SIAMESE-NAS: USING TRAINED SAMPLES EFFICIENTLY TO FIND LIGHTWEIGHT NEURAL ARCHITECTURE BY PRIOR KNOWLEDGE

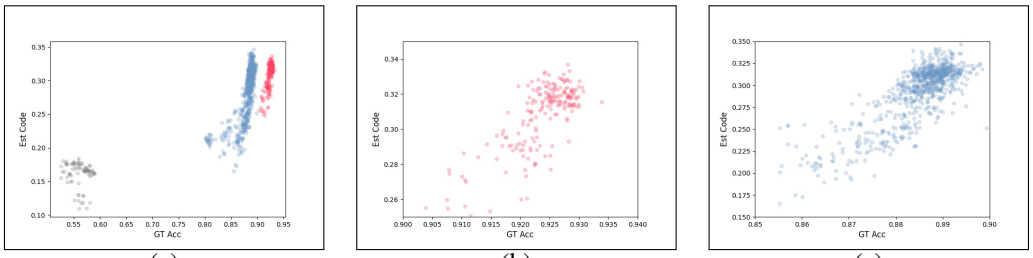
Anonymous authors
Paper under double-blind review

ABSTRACT

In the past decade, many architectures of convolution neural networks were designed by handcraft, such as Vgg16, ResNet, DenseNet, etc. They all achieve state-of-the-art level on different tasks in their time. However, it still relies on human intuition and experience, and it also takes so much time consumption for trial and error. Neural Architecture Search (NAS) focused on this issue. In recent works, the Neural Predictor has significantly improved with few training architectures as training samples. However, the sampling efficiency is already considerable. In this paper, our proposed Siamese-Predictor is inspired by past works of predictor-based NAS. It is constructed with the proposed Estimation Code, which is the prior knowledge about the training procedure. The proposed Siamese-Predictor gets significant benefits from this idea. This idea causes it to surpass the current SOTA predictor on NASBench-201. In order to explore the impact of the Estimation Code, we analyze the relationship between it and accuracy. We also propose the search space Tiny-NanoBench for lightweight CNN architecture. This well-designed search space is easier to find better architecture with few FLOPs than NASBench-201. In summary, the proposed Siamese-Predictor is a predictor-based NAS. It achieves the SOTA level, especially with limited computation budgets. It applied to the proposed Tiny-NanoBench can just use a few trained samples to find extremely lightweight CNN architecture.

1 INTRODUCTION

There are lots of Convolution Neural Network (CNN) models have been proposed that achieved great success in the past decade (Simonyan & Zisserman (2014); He et al. (2016); Huang et al. (2017); Howard et al. (2017)). Nevertheless, designing a handcrafted CNN architecture requires human intuition and experience, which is not easy to get and often not optimal. Neural Architecture Search (NAS) (Zoph & Le (2016)) focus on this problem, it search the best neural network architecture based on specific strategy on the specific search space (Liu et al. (2018b); Ying et al. (2019); Dong & Yang (2020)). Many methods have been proposed in the past. One path is to train a predictor based on Graph Convolution or Multilayer Perceptron. Although the predictor-based method achieved impressive performance with a few trained samples that are randomly CNN architecture trained on CIFAR-10 (Krizhevsky et al. (2009)) or other public datasets. However, to get the ground truth of architectures, we have to train every architecture, which needs so many GPU times, and we all know that GPU times are so expensive. In other words, we believe there is still a huge improvement gap. The proposed Siamese-Predictor combined the extra prior knowledge Estimation Code to improve the search efficiency. The proposed Estimation Code is developed from early losses of the architecture training procedure. It shows a high correlation with the final accuracy of the architecture in our experiments. Although this code shows the potential to boost the predictor, it has a huge computational cost for getting the Estimation Code of each model in the search space. So, we propose the Siamese-Ranking method to reduce this extra cost brought by extra knowledge. Besides, we also propose the training strategy Batch Top Sampling to use trained samples more efficiently by conditional training sample selection. Finally, our method surpasses the baseline on the NASBench-201 (Dong & Yang (2020)) and finds more lightweight CNN architecture on the proposed Tiny-NanoBench.



(a) (b) (c)

Figure 1: The relationship between Estimation Code and ground truth(accuracy) in Tiny-NanoBench. (a) is the relationship about all architecture, (b) is focused on the first high-scoring group in (a), (c) is focused on the second high-scoring group in (a).

2 Related Work

There are various NAS methods have been proposed in latter years, some methods based on reinforcement learning (Zoph & Le (2016); Baker et al. (2016)), some methods developed from evolutionary algorithm (Lu et al. (2019); Real et al. (2019)), and some methods focused on training a predictor to sort the search space (Wen et al. (2020); Dudziak et al. (2020)). In this papaer, we proposes the Siamese-Predictor, it is a predictor-based NAS method. Therefore, We will focus on introducing the other predictors later.

