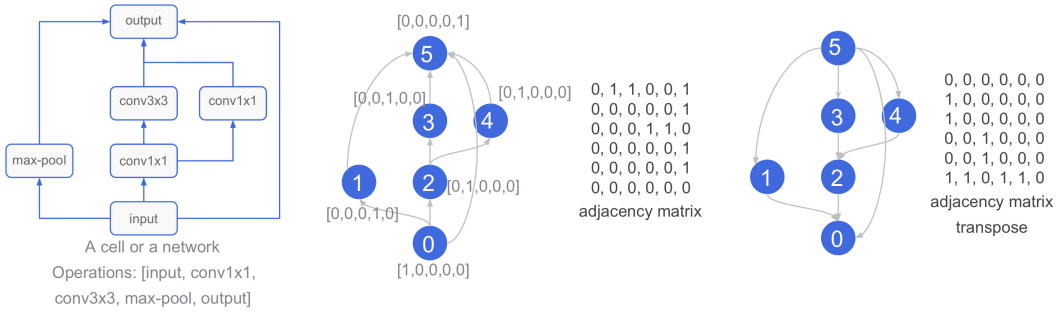

Fig. 3: An illustration of graph and node representations. (Left) A neural network architecture with 5 candidate operations per node. Each node is represented by a one-hot code of its operation. (Middle) The one-hot codes are inputs of a bidirectional GCN, which takes into account both the original adjacency matrix and (Right) its transpose.

A. *Two-stage Neural Predictor*

Instead of a single regressor, the neural predictor behaves like a two-stage predictor consisting of one classifier attached to one regressor. The classifier first filter outs low-potential networks where networks with only 91% validation accuracy or higher remain in the training set. Those high-potential networks, allowing a narrower accuracy range, are then used to accurately train the regressor. Fig. 2 illustrates this process.

B. *Neural Predictor Architecture*

The regression model is constructed from Graph Convolutional Networks (GCNs) as it is suitable for learning representations of graph-structured data. The authors of the original paper decided to apply bidirectionality to typical GCNs as it would help the information flows in both directions. An ablation study on the variants of the predictor's architecture is also provided in Section IV.

Consisting of Graph Convolutional Layers that are represented as matrices, these stacked layers enable the regression model itself to learn high quality node representation. One additional fully-connected layer is attached to the very end as the model will be used to predict the accuracy that considers all layers as a whole.

IV. Main Experiments

This section presents the analysis of Neural Predictor's behavior in the controlled environment from NAS-Bench-101. Then, we show its prediction and ablation study results. The source code is available at https://github.com/itouchz/Neural-Predictor-Tensorflow.

A. *Dataset Description*

1) NAS-Bench-101 [26]: NAS-Bench-101 is a dataset used to benchmark NAS algorithms. This dataset contains train time, validation, and test accuracy from 423,624 models in the search space. Each model was trained and consistently evaluated three times to prevent biases from the implementation from skewing results. It recommends using only validation accuracies during a search and reserving test accuracies for the final report to avoid overfitting. NAS-Bench-101 uses a cell-based NAS [3] on CIFAR-10 [7]. Each cell is a Directed Acyclic Graph (DAG) with up to 7 nodes. There is an input node, an output node, and up to 5 interior nodes. Each interior node can be a 1×1 convolution (conv1x1), 3×3 convolution (conv3x3) or max-pooling layer (max-pool). One example is shown in Fig. 3 (left). In each experiment, we use the validation accuracy from a single run as a search signal. The single run is uniformly sampled from these three records. It simulates training the architecture once. The test accuracy is only used to report the accuracy of the model selected at the end of a search.

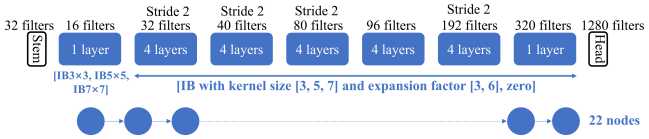

Fig. 4: ProxylessNAS Search Space.

