CSE 847 Project Final Report
Neural Architecture Search using Evolutionary Optimization Algorithm

Submitted by: Ritam Guha (MSU ID: guharita)

Date: April 21, 2021

1 Introduction

Deep learning algorithms have made immense progress over the years in various domains like image classification, speech processing, language translation, fraud detection, virtual assistance and much more. The backbone of these deep learning models are the novel architectures which are proposed by multiple researchers working in the domains for many years. With years of experience, they are able to identify the architectures which are suitable for particular domains. This kind of approach has two major issues:

1. Proposing a novel architecture in a domain requires years of expertise in the domain which makes it an extremely time consuming process. Even after getting a blue print of an architecture, properly optimizing it and customizing it for different problems in the same domain requires hige computational processing. So, it takes a lot of time to come up with an appropriate architecure manually designed by domain experts.

2. Due to the human intervention in the process, it becomes prone to a lot of errors.

The most reasonable way to get rid of these problems is to automate the process of generating a neural architecture customized for a problem which is known as **Neural Architecture Search (NAS)**. It is not an easy task because the architecture space is unbounded and it is practically impossible to search all the possible architectures to find the best architecture for a problem. So, to make the NAS feasbile, domain expertise is used to restrict the search space and find reasonable architectures using available computational resources within a limited time bound. The NAS methods can be cateogorized using three different components which are described below:

- **Search Space:** The search space of ideal NAS is unbounded but practically infeasible. Here domain expertise is used to reduce the search space so that the researchers are able to find appropriate architectures using limited resources and in reasonable time. Restrictions on components like the number of vertices, number of operations, types of operations etc. may reduce the search space a lot because the size grows exponentially with an increase in these factors. People working in the domains for a long time can propose restrictions on these factors providing a starting point for the search.

- **Search Strategy:** After defining the search space, we need a process to iteratively search through the space to find a suitable architecture. Search strategy is really important to reduce the time it takes to find an appropriate architecture on the defined search space. It also requires a balance in the exploration and exploitation procedures as we want to find an architecture as quickly as possible without facing a premature convergence.

- **Performance Estimation:** In order to proceed with the search strategy and to know if our goal is met, we need a performance metric to evaluate the architectures found in the process and to compare two separate architectures. The simplest option is to train the architecture on the training data and compute the validation accuracy which can be then used as a performance metric. But it is computationally expensive and requires a lot of time which significantly restricts the number architectures that can be evaluated. Researchers are trying to find other ways to approximate the performace of the architectures like lower fidelity estimates [1] (train the architecture partially and use that for computing valdiation accuracy), learning curve extrapolation [2] (after some epochs, stop the learning process and extrapolate the learning curve to get an estimate), weight ineritance [3] (Instead of starting from scratch, use some pre-trained weights to speed up the learning process) etc.

So, from the above discussion, we can see that each of these components of NAS can be considered to be a huge research topic and requires proper analysis to speed up the NAS process. [4] summarized the process of NAS using Figure 1.

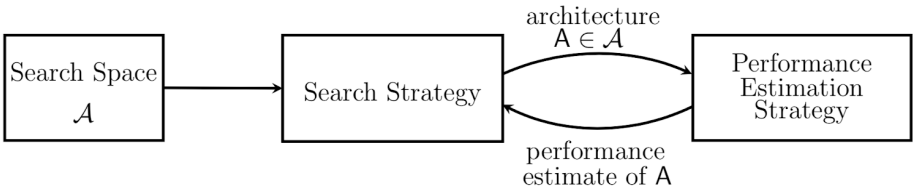

Figure 1: Pictorial representation of the Neural Architecture Search Process

From this introduction, it becomes clear that NAS is a tough process and requires proper exploration-exploitation trade-off to reach to a proper solution. Evolutionary Computation can provide us a sophisticated way to search for architectures in an iterative fashion. As a part of this project, I am using a famous evolutionary apporach called Genetic Algorithm (GA) [5] as the search strategy to explore a pre-defined search space and find the global optimum solution. The main concepts of evolutionary optimization which help us derive an appropriate neural architecture are:

- It uses a population of candidate solutions where each candidate solution is a valid architecture in the search space. Due to the introduction of a population, instead of a single solution, it is way better at finding the global optimum instead of getting stuck at some local optimum.

- It uses evolutionary operators like crossover, mutation, selection which helps to generate new and better solutions from the existing pool of solutions. Although it does not gurantee to find the global optimum, it can provide us a near-optimal solutions within a reasonable time duration.

2 Dataset

The process of NAS is extremely time consuming and computationally expensive. So, it is almost infeasible if there is some restriction on the computational resources. But, in 2019, Ying et. al created a benchmark called NAS-Bench-101 [6] which reduced the computational expense of performing NAS on the proposed search space to a large extent. For the project, I have used a couple of NAS-Benchmarks: NAS-Bench-101 and NAS-Bench-201.

2.1 NAS-Bench-101

NAS-Bench-101 is a table mapping a set of architectures in a proposed search space to various metrics like training accuracy, validation accuracy, training time etc. So, the cost of performance evaluation becomes almost negligible. After coming up with an architecture, it can be evaluated just by a query to this table.

