Hyperparameter Optimization in Deep Learning Using Genetic Algorithm

Akhil Konda (ak4590) Rachita Naik (rvn2103)

Motivation:

Deep learning(CNNs) has gained a lot of attention because of their performance on various computer vision tasks such as Image Classification, Object detection and Localization. However, finding the best model requires numerous experimentations with different hyperparameters such as number of layers, number of filters , activation type etc. which is both time consuming and requires a lot of computational resources. The process of finding best hyperparameters becomes more difficult with increasing depth of the models. As the depth of deep learning models increases, the number of hyperparameters grows exponentially, and searching optimal settings becomes harder.

Genetic algorithms have been used in hyperparameter optimizations. However, traditional genetic algorithms with fixed-length chromosomes may not be a good fit for optimizing deep learning hyperparameters, because deep learning models have variable number of hyperparameters depending on the model depth. It is highly important to have an efficient algorithm to find the best model in reasonable time. In this project, we used a variable length Genetic Algorithm[1] to systematically and automatically tune the hyperparameters of a CNN to improve its performance. Moreover, we also modified the cross-over operation in the Genetic Algorithm to produce fitter individuals (models) in the next generation. We found that this Variable Length Genetic Algorithm has significantly improved the performance of the model by finding the best hyperparameter settings in less time than classical genetic algorithms. Furthermore, our modified cross-over operation is producing better children in terms of various metrics such as average fitness, best fitness compared to the original cross-over operation. To ease the burden of logging and tracking various metrics in different stages of the algorithm, we integrated weights and biases developer tools in our code.

The rest of the report deals with the description of the numerous steps involved in variable length Genetic algorithm and also details the modifications to the cross-over step.

Genetic Algorithms:

Genetic algorithms are biologically inspired. It learns from Darwin's evolutionary theory, i.e, the fittest individuals are selected and they produce offspring. The characteristics of these individuals are passed on the next generations. If the parents have higher fitness, their offspring tend to be fitter and have more chances of survival.

Genetic algorithms learn from this idea. They can be used to solve optimization and search problems. In genetic algorithms, candidate solutions are evolved to generate better ones. A set of solutions form a

search space and the goal is to find the best ones among them. This is similar to finding the fittest individual in a population. Genetic algorithms start with a population that contains random solutions within the search space.

A chromosome represents each solution in the search space. The chromosomes are altered using the following three operations: selections, crossover and mutation.

Selection: It is the process of selecting certain individuals based on some fitness criteria to produce off-springs and generate more solutions.

Crossover: It is the process of producing new off-springs by combining chromosomes of two parents. The off-springs have characteristics of both parents involved in crossover.

Mutation: The Mutation operator is like biological mutation: it changes one or more values in the chromosome. This introduces more diversities in the population.

Variable Length Genetic Algorithms:

Classical GAs have fixed chromosome length. However, the depth of a model is one of the hyperparameters that play a key role in the model's performance, hence having a fixed length chromosome is not suitable for the task at hand.

A chromosome in the variable length GA contains parameters that define a solution, in this case, a hyperparameter configuration for a network model. The following figure-1 is a representation of the entire variable length genetic algorithm.

Steps in Variable length Genetic algorithm:

1. Initialize the size of population p, number of generations g according to user input and number of phases (fixed or variable).
2. Create the initial population. Each individual in the population has randomized hyperparameters.
3. Train and evaluate the fitness of each individual in the current generation. The fitness is the accuracy of the individual on the validation dataset
4. Sort each individual according to their fitness value, from high to low. Select a portion of the population that has the highest fitness values, and let them survive into the next generation. The percentage of this survival rate can be set manually.
5. For the rest of the individuals in the current population, give them a small chance to survive into the next generation to add randomness.
6. Randomly mutate some individuals in the next generation. If an individual is chosen to mutate, one of its hyperparameter’s values will be changed.
7. Perform crossover operation by selecting two parents from the next population based on a weighted probability determined according to the fitness values to produce a child.
8. Each hyperparameter in the child is randomly set to be one of its parent’s hyperparameter values. Repeat the crossover operation on different random individuals many times until the next generation has p individuals.

9. Now the next generation becomes the current generation.
10. Repeat steps 3-9 until the number of generations has reached g.
11. Select the best individual from the current population in the last phase.
12. Produce populations with longer chromosome length. For each individual of the population, one part of the chromosome is from the best individual in Step 11, the other part is randomly generated.
13. Repeat steps 10 - 12 until we reach the number of phases.
14. Select the best individual and train for more epochs until convergence.
15. To save time, the fitness of each individual is evaluated only after five training epochs.
16. Weight transfer is also used to save the experiment running time. If a model survives into a new phase, it does not need to be trained again, we freeze the weights from the previous phase and only train the newly added layers.

