TF-GeneNAS: An evolution-based training-free approach to Neural Architecture Search

Long Doan
VinAI Research
Hanoi, Vietnam
v.longdct@vinai.io

Hieu Le, Huy Dang, Long Tran, Dao Long, Hai Minh Nguyen, Hanh Pham, Nam Nguyen
Hanoi University of Science and Technology
Hanoi, Vietnam
{hieu.ld183530, huy.dq183551, long.th183953, long.dh183579, minh.nh194120, hanh.pv183525, nam.nh183596}@sis.hust.edu.vn

Ngoc Hoang Luong
University of Information Technology
Ho Chi Minh City, Vietnam
hoangln@uit.edu.vn

Huynh Thi Thanh Binh
Hanoi University of Science and Technology
Hanoi, Vietnam
binhht@soict.hust.edu.vn

2022 International Joint Conference on Neural Networks (IJCNN) | 978-1-7281-8671-9/22/$31.00 ©2022 IEEE | DOI: 10.1109/IJCNN55064.2022.9892151

***Abstract*—Neural Architecture Search (NAS) has received much attraction from the research community in recent years. However, due to the massive amount of computational resources required, it is still infeasible to employ NAS into research and production in small labs and companies. Recent methods in NAS that aim to speed up the evaluation process are often limit themselves in other aspects, such as the search space that NAS operates on. In this work, we propose TF-GeneNAS, an evolution-based training-free NAS approach with a dynamic search space and search strategy based on Gene Expression Programming. We conduct experiments on three tasks in both Computer Vision and Natural Language Processing domains to demonstrates the effectiveness of our method. With only 3 CPU days of searching needed, TF-GeneNAS can find network architectures with better performance than previous evolution-based methods, which can require days of GPU resources, thus significantly lower the cost of searching. We also perform further studies to show the impact of our training-free estimation strategy on the NAS process. We hope that our promising results can encourage further research into more efficient evolution-based NAS methods.**

***Index Terms*—Neural Architecture Search**

I. Introduction

With recent development, Deep Learning has achieved astounding results in many fields, including Computer Vision, Natural Language Processing, and Speech Processing. One of the biggest contributors to the advancement of Deep Learning is the discovery of new neural architectures. These modern network structures have achieved state-of-the-art performance on various problems and tasks and even surpass human performance on multiple benchmarks [1]–[3]. However, designing a new neural architecture typically requires knowledge from human experts. As a result, Neural Architecture Search (NAS) emerges as a research direction that aims to automate this engineering process.

Every NAS approach can be divided into three main components: search space, search strategy and performance estimation strategy [4]. For the search strategy, Evolutionary Algorithm is amongst the most widely used method [5]–[10]. Specifically, evolution-based NAS methods encode candidate networks and apply evolutionary operators to explore new solutions, which results in a much more dynamic and versatile search space compared to other approaches. On one hand, this high degree of freedom in the search space has allowed NAS to search for new neural architectures that go beyond human knowledge. On the other hand, because the search space of evolution-based NAS is too large, the NAS process needs to consume a massive amount of computational resources to obtain high-performing solutions. As such, many research with evolutionary approaches have tried to either limit the search space [6], improve the search algorithm [5], [10] or speed up the evaluation process [5], [11], [12]. While these works have contributed greatly to the progress of evolution-based NAS, it is still infeasible to use this approach to search for architectures on large-scale datasets, especially in the case of small research labs and companies.

In the literature, NAS has been used in various tasks and domains [13]–[15]. However, most of the existing research in NAS only focuses on the image classification task, which can be considered as easier to train and evaluate than other problems. For many other problems in other domains, such as NLP, these problems do not get much attention from researchers while being equally important due to them taking up too many computation resources.

In this work, we propose Training-free Gene Expression Programming Neural Architecture Search, or TF-GeneNAS, an evolution-based training-free NAS approach with a dynamic search space and search strategy based on Gene Expression Programming. We conduct experiments on three tasks in both Computer Vision and Natural Language Processing domains to demonstrates the effectiveness of our method. With only 3 CPU days of searching needed, TF-GeneNAS can find network architectures with better performance than previous evolution-based methods, which can require days of GPU resources, thus significantly lower the cost of searching. We also perform fur-

978-1-7281-8671-9/22/$31.00 ©2022 IEEE

ther studies to show the impact of our training-free estimation strategy on the NAS process. To our knowledge, this is the first work that includes training-free performance estimation strategy to the evolution-based NAS.