2.1.1 Search Space

In the domain of NAS, two different types of architectures are considered: Micro-architecture and Macro-architecture. Macro architecture is the bigger snapshot of the architecture or how different components in the architecture are connected to each other. But when we get deep inside different components, that kind of small-scale architecture is known as Micro architecture. NAS-Bench-101 restricts the search to a type of Micro architecure called cells which are small feedforward networks acting as building blocks for bigger architectures. So, the Macro architecture remains fixed while the Micro architecture is optimized. Each cell is stacked 3 times followed by a downsampling layer. This pattern is repeated thrice, followed by global avg pooling and final dense softmax layer. The Macro architecture used in NAS-Bench-101 is displayed in Figure 2

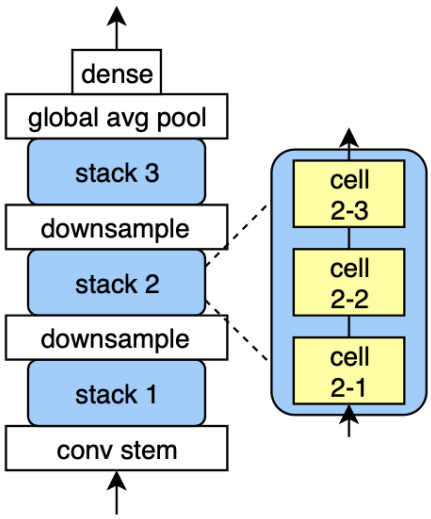

Figure 2: The Micro Architecure used in NAS-Bench-101

The entire search space contains all possible architecures formed by Directed Acyclic Graphs (DAGs) on V nodes and each node has L possibilities in terms of the type of operation. The allowed operations are 3×3 convolution, 1×1 convolution, 3×3 max-pooling and the maximum

size of V is 7 while maximum number of edges is 9. Out of the 7 vertices, 2 are IN and OUT vertices while rest are intermediate vertices.

There maximum number of edges in a graph with 7 vertices is $\binom{7}{2}$=21 and every vertex has 3 possible operations. The maximum number of architectures on such a graph is: $2^{21} \times 3^5 = 510M(approx.)$. But after considering all the restrictions and removing duplications, the space contains $423K$ unique architectures.

2.1.2 Training Information

Each of the architecture are trained and evaluated on CIFAR-10 dataset. Every architecture has been run for 3 times and using 4 epoch schedules $(4, 12, 36, 108)$. So, for every architecture, there are 12 mappings altogether which gives $423K \times 12 = 5M(approx.)$ models.

2.1.3 Metrics

Every entry in the table contains 5 evaluation metrics: training accuracy, validation accuracy, testing accuracy, training time in seconds, number of trainable model parameters.

2.1.4 Complexity in Search

After proper examination of the search space, the authors found out that there are only 11570 models which can be considered as elites among these $510M$ models. So, the probability of hitting one of these elite architectures is 1 to 50000 which makes it a really difficult task.

2.1.5 Issues

NAS-Bench-101 was the first proposed NAS-Benchmark which reduced the amount of computations to a large extent. But it had some associated problems which made it somewhat difficult to work with.

- If we consider the search space of the dataset, it can be seen that it is huge. It was not possible to explore all the architectures in the space. So, the authors restricted the search space by imposing some constraints (For example, the maximum number of edges was restricted to 9). Due to these restrictions, any algorithm should be modified to consider the restrictions before application.
- NAS-Bench-101 has only used CIFAR-10 dataset. So, if researchers want to perform generalized NAS that will provide an architecture working well over different datasets in Image Classification, it is not possible with NAS-Bench-101. The architecture will get biased towards CIFAR-10 dataset.
- The authors of NAS-Bench-101 did not provide the final parameteric values of the architectures. So, at the end of the NAS, the obtained archictecure needs to be retrained on CIFAR-10 to get the parameters.
- The architecture information provided by NAS-Bench-101 is very limited. It can be seen from subsubsection 2.1.3 that only 5 metrics are provided for each run of an architecture. There is no information regarding the training log (epoch-wise loss or accuracy, training cost) in NAS-Bench-101.

2.2 NAS-Bench-201

Although NAS-Bench-101 proposed a new avenue in the direction of NAS Benchmarking, it's popularity reduced after some time when a Dong et. al proposed a new Benchmarked called NAS-Bench-201 [7]. It tried to address the limitations faced by NAS-Bench-101 and was successful in alleviating some of the problems. In recent times, NAS-Bench-201 has become the most popular benchmark among the researchers. This made me motivated to use NAS-Bench-201 for the project as well.

2.2.1 Search Space

Just like NAS-Bench-101, NAS-Bench-201 also limits its search to different cells. The Micro architecture remains the same, while the Micro architecture or cells are updated.