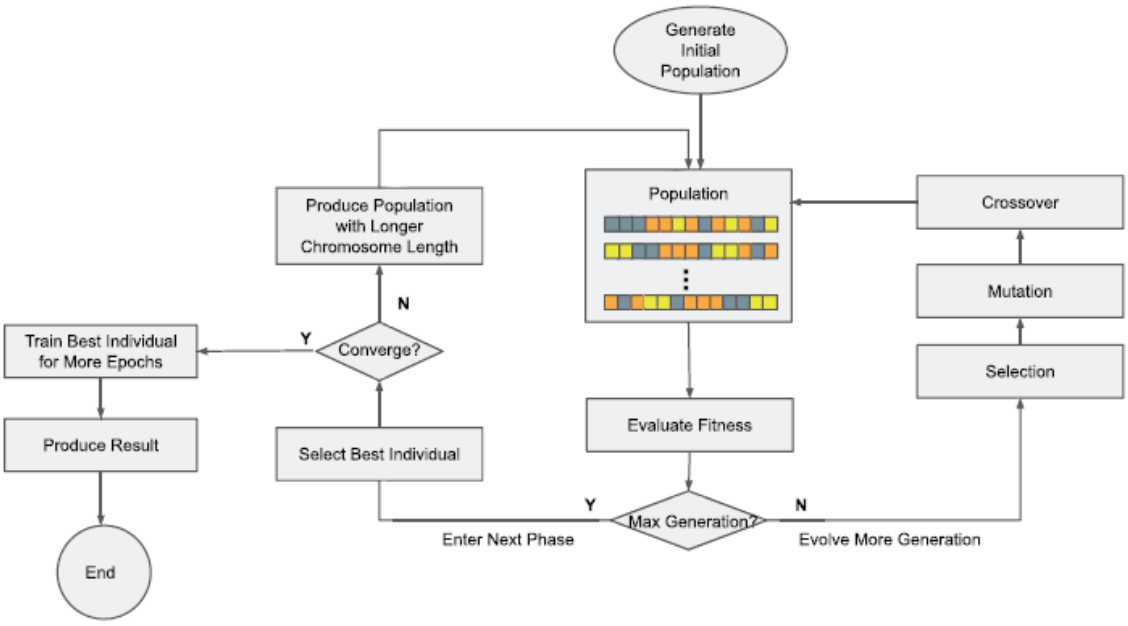


Figure 1: The hyperparameter optimization steps using our variable length genetic algorithm.

Encoding Schema

The chromosomes of our experiments have many fields that contain different values. It defines the hyperparameters of a CNN architecture, such as the number of neurons, activation function type, and soon. Once a chromosome is known, a CNN model can be built according to the chromosome.

In the initial phase (Phase 0) of the algorithm, there are two convolutional layers a and b. Relevant hyperparameters are encoded in the chromosome as shown is Figure 2.

No. of Output Channels for Layer a	Conv kernel Size for Layer a	Activation Type	Include Pool?	Pool Type	Include BN for Layer a?	No. of Output Channels for Layer b	Conv Kernel Size for Layer b	Include BN for Layer b?	Include Skip?

Encoding for Phases after Phase 0.

Chromo From the Previous Phase	No. of Output Channels for Layer a	Conv kernel Size for Layer a	Include Pool?	Pool Type	Include BN for Layer a?	Include Layer b?	No. of Output Channels for Layer b	Conv Kernel Size for Layer b	Include BN for Layer b?	Include Skip?

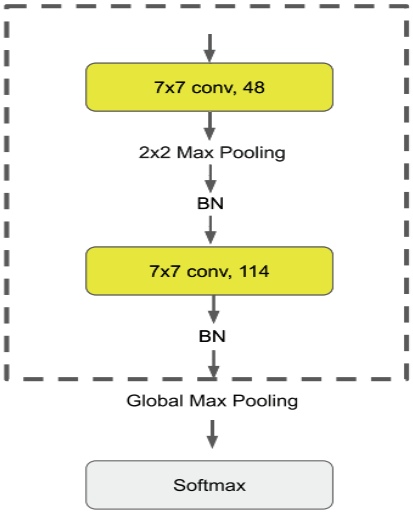
Chromosomes are longer, encoded with properties of new layers in the current phase. Gray background indicates the chromosome from the previous phase. “Include Layer b?” field gives the flexibility of letting the algorithm decide whether to add one or two conv layers.

