NADER: Neural Architecture Design via Multi-Agent Collaboration

Supplementary Material

S1. Detailed Experimental Setting

S1.1. Macro Skeleton

When conducting experiments on CIFAR10, CIFAR100 [5], and ImageNet16-120 [2], in order to compare fairly with the compared methods, we constrain the model designed by our methods to have the same macro skeleton as NAS-Bench-201 [4]. The skeleton is initiated with a stem block. The main body of the skeleton includes three stacks of cells, connected by a downsampling block. Each cell has the same architecture and is stacked five times. Each downsampling block downsamples the input feature map by two times. Unlike NAS-Bench-201, we do not set a fixed model width. When a new architecture is designed, we search for a suitable width for this model to constrain the number of parameters and FLOPs of the model so that it does not exceed the maximum values in NAS-Bench-201.

S1.2. Implement of Agents

All our agents are implemented using GPT-4o. And we use 'text-embedding-ada-002' to extract text embeddings and build vector databases.

Table S1. The training hyperparameter.

optimizer	SGD	initial LR	0.1
Nesterov	✓	ending LR	0
momentum	0.9	LR schedule	cosine
weight decay	0.0005	epoch	200
batch size	256	normalization	✓
random flip	p=0.5	random crop	size=32, padding=4

S1.3. Model Training Details

When training and testing the model, we use the same training dataset, validation dataset, test dataset and training hyperparameters as NAS-Bench-201. Table S1 shows the training hyperparameters for CIFAR-10 and CIFAR-100. ImageNet16-120 has the same training hyperparameters as CIFAR-10 except that the random crop size is 16 and the padding is 2 We repeat the experiment three times using 777, 888 and 999 as the random number seeds.

S2. Prompt design and Output Examples for Agents

S2.1. Reader

The agent Reader has two LLM-based actions: selecting relevant papers and extracting knowledge from the relevant papers. Tables S3 and S4 present the prompt templates for these two actions, respectively. Table S2 shows several examples of knowledge extracted from the papers.

S2.2. Proposer

Table S5 presents the prompt template used by the agent Proposer to select modification suggestions.

S2.3. Modifier

Table S6 presents the prompt used to define the graph-based neural architecture representation. Table S7 presents the prompt template and examples used by the Modifier to generate new architectures through multi-turn dialogues. In the first round, the Modifier generates a architecture containing an undefined operation. In the second round, after providing a prompt to the Modifier, it successfully generates a new architecture.

S2.4. Reflector

Reflector reflects on the historical records of faulty architectures and performance-degrading architectures generated by the Modifier and generating experience in generating correct neural architectures and experience in designing performance-enhanced architectures, respectively. Table S8 presents the prompt template for reflecting on historical records of faulty architectures and Table S9 presents the prompt template for reflecting on historical records of performance-degrading architectures. Tables S10 and Table S11 present several examples of the two types of experience, respectively.

Table S2. Example of knowledge extracted from papers.

PaperId	Knowledge
603 [1]	1. Design the Task Net to leverage knowledge co-embedding features constructed from both image quality and disease diagnosis, using multiple branches to specifically address different aspects of the input data. 2. Implement a Global Attention Block within the Task Net to extract task-specific features, focusing on both channel and spatial attention to capture relevant features for each task.
1530 [7]	1. Adoption of a multi-scale architecture using reversible residual blocks and squeeze modules to capture a wide range of style details while minimizing spatial information loss. 2. Exclusion of normalization layers in the reversible blocks to facilitate learning direct style representation without interference, enhancing the style transfer fidelity. 3. Implementing channel refinement in reversible residual blocks to manage redundant information accumulation and enhance stylization quality.

Table S3. Prompt template for selecting relevant papers.

###Instruction###
You are a computer vision research specialist with a deep background in the field of computer vision, particularly in deep learning models and visual recognition tasks.
You needs to evaluate a given paper to determine if it can inspire to design the better basic blocks architecture of vision models' backbone.
###Goal###
According to the title and abstract of the paper, analyzing whether can get inspiration from the paper to design the better basic architecture of visual model backbone.
###Constraints###
The inspiration must be related to the basic block architecture design of the visual model backbone.
###Workflow###
1. Read and understand the title and abstract of the paper.
2. Summarizing the innovation and contribution of this paper.
3. Analyzing whether you can get inspiration from this paper to design a better basic block architecture for visual models.
###Title###
{{*title*}}
###Abstract###
{{*abstract*}}
###Output###
Answer yes or no prefix with ##response## in the end.