The Micro architecture used in NAS-Bench-201 is shown in Figure 3. It starts with a 3×3 convolution having 16 output channels followed by a batch normalization layer. The main skeleton includes $N = 5$ stacked cells followed by residual blocks having a stride of 2. After the final stack of cells, the feature maps pass through a global average pooling layer, followed by flattening. The flattened vector is connected to a Fully Connected layer which passes the maps to a softmax layer that makes the final classifications.

Similar to NAS-Bench-101, NAS-Bench-201 also uses DAGs to repersent the cells. The difference between these two representations is that NAS-Bench-201 places the operations on the edges while the former one places them in the vertices. All the cells have $V = 4$ vertices and $L = 5$ possible operations: (1) zeroize, (2) skip connection, (3) 1×1 convolution, (4) 3×3 convolution and (5) 3×3 average pooling.

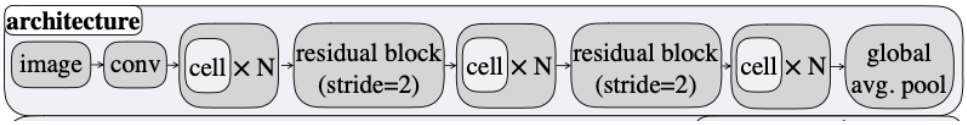

Figure 3: The Micro Architecure used in NAS-Bench-201

2.2.2 Training Information

Each possible architecure in the search space has been tested over three different image classification datasets: CIFAR-10, CIFAR-100 (Fine-grained version of CIFAR-10 with 100 classes), ImageNet-16-120 (Down-sampled variant of ImageNet with 120 classes). The description of these classes and different hyperparameters used by the authors can found in [7].

2.2.3 Diagnostic Information

NAS-Bench-201 comes with a lot of information regarding the training, validation, testing. It provides the training log of the architecture containing epoch-wise training loss, accuracy. The architecture computational costs in terms of FLOPs (Floating Point operations), Latency information are also provided in the benchmark. Finally, the architecures also contain the final parametric values obtained during the time of the training which makes it easier to use the final architeture right away without the need of re-training.

2.2.4 Variation in Search

With the availability of so many diagnostic information, researchers are able to search for architectures based on different criteria. If someone wants to perform resource constrained NAS, that is also possible with NAS-Bench-201 because the training log, count of FLOPs are also available.

2.2.5 Highlights

Although NAS-Bench-101 started the process of NAS Benchmarking, NAS-Bench-201 has been a major improvement in that direction. The key highlights of NAS-Bench-201 that makes it so popular are:

- There is no restriction on the search space. Any architecture defined on the search space is considered to be a valid architecture. Due to this reason, any NAS algorithm becomes applicable to the benchmark in its standard form.
- One of the strongest points of NAS-Bench-201 is the amount of statistical information it offers for each of the architectures. Researchers can easily use these information to gain deeper insights to the NAS process.
- NAS-Bench-201 uses three different Image Classification datasets which makes it easier to perform generalized NAS where the goal is to find a suitable architecture which works well for all the available datasets.

3 Project Description

For the project, I have used NAS-Bench-101 and NAS-Bench-201 as the underlying benchmarks and formulated a GA to apply over them.

3.1 Proposed Method

The flowchart of the implementation is presented in Figure 4. The GA variant has three very important stages as described below:

1. **Initialization:** The GA starts with random initialization of a population of candidate solutions. Each candidate solution is created by randomly choosing values from the allowed values in the two benchmarks. In GA, these candidate solutions are called chromosomes.
2. **Crossover and Mutation:** The heart of GA lies in its crossover and mutation operators. After initializing the chromosomes, two parent chromosomes are selected through tournament selection. The selected parent chromosomes participate in one-point crossover (blending of the parent chromosomes) to create two child chromosomes. The child chromosomes are then mutated for re-adjustment.
3. **Replacement:** If the final child chromosomes are fitter than their parents, they are replaced in the population and can participate in the next iteration of reproduction.