2.1 Neural Predictors

The predictor-based NAS methods are the mainstream approach, creating a predictor to predict the accuracy of CNN architecture. There are many types of these predictors (Liu et al. (2018a); Luo et al. (2018); Wen et al. (2020); Dudziak et al. (2020); Wei et al. (2022)), in recent works, the Neural Predictor (Wen et al. (2020)) is the most representative method. It encodes the architecture to graph structure embedded code that contains the adjacency matrix and feature matrix. The adjacency matrix represents the relationship between nodes. The feature matrix represents the operations used by the nodes in the cell. Since the number of cells in their meta-architecture is fixed and the structure is also fixed. So that uses multiple Graph Convolution (GCN) (Kipf & Welling (2016)) to extract the high-level feature matrix while considering directionality by the adjacency matrix. Finally, flatten it and use a fully connected layer to get the prediction accuracy, then sort search space based on it to find the promising architectures. This GCN-based predictor shows significant performance in NASBench-101 (Ying et al. (2019)). After that, the BRP-NAS (Dudziak et al. (2020)) proposes a binary predictor that takes two different architectures as inputs at the same time, and it predicts which is better in these two instead of directly predicting accuracy. This method greatly improves the performance compared to the neural predictor. We believe that this improvement is caused by the use of prior knowledge, i.e., inputs two architectures at the same time. In this paper, we explore this idea to propose the Siamese-Predictor that using losses of training procedure as prior knowledge, it improves the performance considerably in NASBench-201 (Dong & Yang (2020)).

3 Methods

This section introduces the proposed Estimation Code, Siamese-Predictor, and Siamese-Ranking. The Estimation Code is prior knowledge about the final accuracy of architecture based on training procedure. We believe that the performance of the predictor will be improved by combining prior knowledge. It inspired the past work (Dudziak et al. (2020)). They used two different architectures as inputs at the same time. They think doing this will give the predictor more information. Our observation Fig. 1 also shows a high correlation between Estimation Code and accuracy of architecture. For the proposed Siamese-Predictor, we mixed naive predictor and Estimation Code. It consists of two parts, the basic branch and the estimation branch, the consumption of the two branches in the prediction stage is inconsistent. So, the Siamese-Ranking is about how to use them efficiently to predict every accuracy of search space.

Algorithm 1 Siamese Ranking

Input: (i) search sapce $\mathcal{S}$, (ii) basic branch of siamese-predictor P_b, (iii) estimation branch of siamese-predictor P_e.
Output: ranked search sapce $\mathcal{R}$.

$\mathcal{R}_b \leftarrow \emptyset$
$\mathcal{R}_e \leftarrow \emptyset$
while $s \in \mathcal{S}$ **do**
  $s_{adj}, s_{feat} \leftarrow s$ # getting adjacency matrix and feature matrix.
  $p_b \leftarrow P_b(s_{adj}, s_{feat})$ # getting prediction by basic branch.
  insert s to $\mathcal{R}_b$ based on p_b.
end while
$\mathcal{R}_c \leftarrow Top(\mathcal{R}_b, c)$ # take out the top c.
while $s \in \mathcal{R}_c$ **do**
  $s_{adj}, s_{feat} \leftarrow s$
  $s_{estc} \leftarrow EstCode(s_{adj}, s_{feat})$ # for 3 epochs(0.3% cost of complete training).
  $p_e \leftarrow P_e(s_{adj}, s_{feat}, s_{estc})$ # getting prediction by estimaion branch.
  insert s to $\mathcal{R}_e$ using p_e.
end while
$\mathcal{R} \leftarrow$ replace $\mathcal{R}_c \cap \mathcal{R}_b$ with $\mathcal{R}_e$. # this means resorting the top c.
return $\mathcal{R}$

3.1 Estimation Code

The proposed Estimation Code is developed from the loss of model training procedure. It is an intuitive idea about the relationship between the loss value and the ground truth accuracy of an architecture. Based on this idea, We used the first three losses as the Estimation Code. In order to explore and evaluate this naive idea, we are experimenting with the Tiny-NanoBench we built. The Fig. 1 shows that the Estimation Code has a local positive correlation with accuracy, especially in the high-accuracy group. However, to get an architecture's Estimation Code (first three losses), we have to train this architecture on CIFAR-10 or the other dataset for three epochs. Considering that the whole search space contains thousands of architectures, this will cause much computational cost in the search stage. Therefore, we construct the mini-batch of CIFAR-10 by random sampling with a one-tenth ratio, then train from scratch on this mini-batch for three epochs to get the Estimation Code. In this way, the consumption of computing resources can be significantly reduced. So the consumption of Estimation Code compared to the entire training is 50,000 times 100 epochs and 5,000 times three epochs. The consumption is only 0.3% of the complete training process for an architecture, which is almost negligible. Eq. (1) shows that the idea of the proposed Estimation Code, $g(M)$ denotes the ground truth of architectures, $p(M)$ denotes the prediction accuracy of architectures by predictor, θ denotes the weights of predictor, I denotes the prior knowledge (Estimation Code), S_0, S_1, S_2 denotes the budget using of trained samples for predictor or Estimation Code. Therefore, we believe that the prior knowledge brought by Estimation Code can make the Siamese-Predictor more accurate while $S_1 + S_2$ is more minor than S_0.

$$\begin{gathered} min_{(\theta)}\ (g(M) - p(M|\theta))^2 \simeq (g(M) - p(M|S_0))^2, \\ min_{(\theta,I)}\ (g(M) - p(M|\theta, I))^2 \simeq (g(M) - p(M|S_1, S_2))^2, \\ then\ (S_1{+}S_2) < S_0 \end{gathered} \tag{1}$$

3.2 Siamese-Predictor

This section continues the previous section. We will describe how to integrate the proposed Estimation Code into the native predictor. In general, adding some extra prior knowledge to the predictor may boost the performance. But as mentioned in Eq. (1), it is important to properly control the consumption of budgets $S_1 + S_2$ to be less than S_0. This makes us build a predictor with two modes. The first mode only uses the adjacency matrix and feature matrix to predict the accuracy of architecture. The second mode uses these and the Estimation Code as the third input. So far, the proposed Siamese-Predictor consists of two branches. The Fig. 2 shows that the left branch

Algorithm 2 Batch Top Sampling

Input: (i) search sapce $\mathcal{S}$, (ii) training samples pool $\mathcal{T}$, (iii) batch size b, (iv) max iteration l, (v) training samples budgets m, (vi) update frequency f (default is 10).
Output: siamese predictor P.