The structure of this paper is organized as follows: Section II introduces other researches that influence this paper; Section III presents the proposed method TF-GeneNAS; experiment settings and results to verify the effectiveness of TF-GeneNAS is presented in Section IV; and the final section discusses the conclusions and future works of this study.

II. Related Works

In this section, we introduce existing works that are related to our proposed method. First, we discuss about NASGEP [5], a NAS approach based on Gene Expression Programming. Second, we discuss some performance estimation strategies used in NAS methods with an evolution approach.

A. NASGEP

In [5], the authors proposed a search strategy based on SL-GEP called NASGEP on image classification task. In NASGEP, a candidate network is represented by a SL-GEP chromosome, and each node in the chromosome now represents a neural network module instead of a math function. The search space that NASGEP operates on is based on NASNET [16] search space, which is a cell-based search space for two network blocks: normal cell and reduction cell. Normal cells and reduction cells are evolved separately and combined together to create a candidate network to evaluate. However, the parallel search for normal cell and reduction cell is inefficient as the choosing of which normal cell goes with which reduction cell is random. In our approach, we present a chromosome representation that can overcome the limitation of evolving multiple blocks in parallel of NASGEP.

B. Performance estimation strategy for Evolution-based NAS

There are several works before that aim to speed up the evaluation process in NAS methods with Evolutionary Algorithm as the backbone. NASGEP introduced a new performance estimation strategy that reduced the computation needed by training each candidate network for only 1 epoch, giving rewards to high-performing blocks and saving parameters of each network block into memory. With only a limited GPU search time of 1 day, NASGEP can find networks with competitive performance to SOTA methods. However, this method is still limited as it requires training the candidate network for at least 1 epoch, which is costly on huge datasets or on other search spaces such as recurrent network.

Random Weight Evaluation (RWE) [12] is a performance estimation strategy that relies on training only the classification head. In particular, RWE froze the whole network and performed training on five classification heads, with each head can only see a part of the latent output vector from the candidate network. The final performance result of the network is ensembled from these five classification heads. While reduced the training time significantly, this method still suffered the same problem as NASGEP [5] as it still requires training, and the freezing strategy is heavily prone to poor network initialization.

In [11], the authors assumed that the common evaluation method for candidate network has a bias from the model training. Specifically, the high performance of a candidate network may come from the initialization and the training process but not the architecture itself. This work proposed a NAS method that deemphasizes the importance of weights called Weight Agnostic Neural Network (WANN). WANN modified NEAT [9] as the search space and the search strategy and used a fixed weight for all the connections between neurons and search for the activation function of each neuron. The final performance of each candidate network is evaluated by aggregate the results from 6 shared values of weights. However, this method is based on NEAT and thus only limited to Shallow Neural Networks and not applicable to DNN.

Several training-free methods have been proposed in the current literature to serve as the indicator for the performance estimation [17], [18]. These indicators include trainability, a metric based on neural tangent kernel that correlates with actual performance of networks, and expressivity, which is the number of regions separated by ReLU operators. While these indicators can greatly reduce the time and cost needed for searching, they still have their own weaknesses. As trainability is calculated using the network's gradients, it may have a bias on the search dataset, which is similar to the argument made by WANN [11]. Because expressivity is dependent on ReLU operators, it is more difficult to apply to networks that do not use ReLU as an activation function, such as RNNs or LSTMs.

In this work, we follow the idea of RWE and WANN and propose a performance estimation strategy that eliminate the need of training the model by sampling the parameters from multiple distributions. The next section describes our proposed method in details.

III. Methodology

In this section, we present TF-GeneNAS: an evolution-based training approach to NAS. First, we discuss our search space and search strategy, which is based on SL-GEP algorithm [19]. Then, we introduce our proposed training-free search strategy that based on scoring on a mix of network initializations.