Table S4. Prompt template for extracting knowledge from papers.

###Instruction###
You are a computer vision research expert. Please list the inspirations you get from this paper to design the basic block architecture of the visual model backbone.
###Constraints###
1. A paper usually contains several sections: abstract, introduction, related work, methods, experiments and conclusion. Please focus on the methods of the paper to respond.
2. Inspirations must to be detailed and related to designing the basic block architecture of the visual model backbone.
###Input###
The following is the content of the paper:
{{*paper*}}
###Output###
You response should wrap each inspirations with <inspiration> and </inspiration>, and use ',' to separate different knowledge.

Table S5. Prompt template for selecting modification suggestions.

###Instruction###
You are a computer vision research expert, and you have deep insights into neural arcgitecture design.
You will be given a block to be improved and several candidate inspirations, you need to compare the candidate inspirations and rank them according their usefulness for guiding the improvement of the block.
###Block###
The following is the block to be improved and candidate inspirations. The neural architecture of the block of the model is described in the form of a computational graph.
{{*block*}}
###Candidate inspirations###
The following are the candidate inspirations. Each inspiration is given in the form of 'inspiration index:inspiration'.
{{*candidate inspirations*}}
###Output###
Please rank the all candidate inspirations in descending order according to their usefulness. You response should wrap all inspiration index of the inspirations with <response> and </response>, and use ',' to separate different indexes.

Table S6. Prompt for defining the graph-based neural architecture representation.

Each block starts with "##block_name##". In each line, you can use the "index:operation" to define the node of computation graph or use the "index1->index" to define the edge of computation graph.
The following is a list of available operations:

Conv2d(out_channels, kernel_size, stride, dilation, groups) Two-dimensional convolution operation, 'out_channels' represents the output dimension; 'kernel_size' represents the convolution kernel size; 'stride' represents the step size, default: 1; 'dilation' is the hole convolution size, default: 1; 'groups' groups number of the channels, default:1.
Linear(out_channels) Linear fully connected layer, 'out_channels' represents the output dimension.
AvgPool2d(kernel_size, stride) Two-dimensional average pooling operation, 'kernel_size' represents the kernel size, 'stride' represents the step size.
MaxPool2d(kernel_size,stride) Two-dimensional maximum pooling operation, 'kernel_size' represents the kernel size, 'stride' represents the step size.
AdaptiveMaxPool2d(output_size) Two-dimensional maximum pooling operation pools the input feature map into a feature map with a length and width of output_size. For example, AdaptiveMaxPool2d(output_size=1) pools a feature map of the shape of (B,C,H,W) into (B,C,1,1) shape.
AdaptiveAvgPool2d(output_size) Two-dimensional average pooling operation.
Add Tensor-by-element addition operation, the input tensors' shape must conform to the broadcasting rule.
Mul Tensor-by-element multiplication operation, the input tensors' shape must conform to the broadcasting rule.
Multiply Matrix multiplication operation, the entered tensor shapes must conform to the tensor multiplication rule.
concat(dim) Tensor concating operation, all tensors input to this operation are concated in the dim dimension. The sizes of the concated tensors dimensions other than the dim dimension should be consistent. For example, concat(dim=1) concates all input tensors in the 1 dimension.
mean(dim) Average the tensor in dim dimension. For example, mean(dim=1) pools a input tensor of shape (B,L,D) into the output tensor of shape (B,1,D) by average in the dimension 1.
max(dim) Maximize the tensor in dim dimension. For example, max(dim=2) pools a input tensor of shape (B,L,D) into the output tensor of shape (B,L,2) by max in the dimension 2.
sum(dim) Sum the tensor in dim dimension. For example, sum(dim=0) pools a input tensor of shape (B,L,D) into the output tensor of shape (0,L,D) by sum in the dimension 0.
softmax(dim) Apply a softmax operation at dim dimension. For example, softmax(dim=1) calculate the softmax of input tensor with shape (B,L,D) and the output tensor's shape is (B,L,D).

