Noisy Heuristics NAS: A Network Morphism based Neural Architecture Search using Heuristics

Suman Sapkota [1] **Binod Bhattarai** [2]

Abstract

Network Morphism based Neural Architecture Search (NAS) is one of the most efficient methods, however, knowing where and when to add new neurons or remove dis-functional ones is generally left to black-box Reinforcement Learning models. In this paper, we present a new Network Morphism based NAS called Noisy Heuristics NAS which uses heuristics learned from manually developing neural network models and inspired by biological neuronal dynamics. Firstly, we add new neurons randomly and prune away some to select only the best fitting neurons. Secondly, we control the number of layers in the network using the relationship of hidden units to the number of input-output connections. Our method can increase or decrease the capacity or non-linearity of models online which is specified with a few meta-parameters by the user. Our method generalizes both on toy datasets and on real-world data sets such as MNIST, CIFAR-10, and CIFAR-100. The performance is comparable to the hand-engineered architecture ResNet-18 with the similar parameters.

1. Introduction and Related Works

Neural Architecture Search (NAS) is the process of searching the Architecture of Neural Networks by leveraging the computation rather than doing manually. However, NAS has still not been able to come to the mainstream due to large computational costs and availability of more efficient alternatives such as transfer learning (Zhuang et al., 2020) or reusing architectures. Research has been done on using Reinforcement Learning(RL) (Zoph & Le, 2016; Baker et al., 2016) and Genetic Algorithm(GA) (Desell, 2017) for generating architecture from a given search space, however, these methods have huge computational costs and produce large carbon footprints (Strubell et al., 2019). Gradient-based path selection methods such as DARTS (Liu et al., 2018) and PC-DARTS (Xu et al., 2019) have made NAS more efficient and accessible. However, it still involves training large parameter models and selecting only a subset for the final model.

In a manual architecture search process, we start with a baseline model architecture. If the model has poor performance, we test more and more non-linear models and if the model overfits the dataset we test smaller models, throwing away older models in the process. The initial concern is whether non-linear capacity can be increased or decreased in the same model while reusing the trained function. To our aid, Network Morphing based methods (Elsken et al., 2017; Lu et al., 2018; Dai et al., 2019; Evci et al., 2022) have been used widely to add neurons, which increase the non-linearity and capacity of the model.

However, Network Morphism based methods are generally paired up with Reinforcement Learning (RL) (Cai et al., 2018) or Bayesian Optimization (Jin et al., 2019), which decide the morphism operation to increase the network capacity. This type of solution makes the dynamic nature of neural network a difficult to understand. To understand the dynamics and to simulate heuristics, we need easily controllable models for changing network capacity or the number of neurons or parameters.

Although there are various works on using Network Morphism for Neural Architecture Search, we find that the methods are partial, either only adding neurons (Jin et al., 2019; Cai et al., 2018) and layers or not pruning layers (Gordon et al., 2018) to reduce capacity. Furthermore, those methods that add and prune neurons use it on incremental (Dai et al., 2020) or continual (Zhang et al., 2020) learning settings. Our goal to search for architecture depending on the dynamics requires additional components to change the structure (layers and neurons) of the network itself. This gap motivates us to create a Network Morphism based NAS that can change the number of layers and neurons dynamically during the training phase while keeping the search efficient

We would like to acknowledge Google Cloud for computing credits. [1]NAAMII, Nepal [2]University College London, UK. Correspondence to: Suman Sapkota <suman.sapkota@naamii.org.np>, Binod Bhattarai <b.bhattarai@ucl.ac.uk>.

DyNN workshop at the 39^{th} International Conference on Machine Learning, Baltimore, Maryland, USA, 2022. Copyright 2022 by the author(s).

and simple for the user. We combine multiple ideas and heuristics for creating a framework of Noisy NAS to search for capacity.

Ideas from pruning and dropout support our framework for noisy heuristic-based architecture search. When small neurons are pruned, they typically recover the same accuracy and loss (Molchanov et al., 2019) without recovering the function completely. Furthermore, noisy regularization methods like Dropout (Srivastava et al., 2014) and DropConnect (Wan et al., 2013) suggest that Deep Networks can be trained to be robust to perturbations. We can infer that Neural Networks are robust to the noisy process of addition and pruning of neurons. We can use such a noisy process to try different additions and removals of neurons iteratively which can roughly change the architecture to the desired capacity. The method of adding many neurons and removing poorly performing ones could be used to search for the correct place to add new neurons.

Furthermore, the dynamics of Biological Neural Networks (BNN) (Wan et al., 2019) suggests that there could exist Neural Networks with dynamically changing architecture in a single model. The dynamic nature of BNN is partly due to neurogenesis (Kumar et al., 2019), neuron and synaptic pruning (Fricker et al., 2018) and neuron migration. We aim to understand the internal workings of Artificial Neural Networks(ANN) and apply dynamics from BNN to close the gap between them. We believe that Dynamic Neural Networks along with Spiking Neural Networks (Tavanaei et al., 2019) could model BNN even better.