2) ProxylessNAS [18]: The ProxylessNAS search space does not have the cell-based structure from NAS-Bench-101. It instead requires independent choices for the individual layers. The layers are divided into blocks, each of which has its fixed resolution and a fixed number of output filters. We search over which layers to skip and what operations to use in each layer. Here, the first layer of a block is always present. There are approximately 6.64×10^{17} models in the search space. As illustrated in Fig. 4, only convolutional layers in blue are searched. The optional operations in each layer are six types of Inverted Bottleneck (IB) [27] (with a kernel size 3x3, 5x5, or 7x7 and an expansion factor of 3 or 6) and one zero operation for layer skipping purpose. The expansion factor in the first block is fixed as 1. Additionally, the zero operation is forbidden in the first layer of every block.

N	D	N	D
43	48	172	144
86	72	334	210
129	96	860	320

Fig. 5: Node representation size D under N.

B. Baseline Models

In this project, we follow the experimental settings of the original paper by including **Oracle** (the upper bound baseline) and **Random Search** (the lower bound baseline). Here, we do not include *Regularized Evolution* and *ProxylessNAS* as our baselines since the original paper does not propose them.

C. Implementation Details

We use Python 3 with Tensorflow 2.5, Keras, Scikit-Learn, and Spektral library to implement all variants of Neural Predictor and conduct the experiments in this project. All of the models are trained and tested on the same platform[2].

1) Hyper-parameters of the Neural Predictor for NAS-Bench-101: Neural Predictor consists of three bidirectional GCN layers, whose node representations have the same size D. The node representations from the last GCN layer are averaged to obtain a graph representation, followed by a fully-connected layer with hidden size 128 and an output layer. For the classifier in the two-stage predictor, we use the Adam optimizer [28] with an initial learning rate of 0.0002, the dropout rate of 0.1, and the weight decay of 0.001. The learning rate is gradually decayed to zero by a cosine schedule [29]. We train the classifier for 300 epochs with a mini-batch size of 10. The regressor in the two-stage predictor uses the same hyper-parameters but an initial learning rate of 0.0001. Node representation of size D with different training datasets of size N is listed in Fig. 5.

2) Hyper-parameters of the Neural Predictor for ProxylessNAS: Neural Predictor includes 18 bidirectional GCN layers. The node representations from the last GCN layer are averaged to obtain a graph representation, followed by two fully connected layers with hidden sizes 512 and 128 and an output layer. All Graph Convolutional layers have a node representation size of 96. We have 119 samples, 40 of which are validation samples, and 79 are training samples. We use the Adam optimizer [28] with an initial learning rate of 0.001 and a weight decay of 0.00001. The learning rate is gradually decayed to zero by a cosine schedule [29]. We train for 300 epochs with a mini-batch size of 10.

Even though the original paper provides two sets of hyper-parameters, we decide to use only the former set for all search spaces, i.e., the smaller one, because of the following reasons. First, we decide not to include the ImageNet dataset due to the limitations in computation resources, which will be later discussed. Second, the experimental results from the second set are far worse than the first set, given the dataset we have.

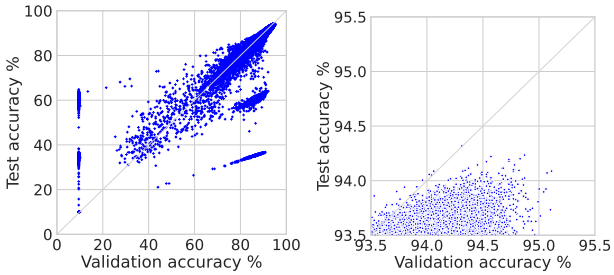

Fig. 6: Validation vs. test accuracy in NAS-Bench-101 (Left). Zoomed in on the highly accurate region. Each model (point) is the validation accuracy from a single training run (Right). Test accuracies are averaged over three runs. This plot demonstrates that even knowing the validation accuracy of every possible model is not sufficient to predict which model will perform best on the *test* set.

D. Experimental Results

1) Oracle (upper-bound baseline): To compute our upper-bound baseline, we assume that the computational resource is unlimited, which is doable due to the provided benchmark, and we find a model with the best performance in a greedy fashion. According to NAS-Bench-101, since each model is trained three times, we iterate through all models and choose one training out of three to compare for the highest validation accuracy. The iteration is performed 100 times, resulting in 95.11% on average. However, the average test accuracy of the best model turns out to be 94.08%, which might be due to overfitting during the training. The global optimum on the test set is 94.32%. However, since this model cannot be found using extensive validation, one should not expect this model to be found using any NAS algorithm. A more reasonable goal is to reliably select a model that has a similar quality to the one selected by the oracle. Our replication is shown in Fig. 6, which is almost identical to the original one.