A. Search space and search strategy

The choice of search space is crucial for every NAS problem, as the search space defined what network can be found through the NAS process. We adopt a GEP-cell search space similar to NASGEP: each candidate in the search space is represented by two representations - genotype and phenotype. The genotype is a list of modules used in the network, and the phenotype is the tree-like structure of the network itself. The choices of neural modules to be used is flexible and can be adapted to any kind of problems. An example of genotype and phenotype is presented in Figure 1. We use a variation of GEP called SL-GEP [19] in TF-GeneNAS, which requires considerably fewer hyperparameters to tune and can

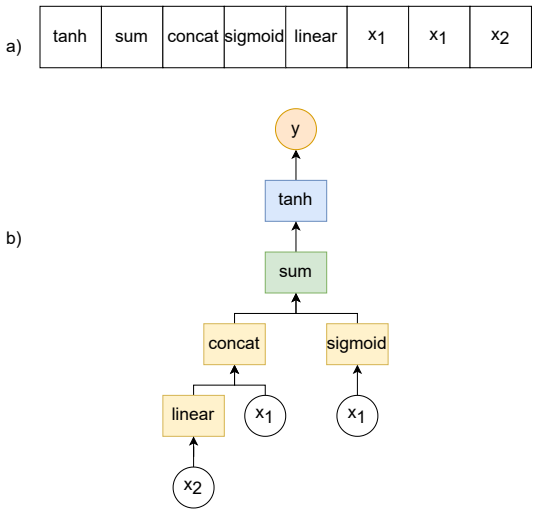

Fig. 1. Genotype and phenotype of a candidate network. a) Genotype b) Phenotype

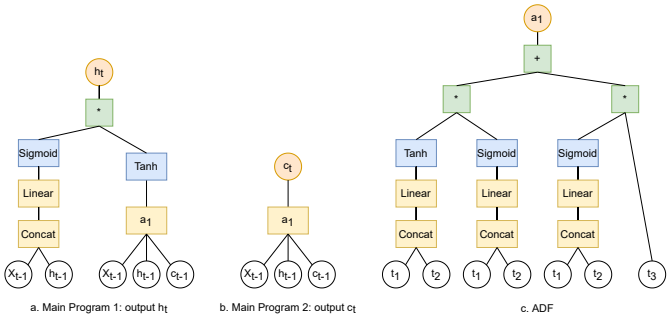

Fig. 2. Genotype of the LSTM cell

obtain optimal solutions quicker. SL-GEP also uses the BFS graph-traversing algorithm to decode from a genotype into a phenotype.

Originally, the genotype of SL-GEP only contains a single main program. The main program then can be decoded into the phenotype with a single tree-like network which returns a single output. In the cell-based search space, it is desirable for each phenotype to have multiple outputs to represent modern architecture like LSTM [20]. Furthermore, for search space that requires finding multiple cells, such as NasNet [16], the above behavior is also desirable as a single phenotype can now represent multiple cells, each cell is corresponding with an output of the phenotype. For these reasons, we increase the number of main programs in a genotype is extended from 1 to num_head, with num_head is a hyper-parameter to configure. We show how our search space can encode a LSTM cell in Figure 2 with $num\_head = 2$. In theory, our method is able to encode every modern cell-based network architectures into the corresponding genotype and phenotype, which allow us to search for the most optimal architecture.

In the search strategy, we use standard SL-GEP algorithm to search for candidate networks. The flow of our search algorithm is presented in Figure 3. We implement the MFGP

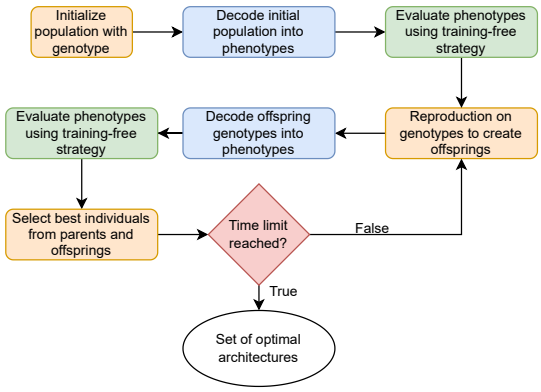

Fig. 3. Overview of TF-GeneNAS.

variant of SL-GEP [21], which represent each module as an integer instead of symbols. For simplicity, we only use uniform mutation, one-point and two-point crossover operators to produce offspring population. To select best individuals for reproduction, we use NSGA-II's multi-objective selection method [22].