$P \leftarrow Init(\theta)$ # initialize the weights of predictor.
$\mathcal{T} \leftarrow RndChoices(\mathcal{S}, \lambda m)$ # initialize the pool, and λ is between 0 and 1.
while $i < \alpha l$ **do**
  $P \leftarrow Grad(P, RndChoices(\mathcal{T}, b))$ # updating predictor by gradient of batch data.
  $i \leftarrow i + 1$
end while
while $i < (1 - \alpha)l$ **do**
  if $i\%f == 0$ **then**
    $\mathcal{S}_f \leftarrow RndChoices(\mathcal{S}, Size(S)/f)$ # random subset of search space.
    $\mathcal{R}_f \leftarrow SimsRanking(\mathcal{S}_f)$ # as shown in Alg. 1.
    $\mathcal{T} \leftarrow \mathcal{T} \cup Top(\mathcal{R}_f, (1 - \lambda)m/f)$. # this means take top $(1 - \lambda)m/f$ into pool.
  end if
  $P \leftarrow Grad(P, RndChoices(\mathcal{T}, b))$
  $i \leftarrow i + 1$
end while
return P

represents the first mode. We call it the basic branch. The right branch represents the second mode. We call it the estimation branch, and as we say, the estimation branch predicts accuracy involving prior knowledge. It should be better than the basic branch. In the fusion process of the 2-d feature matrix and 1-d Estimation Code, we are first upsampling the Estimation Code to the 2-d matrix by a fully-connected layer and reshaping operation. Then the feature matrix and upsampled Estimation Code matrix through the proposed Estimation Fusion Module fuse the feature and prior knowledge. Next, we will introduce the structure of EFM.

3.2.1 ESTIMATION FUSION MODULE (EFM)

The self-Attention has been widely used in vision CNN models in recent years (Dosovitskiy et al. (2020); Liu et al. (2021)), so we think the combination of self-Attention and Graph Convolution is worth trying. The self-Attention split an input as three different - q, k, v by linear or non-linear transformation, then fused q and k to find a self-correlation, and through softmax to factorization. Finally, it multiplied to v to act on itself. In our case, The proposed estimation fusion module (EFM) is a kind of self-Attention method. Actually, EFM is a cross-Attention method between the feature matrix and Estimation Code. We regard the feature matrix as q,v by Graph Convolution, then treat the upsampled Estimation Code as k by Graph Convolution. However, a predictor for NAS and a classifier for classification tasks are not the same, and we have to consider the direction property of the Directed Acyclic Graph (DAG) for the predictor of NAS. So, after we have calculated q multiplied by k, we will multiply the adjacency matrix to prove the direction of information flow. Finally, the skip connects are attached to the transformed feature matrix to enhance the feature passing. Finally, it goes through a Graph Convolution. The part (a) of Fig. 3 shows the structure of the EFM. The part (a) of Fig. 2 shows where EFM was attached.

3.2.2 NODES SELF-ATTENTION MODULE (NSAM)

Unlike the estimation fusion module, the proposed Nodes self-Attention Module(NSAM) is a pure self-Attention method with Graph Convolution focused on the relation of the same input. It used Graph Convolution to generate the q, k, and v matrix from the same input feature matrix. Then we also multiply the adjacency matrix as EFM for the same reason. Part (b) of Fig. 3 shows the structure of the NSAM. In later experiments, we will show more comparisons about NSAM. Part (b) of Fig. 2 shows where NSAM was attached.

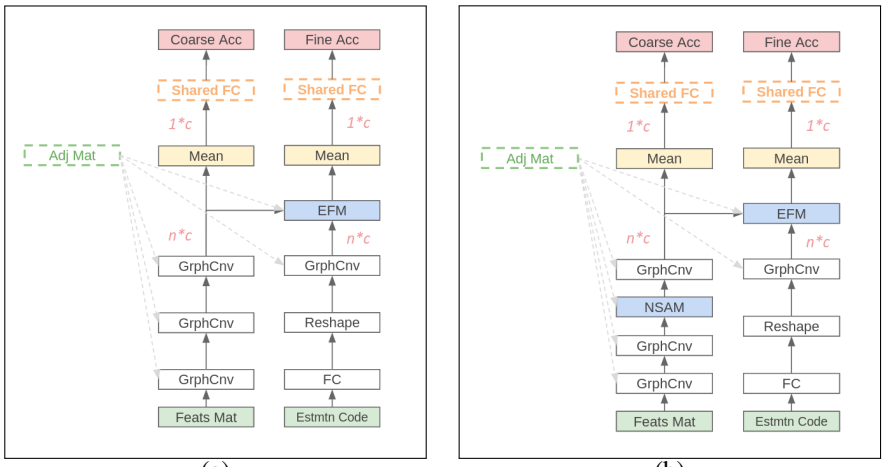


Figure 2: The structure of Siamese-Predictor. Figure (a) shows the Siamese-Predictor. The basic branch only uses the adjacency matrix and feature matrix, and the estimation branch uses an additional Estimation Code as another input. Figure (b) shows the Siamese-Predictor with NSAM. The red character "n" means the number of nodes, and the "c" means the feature length that is used to represent the node.

