Neural Architecture Search (NAS)

Chiranjeev
chiranjeev.1@iitj.ac.in

Gyan Prabhat
prabhat.1@iitj.ac.in

Vibhuti Sharma
sharma.104@iitj.ac.in

***Abstract*—This report depicts the experiments carried out to find the best performing neural architecture for the Fashion-MNIST dataset from search space using Genetic Algorithm. The trade-off has to be maintained between the accuracy and number of parameters so as to find the best performing model with less computation.**

***Index Terms*—CNN, Genome, Search Space**

I. What is NAS?

Neural Architecture Search (NAS) automates network architecture engineering. It aims to learn a network topology that can achieve best performance on a certain task. By dissecting the methods for NAS into three components: search space, search algorithm and child model evolution strategy. [2]

II. CNN Architecture Constraints

Neural Architecture is created with different types of layers: Normal Convolution layer (NC), Reduction layer (RC), Fully Connected Dense layers, Global Average Pool and Final fully connected layer with number of classes(FL), with various activation functions and kernel sizes.

III. Objective

Our interest is to explore the search-space to identify such a genome (to be used as in input to the train function for our CNN architecture formed) which satisfies the given genome-constraints and yields a good accuracy in training and validation set, with less number of parameters.

IV. Understanding the Search-space

Suppose we wish to use a CNN architecture with N-layers ($N \geq 4$). The last (N^{th} layer) will be a FL layer and thus can be formed in 5 ways. (Because we have 5 possible activation functions).

Let us assume that the number of CNN-layers can take M possible unique values ($M > 1, M \in \mathbb{Z}^+$) and the kernel-size can take K possible unique values ($K \in [1, 7]$).This N-layered architecture will consist of a final (FL) layer and exactly 2 RC Layers. Thus, rest $(N-3)$ layers will be NC-layers.

Layer Type	Number of ways the layer can be formed
FL Layer	5
RC Layer	$\binom{(N-1)}{2} \times M \times K \times 5$
NC Layer	$(N-3) \times M \times K \times 5$

TABLE I: Possible ways to form different layers

Total number of ways we can form this architecture is:

$$\frac{5^3 M^2 K^2 (N-1)(N-2)(N-3)}{2}$$

Even with smaller values of N,M and K, we have large search-space. Thus, using grid search will not be a feasible idea to explore the search space.

V. Random Population

We adhere to the following specifications:

- Kernel size $K \in 1, 3, 5, 7$
- CNN-Filter $M \in [5, 64]$
- First layer is always a NC layer
- Number of layers $N \in [3, 10]$
- Activation function : relu,sigmoid,tanh,swish,gelu

We generate a random population of 10 CNN-architectures which makes our initial population. Using these CNN-architectures we train the classification model and capture the training and validation accuracy with.

Input arguments	max_filters (default 64) max_layers (default 10)
Return	A valid CNN architecture (s)

TABLE II: generate_arch function details

VI. Evaluation

Input arguments	population
Return	Sorted list of population arranged wrt to the training accuracy

TABLE III: evaluation function details

For every CNN architecture we generated as a population, we train the model using given train_model function and save its training accuracy, testing accuracy and number of parameters. Using these values, fitness_function calculates the fitness score. Finally, evaluation_function will return the population list sorted with respect to the fitness score.

VII. Fitness Function for evaluation

Input arguments	accuracy, number of parameters and alpha
Return	Fitness Score

TABLE IV: fitness_function function details

The evaluation of each CNN-architecture is carried out using the self-defined fitness function defined below:

$$Fitness = ((1-\alpha) \times p^*) - (\alpha \times n^*)$$

where, $p^* = \text{Accuracy score} * 10$
$n^* = \log_{10}(Floor(\frac{\text{Number of parameters}}{1000}) \times 1000)$

VIII. Crossover

Input arguments	population and generation_size
Return	next_generation

TABLE V: crossover function details

We get a new population such that this new population is generated by the initial population we generated randomly, we do a cross-over and we define the new set of valid CNN architectures as the new-population indexed by its generation.