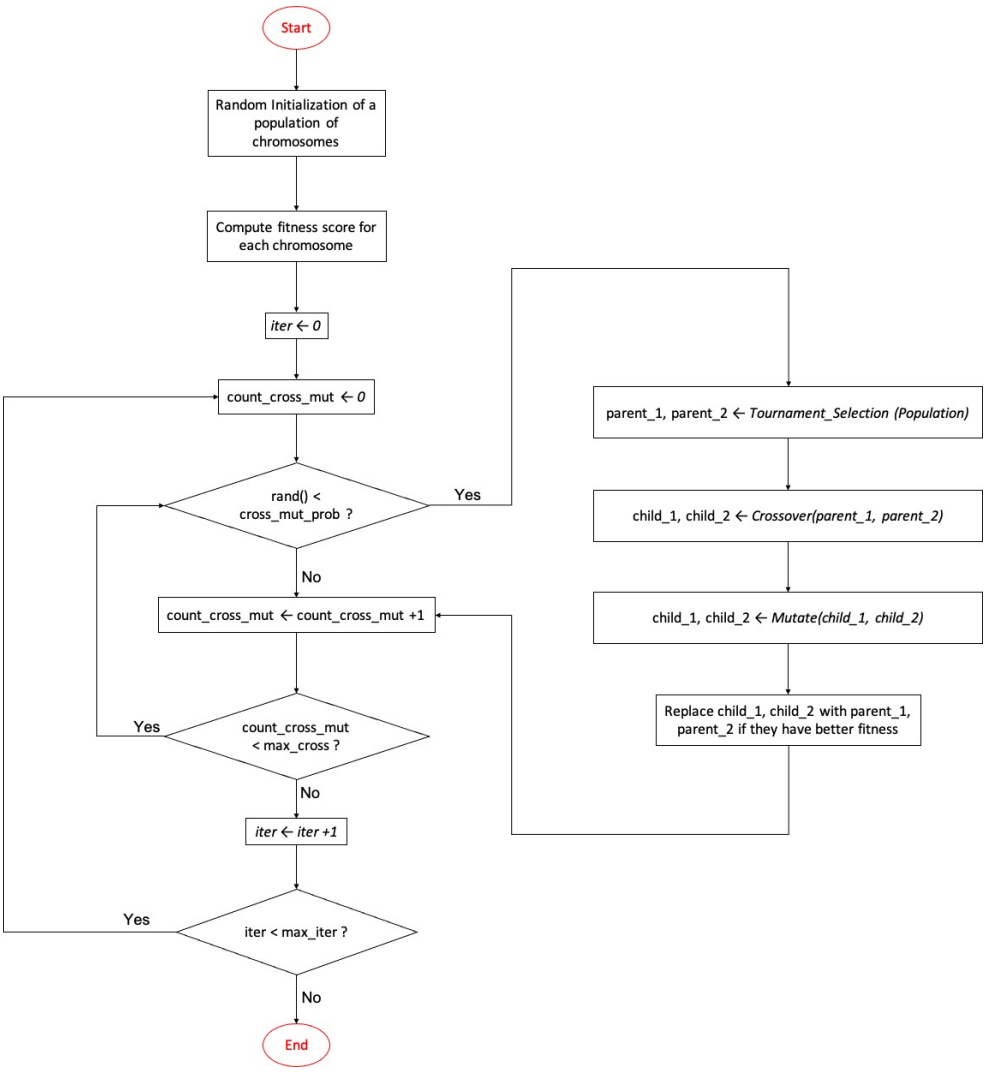

Figure 4: A flowchart of the Used GA Implementation

For formalizing GA, we need to first specify three very important things: Solution Representation, Fitness Function and Position Update.

- **Solution Representation:**

  NAS-Bench-101: Each candidate solution for NAS-Bench-101 is represented as a (matrix, list) pair. Each matrix is upper triangular binary in nature and it represents a 7-vertex DAG where 2 vertices are for INPUT and OUTPUT while the other 5 vertices are for intermediate operations. The list contains the type of operations to be used. For example, consider the following architecture:

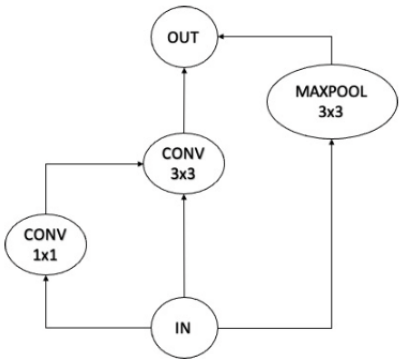

Figure 5: Example of a valid architecture in NAS-Bench-101

This architecture can be represented as a pair of matrix and list as:
$$\begin{pmatrix} 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$, list=[IN, Conv1x1, Conv3x3, Maxpool3x3, Conv1x1, Cov1x1, OUT]

Please note that the final 2 Conv1x1 operations have no meaning in the context. They are added to the list to make the sizes of the matrices and lists the same which allows us to apply evolutionary operators like crossover over the architecture represenetations. The corresponding 5^{th} and 6^{th} rows of the matrix are seen to be all 0s which means they don't hold any meaning in the architecture.

NAS-Bench-201: As mentioned before, NAS-Bench-201 places the operators on the edges instead of the vertices. So, the solution formulation needs to be changed. In case of NAS-Bench-201, an adjacency matrix can be formed by placing the type of operators inside the matrix. Consider the following architecture in NAS-Bench-201:

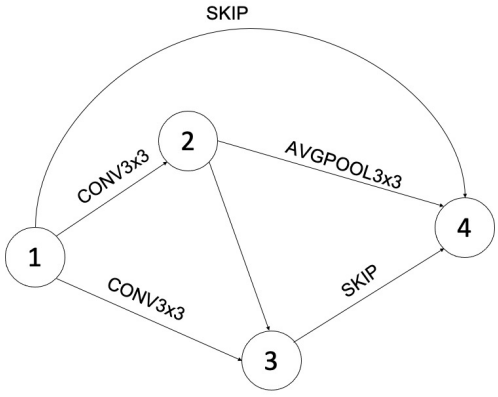

Figure 6: Example of a valid architecture in NAS-Bench-201

This architecture can be represented as an adjacency matrix as mentioned below:

$$\begin{pmatrix} \prime & \text{Conv3x3} & \text{Conv3x3} & \text{Skip} \\ \prime & \prime & \text{Avgpool3x3} & \text{Conv3X3} \\ \prime & \prime & \prime & \text{Skip} \\ \prime & \prime & \prime & \prime \end{pmatrix}$$

But NAS-Bench-201 uses its own string format for input. So, finally this matrix will get converted to

```
|nor_conv_3x3~0|+|nor_conv_3x3~0|avg_pool_3x3~1|+

|skip_connect~0|nor_conv_3x3~1|skip_connect~2|
```

This conversion is done using a function in the implementation.