Our Contribution: Combining the growing and shrinking mechanisms, we are able to get any desired network capacity for the best fitting of the dataset. Such a method is depicted by a generalization curve as shown in Figure 1. We work on the same curve, but instead of trying different capacity models, we change the capacity of the existing models towards the best capacity. To this end, we propose a new method for Network Morphism based Neural Architecture search using heuristics. We simplify our search space using multiple heuristics to a manageable number of meta-parameters. The major contributions of our work are listed below.

1. We introduce a new method to add new neurons and layers heuristically for Network Morphism based Neural Architecture Search.
2. We create a new type of architecture called Hierarchical Residual Network for the ease of changing non-linearity and number of layers during Network Morphism.
3. We combine neuron addition, pruning and Hierarchical Residual Network to change the capacity of the network noisily during training, which we call Noisy Heuristics NAS.
4. We show that our method is successful in getting performance near hand-designed architectures like ResNet.
5. We release code for Network Morphism, Optimizer reusing, Pruning and Noisy Heuristic NAS in the PyTorch (Paszke et al., 2019) framework. https://github.com/tsumansapkota/Noisy-Heuristics-NAS

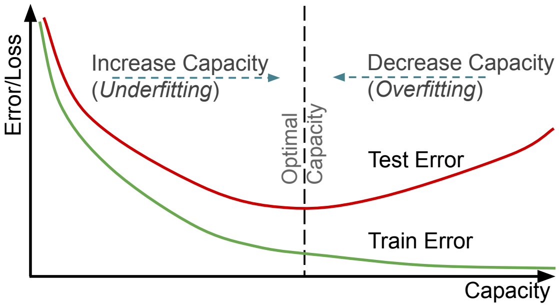

Figure 1. Generalization Curve

2. Methodology

The following concepts are combined to create a Noisy Neural Architecture Search based on Heuristics.

Hiearchical Residual Network We propose a generalization of ResNet (He et al., 2016) called Hierarchical Residual Network (H-ResNet). The main concept is that each linear layer can be made non-linear by adding a residual function to it, which is similar to ResNet, as shown by equation (1). Such residual connections are easy to add and remove without breaking the flow of information in Deep Neural Networks. We extend this idea to the limit. The addition of a residual function to the linear connection can be extended to the linear connection of the residual connection itself as shown by equation (2).

$$f(x) = W(x) + Re(x)\ ;\ Re(x) = W_2(\sigma(W_1(x)) \quad (1)$$

$$Re(x) = f_2(\sigma(f_1(x))) \quad (2)$$

Here, f_1 and f_2 are ResNets with different parameter. The hierarchical nature of this type of network is depicted by the figure 2. Theoretically, all Ordinary Networks, ResNets (He et al., 2016) and UNets (Ronneberger et al., 2015) are special cases of Hierarchical ResNets. If the shortcut connection is zero then it is an Ordinary Network and if the residual function has shortcut as its layers then it is a ResNet.

Where to add new neurons ? The problem of where to add new neurons and how many has large possible choices. To tackle this, we first add large number of neurons (P) distributed to all layers. Secondly, we train the model to fit the new neurons to data. Thirdly, we prune M least important neurons such that change in number of neurons is given by, $\Delta N = P - M$.In this way, we are able to find a good solution for where to add new neurons.

The change in number of neurons in each layer is given by, $\Delta N_i = P_i - M_i$, where P_i is the initial number of neurons

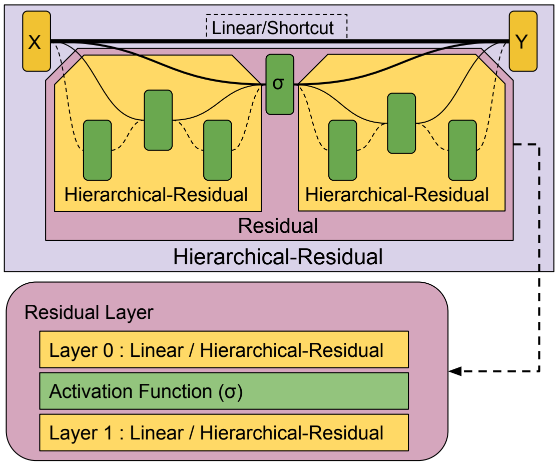

Figure 2. Hierarchical Residual Network

and M_i is the number of pruned neurons. To accelerate the addition of neurons in correct layers in next iterations, we add 70% of new neurons with the probability of each layer $p_i \propto \text{MA}(\Delta N_i)$, where MA represents Moving Average function change in number of neurons.