Table 1: Comparison of BTS and FTS for training time used Siamese-Predictor. The update frequency f is fixed at 10, and the training samples budget c is fixed at 100.

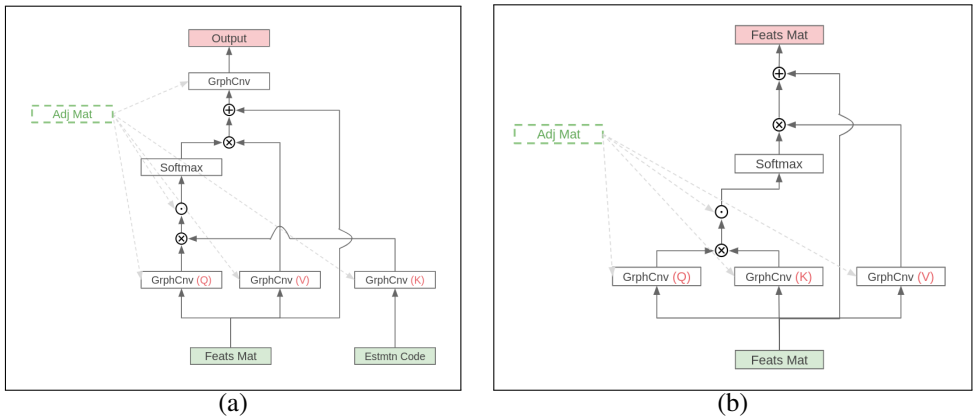
Method	**Search Space**	**Batch Samples**	**Training Time(s)**
Fully Top Sampling	Tiny-NanoBench(1120)	1120	115.67
Batch Top Sampling	Tiny-NanoBench(1120)	112	**67.22**
Fully Top Sampling	NASBench-201(15,625)	15,625	816.43
Batch Top Sampling	NASBench-201(15,625)	1563	**140.62**

3.3 SIAMESE-RANKING

The proposed Siamese-Predictor has two branches: the basic and the estimation branches. The basic branch is used to predict coarse accuracy almost for computation free because no extra trained samples is needed. The estimation branch is used to predict fine accuracy but is more expensive than the basic branch because getting the Estimation Code needs extra trained samples for three epochs. On the other hand, $S_1 + S_2$ must be smaller than S_0. So, we propose a mixture ranking method that first sorts search space by basic branch for almost computational cost-free, then just sorts top c by estimation branch. It ensures the stability of the high-accuracy group and solves the expensive problem. The Alg. 1 shows the implementation detail. The top c is an essential parameter of this algorithm, and if c is too large, the cost will be unacceptable. So the c is set to 30 in BTS, and the c is set to 60 in the evaluation stage. The computational cost for getting the Estimation Code of an architecture is just 0.3% compared to the complete training procedure, so the cost when the c is 30 or 60 is not worth mentioning. So far, The proposed Siamese-Ranking solves the budget problem using estimation branch.

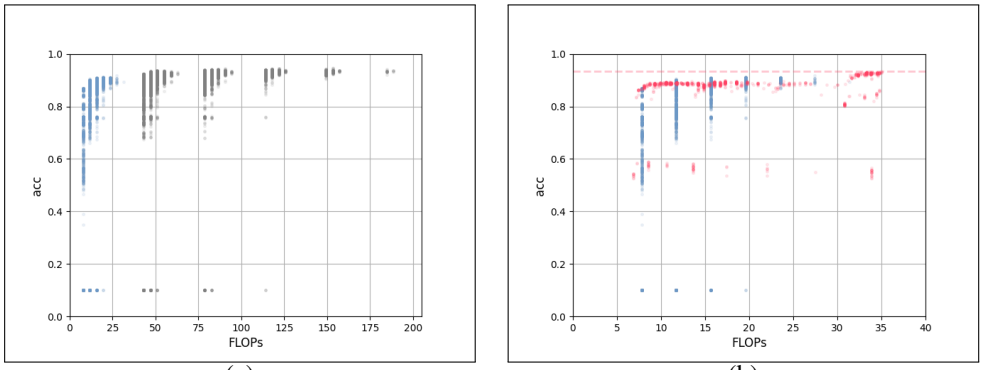
3.4 BATCH TOP SAMPLING (BTS)

The previous sections introduced how to build a powerful predictor with extra prior knowledge Estimation Code. In addition, a sampling strategy is also essential for predictor-based NAS, and a good strategy can train a good predictor using a few trained samples. The proposed Batch Top Sampling (BTS) randomly selects a few training samples to the training pool, then iteratively performs the following steps: 1) Randomly selects some trained samples to train the temporary Siamese-Predictor from the training pool. 2) Randomly selects some samples to form a batch search space from the whole search space. 3) Sorting (Siamese-Ranking) the batch search space based on the temporary



(a) (b)

Figure 3: Figure (a) shows the structure of EFM. Figure (b) shows the the structure of NSAM. The symbol $\bigotimes$ means the operation of matrix multiplication, and the symbol $\bigodot$ means the operation of Hadamard product, the symbol $\bigoplus$ just means the element-wise addition of two matrixes.



