Defined-Filter-Aided Neural Architecture Search

Truong Nguyen Huy
University of Illinois, Chicago

Vaibhav Jolly
University of Illinois, Chicago

Harsh Mishra
University of Illinois, Chicago

Manmohan Dogra
University of Illinois, Chicago

Abstract

We want to propose a Foundations Oriented concept for this project. In detail, we will be experimenting with a relaxation of this aforementioned approach. RL-NAS places the function approximation robustness of neural networks into the RL framework, which elegantly solves the hyperparameter design problem in neural architecture design. Recently, RL-NAS has been proven to be very effective in designing networks that achieve state-of-the-art accuracy in the object recognition[8] task - test error rate of 3.65. In our design, we, first, follow the traditional framework of this algorithm, that is, designing and RL-teaching a controller RNN to optimally generate CNN architectures. However, due to the large quantity of parameters to be learned in this model, that is, the controller's and children's parametrizations, RL-NAS is extremely computational-intensive. We propose a relaxation of this model by introducing a finite convolution operation set, which contains predefined image processing operations introduced in CS 415. With this set, the children's architectures need not learn filters from scratch, which greatly decreases the amount of learnable parameters. The task of the NAS controller is reduced to learning how to recurrently sample operations from this operation set to form children's architectures, and the task of the children is learning parameters in the baseline component. Our experimentation inspects if this relaxed model can carry out object recognition better than baseline CNN methods. We briefly define our proposed model as follows: given an operation set C, the goal is to design a DFA-NAS architecture that, in each iteration, sample operations from C and append them into a child architecture. After construction, it is attached to a baseline CNN, called DFA-CNN, and is evaluated using the CIFAR-10 image classification train-test process. The test results would be used to compute policy gradient[1] signals for the training of the NAS controller.

1. Introduction

Reinforcement Learning[2] (RL) is a branch of machine learning that deals with how to learn control strategies to interact with a complex environment. It's a problem design where an agent learns to maximize their reward-based objectives. RL[5] has many applications in many cutting-edge fields such as self-driving cars, robotics, and even winning Go, Chess, and Atari games. Deep RL[7] places the function approximation robustness of ANNs in the optimization problem.

Neural Architecture Search (NAS)[6] deals with searching for the optimal neural architecture for a given problem. ANN architectural topologies, which traditionally require lots of the programmer's involvement, offer great performance differences if designed efficiently. NAS is a systematic and automatic way of learning high-performance architectures. RL-NAS frames neural architecture design as an RL task, the problem is re-framed as learning a controller ANN that optimally samples child ANNs. The children undergo the traditional loss minimization training process and report an accuracy score that updates the controller's parameters.

2. Literature review

Fast and Practical Neural Architecture[3] framework uses a Directly Acyclic graph as a block of operations. The block-graph vertex represents operations such as element-wise addition, concatenation, and spit operation. The edges represent arithmetic operations, such as convolution or pooling, and identity mapping. The system shows its great generalization ability on ImageNet and ADE20K datasets for classification and semantic segmentation.

Surrogate-Assisted Neural Architecture[4] search which is built for generating task-specific models. The efficiency of this approach stems from surrogate modeling at two levels: one, at the architecture level, to improve sample efficiency, and two, at weights level to improve gradient descent efficiency. The resulting models either match or outperform models from existing approaches with the search

being orders of magnitude more sample efficient.

3. Experimentation setups

We decide to use the CIFAR-10 image dataset for the training and testing of child architectures that would be constructed by the NAS controller. We define our operation set to contain the following Computer Vision defined filters/operations we have learned in CS 415. The operation sets used in the project are as follows:

Harris Corner Detector: This algorithm is commonly used in computer vision to extract corners and infer features of an image. The corner is the point where two edges are joint. This corner is termed as interest points which are invariant to translation, rotation, and illumination.

Sobel Edge detector : It is a gradient-based method based on the first-order derivatives. It calculates the first derivatives of the image separately for the X and Y axes. The operator uses two 3X3 kernels which are convolved with the original image to calculate approximations of the derivatives - one for horizontal changes, and one for vertical.

Image scaling : It is one of the most important operations in Computer Vision problems. We generally scale up the image to get more details about the specific object, and sometimes we need to scale down the images to fit some criteria. We have used Half Scale Up and Half Scale Down in our experiment.