2) Random search (lower-bound baseline): Similar to the oracle method, we perform the random search as our lower-bound baseline. With limited time, one can find the best-performing model by randomly selecting and training one model when only the best model is stored and used after the time constraint is reached. To obtain this baseline, we repeat the search for 60 times of iteration. Here we observe that even when we train and validate 200 models, which requires a relatively high computational budget, the gap to the oracle is large (Fig. 7). For the random search, the average test accuracy is about 91% compared to 94.2% for the oracle. It implies that there is a large margin for improvement over random search. Moreover, the variance is extremely high, with a standard deviation of about 3%. Finally, even evaluating 500 models, there is no improvement over evaluating 200 models at a high computational cost.

3) Neural Predictor on NAS-Bench-101: **Single-stage Predictor.** As depicted in Fig. 7, we used $N = 172$ models to

[2]Ubuntu 18.04 LTS with an NVIDIA GeForce RTX 2080 Ti GPU

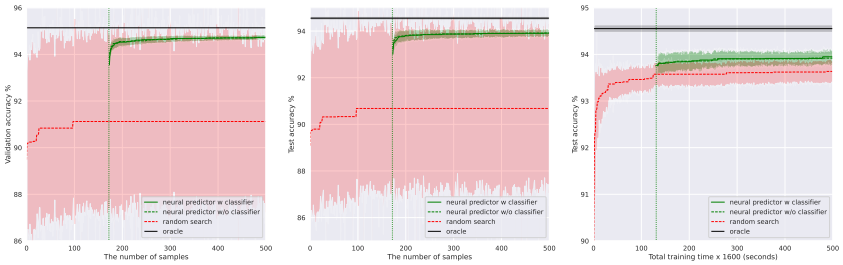

Fig. 7: Comparison of search efficiency among oracle, random search, and the Neural Predictor (with and without a two stage regressor). All experiments are averaged over 60 runs. The x-axis represents the total compute budget $N + K$. The vertical dotted line is at $N = 172$ and represents the number of samples (or total training time) used to build our Neural Predictor. From this line on we start from $K = 1$ and increase it as we use more architectures for final validation. The shaded region indicates standard deviation of each search method.

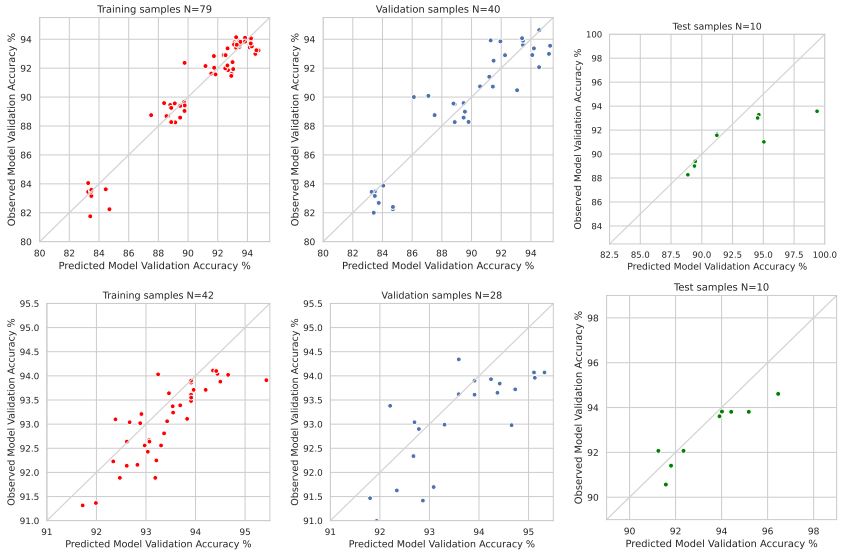

Fig. 8: Performance of Neural Predictor without (top) and with (bottom) the classifier of NAS-Bench-101 on different splits.