Fitness of Individuals:

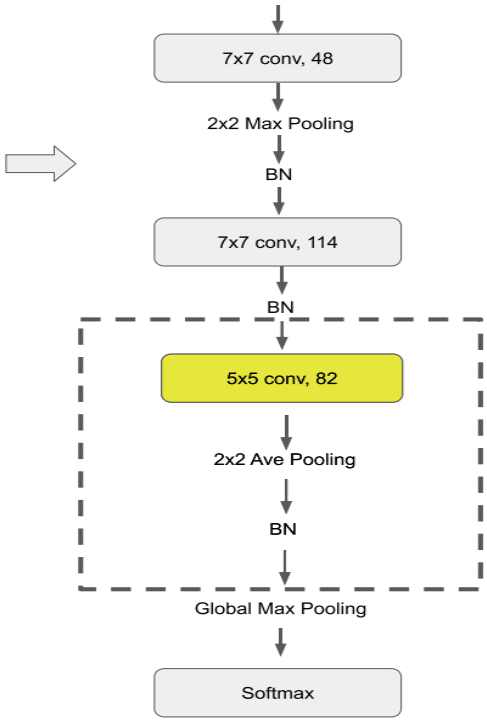
The fitness of an individual is its accuracy on the validation dataset. One way is to train the models to converge, and then compare. While this is more accurate in terms of comparing the models’ performance on the test set, this process takes a very long time and needs a great amount of computing power. To save time, models can be trained for just a few epochs to compare their relative fitness with each other.Models that are fitter in the first few training epochs tend to also perform better later. In the experiments, the number of training epochs for comparing relative fitness is set to be 3.

As we move to next phases, the model depth increases, so training the larger model for a few epochs will not represent its true performance and its accuracy might be lower than a shallow network when trained for the same number of epochs. To address this problem, we introduce weight transfer from shallow models to deeper ones as shown below.

New layers are added to the best model from the previous phase (the left part in figure below). Instead of initializing all the weights randomly in the new models, some of their weights are transferred from the previous best model, and some are done through random initialization



Best Model in Phase 0



A Phase 1 Model

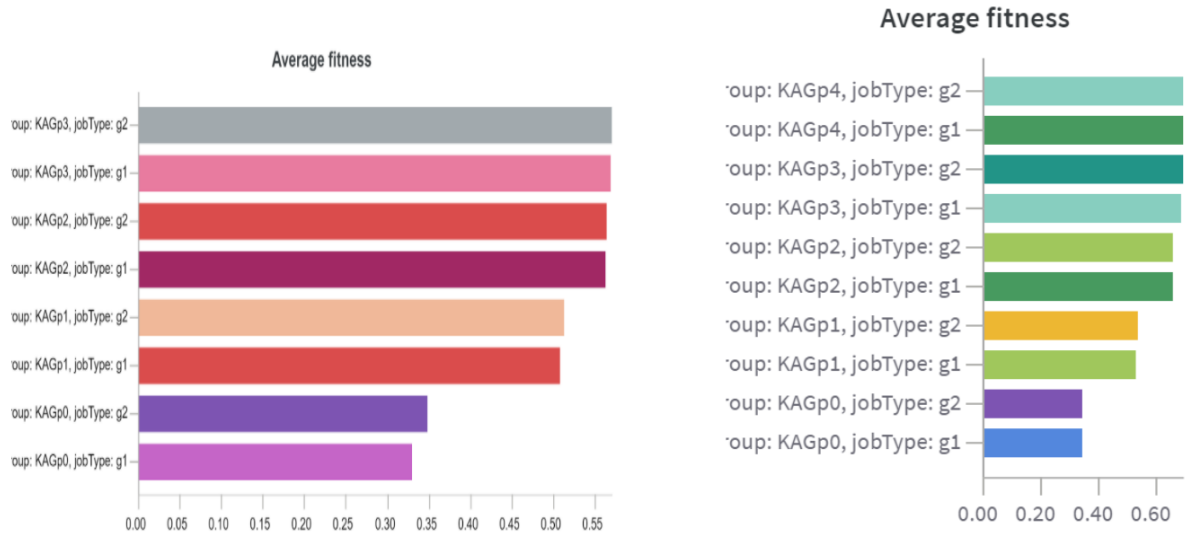
Experimental Evaluation:

The algorithm was run on Google colab with a single GPU (K80). We found the best hyperparameter configuration for the cifar10 classification task. Following are the accuracies we obtained with different hyperparameter optimization algorithms.