B. Training-free performance estimation method

Inspired by WANN [11] and RWE [12], our performance estimation strategy aims to eliminate the need of training the candidate network, which can introduce bias to the search process. Furthermore, by entirely omitting the training, the search process can be accelerated as the computational cost for the training process is much larger than the inference process. The estimation strategy is presented as follows.

1) For each candidate network, we sample all parameters in the network from a distribution, then use these parameters directly for inference without any training.
2) Since there is no training involved in process 1, the result is heavily prone to bad initialization. To reduce such variance from the initialization, we repeat process 1 with multiple distributions. After that, we obtain two objective functions: the average and maximum model performance.
3) To further reduce the variance in the initialization, we employ the cross-validation model selection method. We split our training data into k folds and perform process 2 on each fold. Although there are cross-validation variants used for multi-objective optimization, we simply take the average results of each objective function from each fold as the final results for this process.

Figure 4 illustrates the process of our proposed estimation strategy.

IV. Experiments

To assess the capability of TF-GeneNAS, we want to answer three research questions:

- Can TF-GeneNAS find optimal networks on various tasks and domains?

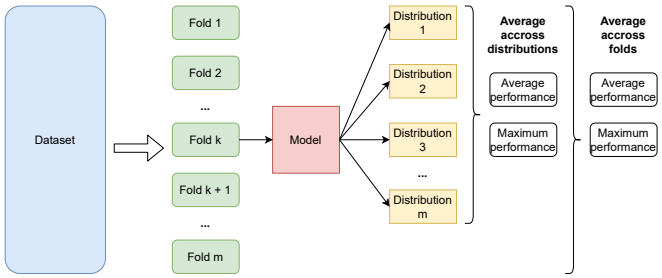

Fig. 4. Our training-free performance estimation strategy process.

TABLE I
STATISTICS OF DATASET USED IN EXPERIMENTS.

Dataset	train	validation	test	# labels
CIFAR-10	50000	-	10000	10
TREC	5452	-	500	6
CoNLL-2003	14041	3250	3453	9
SST2	67349	872	1821	2

- How well do networks found by TF-GeneNAS on one dataset generalize when being transferred to other tasks and datasets?
- What is the impact of our training-free performance estimation strategy?

We set up the experiments and discussion to provide an informed answer to each of these questions.

A. Search on various tasks and domains

1) Experimental Settings: We follow the standard NAS process by experimenting on commonly used benchmark datasets and split each experiment into two phases: *search phase* and *evaluation phase*. For the benchmark datasets, aside from the widely used CIFAR-10 dataset [23] in Computer Vision domain, we also conduct experiments on the NLP domain with two datasets: TREC [24] and CoNLL-2003 [25]. The statistics of these datasets is shown in Table I.

In the search phase, we perform searching for optimal architectures using our proposed method TF-GeneNAS on commonly used benchmarks to obtain a set of candidate networks. The details of the search space and network modules used in each benchmark dataset is listed in the Appendix A. To speed up computation, we only use a subset of 10% of training data of each dataset for searching. For the genotype representation, each genotype contains $num\_main = 2$ outputs and $num\_adf = 2$ ADFs, and the size of head in each main program and ADF is set to $h\_main = 4$ and $h\_adf = 1$ respectively. We set the population size of each generation $num\_population = 10$. As all of our benchmark datasets are in classification task, for each candidate network, we put a 1-layer MLP on top of the phenotype to calculate predictions. For two tasks involve handling text, we initialize our embeddings with GloVe 100-dimension [26]. During the performance estimation, we sample the parameters only in the phenotype and the classification head. As we do not have enough knowledge about the distribution of parameters inside a neural network, and inspired by WANN [11], we sample these parameters from four uniform distributions $\mathcal{U}(-a, a)$ with $a \in \{0.5, 1, 2, 3\}$ to try to cover up all possible parameter values. We also set the number of folds $k = 10$ and conduct this phase entirely on CPU-only machines, with the time limit of 3 days.