Image blurring: We have used Gaussian Blurring and Median Blurring as our operation set. While the "Gaussian" blur filter calculates the mean of the neighboring pixels, the "Median" blur filter calculates the median.

Image transformation : Rotation and translation of images are among the most basic operations in image editing. Both fall under the broader class of Affine transformations. We have used both rotation and translation as an operation set in our experiment.

3.1. The control performance

Because the goal of this project is to compare the performance of traditional CNNs with DFA-CNNs constructed by our DFA-NAS, we define a baseline CNN architecture that consists of a 4-filter convolutional layer, a max-pooling layer, and a densely connected linear layer. Why are we using the convolutional layer and not just the densely connected layer you may wonder? one-dimensional data loses a lot of its two-dimensional information so if we don't have at least a learnable convolution layer, the aided CNNs would do much better than the baseline by a landslide. This is because we hypothesize that post-processed images would be easier to recognize than the more cluttered version. After the training and testing of this baseline CNN, we record its performance as the control performance, called p zero.

3.2. DFA-NAS and DFA-CNNs

We define our DFA-NAS process as follows: Let it produces many iterations of 2-combination of operations withdrawn from the operation set. We only sample two operations for DFA-CNNs because too many operations would badly juggle up the signals in the original images. As we are not implementing identity mapping as seen in ResNet or DenseNet, two have proven to be the most stable number of operations. We, then, prepend these operations to the baseline CNN and call it DFA-CNN. After an arbitrary number of controller training iterations, we collect a DFA-CNN. We call the performance of this child architecture p hat. We can now compare p zero and p hat to see if the baseline CNN with the support of defined filters would do better than the baseline CNN alone.

Additionally, as stated in the project's goals, we would also be interested in the 2-combination that DFA-NAS has determined to work well together in assisting the baseline CNN to generalize object patterns in the CIFAR-10 dataset.

3.3. Assumptions

Gray-scaled CIFAR-10 images is analogous in the context of CNN fitting w.r.t. their RGB format There exists no performance upper-bound when the complexity of the model is scaled up Five epochs are enough for DFA-CNNs to express their generalization capabilities Instead of yielding the last performance, we average the performance over epochs to inspect the convergence rate aided by the defined filters

4. High-level details of our NAS implementation

To kick off this development process, we define the set of Computer Vision operations along with their implementations. each of these operations takes a float32 NumPy array, i.e., the encoded image, as input and output the filtered image. We also initialize our controller RNN, the generative neural network that would pick out operations from the operation set to put them together.

We loop a T number of times. Within each iteration, we sample an arbitrary noise uniformly from 0 to 1 as the initial input to our controller to kick off the generative process. For two times, the controller would sample an operation from the operations set to append into A. Of course, the controller would pick random operations at first. Over time, we hope that the controller would learn to pick operations that work well with previously picked operations (the correctness proof of this process can be found below). That is why we continuously feed previously picked operations as input for the controller to choose the subsequent operation.

Once we have our complete list of two-operation, we input it into an evaluate function that would initialize a base-

Algorithm 1 NAS pseudo-code

$C \leftarrow set\ of\ defined - filters$
$RNN \leftarrow random\ initial\ parameterization$
while $0..t..T-1$ **do**
  $A \leftarrow empty\ list$
  $C_0 \leftarrow random\ noise$
  while $0..t..T-1$ **do**
    $A \leftarrow C_k\ \ using\ RNN(C_{k-1})$
  end while
  $\rho_t \leftarrow evaluate(A)$
  $Compute\ log\ policy\ loss\ using\ \rho_t\ update\ RNN$
end while
$Function$ **evaluate(A)** $returns\ performance\ of\ DFA-CNN$
$DFA-CNN \leftarrow [A + 4\ filter\ convolutiionlayer + linear\ layer\ for\ class\ choosing]$
while $0..e..Epochs$ **do**
  $Train\ DFA\ CNN\ with(X\_train,\ Y\_train)$
  $using\ cross\ entropy\ loss$
  $\rho_e \leftarrow test\ DFA\ CNN\ (X\_test, Y\_test)$
end while
$Return\ avg(\rho_e)\ over\ all\ epochs$

line CNN with this operation list as the image transforms. We train these DFA-CNNs for three epochs with training data from the CIFAR-10 dataset and then collect its accuracy on the test set. We return the performance of this DFA-CNN as an average over three epochs because we want to capture the convergence rate of DFA-CNNs to compare with that of a normal baseline CNN.