Where to add new layers ? In Hierarchical ResNet, we can add layers to any linear connection of any hierarchy. To solve the problem of having a large number of possible choices, we use a rough heuristic to add new layers to *Residual Layer* automatically and heuristically after *Shrink Phase*. The heuristic is that if hidden neurons (H) on Residual Layer with Linear connection is greater than $M = (I * O^2)^{\frac{1}{3}}$, we convert the inner layers, *Layer 0* and *Layer 1*, from Linear to Hierarchical-Residual Layer. The Hierarchical-Residual Layer can again grow layers on its Residual Layer in a recursive way. (See Figure 2)

How to prune neurons ? We use global importance estimation based pruning similar to previous methods (Molchanov et al., 2019; Yu et al., 2018) which prunes the least important neurons. The importance score per neuron (I) is given by $I = A * (1 - B^{33})$. Where,

$$A = \Sigma_{i=0}^{N} |\sigma_i * \delta\sigma_i| \; ; \; B = \frac{1}{N}\Sigma_{i=0}^{N}(1[\sigma_i > 0])$$

σ_i is the activation for i^{th} data point, $\delta\sigma_i$ is the gradient of the activation and N is the total number of data in the dataset. The term B gives fraction of non-zero activations. We scale the term A by B to give low importance to always firing neurons.

Since pruning removes non-zero neurons as well it creates sudden change in function learned. To make removal gradual, we decay the outgoing connection of least important neurons while finetuning other neurons for some epochs.

How to remove layers ? During each iteration of addition and pruning, we morph the network at the end of the pruning stage. After pruning, if the number of neurons in a certain Residual Layer (see Figure 2) decreases to near zero (say 1), then we prune away the neurons from the layer and the layer is removed altogether by our algorithm. This makes the parent Residual layer contain Linear connection as *Layer 0* or *Layer 1*. This happens just the opposite way of adding a new Layer. New layers added in the previous iteration are susceptible to removal.

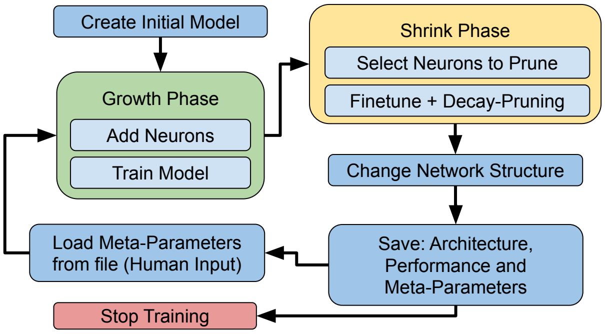

Figure 3. Pipeline of Noisy Heuristic NAS

When to know when model is fitted ? In a general training setting, we stop the training when the loss curve flattens out. First, we normalize steps and loss values to range [0, 1] and fit the steps vs loss data to $y = e^{ax}(1 - x)$, where a is the parameter (See Appendix A). We set $a < -5$ as threshold for determining the flattening of curve. The training is stopped if $a < -5$ or epochs $\geq$ maximum train epochs.

Noisy Heuristic NAS Our method combines the above-mentioned operations to search for the architecture. The pipeline for the Noisy Heuristic NAS is shown in Figure 3. The pipeline is explained as follows.

Firstly, an initial model is created. The initial model can be Linear Layer or some standard architecture like ResNet. Secondly, during the *Growth Phase*, we add neurons randomly. This model is then trained for given epoch or until convergence. Thirdly, during the *Shrink Phase*, neurons are pruned by importance score.

Next, we save the model, performance and current meta-parameters. Then we take meta-parameters as input from a file, which is an interface between user and the NAS system. We repeat the process of growth, shrink, save and load for multiple iterations until desired network is achieved.

Expanding or shrinking : Our method allows the network to either expand or shrink in capacity allowing us to navigate the capacity dimension of Generalization Curve as shown in Figure 1. If the number of neurons added is more than pruned, then the network is expanding, otherwise the network is shrinking. This allows us to change the capacity dynamically. We may start from pre-trained large network