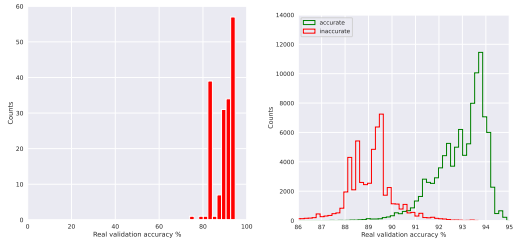

Fig. 9: The classifier filtering out inaccurate models in NAS-Bench-101. 172 models (left) are sampled to build the classifier, which is tested by unseen 100,000 samples (right).

train the predictor. Then, we vary K, the number of architectures with the highest predicted accuracies to be trained and validated to select the best one. Therefore, "the number of samples" in the figure equals $N + K$ for Neural Predictor. In Fig. 7 (left), our Neural Predictor significantly outperforms the random search. The mean validation accuracy is close to that of the oracle after about 400 samples. The sample efficiency in validation accuracy transfers well to test accuracy in terms of the total number of trained models in Fig. 7 (middle) and wall-clock time in Fig. 7 (right). Another observation is that Neural Predictor has a minor search variance.

Two-stage Predictor. If we only use a single stage, the MSE for the validation accuracy on the test set is 5.691. By introducing the filtering stage, this reduces to 0.859. Here, Fig. 9 shows how accurate the classifier is. It has very low False Negative Rate. In Fig. 7, we observe that the predictor outperforms the random search baseline even without the filtering stage. Therefore, the two-stage approach should be seen as a non-essential fine-tuning of the proposed method. Even though the evaluation scores are significantly different, we can see that using either a single- or two-stage predictor does not affect the final model selection. The total evaluation results are reported in Table I.

4) Ablation Study: **N vs K.** In this experiment, we study the problem of choosing an optimal N when the total number of models are allowed to train $N + K$ is fixed. Fig. 10 illustrates the study on N. A Neural Predictor underperforms with a

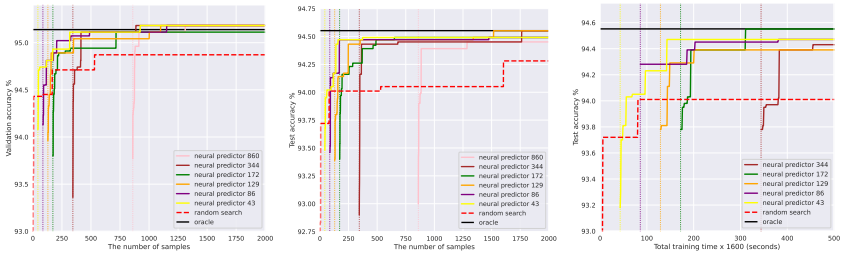

Fig. 10: Analysis of the trade-off between N training samples vs K final validation samples in the neural predictor. The x-axis is the total compute budget $N + K$. The vertical lines indicate different choices for N — the number of training samples and the point where we start validating K models. All experiments are averaged over 60 runs.

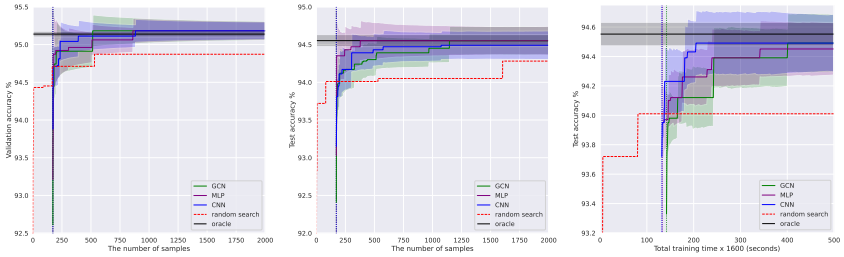

Fig. 11: The ablation study of our Neural Predictor under different architectures. Experiments are performed in NAS-Bench-101. In this study, a one stage predictor without a classifier is used. All methods are averaged over 60 experiments. The shaded region indicates standard deviation of each search method. The x-axis represents the total compute budget $N + K$. The vertical dotted line is at $N = 172$ and represents the number of samples (or total training time) used to build our Neural Predictor. From this line on we start from $K = 1$ and increase it as we use more architectures for final validation.

minimal N (e.g., 43), as it cannot accurately predict which models are interesting to evaluate. Finally, we consider the case where N is large (e.g., 860) but K is small. In this case, we notice that the increase in the quality of the GCN cannot compensate for the decrease in the evaluation budget. Note that, in Fig. 6, we show that some models are higher ranked according to validation accuracies than test accuracies. Therefore, it can cause the test accuracy to degrade as we increase K.