In the evaluation phase, we fully train each candidate networks and evaluate to obtain the final performance. We train our model until there is no improvement in evaluation metrics on the validation set after $train\_patience = 20$ epochs or reach the maximum number of training steps $max\_steps = 100000$. For datasets without a validation set, we randomly split with stratified 10% of training set as a new validation set. For simplicity, we use a fixed learning rate of $1e-3$ for all candidate networks and train without any data augmentation. To reduce the variance that occurs in the training process, for each network, we repeat the training and evaluation five times independently using a different seed and report the average result. We report the result of the best candidate architecture after training. All the experiments in this phase is conducted in Google Colab environment with a P100 GPU. [1]

2) Result on CIFAR-10 dataset: In this experiment, we compare the result of TF-GeneNAS with two baselines, one is ResNet – a popular architecture in Computer Vision, and the other is NASGEP [5] that we directly based on. For the ResNet baseline, we use ResNet-18 variant from torchvision library.[2] For the NASGEP, as the authors did not publish the code, we both use the best architecture presented in the paper and re-implement the algorithm from scratch. As NASGEP search need a single GPU, we run NASGEP on the Google Colab with a P100 GPU, which has a similar memory size compare to the Titan XP used in the paper. Table II shows the result of this experiment, and Figure 5 illustrates the best performing network found by TF-GeneNAS on CIFAR-10 dataset. Here, the estimated search cost is calculated based on the time consumed on AWS instances with similar configurations.[3]

From the table, it is clear that the best network found by TF-GeneNAS outperforms both baseline methods on CIFAR-10 dataset, at 84.70% accuracy. While our TF-GeneNAS take double the search time, it costs five times less than NASGEP, which may be beneficial for small labs or individual researchers to use. Here, our best architecture has a limited amount of convolution operators (two Separable depth-wise convolutions 3x5, two Point-wise convolutions) and result in a much lower number of parameters. Note that our model performance is lower than the reported number from [5], which can be explained as we do not include any data augmentation methods or training tricks in the evaluation phase.

3) Result on TREC and CoNLL-2003 dataset: Originally, NASGEP is proposed for the Image Classification task only, making it difficult to apply to tasks in other domains like NLP. Furthermore, NLP is underexplored in the NAS literature,

[1] https://colab.research.google.com/
[2] https://pytorch.org/vision/stable/index.html
[3] https://calculator.aws

TABLE II
RESULT ON CIFAR-10 DATASET.

NAS	Search computation	Model size	Accuracy	Estimated search cost
ResNet-18	-	11.18M	78.56	-
NASGEP (paper)	1 GPU day	3.48M	66.06	$32.41
NASGEP (our implement)	1 GPU day	3.54M	75.37	$32.41
TF-GeneNAS	3 CPU day	249K	84.70	$6.07

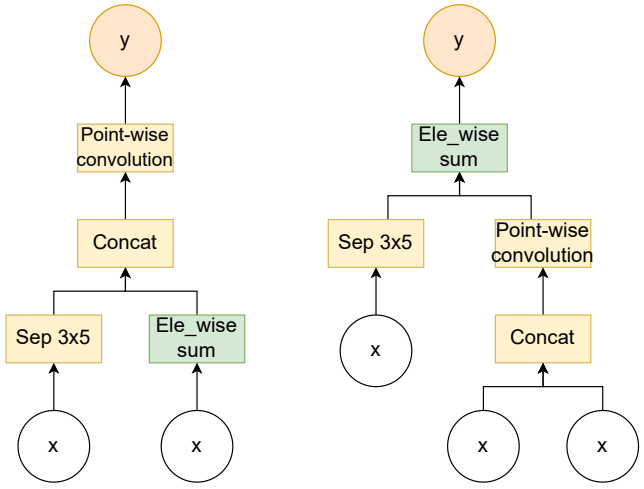

Fig. 5. Best network architecture found on CIFAR-10 dataset. Left: normal cell, right: reduction cell

TABLE III
RESULT ON TREC AND CONLL-2003 DATASET.