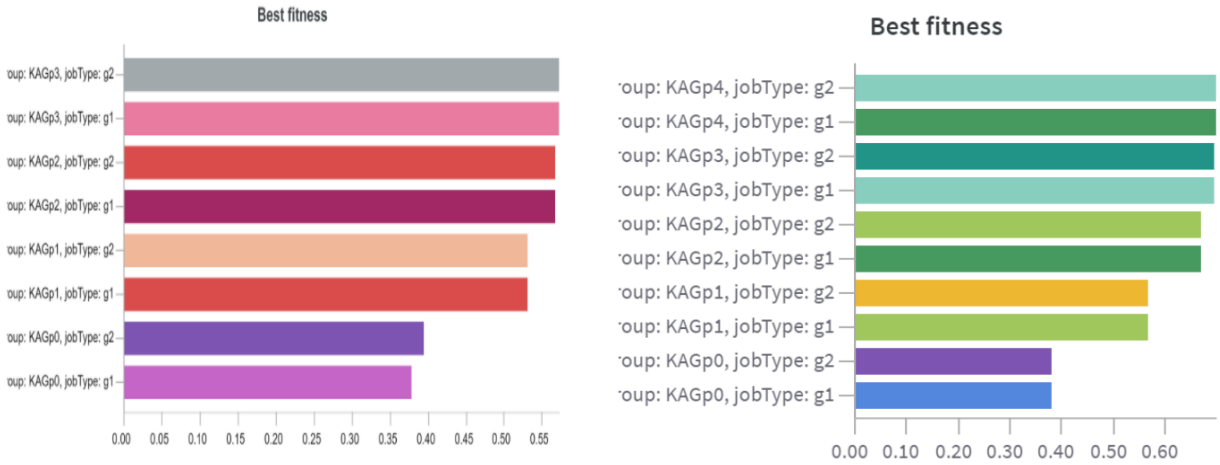
Method	Accuracy
Random Search	58.66%
Classical Genetic Algorithm	80.75%
Variable Length GA	87.25%

Comparison of variable GA with other methods

We also compared algorithm performance with and without cross-over operation.

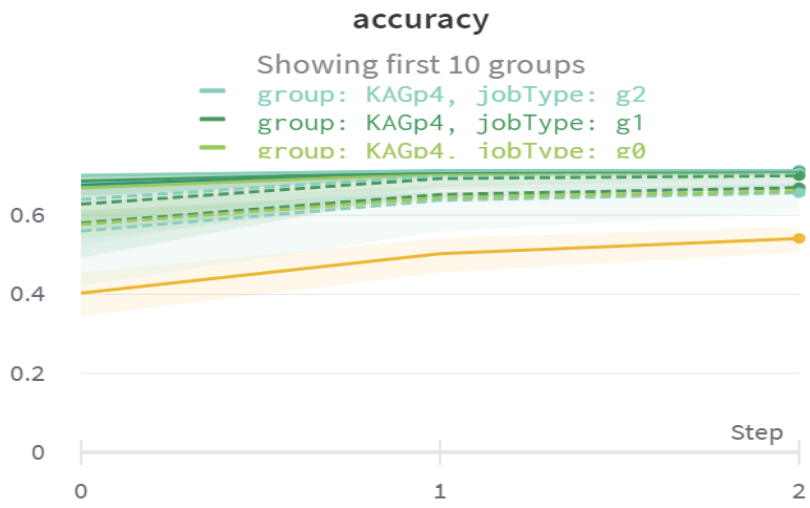


The above plots represent average fitness of the models in different phases of the algorithm, the left plot is the baseline implementation without crossover modification while the right plot is our modified version. Clearly the average fitness values for the models in different phases on the right (eg: phase 3, generation 2: 0.6925) are higher as compared to those on the left (eg: phase 3, generation 2: 0.5699)



The above plots represent the best fitness of the models in different phases of the algorithm, the left plot is the baseline implementation without crossover modification while the right plot is our modified version. Clearly the best fitness values for the models in different phases on the right (phase 3, generation 2: 0.6969) are higher as compared to those on the left (eg: phase 3, generation 2: 0.572)

Accuracy of the models increases as they evolve over generations through the phases as seen below:



Conclusion

We modified the variable length genetic algorithm to efficiently find good hyperparameter settings for deep convolutional neural networks. Experiments performed on CIFAR-10 data show better performance results as compared with the actual variable length genetic algorithm, random search, classical genetic algorithm, and large scale evolution (results from paper). However, this algorithm can find much better models within a time constraint. In the future, to improve the efficiency of the algorithm, early stopping can be incorporated while training models during each phase, such that if a model has poor hyperparameter configurations more computational resources are not spent in training them. Also, other evolutionary algorithms such as ant colony can also be applied to this problem

References

- Classical fixed length genetic algorithm (GA) to optimize a CNN model for the CIFAR-10 dataset (https://dl.acm.org/doi/10.1145/2834892.2834896)
- Mutation only evolutionary algorithm (computationally expensive - the whole process takes 250 parallel computers and over 250 hours to complete) (https://arxiv.org/abs/1703.01041)
- Efficient Hyperparameter Optimization in Deep Learning using a Variable Length Genetic Algorithm (https://arxiv.org/pdf/2006.12703v1.pdf)
- https://wandb.ai/authors/vlga/reports/Survival-of-the-Fittest-CNN-Model--VmlldzoyODA1Nzc