Neural Predictor Architectures. Fig. 11 presents an ablation study of different architectures for the Neural Predictor on NAS-Bench-101. We compare Graph Convolutional Networks (GCN), Convolutional Neural Networks (CNN), and Multi-layer Perceptrons (MLP) in the figure. To generate inputs for the MLP and CNN, we concatenate the one-hot codes of node operations with the upper triangle of the adjacency matrix. From the figure, we can see that all architectures work comparatively. The exciting finding is that, contradictory to the original paper, our CNN performs competitively to the GCN architecture that the original paper reported that they have failed to implement the Neural Predictor with the CNN architecture.

V. Additional Experiments and Modifications

A. Dataset Description

1) NAS-Bench-NLP [10]: This dataset has the same goal as the NAS-Bench-101 but is trained with NLP datasets. Specifically, it was trained with the Penn Tree Bank dataset. There are

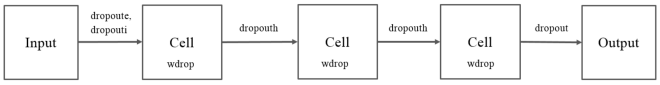

Fig. 12: Macro-level of the NAS-Bench-NLP search space.

two levels of search space, including macro-level and micro-level search. For the macro-level, the network consists of three stacked cells with weightdrop regularizations in each and locked dropouts between them and input/output layers, as well as a dropout, applied to input embedding (see Fig. 12). For the micro-level, the search space is defined for cells (micro-level models) to include all conventional recurrent cells (e.g., RNN, LSTM, and GRU) as particular instances. Cell computations are encoded as attributed graphs: each node is associated with an operation, and edges encode its inputs. In each node, there is one of the following operations: Linear, Element-wise blending, Element-wise product, and sum, and Activations (e.g., Tanh, Sigmoid, and LeakyReLU). The number of nodes is up to 24, the number of hidden states is up to 3, and the number of linear input vectors is up to 3.

The following datasets are selected to replace the ImageNet dataset that is used in the original paper.

2) CIFAR-100 [7]: This dataset is just like the CIFAR-10, except it has 100 classes containing 600 images each. There are 500 training images and 100 testing images per class.

3) Caltech-101 [8]: Caltech-101 consists of pictures of objects belonging to 101 classes, plus one background clutter class, each of which is labeled with a single object. There are roughly 40 to 800 images per class, totaling around 9k images.

4) Oxford-IIIT Pet [9]: This dataset contains 37 category pet images, with roughly 200 images for each class. The images have significant variations in scale, pose, and lighting. In addition, all images have an associated ground truth annotation of the breed. As it was introduced during the class exercise, we also decide to include this one.

5) Hyper-parameters of Image Classification Models: We apply the following settings provided by the original paper to train and test the randomly selected models from the ProxylessNAS search space for generating train, validation, and test sets for the Neural Predictor. Here, the batch size is 128. The number of epochs is 90. We adopt the RMSprop optimizer with momentum 0.9, decay 0.9, and epsilon 0.1. The learning rate is decayed according to a cosine schedule. Then, batch normalization is applied with epsilon 0.001 and momentum 0.99. Convolutional kernels are initialized with He Initialization, and bias variables are initialized to zero. The fully connected layer is initialized with a mean of 0 and a standard deviation of 0.01. We used an L2 regularization with a decay rate of 4×10^{-5} for all convolutional layers except for the final fully connected layer. The input image size is 224×224 with MobileNetv2 preprocessing (changed from ResNet due to the accuracy improvement).

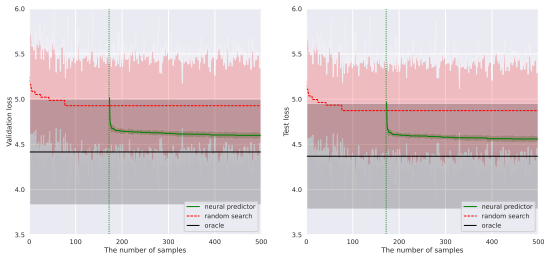

Fig. 13: Comparison of search efficiency among oracle, random search, and the Neural Predictor (with and without a two stage regressor). All experiments are averaged over 60 runs. The x-axis represents the total compute budget $N + K$. The vertical dotted line is at $N = 172$ and represents the number of samples used to build our Neural Predictor. From this line on we start from $K = 1$ and increase it as we use more architectures for final validation. The shaded region indicates standard deviation of each search method.