With this performance score, we compute the gradient signal to update the controller using the vanilla policy gradient method.

5. Our method for Correctness Proofing

5.1. Correctness of Controller RNN training and Nuances

We define a further relaxation of the Controller RNN training process to inspect the correctness of our NAS implementation. Let the controller takes a deterministic noise as input and samples only one operation from a reduced operation set, i.e., the set containing 3-4 operations. Why is this relaxation needed? Policy gradient converges extremely slowly when the state space is large. In particular, given some initial stochastic noises, the controller would sample different initial operations each time. These initial operations affect the input state space for the sampling of the subsequent operations. Concretely, we see that the cardinality of the state space for sampling an operation at time step t is equal to the cardinality of operation set raised to the power of (t - 1).

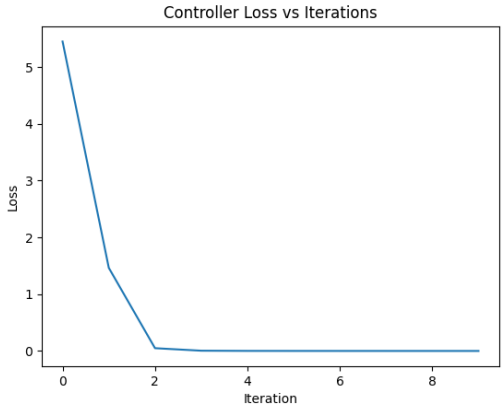

Figure 1: Controller loss over iterations

Furthermore, there is also the problem of high variance in computing gradient signal given a selected operation. There is no guarantee that a CNN would converge to the same optima at every iteration given a particular operation. Therefore, this hinders the convergence speed of our controller greatly. Ultimately, due to our limited computational resources, we use this relaxation to efficiently inspect the correctness of the policy gradient loss computation. Of course, it's worth priming that this assumes the convergence properties of our controller pertain to an increasing number of demanded operations.

If our implementation of policy gradient is correct, we would observe the controller's loss decreasing as the number of iterations increases. In detail, the controller should learn a probability distribution that is skewed towards the one operation that is most helpful in aiding the CNN in generalizing CIFAR-10 dataset. As seen in the plot below (also available in the correctness proof folder), the controller's loss decreases. In our log, we see that the controller becomes more and more interested in the Gaussian Blur operation until it sets its probabilities to 100 percent. This operation, indeed, does yield an expected performance higher than the average expected performance. Note that collecting the control performance is irrelevant here because with only one operation, DFA-CNNs are not receiving much advantage over the baseline.

5.2. Correctness of DFA-CNN evaluation

Our implementation of DFA-CNN evaluation follows the standard protocol of training and testing CNNs. We use a small learning rate of 1e-4 to minimize the possibilities that our CNN children would diverge and throw off the policy gradient signal. To further lower the variance of the gradient signal, we train these DFA-CNNs for only three epochs. Additionally, because their performance is to be contrasted

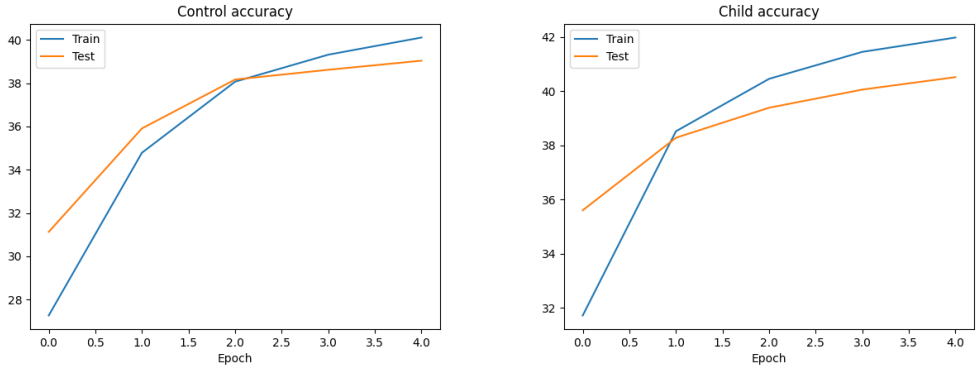

Figure 2: Control(baseline CNN) accuracy vs. Child DFA-CNN accuracy

with a baseline, we need not spend precious computation effort on maximizing their generalization aptitudes, i.e., letting each child trained until convergence.