IX. Mutation

Input arguments	top_k_generations
Return	new_top_k_generations

TABLE VI: mutation function details

Among the CNN-architectures generated in population in various generations, we select the top-k generations based on their fitness score in performing the classification task on the Fashion-MNIST data. We randomly mutate a genome based on random probability.

Probability	Action
< 0.13	Change filter size
$0.13 - 0.55$	Change number of filter
≥ 0.55	Change activation function

TABLE VII: probabilistic decision for mutation

Based on these actions, we get an entirely new set of CNN-architectures which are having new parameters but are generated from the initial population of genomes we had. Thus they are the mutated child architectures. This is equivalent of exploring the new elements in the search space.

X. Results

We conduct three experiments (four generations for each experiment) for various values of α. The results of the experiment with $\alpha = 0.3$ are shown below:

Genome	Architecture
S_1	NC 48 5 swish;NC 5 5 gelu;RC 22 7 gelu;RC 39 3 tanh; FL swish;
S_2	**NC 33 3 swish;NC 14 5 tanh;NC 14 3 tanh;NC 28 3 gelu; NC 24 5 relu;RC 63 3 tanh;RC 34 3 gelu;FL gelu;**
S_3	NC 24 5 gelu;NC 21 7 tanh;NC 43 3 tanh;RC 14 7 relu; NC 44 3 gelu;RC 41 5 relu;NC 9 3 relu;NC 12 1 sigmoid; NC 25 1 tanh;FL swish;

Model	Score(Accuracy)	Trainable Parameters	Fitness
S_1	85.55%	23636	4.6806
S_2^*	**88.45%**	**69925**	**4.7398**
S_3	89.09%	119841	4.6988

TABLE VIII: For $\alpha = 0.3$, Best genome string: S2

The results of the experiment with $\alpha = 0.5$ are shown below:

Genome	Architecture
S_1	NC 48 5 swish;NC 5 5 gelu;RC 22 7 gelu; RC 39 3 tanh;FL swish;
S_2	NC 25 5 relu;NC 30 1 tanh;NC 15 3 relu;NC 25 5 swish; RC 21 7 tanh;RC 26 7 tanh;FL sigmoid;
S_3	**NC 14 5 gelu;NC 6 5 gelu;RC 5 7 relu; RC 15 7 gelu;NC 9 7 sigmoid;FL gelu;**

TABLE IX: Selected genome architecture α = 0.5

Model	Score(Accuracy)	Trainable Parameters	Fitness
S_1	83.96%	14840	2.105
S_2	89.21%	69799	2.04
S_3^*	**85.65%**	**15549**	**2.19**

TABLE X: For α = 0.5, Best genome string: S3

The results of the experiment with $\alpha = 0.7$ are shown below:

Genome	Architecture
S_1	NC 7 7 gelu;NC 47 7 swish;RC 46 1 tanh;RC 7 7 tanh;FL gelu;
S_2	NC 46 3 tanh;RC 51 1 relu;RC 47 3 tanh;NC 6 5 gelu; NC 8 1 sigmoid;FL gelu;
S_3	**NC 5 3 relu;RC 17 5 tanh;NC 6 7 swish;NC 7 1 swish; NC 6 1 relu;RC 25 5 tanh;FL gelu;**

TABLE XI: Selected genome architecture α = 0.7

Model	Score(Accuracy)	Trainable Parameters	Fitness
S_1	86.11%	35673	-0.5975
S_2	80.72%	32815	-0.7317
S_3^*	**85.54%**	**13382**	**-0.3134**

TABLE XII: For α = 0.7, Best genome string: S3

From models mentioned in above tables tried with different alpha values, we observed that with α = 0.7 we get better performing model with accuracy 85.54% with very few parameters(13382). This is the best model as we compare it to other best performing models in their respective alpha values tables, these models have more number of parameters as compared to the overall best performing model. So, the Best genome is: NC 5 3 relu;RC 17 5 tanh;NC 6 7 swish;NC 7 1 swish;NC 6 1 relu;RC 25 5 tanh;FL gelu;

References

[1] David Laredo,Yulin Qin,Oliver Schutze and Jian-Qiao Sun. *Automatic Model Selection for Neural Networks*
[2] Lil'Log (Online Blog) [Link]