Optimizing a Convolutional Neural Network with Genetic Algorithm

1st Adewuyi Israel Oluwajuwon
Department of Computer Science
Innopolis University
Innopolis, Russia
i.adewuyi@innopolis.university

2nd Khush Patel
Department of Computer Science
Innopolis University
Innopolis, Russia
k.patel@innopolis.university

3rd Chulpan Valiullina
Department of Computer Science
Innopolis University
Innopolis, Russia
c.valiullina@innopolis.university

***Abstract*—This paper discusses the optimization of Convolutional Neural Networks (CNN) using a Genetic Algorithm (GA). We explore the adaptation of GA for automated hyperparameter tuning and architectural configuration of CNNs. The method allows for dynamic adjustments of parameters such as filter sizes, number of filters, and batch sizes, and shows promising results in optimizing network performance on image classification tasks.**

***Index Terms*—Convolutional Neural Networks, Genetic Algorithms, Hyperparameter Optimization, Deep Learning, Image Classification**

I. Introduction

Convolutional Neural Networks have revolutionized image recognition and detection tasks by automatically learning features from raw data. The optimization of CNN parameters is crucial for achieving peak performance in various image recognition tasks. Genetic Algorithms, on the other hand, utilize principles of natural selection and evolution to efficiently search and optimize complex solution spaces.

When it comes to optimizing CNN parameters, traditional methods like grid search or random search can become computationally expensive as model complexity increases. This is where GAs come into play, offering a promising alternative by exploiting the principles of selection, crossover, and mutation to efficiently traverse the parameter space and find optimal configurations. The application of GAs in the context of CNN parameter optimization brings several benefits, including the ability to handle non-linear and non-convex search spaces more effectively, as well as the potential to discover novel parameter configurations that traditional methods might overlook.

This paper presents a method using Genetic Algorithms (GA) to automate the optimization process, potentially reducing time and improving accuracy and in the following sections, we will delve deeper into the mechanics of GAs and their specific application to the optimization of CNN parameters, highlighting their potential to revolutionize the process of tuning CNNs for optimal performance in image recognition and detection tasks.

II. Related Work

The integration of evolutionary algorithms for neural network optimization has been explored in various studies. It is worthy of note that one of the major problems with solving a problem with GA is representing the problem in a format that's solvable by GA [3]. Study [3] demonstrates the potential of GAs to optimize deep learning architectures, study [5] discusses problem representation and fitness evaluation and presents experimental results, showing successful CNN architectures generated by the GA tuner.

CNNs have demonstrated great success in visual image analysis, but determining their architecture needed human intervention, which is more an error laden process. The proposed method utilizes a genetic algorithm to evolve the hyper-parameters of the CNN architecture, achieving successful results without human intervention.

There have also been different domains or applications where the combination of CNN and GA have been used to achieve rather impressive results. Yinan, Pingan and Lu [6] used them to optimize the design of the main body shape of underwater vehicles. Salih and Diffy [1] used CNN and GA for diagnosis of Skin Lesion. Lee et. al [2] used CNN and GA on PET/CT dataset for classification as well.

Reference [3] implemented the GA using direct-encoding representation where hyper-parameters are extracted from a binary format GA chromosome, and this

allows for the generation of successful CNN architectures. Every detail from the number and size of filters in each layer, to the size of each layer, to the activation functions are encoded in binary format. [2] also took a similar approach, but they encoded the relationship between convolution layers.

In this study, we build on the work on some of these authors, while experimenting with novel insights.

III. Github

Here is the link to the github repository containing the code for this paper: https://github.com/Chehmet/Triple_mnist.

IV. Dataset

We used the triple MNIST dataset. The original source of the dataset is hard to ascertain, but the dataset was provided by Professor Adil Khan to the students of the Spring 2022 on the Introduction to Machine Learning Course.

The triple MNIST handwriting writing dataset is similar to the famous MNIST handwriting dataset, but in this case, it's three digits in a 28 x 28 image, instead of a single digit.