- **Fitness Function:**

  NAS-Bench-101: The validation accuracies for the architectures are used as fitness scores for this benchmark. To evaluate every candidate solution, the validation accuracy can be obtained just by querying the dataset using the solution representation as the parameter.

  NAS-Bench-201: With NAS-Bench-201, the goal is to find a generalized architecture for all the three datasets. For this reason, the test accuracies averaged over all the three datasets are used as fitness measures for the architectures.

- **Position Update:** For position update, I have used three important evolutionary operators: tournament selection, crossover and mutation.

  Tournament Selection: It is a process for selecting parents in the population of candidate solutions. From the population, k random solutions are selected and are said to participate in a tournament. The best solutions out of those k solution in terms of fitness scores are the winners of the tournament and are eligible to become parents.

  Crossover: After the parents are selected, they undergo crossover which is the process of crearting child solutions by intermixing information from multiple parent solutions. For performing crossover, I am selecting a crossover point and the first child is created by taking the operations and corresponding columns of parent 1 till the crossover point and the same things after crossover point from parent 2. In a similar way, the second child is also created by swapping the parent numbers for the first child creation process.

  Mutation: The child solutions created from crossover are mutated by randomly changing the values for the matrices and operators. For NAS-Bench-101, sometimes this leads to invalid representations because of the restrictions imposed by the benchmark. The process is continued till a valid representation is found. But for NAS-Bench-201, it is not an issue.

3.2 Code

The codes used for both the benchmarks are available at the following Github Repository: NAS_Project.

3.3 Experimental Results

This section contains the experimental outcomes of the project. The GA implementation mentioned in the previous section was applied over NAS-Bench-101 and NAS-Bench-201 for proper testing of the implementation.

3.3.1 NAS-Bench-101

After implementing the code, it was tested on NAS-Bench-101 dataset. The generation-wise convergence graph for the implementation is presented in Figure 7. The Figure represents the situation for 30 independent runs of GA. At each run, we can see that the behavior of the runs are almost the same with little to no variance. It denotes that the GA implementation is pretty robust for NAS-Bench-101. The main goal of the optimization algorithm is to improve the average value over the generations. From the trend of the fitness scores, it can be observed that the GA implementation is working great over NAS-Bench-101 dataset.

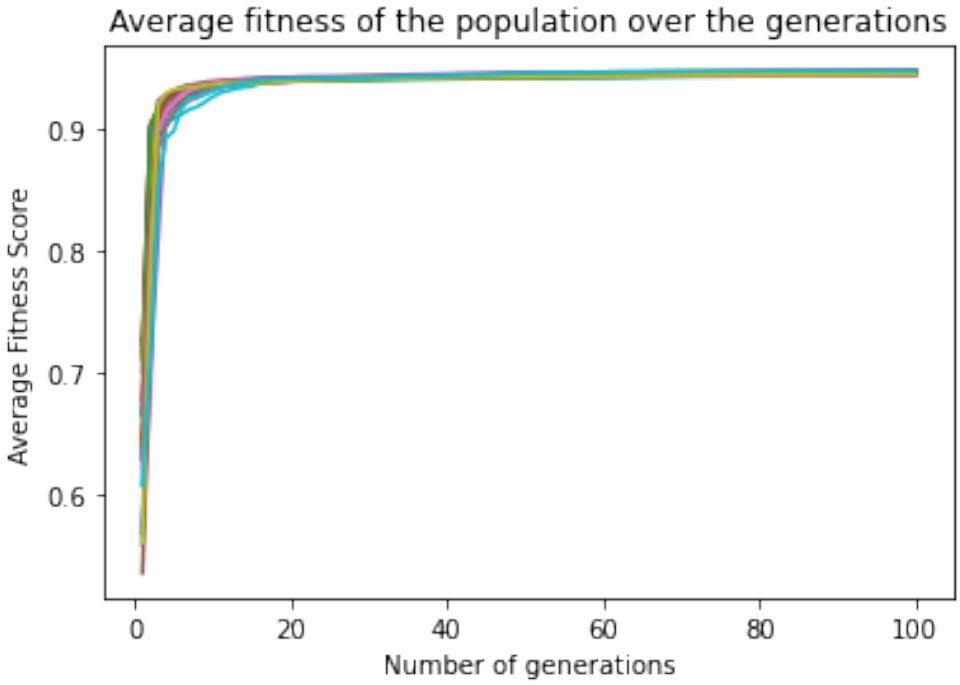

Figure 7: Convergence of the average fitness score over the generations

After checking the performance of the GA implementation, the next step was to compare it against some other algorithms. For this purpose, Random Search (RS) [8] and Regularized Evolutions (RE) [9] were run on the same benchmark. The comparison illustration can found in Figure 8.