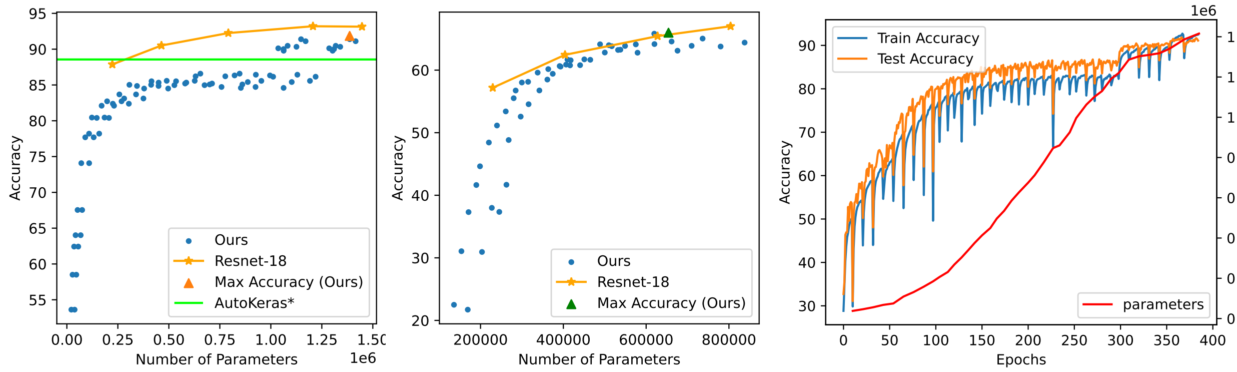

Figure 4. **Left:** Parameter vs Accuracy plot for CIFAR-10 dataset. AutoKeras (Jin et al., 2019) has unknown number of parameters. **Center:** Parameter vs Accuracy plot for CIFAR-100 dataset. **Right:** Epoch vs (Accuracy and Parameter) plot for CIFAR-10 dataset.

and decrease its capacity or start from linear model and increase its capacity towards large model all in a single continuous training, without throwing away models completely.

Workflow: The typical workflow of Noisy Heuristics NAS is to observe the performance and capacity and tweak the meta-parameters during the training. The major meta-parameters that are tuned manually during the search process are (1) number of neurons to add, (2) number of neurons to remove / add-to-remove-ratio and (3) learning rate.

Using these parameters we can control the type of growth of the network. The architecture found is dependent on the dynamics of the process and the meta-parameters.

Genetic Algorithm View of Noisy Heuristic NAS : We find our addition and pruning based approach of search similar to Genetic Algorithm. If we view individual neurons as an organism, the pruning is equivalent to the selection process and new neurons are equivalent to new generation/mutation. Furthermore, we could apply better genetic algorithms for better selection of neurons and faster convergence of the search process.

3. Experiments

Experiments on Toy Datasets: We experiment on small 2D toy regression and classification problems to verify our method (See Appendix B). We find that our method effectively finds a suitable network for fitting the given dataset. We do not compare our method to any other architecture due to the simple solution to these datasets. However, it opens our method to be tested on large scale datasets like MNIST, CIFAR-10 and CIFAR-100.

Experiments on Large Scale Datasets : We find that our method improves the architecture search by modifying the network without changing the function. The use of heuristics for network morphism helps to avoid the computational cost of training deep RL models. We extend our method to MNIST dataset with Dense and Convolutional Layers where we get an accuracy of 97.87% and 99.34% respectively. We find that our method produces satisfactory results.

Since our code is written mostly on Python/PyTorch and Hierarchical ResNet is not optimized, we find that it takes a longer time. We instead compare the methods in terms of the number of epochs for optimization. We find that our method performs comparatively with hand-designed ResNet-18 architecture on both CIFAR-10 and CIFAR-100 datasets (see Figure 4). Our method gradually changes (increases) parameters dynamically during the training which results in a gradual increase in test accuracy. A similar trend follows in the CIFAR-100 experiment as well.

For ResNet-18 architecture, we change the number of parameters by changing the width (number of channels) of the residual blocks. We choose ResNet with similar number of parameters for relative comparison. The experiments are carried out for 200 epochs each. And for our method, meta-parameters were changed during the training to search desired capacity and for better performance.

While testing for 5 different parameter ResNets, we spend a total of 1000 epochs which is higher than we train our method for (≈ 400). Our method produces models of varying size and similar performance in one training. Further details of the experiments and findings are in Appendix C.

4. Conclusion

In this work, we presented Noisy Heuristics NAS working on the Network Morphism framework. We efficiently train variable-capacity models by morphing the same network with minimal loss in trained parameters. Our method has performance comparable to hand-designed architecture like ResNets while allowing the network to change in capacity by meta-parameters adjusted during training.

References

Baker, B., Gupta, O., Naik, N., and Raskar, R. Designing neural network architectures using reinforcement learning. *arXiv preprint arXiv:1611.02167*, 2016.

Cai, H., Chen, T., Zhang, W., Yu, Y., and Wang, J. Efficient architecture search by network transformation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Dai, X., Yin, H., and Jha, N. K. Nest: A neural network synthesis tool based on a grow-and-prune paradigm. *IEEE Transactions on Computers*, 68(10):1487–1497, 2019.

Dai, X., Yin, H., and Jha, N. K. Incremental learning using a grow-and-prune paradigm with efficient neural networks. *IEEE Transactions on Emerging Topics in Computing*, 2020.

Desell, T. Large scale evolution of convolutional neural networks using volunteer computing. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pp. 127–128, 2017.

Elsken, T., Metzen, J.-H., and Hutter, F. Simple and efficient architecture search for convolutional neural networks. *arXiv preprint arXiv:1711.04528*, 2017.

Evci, U., Vladymyrov, M., Unterthiner, T., van Merriënboer, B., and Pedregosa, F. Gradmax: Growing neural networks using gradient information. *arXiv preprint arXiv:2201.05125*, 2022.