The activation functions that can be used are: **ReLU**, **GELU**, **Sigmoid**.
The normalization methods that can be used are:

BN Batch normalization
LN Layer normalization.

The tensor can be transformed by using the following operations:

permute(*dims) rearranges the tensor dimensions, 'dims' is the order of the new dimensions, for example: permute(0, 2, 3, 1) changes the tensor shape from (B, C, H, W) to (B, H, W, C).
repeat(*sizes) repeats the tensor along the specified dimensions. 'sizes' is a list containing the number of repetitions along each dimension. For example: repeat(1, 3, 2, 4) repeats the tensor 1 times in the first dimension, 3 times in the second dimension, 2 times in the third dimension, 4 times in the forth dimension.
reshape(*shape) changes the shape of the tensor to the specified shape; 'shape' is an array representing the shape of new tensor; you can use -1 as the size of a dimension to automatically calculate the size of the dimension to ensure that the total number of elements remains unchanged; for example: reshape(B, H, W, C) means changing the shape of the tensor to (B, H, W, C).

Variables you may use include:

input: input feature map, the shape is (B,C,H,W).
output: output feature map, the shape is (B,dim,H,W). Noting that the output node can have only one input.
C: the number of channels of the input feature map.
dim: the number of channels of the output feature map.
H: the height of the input feature map.
W: the width of the input feature map.

You can use the basic $+, -, \times, /$ operations.

Table S7. Prompt template and example for generating new architecture.

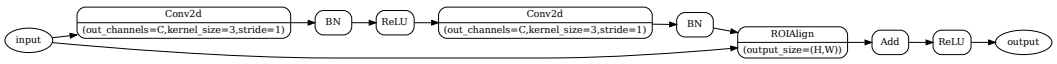
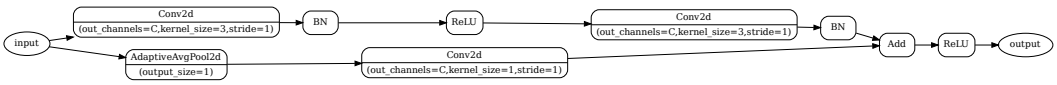
Human	###Instruction### You are an expert who is proficient in various model structures of deep learning. Please make reasonable modifications to the specified block based on the characteristics of the block and the inspiration. ###Constraints### 1. Please ensure that the number of input channels and output channels of the generated block are both C. 2. Note that structures in the modified block that unrelated to the proposal should be kept as original as possible. 3. The new neural architecture you output must comply with the BlockDefinition format. ###BlockDefinition### {{*Definition of the graph-based neural architecture representation*}} ###block### {{*block*}} ###inspiration### {{*inspiration*}} ###Design Experience### Refer to the following suggestions to help you generate a block that better meets block definition. {{*Experience in generating the correct architectures*}} Refer to the following suggestions to help you generate a block that has better performance. {{*Experience in generating better-performing architectures*}} ###Output### When outputting, you only need to output the block that meet the defined rules, and do not output other irrelevant information.
Assistant	
Reflector	{'status':'error','context':'node 8 error: Undefined computation ROIAlign is used'}
Human	The block you generate has following error: {'status':'error','context':'node 8 error: Undefined computation ROIAlign is used'}, please fix it and generate a new one.
Assistant	
Reflector	{'status':'success'}

Table S8. Prompt template for reflecting on the historical records of faulty architectures.

###Instruction### You are an expert who is proficient in neural architecture design. The structure of neural networks is now described in terms of directed acyclic graphs. The following is the definition of directed acyclic graphs. {{*Definition of the graph-based neural architecture representation*}} ###Input### Now there is a network that does not fully meet the above definition and a hint of reason: {{*block*}} Error reason: {{*error*}} ###output### Please analyze the reason of network design errors, and based on this, give a general design tip to prompt users to accurately design a network that fully meets the requirements. The tip should be wrapped in <tip> and </tip>.

Table S9. Prompt template for reflecting on the historical records performance-degrading architectures.