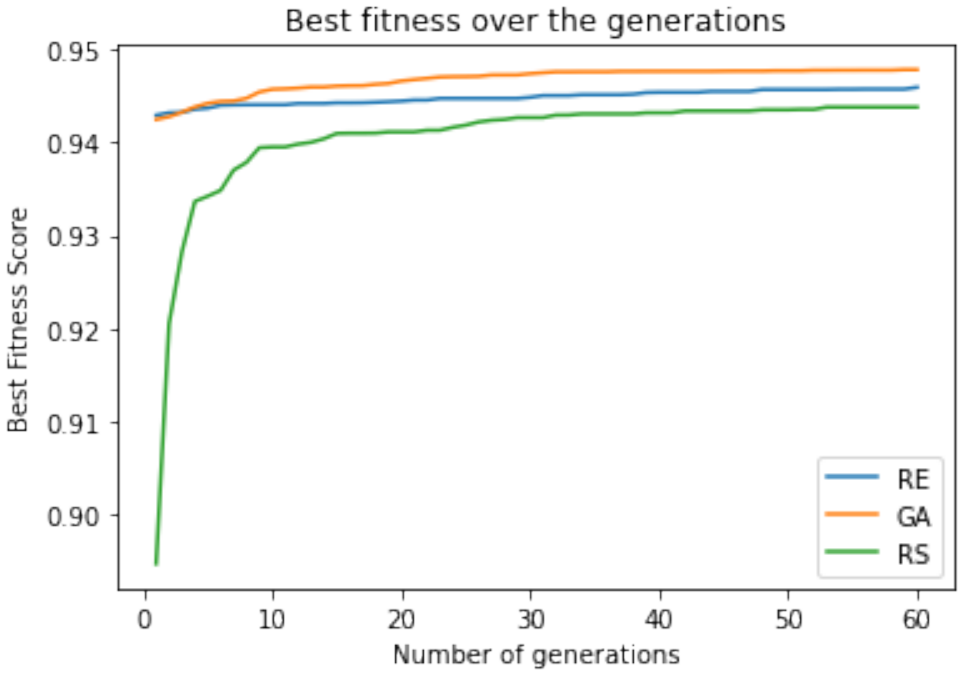

Figure 8: Comparison of different algorithms over NAS-Bench-101

It can be observed that the GA implementation was able to surpass the other algorithms. But from the plots, it is visible that it is quite easy to get to a ≥ 0.93 fitness score. Even RS is getting to a decent accuracy in a very short time. This makes NAS-Bench-101 quite simple and the elite architectures of the benchmark can be easily found. So, not much effort is required from the perspective of the NAS algorithm. This is the reason for moving towards more difficult benchmark like NAS-Bench-201.

3.3.2 NAS-Bench-201

After checking the outcome of the experimentation with NAS-Bench-101, I got interested to explore other benchmarks as well. NAS-Bench-201 is currently the most popular NAS Benchmark. After application of the GA implementation over NAS-Bench-201, the trend of convergence in Figure 9 was found to be completely different than NAS-Bench-101. In case of NAS-Bench-101, all the GA runs converged within first 20 generations. But in case NAS-Bench-201, GA needed almost 300 generations to get converged. It clearly shows the difference in complexity of the two benchmarks. Although their is some variance among different runs, all the runs were able to get to ≥ 60 in the fitness score. Please note that this is the average of the fitness scores over 3 datasets and the original test scores are considered.

