Neural Architecture Search

Course Project: CSL 7540 - Artificial Intelligence - 1

Abhijit S Iyer
Department of CSE
Indian Institute of Technology, Jodhpur
Jodhpur, India
M21CS001

Jayesh Budhwani
Department of CSE
Indian Institute of Technology, Jodhpur
Jodhpur, India
M21CS007

Satwik Srivastava
Department of Mathematics
Indian Institute of Technology, Jodhpur
Jodhpur, India
M21MA006

***Abstract*—This report provides a sneak peek into a search algorithm implemented for 'Neural Architecture Search' (NAS), that was devised to find a neural network architecture that would return the best neural network in the form of a genome string, which would in-turn be fed to the training function to get the training accuracy. In our implementation of the search algorithm, we have made use of an 'Evolutionary Search Technique known as the 'Genetic Algorithm' to search for an efficient search architecture by mutating the existing architecture with other architectures try find out the best trade-off between accuracy and the number of parameters used.**

I. Introduction

Network Architecture Search(NAS) is a concept where a machine learns to find out the best possible architecture for a given purpose. Although human beings have manually designed neural networks and have come up with some of the best neural networks by designing them manually, we would rather prefer a machine generate it's own architecture for a given purpose going ahead. Therefore, NAS aims at making use of efficient search algorithms to generate an efficient neural network, achieving the right trade-off between accuracy and parameter count.

II. Problem Statement

The task was to implement the NAS algorithm to search in a space of Convolutional Neural Networks (CNN) to find out the best network that would work on the fashion mnist dataset. The best performing network would be one which would achieve an accuracy of at least 75% on the given test dataset. Another constraint was to achieve a trade-off between the accuracy and parameter count i.e. the search algorithm should be able to come up with a network that would achieve high accuracy with as less parameters as possible. The network would be represented with the help of a genome string, which the search algorithm should return.

A. Constraints

The following constraints had to be satisfied when searching for the required network:

- The network should be a sequential model i.e. it should not contain any skipped connections.
- The network may contain any number of normal (NC) CNN layers with stride = 1
- The network must contain exactly two reduction (RC) CNN layers with stride = 2
- The activation function applied at nodes could by any of the following: relu, sigmoid, tanh, swish, gelu
- The final layers must contain a global average pooling 2D layer, a fully connected layer with 64 units and a fully connected layer with 10 units
- The reduction and normal layers can be in any order as long as the above constraints are satisfied

III. Search Algorithms Experimented

To facilitate our purpose, we have experimented with two search algorithms:

A. Evolutionary Search: Genetic Algorithm

Evolutionary Search technique is based on Darwin's principles of evolution found in nature modelled as a search technique in computing. Here, we pick random samples from the search state space of the neural networks, and we mutate two or more networks at each iteration to generate new networks, which give rise to a new neural network architecture [1]. The newly generated architecture is then tested for its accuracy and parameters, before we proceed to mutate again for better results. Various types of mutations, such as swap mutations, inverse mutations, filter change mutations and mutation by adding layers from the network were experimented upon in the algorithm.

B. Random Search Algorithm

Random Search Technique is another local search technique where we generate random neural network architectures with the given constraints, and test each of the generated neural network for its accuracy and parameter count. The difference between the genetic algorithm and the random search algorithm is that in case of random algorithm, a new state/ network generated will have absolutely no relation to the previously generated network, and there is every chance that the same network may appear twice during the process of searching, something which isn't possible with genetic algorithms.

IV. Methodology

The algorithm that we had used [2] is as follows:

Algorithm 1 Aging Evolution

```
population ← empty queue                          ▷ The population.
history ← ∅                                       ▷ Will contain all models.
while |population| < P do                         ▷ Initialize population.
    model.arch ← RandomArchitecture()
    model.accuracy ← TrainAndEval(model.arch)
    add model to right of population
    add model to history
end while
while |history| < C do                            ▷ Evolve for C cycles.
    sample ← ∅                                    ▷ Parent candidates.
    while |sample| < S do
        candidate ← random element from population
                                                  ▷ The element stays in the population.
        add candidate to sample
    end while
    parent ← highest-accuracy model in sample
    child.arch ← Mutate(parent.arch)
    child.accuracy ← TrainAndEval(child.arch)
    add child to right of population
    add child to history
    remove dead from left of population           ▷ Oldest.
    discard dead
end while
return highest-accuracy model in history
```

To start off, we generate neural networks based on the given constraints, and the layers of this neural network are also generated based on the given constraints of filters, kernels and functions. The number of neural networks generated depends on the value of the population. Once the initial population is generated, we then proceed towards mutating the neural networks.

The different types of mutations that were experimented were:

- Swap Mutations
- Inverse Mutations
- Filter Change Mutations
- Layer change Mutations

With swap mutations, we swapped layers that were present in the existing neural network and then re-checked them for accuracy. In case of inverse mutations, a sub-string of the genome of the neural network would be picked randomly and would be inverted except for the final layer would be inverted and then the network would be checked for accuracy. With filter change mutations, we would choose any layer in the neural network and change the number of filters in it and then recheck for accuracy. In the layer change mutations, we would replace an existing layer in the neural network with a totally new layer, post which accuracy of the network on the given dataset would be calculated.

Out of the given list of activation functions, we observed that relu, sigmoid and tanh gave us the best results in terms of accuracies, hence we stuck to using the three.

The random search algorithm was another search algorithm that was experimented upon for comparison purposes, but it was observed that the execution time of the random search algorithm given the limitations was extremely high, and therefore we chose not to proceed with it further.

V. Results

Our experiments gave us results based on each of the different mutation techniques we had adopted. Each of them gave results that were highly accurate and also satisfied the necessary constraints.

The following table depicts the best results we observed on the various mutations we applied to the genetic algorithm, in terms of accuracy of model and parameters generated:

Mutation	Accuracy	Parameters
Filter	0.7470999956130981	1692572
Inversion	0.7447999715805054	170818
Swap	0.8227999806404114	60542
Add Layer	0.8995000123977661	220858

VI. Observations

- The number of filters used was kept to be lesser than 512, since beyond this limit the algorithm ended up taking extremely large amounts of time to compute results.
- By increasing model complexity, such as the number of layers or parameters, we observed that the accuracy of the model was affected really badly. Through experiments conducted during the process, we observed that the sweet spot with respect to the number of layers in the model is 5-6.
- The parameter count vs accuracy percentage tradeoff was not exactly observed, i.e. we did not come up with any exact relation between the two units. There were cases where high accuracy was observed with high number of parameters while lesser accuracy was observed with lesser parameters.
- Given that we had limited GPU capacity and time constraints of Google Collab, we were only able to test our algorithm with minimal sample and population sizes. 30 mins to 1 hr is the time taken to execute our algorithm for such cases, which is quite efficient. Given better infrastructure, we would be able to generate better results and better observations.
- Generating the set of the entire search space and checking for accuracy on each one of the samples would require immense computing power. Hence, we used the evolutionary search technique of genetic algorithm.

References

[1] Weng, L. (2020, August 6). Neural Architecture Search. Lil'Log. Retrieved November 17, 2021, from https://lilianweng.github.io/lil-log/2020/08/06/neural-architecture-search.htmlevolutionary-algorithms.

[2] Real, E., Aggarwal, A., Huang, Y., amp; Le, Q. V. (2019). Regularized evolution for Image Classifier Architecture Search. Proceedings of the AAAI Conference on Artificial Intelligence, 33, 4780–4789.