Fricker, M., Tolkovsky, A. M., Borutaite, V., Coleman, M., and Brown, G. C. Neuronal cell death. *Physiological reviews*, 98(2):813–880, 2018.

Gordon, A., Eban, E., Nachum, O., Chen, B., Wu, H., Yang, T.-J., and Choi, E. Morphnet: Fast & simple resource-constrained structure learning of deep networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1586–1595, 2018.

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

Jin, H., Song, Q., and Hu, X. Auto-keras: An efficient neural architecture search system. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 1946–1956, 2019.

Kumar, A., Pareek, V., Faiq, M. A., Ghosh, S. K., and Kumari, C. Adult neurogenesis in humans: a review of basic concepts, history, current research, and clinical implications. *Innovations in Clinical Neuroscience*, 16 (5-6):30, 2019.

Langley, P. Crafting papers on machine learning. In Langley, P. (ed.), *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pp. 1207–1216, Stanford, CA, 2000. Morgan Kaufmann.

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

Lu, J., Ma, W., and Faltings, B. Compnet: Neural networks growing via the compact network morphism. *arXiv preprint arXiv:1804.10316*, 2018.

Molchanov, P., Mallya, A., Tyree, S., Frosio, I., and Kautz, J. Importance estimation for neural network pruning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11264–11272, 2019.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Ronneberger, O., Fischer, P., and Brox, T. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pp. 234–241. Springer, 2015.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

Strubell, E., Ganesh, A., and McCallum, A. Energy and policy considerations for deep learning in nlp. *arXiv preprint arXiv:1906.02243*, 2019.

Tavanaei, A., Ghodrati, M., Kheradpisheh, S. R., Masquelier, T., and Maida, A. Deep learning in spiking neural networks. *Neural networks*, 111:47–63, 2019.

Wan, L., Zeiler, M., Zhang, S., Le Cun, Y., and Fergus, R. Regularization of neural networks using dropconnect. In *International conference on machine learning*, pp. 1058–1066. PMLR, 2013.

Wan, Y., Wei, Z., Looger, L. L., Koyama, M., Druckmann, S., and Keller, P. J. Single-cell reconstruction of emerging population activity in an entire developing circuit. *Cell*, 179(2):355–372, 2019.

Xu, Y., Xie, L., Zhang, X., Chen, X., Qi, G.-J., Tian, Q., and Xiong, H. Pc-darts: Partial channel connections for memory-efficient architecture search. *arXiv preprint arXiv:1907.05737*, 2019.

Yu, R., Li, A., Chen, C.-F., Lai, J.-H., Morariu, V. I., Han, X., Gao, M., Lin, C.-Y., and Davis, L. S. Nisp: Pruning networks using neuron importance score propagation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 9194–9203, 2018.

Zhang, J., Zhang, J., Ghosh, S., Li, D., Zhu, J., Zhang, H., and Wang, Y. Regularize, expand and compress: Nonexpansive continual learning. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 854–862, 2020.

Zhuang, F., Qi, Z., Duan, K., Xi, D., Zhu, Y., Zhu, H., Xiong, H., and He, Q. A comprehensive survey on transfer learning. *Proceedings of the IEEE*, 109(1):43–76, 2020.

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

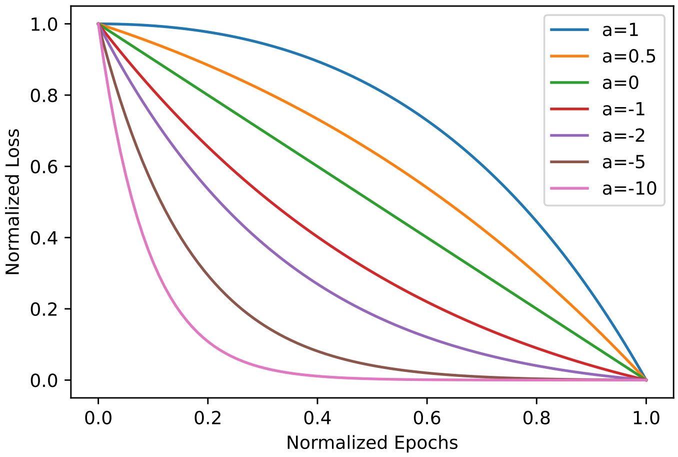

Figure 5. Model for flattening of loss curve with different parameters. Here the number of epochs and loss values are normalized in range [0,1] to fit the model.