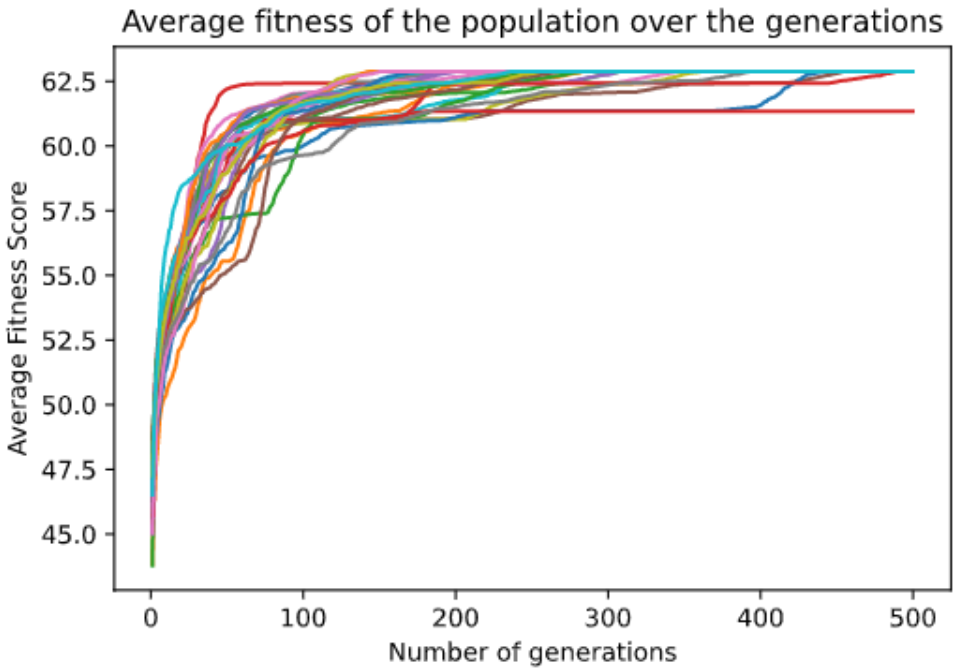

Figure 9: Convergence of the average fitness score over the generations

After getting the outcome for the fitness involving all the three image classification datasets, I was curious to check the performance of GA for every dataset. Figure 10 shows the performance of GA for each dataset. Two types of performances are plotted in the Figure. For each dataset, the generation-wise average fitness of the population is plotted in the left column of the diagram while the best fitnesses are plotted in the right column. From the diagrams, it is visible that the GA implementation provides pretty robust performance across the datasets.

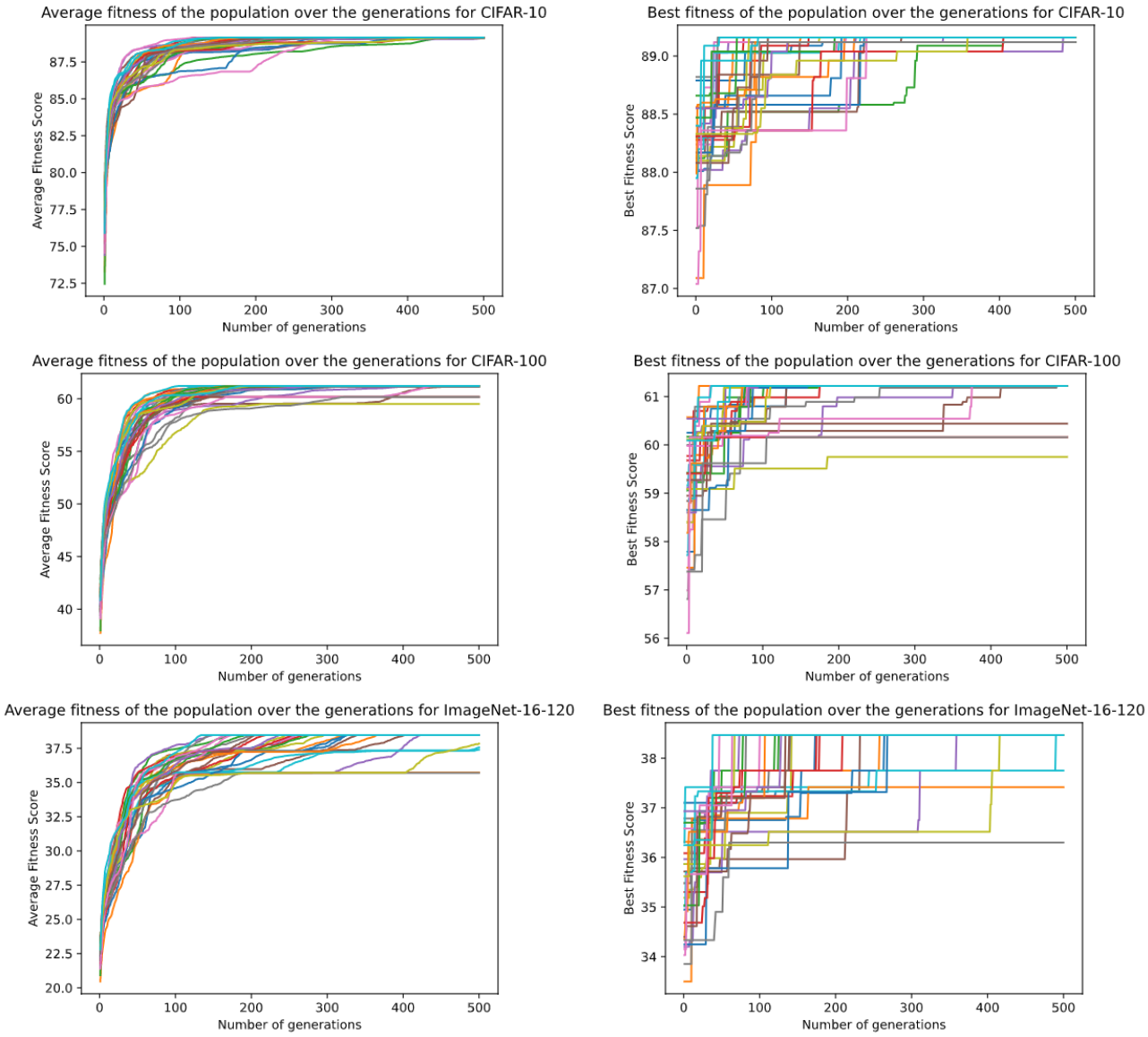

Figure 10: Performance of the GA implementation for every individual dataset

A similar study was made to find the superiority of GA over RE and RS. The plot can be found in Figure 11. The GA implementation improved the quality of the solutions more than the others throught the generations.