B. Experimental Results

1) ProxylessNAS with Image Datasets: Here, as shown in Fig. 14 and Table I, we test the generalization of our Neural Predictor to unseen test architectures (right). Our Neural Predictor can still work, yet not with satisfactory results in some cases (low Kendall's rank coefficient and R-squared). These results are from the models we directly train from the ProxylessNAS search space with the above training hyperparameters. All trained (architecture, validation accuracy) pairs are provided in our GitHub repository.

2) Performance Comparison on NAS-Bench-NLP: Similarly to the above case, we sample the model from the NAS-Bench-NLP and train our Neural Predictor. Even though we make the Neural Predictor work with this search space, the performance is still insufficient due to the fixed set of hyperparameters. The results are presented in Fig. 13 and Table I.

C. Modification on Neural Predictor

1) Reshifted Sigmoid Activation: In the original paper, they shifted output to be between 10% and 100%. However, the model with this setting cannot correctly predict architectures with very low validation accuracy. Therefore, we introduce an argument to the model whether to shift the sigmoid function back or remain the original one.

2) Loss Mode: Since the metric in NAS-Bench-NLP is loss values, not accuracy, as in other datasets, we need to replace the sigmoid activation function of the output layer with other activation functions. Thus, we introduce an argument for selecting a mode for the Neural Predictor to predict whether accuracy or loss by replacing the sigmoid activation function with ReLU.

3) New Input Preprocessing: For the same reason as the previous case, we need a new input preprocessing to represent the graph and its nodes for RNN-based architectures. Therefore, we make a new input preprocessing function by directly traversing the architecture generated from NAS-Bench-NLP search space to produce the node representation and adjacency matrix that can work with the Neural Predictor.

VI. Limitations and Discussion

A. Unable to Parallelize

As parallelizability is the original paper's key idea, they conduct their experiments with a set of multi-cores TPUs. Given this setting, we cannot exactly follow all the original experiments. Therefore, we decide to reduce the scale of the experiment, as presented above. Second, we decide to replace the ImageNet datasets with the smaller ones because training on ImageNet datasets, even with a few epochs, requires a large amount of time. It will be roughly a month if we follow the same experimental settings in the original work.

B. Hyperparameter Optimization

As shown above, the Neural Predictor performance is somewhat insufficient and underperform. It is because we do not perform any hyperparameter tuning due to the time limit. Undoubtedly, the performance will be improved if we conduct the hyperparameter optimization for each dataset.

C. Inaccessible Resources

Another experiment that we exclude is finding a high-quality mobile model with frontier models. The reason is that they use an inference latency prediction model to predict and filter the inference latency (when test on Pixel 1 device) time of selected

TABLE I: Performance of Neural Predictor

Datasets	Training			Validation			Test		
	MSE	**Kendall's τ**	R^2	**MSE**	**Kendall's τ**	R^2	**MSE**	**Kendall's τ**	R^2
NAS-Bench-101	0.576	0.755	0.963	2.677	0.684	0.842	5.691	0.822	0.379
NAS-Bench-101 w/ Classifier	0.379	0.705	0.524	1.063	0.651	0.407	0.859	0.796	0.703
ProxylessNAS on CIFAR-100	0.839	0.435	0.340	1.457	0.000	-0.462	0.274	0.400	0.282
ProxylessNAS on Caltech-101	6.959	0.404	0.374	9.021	0.227	0.030	9.514	0.186	-0.035
ProxylessNAS on Oxford Pet	0.865	0.283	0.178	0.224	0.225	0.179	0.768	-0.021	-0.356
NAS-Bench-NLP	0.322	0.255	0.252	0.269	0.238	0.079	0.483	0.067	-0.079