###Instruction###
You are an expert who is proficient in neural architecture design. You will be given a raw model and a modified model and their corresponding accuracy on test dataset and you need to analyze why the accuracy of the raw model decreases after modification, and give a suggestion to avoid this error in the next modification.
The structure of neural networks is now described in terms of directed acyclic graphs. The following is the definition of directed acyclic graphs:
{{*Definition of the graph-based neural architecture representation*}}
###Input###
Their is the raw model and its accuracy:
{{*raw block*}}
Accuracy: {{*raw block's accuracy*}}
Their is the modified model and its accuracy:
{{*new block*}}
Accuracy: {{*new block's accuracy*}}
###Constrain###
1. The suggestion must be relevant to the neural network structure.
2. The suggestion must be a sentence no more than 50 words.
3. The suggestion must be must be general.
###Output###
Please think step by step about the reasons why the accuracy of the model decreases after modification and give a suggestion to avoid this error in the next modification. The suggestion should be wrapped in <suggestion> and </suggestion>.

Table S10. Example of experience in generating correct archs.

1. When using LN, always ensure that the last dimension of the input tensor is the channel dimension. You can achieve this by using the 'permute' operation to rearrange the tensor dimensions appropriately before applying LN.
2. When designing a neural network, ensure that the 'out_channels' parameter in the 'Conv2d' operation is divisible by the 'groups' parameter. This is necessary to maintain valid configurations and avoid errors. Specifically, if you set 'groups' to be equal to 'C' (the number of input channels), make sure that 'out_channels' is also a multiple of 'C'.
3. When designing a neural network with multiple branches that will be concatenated or added together, ensure that all branches maintain consistent spatial dimensions (height and width) throughout their respective operations. Use padding in convolution and pooling layers to preserve the input dimensions, or carefully adjust stride and kernel sizes to ensure outputs are compatible for concatenation or addition.

Table S11. Example of experience in generating correct archs.

1. Ensure modifications retain the original block's dimensional consistency and spatial information to maintain performance.
2. Ensure that added layers and operations contribute to meaningful feature extraction and avoid unnecessary complexity.
3. Consider maintaining intermediate bottleneck layers and incorporating attention or pooling mechanisms to enhance feature extraction.
4. Ensure that any added skip connections or operations do not disrupt the learning process by excessively altering the network's expected data flow.
5. Ensure the newly added paths or operations do not interfere destructively with existing paths, and validate their impact on gradient flow.

S3. Detailed Experimental Results

S3.1. Detailed Numberical Results

Table S12, Table S13, and Table S14 show the experimental results on CIFAR10, CIFAR100, and ImageNet16-120 using ResNet as the initial model. Table S15, Table S16, and Table S17 show the experimental results on CIFAR10, CIFAR100, and ImageNet16-120 using random models in NAS-Bench-201 as the initial models. We repeat each experiment with different random seeds for three times.

S3.2. Analysis of Large-scale NAD Experiment Results

Figure S1 illustrates the distribution of test accuracy of 500 models designed by NADER on CIFAR-100. It can be seen that 30.8% of the models have an accuracy that exceeds the optimal model in the neural architecture search space designed by NAS-Bench-201. 58.8% of the proposals generated by the Proposer are effective, and the performance of the model based on these proposals surpasses that of the initial model. 19.2% of the models failed to converge during training due to unreasonable design. At the same time, these bad models exhibit noticeable differences in loss values within the initial few epochs, allowing for early detection and termination to reduce unnecessary training.

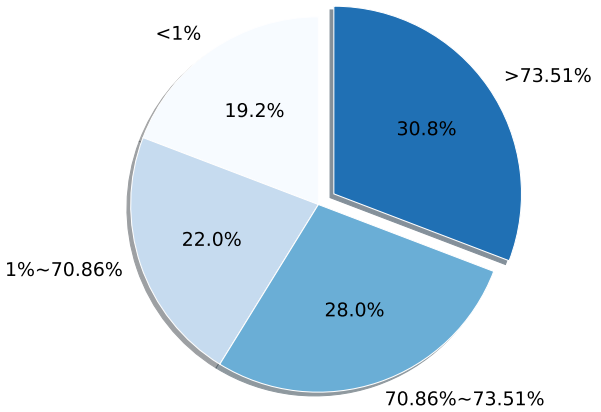


Figure S1. Distribution of test accuracy of 500 models designed by NADER on CIFAR-100.