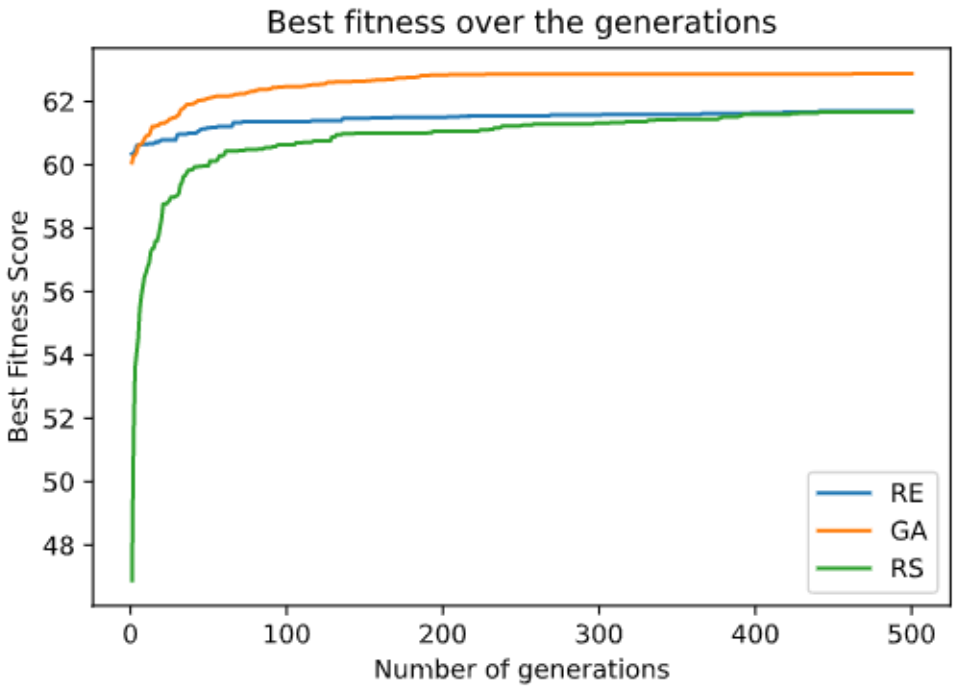

Figure 11: Comparison of different algorithms over NAS-Bench-201

3.3.3 Objective Injection

With NAS-Becnh-201, it is possible to work with multiple contradicting or non-contradicting objectives. From the diagnostic information provided in the benchmark, the algorithms can be adjusted to consider different objectives or even multiple objectives at a time. The only step needed to change the GA's objective is to modify the fitness function metnioned in subsection 3.1. So, for example, if someone wants to do resource-constrained NAS, it can be done by simply including the terms like Latency, FLOPs in the definition of the fitness function.

4 Conclusion

As a part of this project, an evolutionary algorithm namely Genetic Algorithm has been used to perform Neural Architecture Search in different search spaces. The process of NAS is extremely time consuming and resource intensive. In order to cut the computational cost, two different NAS Benchmarks have been used to get the associated values of searched architectures without training them. From the experimental outcomes, it is clear that the GA implementation has been able to perform decently in different scenarios.

The two benchmarks used for the project are: NAS-Bench-101 and NAS-Bench-201. The limitations of NAS-Bench-101 has been explored in detail and how NAS-Bench-201 alleviates some of those limitations are also discussed. The project was able to reach the goal of exploring GA's applicability in the field of NAS. From careful observation, it can concluded that GA is able to achieve performances similar to state-of-the-art algorithms in NAS. NAS-Bench-101 is easier to explore than NAS-bench-201 but NAS-Bench-201 is more appropriate for challenging different NAS algorithms. It comes with a lot of diagnostic information as well which makes it more insighful

than NAS-Bench-101.

References

[1] Lisha Li, Kevin G Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: Bandit-based configuration evaluation for hyperparameter optimization. In *ICLR (Poster)*, 2017.

[2] Kevin Swersky, Jasper Snoek, and Ryan Prescott Adams. Freeze-thaw bayesian optimization. *arXiv preprint arXiv:1406.3896*, 2014.

[3] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V Le, and Alexey Kurakin. Large-scale evolution of image classifiers. In *International Conference on Machine Learning*, pages 2902–2911. PMLR, 2017.

[4] Thomas Elsken, Jan Hendrik Metzen, Frank Hutter, et al. Neural architecture search: A survey. *J. Mach. Learn. Res.*, 20(55):1–21, 2019.

[5] John H Holland. Genetic algorithms. *Scientific american*, 267(1):66–73, 1992.

[6] Chris Ying, Aaron Klein, Eric Christiansen, Esteban Real, Kevin Murphy, and Frank Hutter. Nas-bench-101: Towards reproducible neural architecture search. In *International Conference on Machine Learning*, pages 7105–7114. PMLR, 2019.

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

[8] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(2), 2012.

[9] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pages 4780–4789, 2019.