V. Methodology

A. Overview

Convolutional Neural Networks have a lot of different possible configurations and depending on what is being optimized, it could have different Genetic Algorithm representations. For the CNN models, manual choice of the configurations is always often based on intuition, random choice or expert opinion. The major architectural decisions in a CNN model are usually the number of layers, the number and the size of the filters at each layer, the size of padding, stride, the choice of activation functions e.t.c. Each layer in the CNN involves multiple layers, each with its specific role in extracting features from the dataset.

B. Genetic Algorithm Design

The Genetic Algorithm is an evolutionary technique inspired by natural selection principles. It is particularly well-suited for solving optimization problems where the search space is large and complex. In our context, the GA optimizes CNN parameters by iteratively selecting, mutating, and recombining a population of parameter sets (genes) toward optimal performance over several generations.

C. CNN Configuration

For simplicity and because of the bottleneck of compute available, we made the choice to fix some set of parameters as constant and attempt to modify other parameters through the Genetic Algorithm. This choice is discussed below.

1) Fixed Parameters: : The activation functions and the number of convolutional layers were held constant. We used the ReLU activation function for its effectiveness in non-linear transformations and gradient propagation, minimizing the vanishing gradient problem. The choice of ReLU was made because it's a fairly common activation function to use for the specific dataset we were dealing with.
The CNN consisted of two convolutional layers followed by pooling layers and a fully connected layer for classification. This fixed design allowed us to isolate the impact of other variables under study. This choice of fixing two layers was informed by [4]

2) Variable Parameters:: The parameters we chose to optimize with the GA included the batch size, the number of filters, and the filter sizes in each convolutional layer. These parameters were selected based on their significant impact on model performance and training dynamics:

- Batch Size: Batch size affects the gradient estimation and training stability. Larger batch sizes provide a more accurate estimate of the gradient but at a higher computational cost and potentially slower convergence on smaller datasets.

- Number of Filters: Determines the depth of the network at each layer, influencing the network's ability to process various features from the input images.

- Filter Sizes: Affect the receptive field of the convolution operations. Smaller filters can capture finer details, while larger filters help in extracting broader features. Based on [4], the size of filters should range between 3 - 7, for small feature detection and 9 - 11 for huge data images. The dataset we are working with is quite small, so we decided to limit the search space of filters to 3 to 9.

D. Implementation Details

1) Gene Representation:: Each gene in the population represents a set of CNN parameters. A gene is structured as an array of five elements, where the first element represents the batch size of the current CNN individual.

The next two elements represents the number of filters and the size of the filters in the first layer of the CNN. The fourth and fifth element represent the number of filters and the size of the filters in the second layer of the CNN. As mentioned earlier, all the layers of the CNN use ReLU activation function.

```
def create_gene(self):
    filter_sizes = [3, 5, 7, 9]  # Define possible filter sizes
    gene = [
        random.randint(8, 150),  # Batch size randomly chosen between 8 and 100
        random.randint(8, 100),  # Number of filters for the first layer
        random.choice(filter_sizes),  # Filter size for the first layer
        random.randint(8, 100),  # Number of filters for the second layer
        random.choice(filter_sizes)  # Filter size for the second layer
    ]
    return gene
```

Listing 1. Gene Creation in Genetic Algorithm

2) Fitness Evaluation:: Each gene's fitness is evaluated based on the CNN's performance metrics, primarily its accuracy on a validation dataset. Higher fitness scores are assigned to configurations that yield better classification performance. This decision was made simply because how well a particular configuration works is based on how well it can detect the necessary features in the train data and also generalize to unseen data.

3) Selection:: We employed a tournament selection process, where a subset of genes is chosen randomly, and the best-performing gene from this subset is selected for reproduction. This method helps maintain diversity within the gene pool while favoring genes with higher fitness.