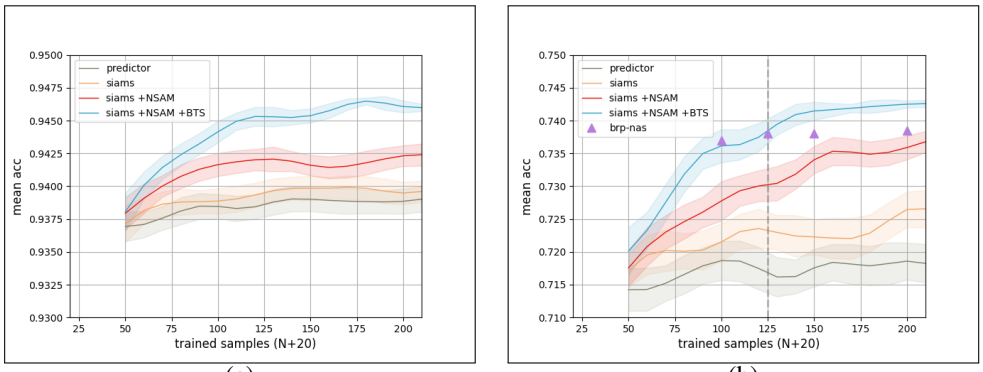
(a) (b)

Figure 4: Figure (a) is the distribution of NASBench-201, and figure (b) is the distribution comparison of NASBench-201 (blue) and the proposed Tiny-NanoBench (red).

Siamese-Predictor. 4) Choosing the top n samples of batch search space to the training pool. The Alg. 2 shows the implementation detail. This strategy of coarse to fine during training keeps the predictor focused on the high-accuracy groups. However, it also consumes considerable computing resources to predict architecture in the whole search space. The proposed BTS randomly selected to form the batch search space at the first step solved this problem. The Table. 1 shows that the training time of BTS is faster than Fully Top Sampling(FTS) due to the BTS using batch search space and FTS using whole search space. Finally, the BTS can help the predictor to boost the performance and reduce the variance from training randomness in later experiments.

3.5 Well-Designed Lightweight Search Space

In the previous sections, we proposed Siamese-Predictor, components, and tricks. Although this robust NAS algorithm shows considerable performance in later experiments. Our another purpose is focused on lightweight architecture search. As part (a) of Fig. 4 shown, we record the relationship between FLOPs and accuracy for NASBench-201. We can find that it consists of several groups at the FLOPs axis. Based on this observation, we set a constraint of about 35 MFLOPs to focus on the smallest FLOPs group (blue group in part (a) of Fig. 4), we call this subset Tiny-NASBench-201. We believe that Tiny-NASBench-201 is not the optimal search space at the tiny FLOPs (35M) level, so we proposed the Tiny-NanoBench trained on CIFAR-10. It is a well-designed lightweight search space based on a series of lightweight operations, depthwise convolution, pointwise convolution (Sandler et al. (2018)), splitting on the channel (Wang et al. (2020b)), expanding channel block (Zhang et al. (2021)), ..., etc. We also record the relationship between the FLOPs and accuracy. We



(a) (b)

Figure 5: The experiment of Siamese-Predictor on NASBench-201. Figure (a) shows the results of CIFAR-10, and figure (b) shows the results of CIFAR-100. The larger the area of the light-colored part, which means the more extensive the standard deviation.

found that the distribution of Tiny-NanoBench is more stable than Tiny-NASBench-201, especially at 10 15 MFLOPs. We also found that the Tiny-NanoBench has better architectures than Tiny-NASBench-201 at 30 35 MFLOPs.

4 EXPERIMENTS

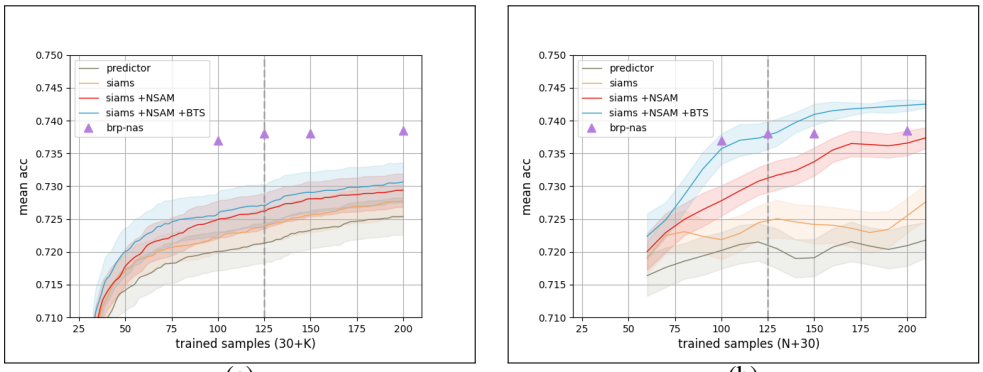
In this section, we conduct a series of experiments to validate our proposed methods on two benchmarks. We introduced the public search space NASBench-201 and evaluated the proposed Siamese-Predictor on it. Then we found that the typical training budget of $N+K$ is something worth thinking about the trade-off. Besides, We also introduced the proposed search space Tiny-NanoBench and evaluated Siamese-Predictor on it for lightweight architecture search.