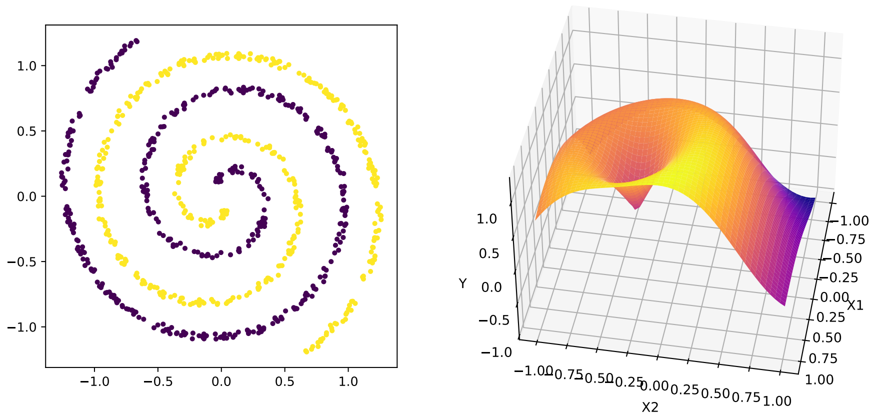

Figure 6. **Left:** 2-Spiral Dataset for binary classification. **Right:** A regression dataset on a 2D grid.

A. Stopping Criterion using Loss Curve

We use the model $y = e^{ax}(1 - x)$ for detecting the flattening of the loss curve. The Figure 5 shows the model for different values of a. We set $a < -5$ for determining that the loss curve has flattened. We also set maximum training epochs threshold for stopping if curve does not flatten out. The maximum training epochs is also a meta-parameter which can be changed by the user. The experiments also show changing maximum training epochs change during the experiment.

B. Toy Experiments

We use 2D synthetic datasets for verifying that our algorithm works. The spiral dataset shown in Figure 6 is used for classification. In the experiment, we get accuracy of 100% in few iterations. Furthermore, we also test our algorithm on 2D regression dataset as shown in Figure 6. We find that our method produces neural network that fits the dataset well.

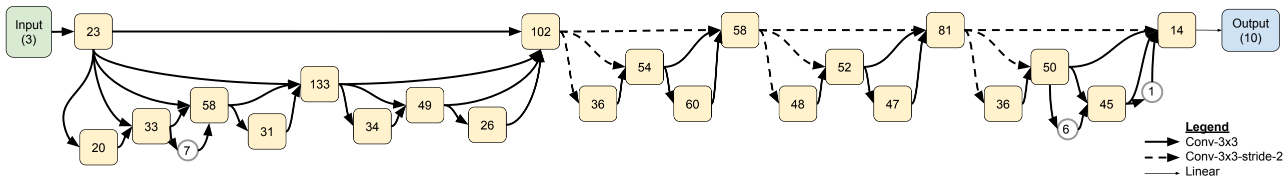

Figure 7. Architecture Found in CIFAR-10 experiment. The backbone network is inspired by ResNet architecture. We consider each block as Hierarchical Residual Network which is allowed to expand. The depth of this architecture is 23 (not counting the layers with the number of neurons < 10). We apply Global Average Pooling (GAP) before the Linear Layer for Classification.

C. CIFAR Experiments: Extended

The experiments carried out in the Experiment Section (3) are extended and elaborated in this section. The experiments for ResNet-18 architectures on both CIFAR-10 and CIFAR-100 datasets are carried out with Adam optimizer with a learning rate of 0.001 with Cosine Scheduler with a 200 time period. For Noisy Heuristics NAS, we use Adam optimizer with learning rates manually changed during the training. We use dropout with drop probability $p = 0.1$ with ReLU activation function. The use of dropout causes the Train Accuracy to be lower than Test Accuracy.

The meta-parameters were changed during the training to search desired capacity and for better performance. The exact meta-parameter values and observations are elaborated next.

CIFAR-10 Experiment: Firstly, experiment on the CIFAR-10 dataset using our method, as shown in the Experiment Section produces best architecture with 1.42M parameters and 91.85% accuracy. Furthermore, Residual Network produces 93.14% accuracy with 1.45 parameters. The accuracy mentioned in Auto-keras paper (Jin et al., 2019) is 88.56% with an unknown number of parameters. The CIFAR-10 experiment carried out in the Experiment Section has varying meta-parameters in different epochs. We plot the meta-parameters in the Figure 8. Furthermore, we plot the architecture found using our search method in its best epoch in Figure 7

Furthermore, to verify that our algorithm can also reduce the parameters from a large model, we first grow the model (till epoch 400) and later shrink the model. The observations from this experiment are shown in Figure 9. It verifies that our algorithm can effectively reduce neurons and layers according to the shrinking or expanding nature of the search process. The architecture also has a varying number of depths while increasing or decreasing the number of neurons. In the initial stage, the depth of the network is 10, at peak accuracy, it is 26 and at the later stage of shrinking, the depth is 16. Furthermore, it can be observed that the accuracy is correlated with the number of parameters.