S3.3. Scalability under different LLMs.

NADER is flexible and can be driven by different LLMs (e.g., the open-source DeepSeek-V3). Table S18 reports av-

Table S12. Detailed Experimental Results on CIFAR-10 with ResNet initialization.

Trail	5 archs		10 archs	
	validation	test	validation	test
Trail 1	91.28	94.83	91.28	94.83
Trail 2	90.83	94.37	90.86	94.37
Trail 3	91.39	94.35	91.39	94.35

Table S13. Detailed Experimental Results on CIFAR-100 with ResNet initialization.

Trail	5 archs		10 archs	
	validation	test	validation	test
Trail 1	74.18	74.18	75.11	75.11
Trail 2	73.84	73.57	74.34	74.45
Trail 3	71.64	71.62	74.44	74.38

Table S14. Detailed Experimental Results on ImageNet16-120 with ResNet initialization.

Trail	5 archs		10 archs	
	validation	test	validation	test
Trail 1	47.53	47.72	49.27	49.58
Trail 2	49.00	48.52	49.00	48.52
Trail 3	47.40	47.72	47.40	47.72

Table S15. Detailed Experimental Results on CIFAR-10 with Random initialization.

Trail	5 archs		10 archs	
	validation	test	validation	test
Trail 1	90.39	93.67	90.96	94.13
Trail 2	90.83	94.37	90.86	94.37
Trail 3	91.50	94.55	91.66	94.69

Table S16. Detailed Experimental Results on CIFAR-100 with Random initialization.

Trail	5 archs		10 archs	
	validation	test	validation	test
Trail 1	74.04	74.35	74.04	74.35
Trail 2	73.84	73.57	74.34	74.45
Trail 3	74.44	74.15	74.86	74.72

Table S17. Detailed Experimental Results on ImageNet16-120 with Random initialization.

Trail	5 archs		10 archs	
	validation	test	validation	test
Trail 1	48.60	48.82	50.63	50.38
Trail 2	49.00	48.52	49.00	48.52
Trail 3	48.60	48.82	50.57	50.00

erage token cost, executability (E), quality (Q), and success rate (SR) for different LLMs.

Table S18. Evaluation of different LLMs.

Task	Model	# Tokens (K)	E	Q	SR
Macro	DeepSeek-V3	0.52±0.76	0.80	0.80	0.64
	GPT-4o	0.53±0.68	0.78	0.87	0.68
Micro	DeepSeek-V3	0.32±0.41	0.96	0.88	0.84
	GPT-4o	0.31±0.45	0.92	0.96	0.88

S3.4. Generalization Ability of the Model Designed by NADER

After 500 designs, NADER designed a model on the CIFAR-100 dataset that showed significant improvements compared to the optimal model in the NAS method's search space (NB201-Optimal). We refer to this model as NADER-500. We further retrain and test the NADER-500 on several other datasets to evaluate the generalization ability of the model designed by NADER. The experimental results are presented in Table S19. It show that NADER-500 demonstrates significant advantages on datasets involving natural scenes, such as animals and vehicles (e.g., CIFAR-100 [5], STL-10 [3], GT SRB [6]).

Table S19. Experimental results of NADER-500 and NB201-Optimal on different datasets. The test accuracy is presented.

Dataset	NB201-Optimal (#Params:1.30M)	NADER-500 (#Params:1.28M)
CIFAR100 [5]	73.23	75.97
GTSRB [6]	96.16	96.94
STL10 [3]	69.65	72.13

S3.5. Extension to ImageNet-1K.

We validate our method's scalability on ImageNet-1K. Table S20 presents test accuracy for the best models obtained after 15 design iterations starting with ResNet and 30 iterations starting with ConvNeXt, respectively. These models were trained using the same configuration as ConvNeXt, except with reduced training epochs (50) and warm-up epochs (5) for efficiency. These results demonstrate that NADER scales well to larger datasets.

Table S20. Results on ImageNet-1K.