NAS	TREC		CoNLL-2003	
	Model size	Acc	Model size	F1 micro
BiLSTM	2.08M	66.20	2.08M	87.98
TF-GeneNAS	1.85M	84.40	4.32M	90.81

especially with evolution-based methods, which make it hard to compare our TF-GeneNAS with other NAS approaches. Thus, we only compare results of TF-GeneNAS on TREC and CoNLL-2003 with a single Bidirectional LSTM baseline. We present our results in Table III and the best models in Figure 6 and 7. Similar to results in Section IV-A2, the best network architecture from our methods still outperforms the baseline method in both tasks. However, the optimal architecture found on TREC dataset have a similar amount of parameters to LSTM, while the one discovered on CoNLL-2003 has more than double of the model size.

B. Transfer network architectures to other tasks

In this experiment, we aim to evaluate whether our best network architectures can still obtain good performance when being applied to other tasks and datasets. To do this, we design two scenarios for experiments: intra-task, in which architecture is transferred between different datasets in the same task, and inter-task, in which architecture is transferred between different tasks. For intra-task scenario, we transfer the optimal network found on TREC dataset and evaluate it on SST2 dataset [27]. The statistics of SST2 dataset is included

TABLE IV
RESULT ON TRANSFER NETWORK FROM TREC TO SST2 DATASET.

NAS	Accuracy
BiLSTM	79.56
Search on TREC	81.24

TABLE V
RESULT ON TRANSFER NETWORK FROM CONLL-2003 TO TREC DATASET.

NAS	Accuracy
BiLSTM	66.20
Search on TREC	84.40
Search on CoNLL-2003	88.60

in Table I. Because the labels of the test set is not publicly available, we consider the validation set as the new test set and split 10% of the train set as the new validation set. For inter-task scenario, we transfer the best architecture found on CoNLL-2003 dataset and evaluate it on TREC dataset. Table IV and Table V show results of this experiment. As both tables show, the optimal network found by our TF-GeneNAS can still generalize well when being transferred to other tasks and datasets. It is worth noting that our best architecture found on CoNLL-2003, which is on a different task, outperforms the one found on TREC, which is on a similar task. This can be explained that the architecture discovered on CoNLL-2003 has a larger model size than the network found on TREC, which allowing for greater learning capacity.

C. Effectiveness of our training-free performance estimation strategy

To verify the effectiveness of our training-free performance estimation strategy, during the search process, we replace it with two other baseline approaches. One approach is RWE [12], as our strategy is directly influenced by it, and the other is to train each candidate network for 20 epochs. While the second strategy is not perfect as some networks can achieve better results regardless of poor performance in first several epochs, it serves as a suitable baseline for this study. Here, we do not include NASGEP baseline as its performance estimation strategy is deeply connected with the search strategy, which is quite different from our TF-GeneNAS. We perform our experiment on CIFAR-10 dataset and use the same setup as Section IV-A1. As both these baselines require training, we use Google Colab instances with a P100 GPU and limit the search time to 1 GPU day. We present our result in Table VI. The best network discovered by our training-free performance estimation strategy obtains better performance

TABLE VI
RESULT WITH DIFFERENT ESTIMATION STRATEGY ON CIFAR-10 DATASET.

Strategy	Model size	Accuracy
Training-free	249K	84.70
RWE	245K	80.08
20 epochs	221K	82.42

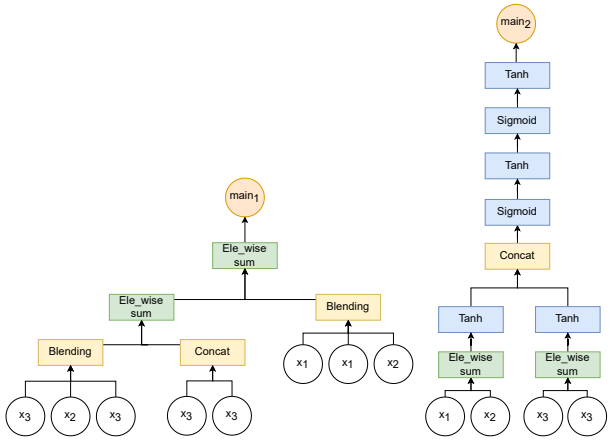

Fig. 6. Best architecture found on TREC dataset. $main_1$ and $main_2$ denote two output of the recurrent cell with $num\_head = 2$. x_1 denotes the input token, x_2 and x_3 denotes two outputs of the cell in the previous step.

than other training-limited methods, at 84.70 versus 80.08 and 82.42. This result re-confirms the effectiveness of our training-free method in both performance and cost-wise.