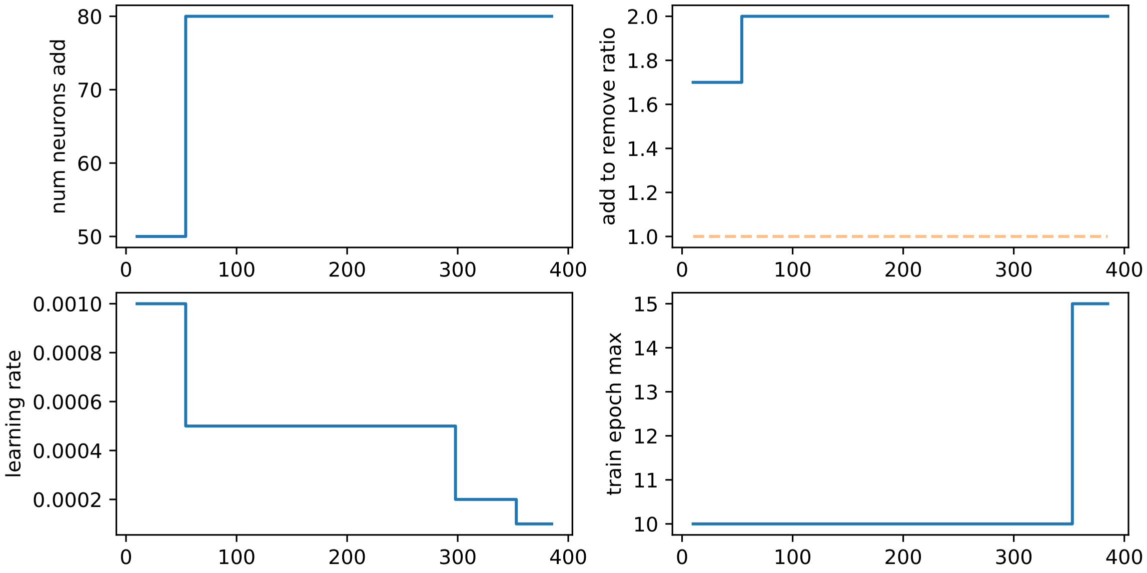

Figure 8. Different Meta-Parameters changed during the training phase. The x-axis is the number of epochs and y-axis (labeled) are meta-parameters