Model	**Accuracy**
ResNet	73.60
NADER-ResNet-15	**74.62**
ConvNeXt	77.42
NADER-ConvNeXt-30	**78.10**

S3.6. Time cost analysis.

The total time cost consists of the neural architecture design (NAD) cost and the model validation cost. The average time cost for 500 archs is shown in the Table S21. (1) NADER focuses on NAD, whose time cost is negligible. (2) The primary time-consuming step is model validation, with a cost proportional to the number of searched architectures (2781.79s per model). Extending the test-time model search allows for evaluating more models, increasing the chances of discovering higher-performing architectures. Notably, our method achieves superior performance with fewer searches.

Table S21. Time cost of NADER.

Proposer	**Modifier**	**Model Validation**
1.01 s	21.68 s	2781.79s

S3.7. Architecture Details

Figure S2, Figure S4, Figure S3 and Figure S5 show several examples of neural architectures designed by NADER with ResNet as the initial model. It can be found that the multi-agent collaboration method we designed can stimulate the creativity of LLMs and generate novel and effective architectures, rather than simply generate architectures that LLMs may have seen in the training process.

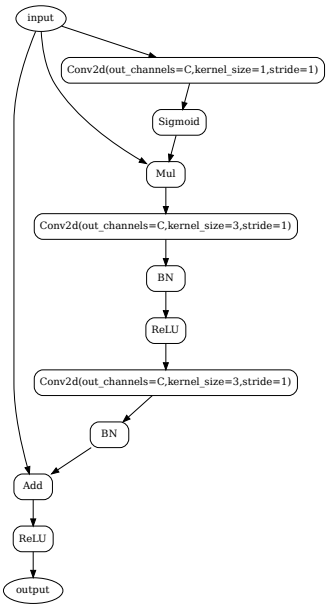


Figure S2. The neural architecture of the optimal model obtained by performing 10 iterations of search on CIFAR-10. The accuracy on the test dataset is 94.83%.

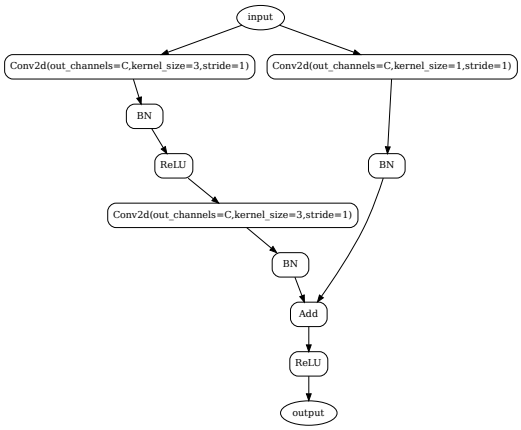


Figure S3. The neural architecture of the optimal model obtained by performing 10 iterations of search on ImageNet16-120. The accuracy on the test dataset is 49.58%.

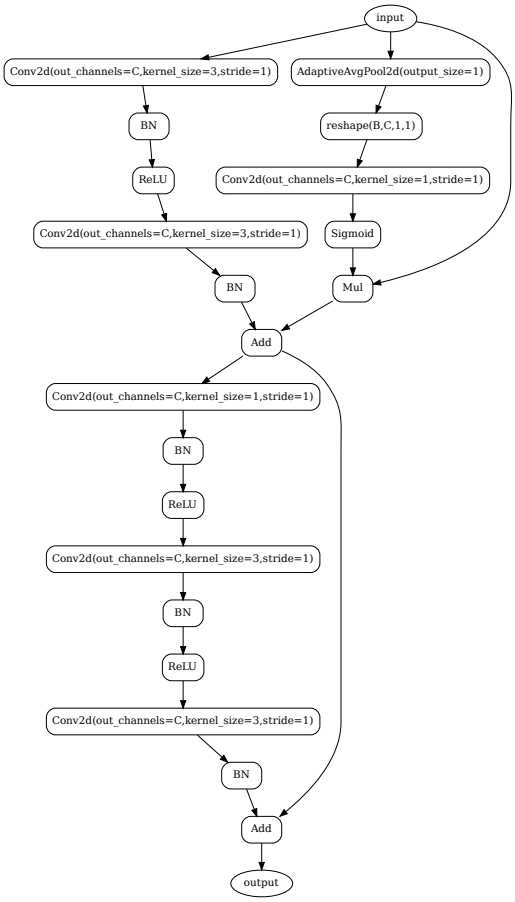


Figure S4. The neural architecture of the optimal model obtained by performing 10 iterations of search on CIFAR-100. The accuracy on the test dataset is 75.11%.