4.1 GENERAL EVALUATION ON NASBENCH-201

NASBench-201 is a cell-based search space, it is public, credible, and easy to use, it records the trained accuracy about three dataset - CIFAR-10, CIFAR-100 (Krizhevsky et al. (2009)), ImageNet-16-120 (Chrabaszcz et al. (2017)). NASBench-201 contains 15,625 different architectures. These architectures are built on the same meta-architecture by different cells with six operations. We put our Siamese-Predictor up for comparison with the naive predictor. The Fig. 5 shows the 100 run results of the comparison. The "trained samples" represent the number of trained samples from the search space used by the predictor during the training process. It also contains the top-k choices at evaluating stage. The "mean acc" represents the average best accuracy of 100 runs. Overall, we trained a predictor with part of "trained samples" and sorted search space by this predictor. Finally, we took the top 20 and trained them one by one from scratch to get the architecture with the highest ground truth for each run. As shown in the Fig. 5, our method surpasses the naive predictor in the case of CIFAR-10, especially in CIFAR-100, the performance gap is greater than CIFAR-10. We also found that the NSAM and BTS bring considerable improvements in these two dataset. The proposed Siamese-Predictor with NSAM and BTS surpasses the SOTA brp-nas when trained samples are greater than 125, and the 125 samples only account for 0.8%(125/15265) of NASBench-201, even using 200 trained samples, that is only 1.3%, and it still achieves SOTA level with such few budgets. When trained samples are about 200, we also found that the standard deviation of Siamese-Predictor +NSAM +BTS is more stable than the other version. We believe this is due to BTS picking samples more efficiently.

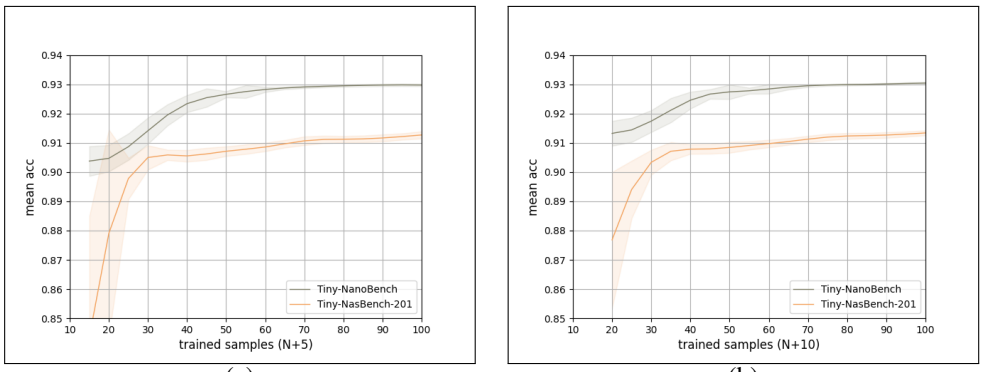
4.1.1 TRADE-OFF ABOUT $N + K$

The "trained samples" is a critical metric for predictor-based NAS. It consists of two variables, the number of training architectures for predictor N and the top K of sorted search space at the evaluation stage. In past, neural predictors (Wen et al. (2020); Dudziak et al. (2020)) prefer to fixed



(a) (b)

Figure 6: Comparison of the trade-off between N and K. Figure (a) is fixed N and increase K, and figure (b) is to fix K then increase N.



(a) (b)

Figure 7: The comparison of lightweight searching is between the Tiny-NanoBench and the Tiny-NASBench-201. Figure (a) is to fix K as 5, and figure (b) is to fix K as 10.

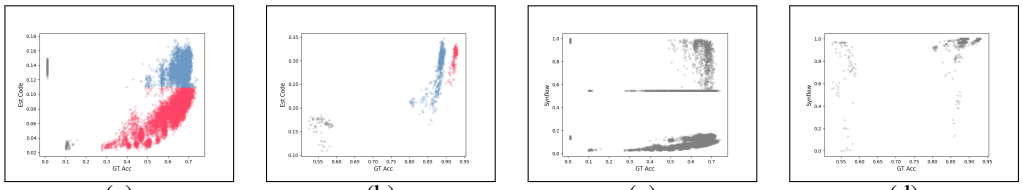
N and increase the K gradually. Nevertheless, we believe that it is to fix K, and increasing N may improve the performance because the number of N directly correlates with training samples for predictor training. The Fig. 6 shows the experiments on NASBench-201, which is fixed N or K and evaluates these predictors. Part (a) of Fig. 6 is fixed N to 30, then gradually increases K to draw the entire curve. We can see that when N even increased to 170 (the sum is 200), it is still far away from SOTA. On the other hand, part (b) of Fig. 6 is fixed K to 30, then gradually increases N. We found that spending the bulk of the budget to N can significantly improve. Not only that, it can be seen that as N increases, the standard deviation also decreases, and increasing K has no noticeable effect. These obvious conditions lead us to believe that increasing N is a better choice.