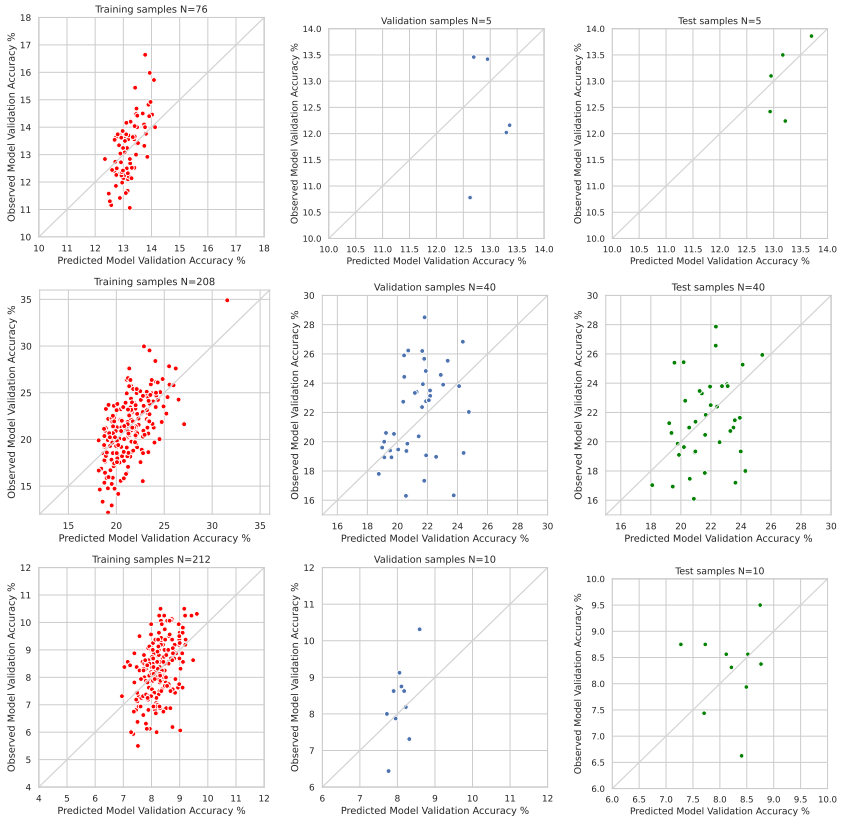

Fig. 14: The performance of Neural Predictor of ProxylessNAS with various datasets: CIFAR-100 (top), Caltech-101 (middle), and Oxford-IIIT Pet (bottom) on training, validation and test samples.

models from ProxylessNAS search space, but we cannot access those model and do not have the Pixel device. Therefore, the Pareto front is impossible to be computed in our settings.

D. Exclusion of Other Sampling Techniques

Since the search space cannot freely be accessed as the regular datasets, we cannot apply other sampling techniques here. Also, accessing already precomputed accuracy breaks the regular machine learning protocol when we separately train and test the model in real-world environments.

VII. Conclusion

This project replicates and demonstrates the results from several experiments conducted in the original paper with smaller scales. Also, we improve the Neural Predictor to work with a new search space, the NAS-Bench-NLP. Finally, we provide the trained architecture and validation accuracy pairs for further testing on Neural Predictor by extensive training on three popular image datasets.

References

[1] T. Elsken, J. H. Metzen, F. Hutter *et al.*, "Neural architecture search: A survey." *J. Mach. Learn. Res.*, vol. 20, no. 55, pp. 1–21, 2019.

[2] P. Ren, Y. Xiao, X. Chang, P.-Y. Huang, Z. Li, X. Chen, and X. Wang, "A comprehensive survey of neural architecture search: Challenges and solutions," *arXiv preprint arXiv:2006.02903*, 2020.

[3] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 8697–8710.

[4] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy, "Progressive neural architecture search," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 19–34.

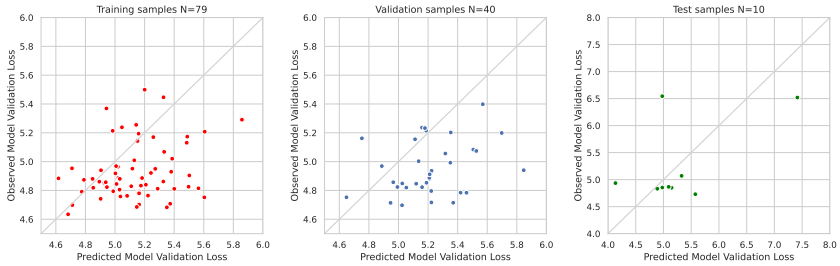

Fig. 15: The performance of Neural Predictor of NAS-Bench-NLP on training, validation and test samples.