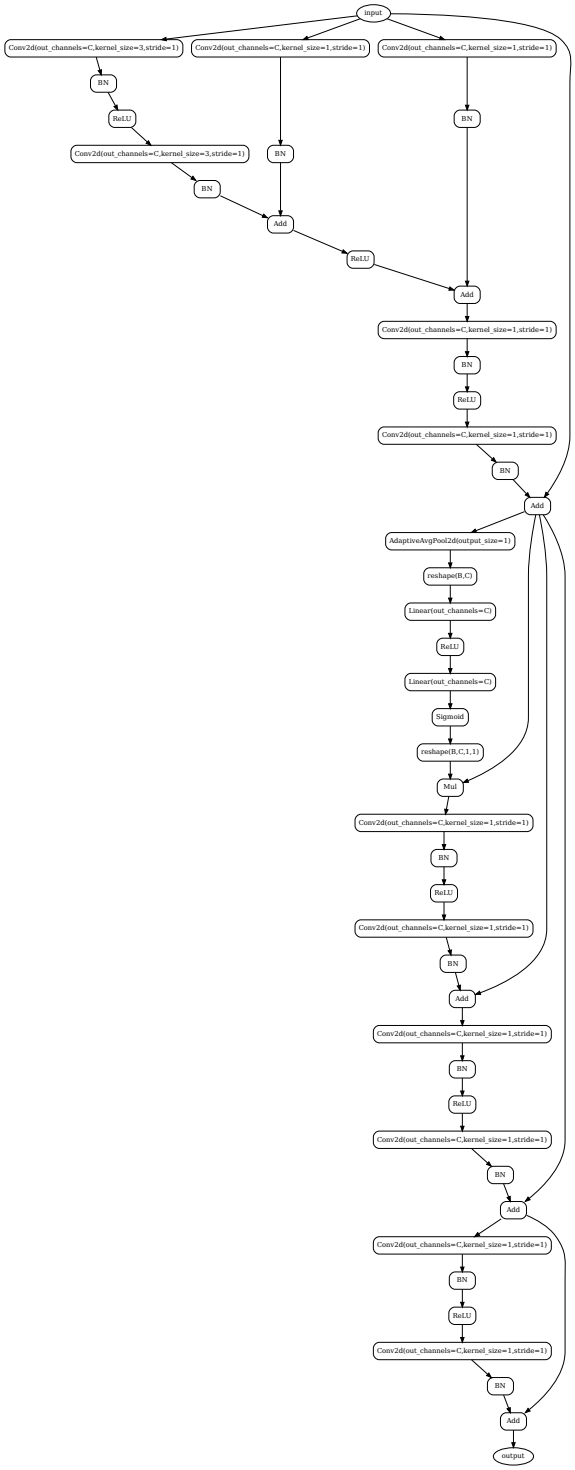


Figure S5. The neural architecture of the optimal model obtained by performing 500 iterations of search on CIFAR-100. The accuracy on the test dataset is 76.00%.

References

[1] Haoxuan Che, Siyu Chen, and Hao Chen. Image quality-aware diagnosis via meta-knowledge co-embedding. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 19819–19829, 2023. 1

[2] 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. 1

[3] Adam Coates, Honglak Lee, Andrew Y Ng, Adam Coates, Honglak Lee, and Andrew Y Ng. An analysis of single-layer networks in unsupervised feature learning. In *Aistats*, 2011. 7

[4] Xuanyi Dong and Yi Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search. *arXiv preprint arXiv:2001.00326*, 2020. 1

[5] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. *Master's thesis, University of Tront*, 2009. 1, 7

[6] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural Netw*, 32: 323–332, 2012. 7

[7] Linfeng Wen, Chengying Gao, and Changqing Zou. Cap-vstnet: Content affinity preserved versatile style transfer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 18300–18309, 2023. 1