4) Crossover:: The crossover function combines genes from two parent configurations to produce a new gene. For all of the 5 gene positions in an individual, a new child gene is formed by taking either of the alleles of the parent at that position.

```
def crossover(self):
    parent1, parent2 = random.sample(self.population, 2)
    child = [random.choice(gene) for gene in zip(parent1, parent2)]
    child_gene_tuple = tuple(child)
```

Listing 2. Crossover function

There are also checks for if the new child gene generated is unique, before added to the current population as well as the maximum number of attempts before it stops attempting to generate a new individual.

5) Mutation:: A mutation rate is set at initialization. This represents the proportion of individuals to be mutated. For all individuals in the list of individuals to be mutated, a random index is selected and depending on the selected gene index, either a random batch size within the range [8, 150] or a random filter configuration (number and size) is applied. Mutations introduce variability into the gene pool, helping the GA escape local optima and explore new areas of the solution space.

```
def mutation(self):
    num_to_mutate = int(len(self.population) * self.mutation_rate)
    selected_indices = random.sample(range(len(self.population)), num_to_mutate)

    for index in selected_indices:
        gene = self.population[index].copy()  # Make a copy to mutate
        mutation_index = random.randint(0, 4)  # Choose a random index in the gene to mutate

        if mutation_index in [2, 4]:
            gene[mutation_index] = random.choice([3, 5, 7, 9])
        elif mutation_index in [1, 3]:
            gene[mutation_index] = random.randint(8, 100)
        elif mutation_index == 0:
            gene[mutation_index] = random.randint(8, 150)  # Batch sizes can also be dynamic
```

Listing 3. Mutation function

6) Number of generations and convergence:: The GA runs for a predetermined number of generations or until improvement in fitness scores plateaus, indicating that a near-optimal solution has likely been found. We decided to set a fixed number of generations.

E. Simplification Justification

Fixing certain parameters like the activation function and the number of layers allowed us to reduce the complexity of the hyperparameter space, making the optimization process more manageable and focused. This decision was guided by preliminary experiments we made. More details are discussed below.

VI. Experiments and Evaluation

We implemented our approach using Python and TensorFlow, applying our GA to optimize a CNN for the MNIST dataset. Google Colab CPU was what we used for training.

Initially, we had the variable parameters to be

- batch size, fixed in the range [32, 64, 128]
- number of filters fixed in the same range of [32, 64, 128]

and the fixed parameters include a fixed filter size across both layers at (3 x 3). We got a fairly impressive 88% accuracy. But because of the tight constraints on the possible values, we decided to expand the variable parameters to what it is now.
We ran the GA with 10 randomly generated individuals and initiated evolution process for 5 generations.

For each individual, it was trained on the entire dataset, for a single epoch. After training, the evaluating the current individual model on the validation set, we then retrieve the best individual so far, and then instantiate a model based on the parameters of the best CNN model. This model is then trained for 10 epochs. The results are presented below.

VII. Analysis and Observations

Analysis shows that our GA was able to find configurations that performed fairly well. Notably, the evolution process gradually converged to optimal batch sizes and filter configurations, which were further validated through extended training. The result of the best individual in each generation is presented below:

TABLE I
Best Individuals and Their Fitness Scores by Generation

Generation	Best Individual	Fitness Score
1	(110, 25, 7, 72, 9)	0.93
2	(110, 25, 7, 72, 9)	0.93
3	(126, 65, 5, 69, 7)	0.94
4	(110, 65, 7, 72, 7)	0.94
5	(102, 65, 7, 72, 9)	0.94

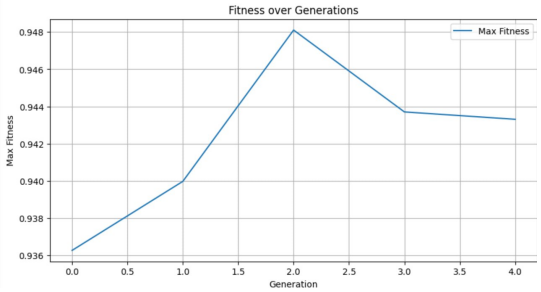

Fig. 1. Best fitness score over generations

The best individual from all generations, which happens to be individual 3, is then trained for 20 epochs, on the dataset, with early stopping. This resulted in an accuracy of approximately 95.7% after 5 epochs.