[5] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *Proceedings of the aaai conference on artificial intelligence*, vol. 33, no. 01, 2019, pp. 4780–4789.

[6] W. Wen, H. Liu, Y. Chen, H. Li, G. Bender, and P.-J. Kindermans, "Neural predictor for neural architecture search," in *European Conference on Computer Vision*. Springer, 2020, pp. 660–676.

[7] A. Krizhevsky, "Learning multiple layers of features from tiny images," Tech. Rep., 2009.

[8] L. Fei-Fei, R. Fergus, and P. Perona, "Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories," *Computer Vision and Pattern Recognition Workshop*, 2004.

[9] O. M. Parkhi, A. Vedaldi, A. Zisserman, and C. V. Jawahar, "Cats and dogs," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2012.

[10] N. Klyuchnikov, I. Trofimov, E. Artemova, M. Salnikov, M. Fedorov, and E. Burnaev, "Nas-bench-nlp: neural architecture search benchmark for natural language processing," *arXiv preprint arXiv:2006.07116*, 2020.

[11] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *arXiv preprint arXiv:1611.01578*, 2016.

[12] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin, "Large-scale evolution of image classifiers," in *International Conference on Machine Learning*. PMLR, 2017, pp. 2902–2911.

[13] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *Proceedings of the aaai conference on artificial intelligence*, vol. 33, no. 01, 2019, pp. 4780–4789.

[14] X. Dai, P. Zhang, B. Wu, H. Yin, F. Sun, Y. Wang, M. Dukhan, Y. Hu, Y. Wu, Y. Jia *et al.*, "Chamnet: Towards efficient network design through platform-aware model adaptation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 11 398–11 407.

[15] K. Kandasamy, W. Neiswanger, J. Schneider, B. Poczos, and E. Xing, "Neural architecture search with bayesian optimisation and optimal transport," *arXiv preprint arXiv:1802.07191*, 2018.

[16] G. Bender, P.-J. Kindermans, B. Zoph, V. Vasudevan, and Q. Le, "Understanding and simplifying one-shot architecture search," in *International Conference on Machine Learning*. PMLR, 2018, pp. 550–559.

[17] A. Brock, T. Lim, J. M. Ritchie, and N. Weston, "Smash: one-shot model architecture search through hypernetworks," *arXiv preprint arXiv:1708.05344*, 2017.

[18] H. Cai, L. Zhu, and S. Han, "Proxylessnas: Direct neural architecture search on target task and hardware," *arXiv preprint arXiv:1812.00332*, 2018.

[19] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," *arXiv preprint arXiv:1806.09055*, 2018.

[20] H. Cai, T. Chen, W. Zhang, Y. Yu, and J. Wang, "Efficient architecture search by network transformation," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.

[21] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, "Efficient neural architecture search via parameters sharing," in *International Conference on Machine Learning*. PMLR, 2018, pp. 4095–4104.

[22] B. Deng, J. Yan, and D. Lin, "Peephole: Predicting network performance before training," *arXiv preprint arXiv:1712.03351*, 2017.

[23] B. Baker, O. Gupta, R. Raskar, and N. Naik, "Accelerating neural architecture search using performance prediction," *arXiv preprint arXiv:1705.10823*, 2017.

[24] R. Luo, F. Tian, T. Qin, E. Chen, and T.-Y. Liu, "Neural architecture optimization," *arXiv preprint arXiv:1808.07233*, 2018.

[25] Y. Tang, Y. Wang, Y. Xu, H. Chen, B. Shi, C. Xu, C. Xu, Q. Tian, and C. Xu, "A semi-supervised assessor of neural architectures," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 1810–1819.

[26] C. Ying, A. Klein, E. Christiansen, E. Real, K. Murphy, and F. Hutter, "Nas-bench-101: Towards reproducible neural architecture search," in *International Conference on Machine Learning*. PMLR, 2019, pp. 7105–7114.

[27] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520.

[28] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[29] I. Loshchilov and F. Hutter, "Sgdr: Stochastic gradient descent with warm restarts," *arXiv preprint arXiv:1608.03983*, 2016.