V. CONCLUSIONS

In this work, we have introduced TF-GeneNAS, a novel evolution-based approach for NAS. TF-GeneNAS employs a search space and search strategy with a powerful representation based on SL-GEP, which can virtually encode most modern architectures made by human experts. Furthermore, unlike previous works in NAS where the performance estimation strategy involves the training process, which can induce bias to the candidate network, TF-GeneNAS implements training-free method that can speed up the evaluation process while still maintain the performance. To demonstrate the effectiveness of TF-GeneNAS, we conduct extensive experiments on both Computer Vision and NLP domains. The experimental results show that TF-GeneNAS outperforms all baseline methods in all experiments with a reasonable search cost. To our knowledge, this is the first work that includes training-free performance estimation strategy to the evolution-based NAS.

Future works based on this study may fall into three categories: (1) Improve the search algorithm and the training-free strategy to speed up the NAS process; (2) Study on the characteristics of distributions of parameters inside a deep neural network to improve the performance estimation strategy; (3) Explore the effectiveness of TF-GeneNAS on more advanced search space, such as transformers-based search space similar to Evolved Transformer [28].

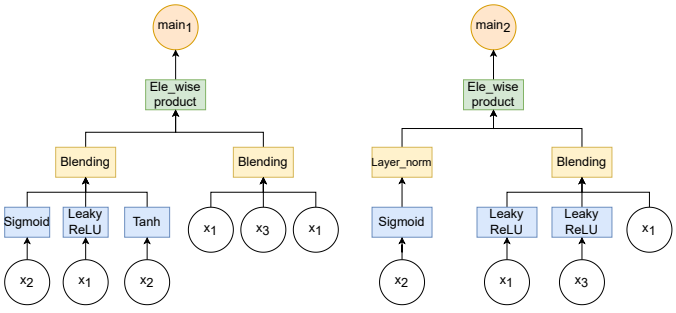

Fig. 7. Best architecture found on CoNLL-2003 dataset. $main_1$ and $main_2$ denote two output of the recurrent cell with $num\_head = 2$. x_1 denotes the input token, x_2 and x_3 denotes two outputs of the cell in the previous step.

ACKNOWLEDGMENT

This research is supported by the International Technology Center Pacific (ITC-PAC) under Contract No. FA520919PA148 for Huynh Thi Thanh Binh.

REFERENCES

[1] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4171–4186, June 2019.

[2] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," in *International Conference on Learning Representations*, 2021.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[4] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *Journal of Machine Learning Research*, vol. 20, no. 55, pp. 1–21, 2019.

[5] J. H. Alves and L. F. de Oliveira, "Optimizing neural architecture search using limited gpu time in a dynamic search space: A gene expression programming approach," in *2020 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1–8, IEEE, 2020.

[6] Z. Lu, I. Whalen, V. Boddeti, Y. Dhebar, K. Deb, E. Goodman, and W. Banzhaf, "Nsga-net: neural architecture search using multi-objective genetic algorithm," in *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 419–427, 2019.

[7] R. Miikkulainen, J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, H. Shahrzad, A. Navruzyan, N. Duffy, *et al.*, "Evolving deep neural networks," in *Artificial intelligence in the age of neural networks and brain computing*, pp. 293–312, Elsevier, 2019.

[8] 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, pp. 4780–4789, 2019.

[9] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary computation*, vol. 10, no. 2, pp. 99–127, 2002.

[10] M. Suganuma, S. Shirakawa, and T. Nagao, "A genetic programming approach to designing convolutional neural network architectures," in *Proceedings of the genetic and evolutionary computation conference*, pp. 497–504, 2017.

[11] A. Gaier and D. Ha, "Weight agnostic neural networks," in *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[12] S. Hu, R. Cheng, C. He, and Z. Lu, *Multi-objective Neural Architecture Search with Almost No Training*, pp. 492–503. 03 2021.

[13] 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.