VIII. Conclusion

The results affirm that Genetic Algorithms can effectively optimize CNN configurations, providing a robust tool for automating neural network design. This method can significantly reduce the dependency on manual tuning and accelerate the deployment of efficient models.

IX. Further Work

The methodology and experiments presented in this study provide a robust foundation for optimizing convolutional neural networks (CNNs) using genetic algorithms (GAs). While the results are promising, there are several avenues for further research that could enhance the efficacy and applicability of this approach. The following sections outline potential areas for future work:

1) **Expansion of Parameter Space** Currently, the GA focuses on optimizing three primary parameters: batch size, number of filters, and filter size. Future work could consider expanding the parameter space to include other significant hyperparameters such as the learning rate, number of layers, the activation function at each layer, max pool size, max pool layer after each layer [3].
   Expanding the parameter space would likely improve the model's performance but would also increase the complexity of the genetic algorithm. Advanced techniques such as multi-objective optimization could be employed to balance the trade-offs between different performance metrics like accuracy and computational efficiency.

2) **Complex Architectures and Transfer Learning:** The current implementation uses a relatively straightforward CNN architecture. Investigating more complex architectures or pre-trained models could yield better performance, especially on more challenging datasets. Utilizing GAs to fine-tune the layers of a pre-trained model on a new dataset could be particularly effective, combining the strengths of transfer learning with the adaptive capability of GAs.

3) **Dynamic Mutation and Crossover Strategies** The mutation and crossover strategies used in the GA are static throughout the optimization process. Implementing adaptive or dynamic strategies, or

even just a different approach to mutation and crossover altogether, that adjust according to the population's state could enhance the GA's ability to escape local optima and converge faster. For example, increasing mutation rates when the population's fitness improvement plateaus or varying the crossover strategy based on the diversity of the population could be investigated.

4) **Initial Number of individuals and number of generations** The number of individuals used are were majorly constrained by the compute resources and with the current results on the little number of iterations, it's reasonable that increasing the number of individuals and the number of generations will give a more diverse initial search space and is more likely to prevent the GA getting stuck at a local minimum.

REFERENCES

[1] Salih, O.; Duffy, K.J., "Optimization Convolutional Neural Network for Automatic Skin Lesion Diagnosis Using a Genetic Algorithm," Appl. Sci., 13, 2023.

[2] Lee, S.; Kim, J.; Kang, H.; Kang, D.-Y.; Park, J., "Genetic Algorithm Based Deep Learning Neural Network Structure and Hyperparameter Optimization," Appl. Sci., 11, 2021.

[3] Bhandare, Ashray and Kaur, Devinder, "Designing Convolutional Neural Network Architecture Using Genetic Algorithms," International Journal of Advanced Network, Monitoring and Controls, 26-25, 2021.

[4] Shashank Ramesh, "A guide to an efficient way to build neural network architectures- Part II: Hyper-parameter selection and tuning for Convolutional Neural Networks using Hyperas on Fashion-MNIST," [Online], 2018, https://towardsdatascience.com/a-guide-to-an-efficient-way-to-build-neural-network-architectures-part-ii-hyper-parameter-42efca01e5d7.

[5] Yanan Sun, Bing Xue, Mengjie Zhang, Gary G. Yen, and Jiancheng Lv, "Automatically Designing CNN Architectures Using Genetic Algorithm for Image Classification", IEEE, 2020, https://arxiv.org/pdf/1808.03818.

[6] Yinan Xu, Pingan Liu, Lu Wang, "Main body shape optimization of non-body-of-revolution underwater vehicles by using CNN and genetic algorithm", Ocean Engineering, 2024, https://www.sciencedirect.com/science/article/abs/pii/S0029801824002750

LeCun, Yann Bengio, Y. Hinton, Geoffrey. (2015). Deep Learning. Nature. 521. 436-44. 10.1038/nature14539.