4.2 Lightweight Evaluation on Tiny-NanoBench

In order to verify the benefit of the proposed search space Tiny-NanoBench. We extract a small subset Tiny-NASBench-201 from NASBench-201. The exact extraction method is described in section 3. Both search spaces have the same constraints on FLOPs ($< 35M$). We use the proposed Siamese-Predictor to find the best architecture on these two search spaces, as shown in part (a) of Fig. 7. We found that it finds the better architecture faster under Tiny-NanoBench, especially when K and N are small. Part (a) and (b) of Fig. 7 show that it also can find the architecture of higher accuracy than Tiny-NASBench-201 when N is large.

4.3 Analysis for Estimation Code

The proposed Estimation Code is the critical part of this paper. In order to further analyze it, we introduce the other cost-free metric of prior knowledge. This work (Abdelfattah et al. (2021))de-



(a) (b) (c) (d)

Figure 8: The comparison of Estimation Code and synflow. Figure (a) is the distribution of NasBench-201 (CIFAR-100). Figure (b) is the distribution of Tiny-NanoBench. Figure (c) is the distribution of NASBench-201 about accuracy and synflow. Figure (d) is the distribution of Tiny-NanoBench about accuracy and synflow.

tailed introduces several metrics. There are snip (Lee et al. (2018)), grasp (Wang et al. (2020a)) and synflow ,and the synflow (Tanaka et al. (2020)) shows the considerable performance for many NAS methods in their experiments. So, we take the synflow to replace the Estimation Code as the prior knowledge for comparison. The Fig. 8 shows the results on NASBench-201 and Tiny-NanoBench. The red and blue regions of Fig. 8 have a significant positive correlation between prior knowledge (estimated code or synchronous flow) and accuracy. The gray region is the opposite. So, we can see that part (c) and part (d) of Fig. 8 are more disorders than part (a) and part (b) of Fig. 8, which means the Estimation Code has a higher correlation with accuracy than synflow.

5 Conclusion

This paper focuses on building a powerful predictor to find promising CNN architecture for training samples efficiently. The proposed Siamese-Predictor using EFM to fuse prior knowledge - Estimation Code. The proposed Estimation Code and accuracy of architectures are highly correlated based on our observation (Fig. 1). Besides, we also proposed Graph Convolution-based self-Attention method NSAM and fast and efficient sampling strategy BTS. These two tricks greatly enhanced the performance of the Siamese-Predictor in our experiments. It surpass BRP-NAS and achives the SOTA level on NASBench-201 (Fig. 5). We also proposed a search space Tiny-NanoBench, for lightweight CNN architectures searching. In this search space, our predictor can find better lightweight architecture more easily than Tiny-NASBench-201, the lightweight subset of NASBench-201. In summary, we proposed a powerful and efficient predictor-based NAS method Siamese-Predictor and a search space Tiny-NanoBench designed for lightweight CNN architecture.

6 Reproducibility

All ”mean acc” mentioned in this article, all results are run 100 times, and we also provide the source code about all versions of Siamese-Predictor. The main function can be found in the file ”train_eval.py”. It is the main function of the project. It can be traced to the code and find more detail. Now, we introduce key points that may affect reproduction in our implementation.

6.1 Bidirectional Graph Convolution

The proposed siamese-predictor is constructed by graph convolutions. Its structure is to follow the advice from (Wen et al. (2020)), not the naive version. They suggest combining two graph convolutions, one using the adjacency matrix of the original direction and the other using the transposed to reverse direction. We are also adding batch normalization and using mish to replace all relu. This detail about graph convolution can refer the file ”graph_conv.py”.

6.2 Estimation Code

In the previous section, we have already described how to get the estimation code. Here we re-explain it in more detail. We randomly sampled 5000 images from CIFAR-10, then trained every architecture of NasBench-201 or Tiny-NanoBench on it for three epochs. The three epochs are

just 3/100 of the complete training procedure. The training images are just 5000/50000 of fully CIFAR-10. That is why we always say this code is almost cost-free, especially since the training is only required if there are used samples. More info can refer the function "TrainEstCode" and "TrainNas201EstCode" of file "nas_training.py".

6.3 EFM AND NSAM AND BTS

These three proposed components of our paper play important roles. The details of these methods have been presented clearly. So, we just point out the exact location of the source code for reference. The EFM is denoted from class "NodesAttention" of file "modules.py". The NSAM is denoted from class "EstmtnFusion" of file "neural_predictor.py". The BTS is denoted from class "BatchTopSampler" of file "nas_callbacks.py".

REFERENCES

Mohamed S Abdelfattah, Abhinav Mehrotra, Łukasz Dudziak, and Nicholas D Lane. Zero-cost proxies for lightweight nas. *arXiv preprint arXiv:2101.08134*, 2021.

Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. *arXiv preprint arXiv:1611.02167*, 2016.

Patryk Chrabaszcz, Ilya Loshchilov, and Frank Hutter. A downsampled variant of imagenet as an alternative to the cifar datasets. *arXiv preprint arXiv:1707.08819*, 2017.

Xuanyi Dong and Yi Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search. *arXiv preprint arXiv:2001.00326*, 2020.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

Lukasz Dudziak, Thomas Chau, Mohamed Abdelfattah, Royson Lee, Hyeji Kim, and Nicholas Lane. Brp-nas: Prediction-based nas using gcns. *Advances in Neural Information Processing Systems*, 33:10480–10490, 2020.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700–4708, 2017.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

Namhoon Lee, Thalaiyasingam Ajanthan, and Philip HS Torr. Snip: Single-shot network pruning based on connection sensitivity. *arXiv preprint arXiv:1810.02340*, 2018.