[14] A. Mehrotra, A. G. C. P. Ramos, S. Bhattacharya, Ł. Dudziak, R. Vipperla, T. Chau, M. S. Abdelfattah, S. Ishtiaq, and N. D. Lane, "{NAS}-bench-{asr}: Reproducible neural architecture search for speech recognition," in *International Conference on Learning Representations*, 2021.
[15] 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*, pp. 7105–7114, PMLR, 2019.
[16] 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*, pp. 8697–8710, 2018.
[17] W. Chen, X. Gong, and Z. Wang, "Neural architecture search on imagenet in four gpu hours: A theoretically inspired perspective," *arXiv preprint arXiv:2102.11535*, 2021.
[18] J. Mellor, J. Turner, A. Storkey, and E. J. Crowley, "Neural architecture search without training," in *International Conference on Machine Learning*, pp. 7588–7598, PMLR, 2021.
[19] J. Zhong, Y.-S. Ong, and W. Cai, "Self-learning gene expression programming," *IEEE Transactions on Evolutionary Computation*, vol. 20, no. 1, pp. 65–80, 2015.
[20] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, p. 1735–1780, Nov. 1997.
[21] J. Zhong, L. Feng, W. Cai, and Y.-S. Ong, "Multifactorial genetic programming for symbolic regression problems," *IEEE transactions on systems, man, and cybernetics: systems*, vol. 50, no. 11, pp. 4492–4505, 2018.
[22] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE transactions on evolutionary computation*, vol. 6, no. 2, pp. 182–197, 2002.
[23] A. Krizhevsky *et al.*, "Learning multiple layers of features from tiny images," 2009.
[24] X. Li and D. Roth, "Learning question classifiers," in *COLING 2002: The 19th International Conference on Computational Linguistics*, 2002.
[25] E. F. Sang and F. De Meulder, "Introduction to the conll-2003 shared task: Language-independent named entity recognition," *arXiv preprint cs/0306050*, 2003.
[26] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 1532–1543, 2014.
[27] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Ng, and C. Potts, "Recursive deep models for semantic compositionality over a sentiment treebank," in *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pp. 1631–1642, Association for Computational Linguistics, Oct. 2013.
[28] D. So, Q. Le, and C. Liang, "The evolved transformer," in *International Conference on Machine Learning*, pp. 5877–5886, PMLR, 2019.
[29] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.

APPENDIX

A. Search space on CIFAR-10 dataset

For CIFAR-10 dataset, we follow the search space from NASGEP [5]. To handle shape mismatch from the "Concatenation" module, we add another point-wise convolution right after to project into the same feature size.

B. Search space on TREC and CoNLL-2003 datasets

For TREC and CoNLL-2003 datasets, we aim to find the optimal architecture for the recurrent cell in the recurrent network, similar to LSTM cell. Inspired by NAS-bench-NLP [13], we list the network modules used in this search space as follows:

- Linear: $f(X) = XW + b$
- Element-wise sum: $f(X_1, X_2) = X_1 + X_2$
- Element-wise product: $f(X_1, X_2) = X_1 \odot X_2$
- Element-wise blending: $f(X_1, X_2, X_3) = X_3 \odot X_1 + (1 - X_3) \odot X_2$
- Concatenation: $f(X_1, X_2) = X_1 \oplus X_2$
- Sigmoid: $f(X) = \frac{1}{1+e^{-X}}$
- Tanh: $f(X) = \frac{e^{2X}-1}{e^{2X}+1}$
- ReLU: $f(X) = max(0, X)$
- Layer norm [29]: $f(X) = \frac{X - E[X]}{\sqrt{Var[X]+\epsilon}} * \gamma + \beta$

After the genotype is decoded into the phenotype – or the recurrent cell, it will be put into the recurrent network with the architecture in Figure 8. We set the hidden size of the network to 128, number of layers to 1 and use bidirectional variant.

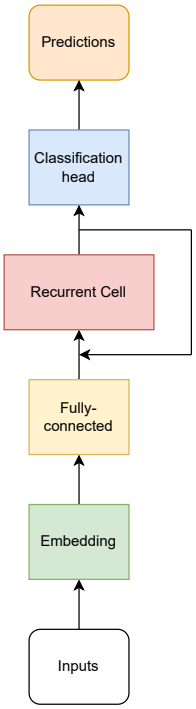

Fig. 8. Overall representation of the recurrent network in the search space of TREC and CoNLL-2003 datasets.