6. Results and Conclusion

After many iterations of experimentation, we observe that the expected performance of DFA-CNNs is between 20-40. Therefore, we choose a baseline value of 30 for our policy gradient calculation. This value illustrates, that upon observing a state, i.e., a sequence of pre-selected operations, whether the to-be-selected operation gives more or less advantage over the average expected performance of said state. With this advantage value consideration, given an operation sequence, the controller RNN learns to improve probabilities of operations that would yield above 30 performances for said sequence and reduce probabilities of those that yield below 30.

We have also observed that, in the context of 2-combination of operations, DFA-NAS favors the choice of Sobel Filtering followed by Median Blurring/Half-up Scaling. In contrast, in the context of 1-operation, DFA-NAS favors rotation transformation or gaussian blurring while "disliking" Sobel Filtering. Our hypothesis is that Sobel Filtering alone would greatly reduce the information in images, rendering the dataset harder for CNN generalization. Applying a similar rotation transformation or blurring to all images in the dataset would not affect CNN generalization capabilities. However, if DFA-NAS is allowed to pick two operations, Sobel Filtering followed by Half-up Scaling would extrapolate information in the image, making CNN feature extraction easier. We note that the expected performance achievable by optimal children CNNs are higher than the control performance of the baseline CNN. Figure 2 represents the comparison of the accuracy between the baseline CNN model and the child model. This denotes that DFA-CNNs perform better than baseline CNNs. All resulting performance plots are in the results folder.

7. Prospective Improvement Directions

We can further extrapolate DFA-NAS capabilities by implementing identity mapping at each operation, which allows for a higher number of operation layers. We can increase the complexity of our baseline CNN to the state-of-the-art architectures to see how DFA-CNNs behave in contrast to this new baseline alone. We can add more complex Computer Vision operations into the operation set, or even pre-trained filters collected from other CNNs. Ultimately, there are still many directions for further experimentation with this Defined-Filter-Aided concept.

References

[1] Sutton, R. S., McAllester, D., Singh, S., & Mansour, Y. (1999, January 1). *Policy gradient methods for reinforcement learning with function approximation*, Advances in Neural Information Processing Systems. Retrieved December 8, 2021, from https://papers.nips.cc/paper/1999/hash/464d828b85b0bed98e80ade0a5c43b0f-Abstract.html. 1

[2] Zoph, B., & Le, Q. V. (2017, February 15) *Neural architecture search with reinforcement learning. arXiv.org*, Retrieved December 8, 2021, from https://arxiv.org/abs/1611.01578 1

[3] Cui, J., Chen, P., Li, R., Liu, S., Shen, X., & Jia, J. (n.d.) *Fast and practical neural architecture search*, Advances in Neural Information Processing Systems. Retrieved December 9, 2021, from https://openaccess.thecvf.com/content_ICCV_2019/papers/Cui_Fast_and_Practical_Neural_Architecture_Search_ICCV_2019_paper.pdf. 1

[4] Lu, Z., Boddeti, V. N., Banzhaf, W., Goodman, E., & Deb, K. (n.d.). *Michigan State University, East Lansing, MI 48824*, ... - arxiv. Retrieved December 9, 2021, from https://arxiv.org/pdf/2007.10396.pdf. 1

[5] Arash V., Arun M., Ming-Yu L., Jan K., *UNAS: Differentiable Architecture Search Meets Reinforcement Learning*, ... - arxiv. 2020, from https://arxiv.org/abs/1912.07651. 1

[6] Pengzhen R., Yun X., Xiaojun C., Po-Yao H., Zhihui Li, Xiaojiang C., Xin W. *A Comprehensive Survey of Neural Architecture Search: Challenges and Solutions*, ... - arxiv. Retrieved 2021, from https://arxiv.org/abs/2006.02903. 1

[7] Ngan Le, Vidhiwar Singh R., Kashu Y., Khoa L., Marios S. *Deep Reinforcement Learning in Computer Vision: A Comprehensive Survey*, ... - arxiv. Retrieved 2021, from https://arxiv.org/abs/2108.11510. 1

[8] Ning W., Yang G.,Hao C., Peng W., Z. Tian, C. Shen,Y.Zhang, *NAS-FCOS: Fast Neural Architecture Search for Object Detection*, ... - arxiv. Retrieved 2020, https://arxiv.org/abs/1906.04423. 1