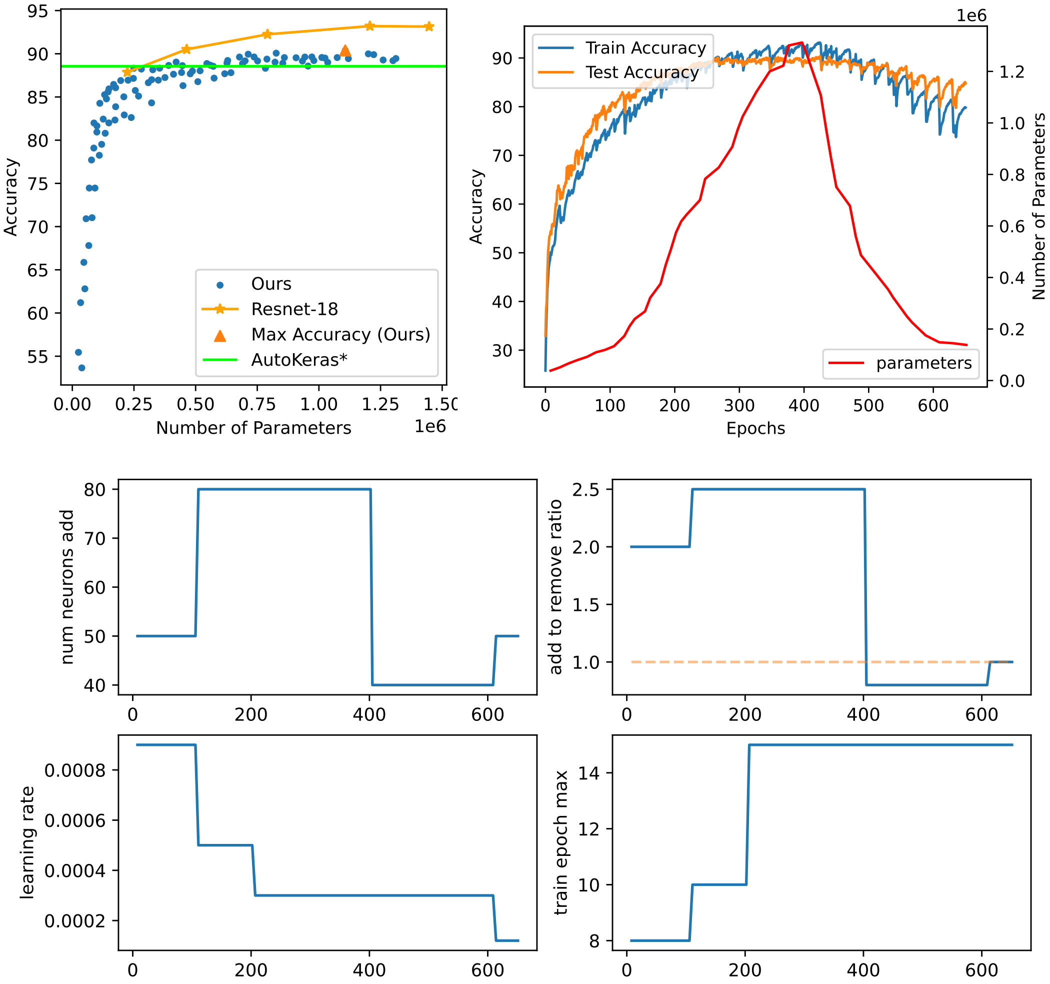

Figure 9. **Top Left:** Parameter vs Accuracy plot for CIFAR-10 dataset. **Top Right:** Epoch vs (Accuracy and Parameter) plot for CIFAR-10 dataset. **Bottom Four:** Meta-parameters that guide the architecture search process.

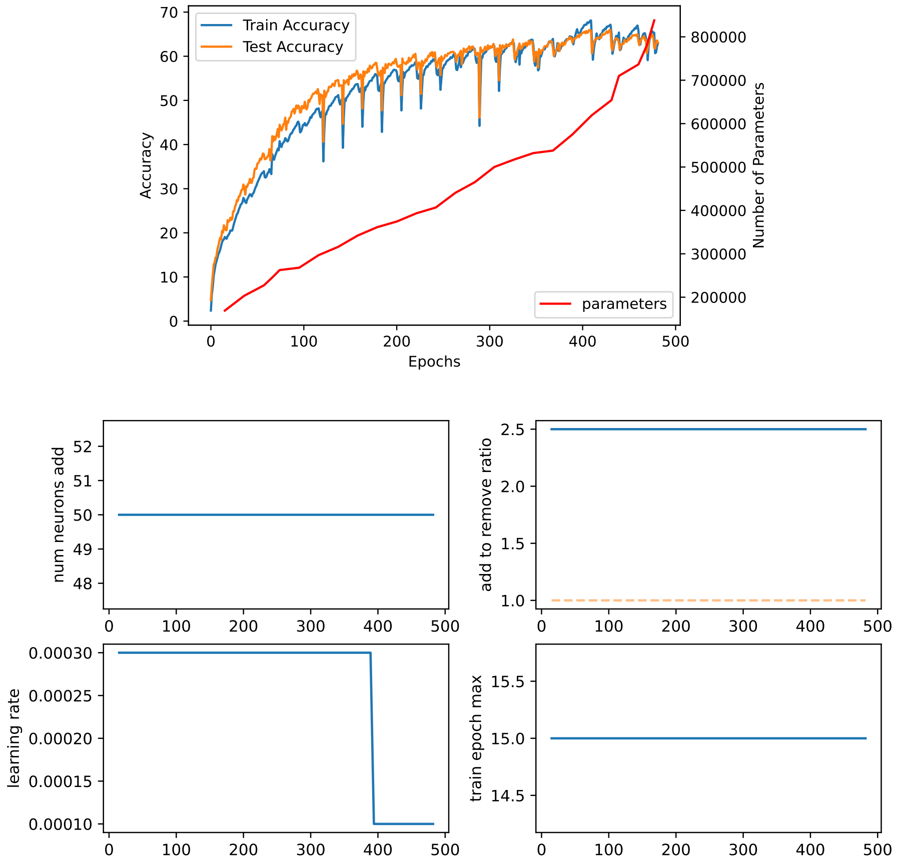

Figure 10. **Top:** Epoch vs (Accuracy and Parameter) plot for CIFAR-100 dataset, **Bottom Four:** Meta-parameters used during CIFAR-100 training. We only modify the learning rate once during the search.

CIFAR-100 Experiment: This paragraph is extension of the CIFAR-100 experiment mentioned in the Experiments Section (3). We also show the relationship between Epochs, Accuracy and Number of parameters in the Figure 10. Furthermore, we also show the values of meta-parameters used in the experiment in the same Figure. In CIFAR-100 experiments, we find trends similar to experiments on CIFAR-10. Our method produces architecture having 65.98% accuracy with 0.654M parameters whereas ResNet-18 with comparable parameters of 0.627M has an accuracy of 65.43%.

D. Limitations and Future Work

We implement Convolutional Hierarchical Residual Network using only 3x3 convolution operations. Many NAS algorithms search for kernel size as well as strides and padding. We could add another variety of Convolution parameters in search space in future work. Furthermore, we can also add other search variables such as dropout rate and activation functions.

In future works, we could also use better neuron initialization methods such as GradMax (Evci et al., 2022) to fit the residual better, and use better importance estimation algorithm for pruning as well. We can also implement better optimization techniques to avoid accuracy drops while training newly added neurons. Furthermore, we can apply meta models to learn the meta-parameters which in turn modifies the network. We also plan to optimize the Hierarchical Residual Architecture for better performance.