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)*, pp. 19–34, 2018a.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018b.

Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 10012–10022, 2021.

Zhichao Lu, Ian Whalen, Vishnu Boddeti, Yashesh Dhebar, Kalyanmoy Deb, Erik Goodman, and Wolfgang Banzhaf. Nsga-net: neural architecture search using multi-objective genetic algorithm. In *Proceedings of the genetic and evolutionary computation conference*, pp. 419–427, 2019.

Renqian Luo, Fei Tian, Tao Qin, Enhong Chen, and Tie-Yan Liu. Neural architecture optimization. *Advances in neural information processing systems*, 31, 2018.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pp. 4780–4789, 2019.

Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4510–4520, 2018.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Hidenori Tanaka, Daniel Kunin, Daniel L Yamins, and Surya Ganguli. Pruning neural networks without any data by iteratively conserving synaptic flow. *Advances in Neural Information Processing Systems*, 33:6377–6389, 2020.

Chaoqi Wang, Guodong Zhang, and Roger Grosse. Picking winning tickets before training by preserving gradient flow. *arXiv preprint arXiv:2002.07376*, 2020a.

Chien-Yao Wang, Hong-Yuan Mark Liao, Yueh-Hua Wu, Ping-Yang Chen, Jun-Wei Hsieh, and I-Hau Yeh. Cspnet: A new backbone that can enhance learning capability of cnn. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, pp. 390–391, 2020b.

Chen Wei, Chuang Niu, Yiping Tang, Yue Wang, Haihong Hu, and Jimin Liang. Npenas: Neural predictor guided evolution for neural architecture search. *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

Wei Wen, Hanxiao Liu, Yiran Chen, Hai Li, Gabriel Bender, and Pieter-Jan Kindermans. Neural predictor for neural architecture search. In *European Conference on Computer Vision*, pp. 660–676. Springer, 2020.

Chris Ying, Aaron Klein, Eric Christiansen, Esteban Real, Kevin Murphy, and Frank Hutter. Nas-bench-101: Towards reproducible neural architecture search. In *International Conference on Machine Learning*, pp. 7105–7114. PMLR, 2019.

Yu-Ming Zhang, Chun-Chieh Lee, Jun-Wei Hsieh, and Kuo-Chin Fan. Csl-yolo: A new lightweight object detection system for edge computing. *arXiv preprint arXiv:2107.04829*, 2021.

Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

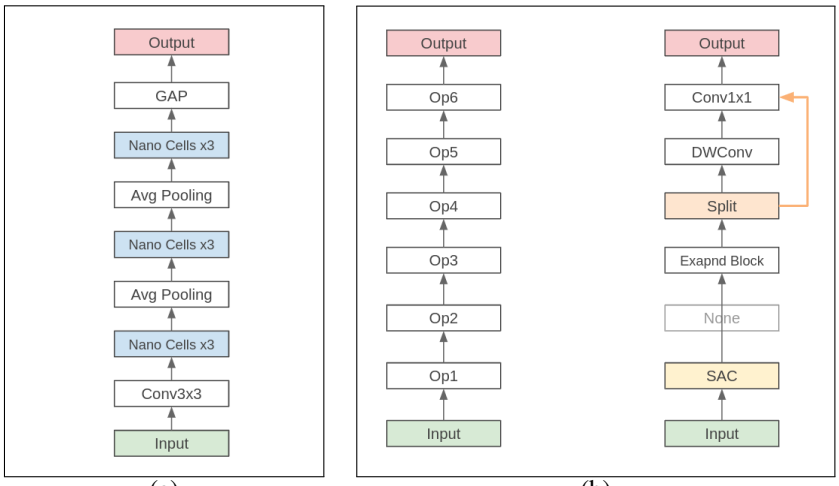


Figure 9: Figure (a) shows the meta-architecture of Tiny-NanoBench. The left of figure (b) shows the structure of Nano Cell that contains six nodes(op 1 op 6). The right of figure (b) shows an example of Nano Cell, and it excatly contains all optional operations.

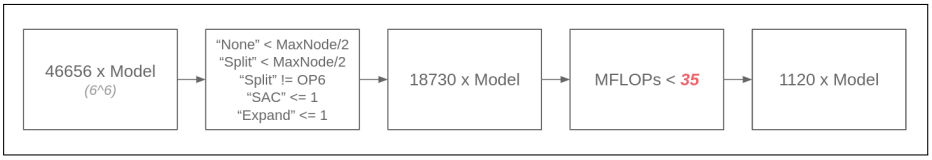


Figure 10: The flow chart of making Tiny-NanoBench shows the constraint about FLOPs, and the number of architectures.

A TINY-NANOBENCH

The proposed Tiny-NanoBench is a search space that focuses on lightweight CNN architecture. It is a cell-based search space and imitated NASBench-101. Part (a) of Fig. 9 shows the meta-architecture. It stacks three Nano Cells at each stage. There are nine cells in total. Part (b) of Fig. 9 shows an example of chosen operations in each cell. Fig. 10 shows why does Tiny-NanoBench have 1120 architectures. For more info, it can refer to files "nanonas_model.py" and "modules.py."