Technical note

Optimization of Convolutional Neural Network hyperparameters using Genetic Algorithms

Shah N

Submitted: October10, 2023, Revised: version 1, December 5, 2023, version 2, February 3, 2024
Accepted: February 12, 2024

Abstract

The ever-increasing complexity of deep learning models leads to larger sizes and computational costs (greater execution time), making models extremely difficult to implement in resource constrained environments. Moreover, extensive hyperparameter tuning is necessary to create successful models. One approach to this problem is pruning, which removes redundant connections and neurons within the model. However, no approach fundamentally changes the structure of the model (layers, width, depth, filters, etc.). This would inherently bypass the necessity for tediously removing connections within the model and rather remove unnecessary layers and filters (kernels) which may not substantially contribute to performance. The proposed research introduces a approach to reduce inference times of Convolutional Neural Networks (CNNs) by dynamically changing the number of convolutional layers and the kernel parameters within those layers using Genetic Algorithms (GAs). By parameterizing architecture and layer configurations, this approach found the highest efficiency while maintaining a competitive accuracy. Increasing efficiency means decreasing the number of parameters within a model, thus reducing resource consumption during use and reducing inference time. In this paper, a dynamically trained model was tested on three datasets, MNIST, CIFAR-10, and CIFAR 100. Utilizing concepts from GA's such as selection, crossover, and mutation, each model was iteratively trained and tested on each dataset. The resulting model maintained an accuracy at a given threshold while demonstrating an increase in efficiency by changing the number of convolutional layers and the corresponding filter parameters.

Keywords

Convolutional Neural Networks, Genetic Algorithms, Kernels, Hyperparameters, Inference Time, CNN, Computational cost, Deep learning, Image classification, Network architecture

Nirmit Shah, Portola High School, 1001 Cadence, Irvine, CA 92618, USA.
nirmitmshah@gmail.com

Introduction

Convolutional Neural Networks (CNNs) have demonstrated remarkable success in various computer vision tasks, achieving state-of-the-art performance in image classification (1). However, as models become more sophisticated to handle complex visual patterns, they also become more resource-intensive, demanding larger memory footprints and longer inference times (2). This trade-off between model complexity and efficiency poses a significant challenge, especially in scenarios with limited computational resources and strict latency requirements. Moreover, tedious hyperparameter manipulation with sophisticated knowledge is required to create accurate models (3).

To address this challenge, researchers have explored different approaches to optimize CNN architectures and hyperparameters. One widely studied technique is model pruning, which aims to reduce the number of parameters and operations in a pre-trained CNN, resulting in a more efficient network (4). However, model pruning typically involves modifying individual connections or neurons within the network, while keeping the overall structure unchanged (5). Additionally, Hyperparameter optimization software has been developed. Utilizing Bayesian (Probabilistic) models, researchers have developed an algorithm called Tree-Structured Parzen Estimator (TPE) which handles conventional hyperparameter optimization without having to search the entire search space (possibilities) exhaustively (6). However, convolutional layers and their corresponding filter sizes could not be optimized using these software's due to the complexity of training individual model architectures.

In contrast, this research introduces a approach to enhance the efficiency of CNNs by dynamically adapting the network architecture and layer configurations using Genetic Algorithms (GAs). Genetic Algorithms draw inspiration from the principles of natural selection and evolution, enabling them to handle discrete values, such as filter sizes and the number of convolutional layers in a CNN (7). Convolutional Neural Networks differ from conventional NN's because of the presence of convolutional layers, which decrease the complexity of images and extract useful information by passing filters (kernels) over the image iteratively. Each layer has a distinct number and corresponding size of filters. Because filters only come in odd sizes, gradient descent methods are rendered insufficient as they deal with continuous values. The number of convolutional layers, the number of filters within those layers, and the sizes of those filters are all hyperparameters within a CNN, input manually when creating a model. However, Genetic Algorithms can be used to discreetly parameterize these values and iteratively train them, creating a more accurate and efficient model (8).

The utilization of Genetic Algorithms for optimizing CNNs affords several advantages. Firstly, it allows for the exploration of a diverse range of architectures that would be computationally prohibitive with traditional exhaustive search methods (9). Secondly, the selection, mutation, and crossover operations in GAs mimic the concepts of ecological

evolution, providing a principled and effective framework for generating new generations of CNN architectures (10). By iteratively training a model with Genetic Algorithms, the model should converge to an efficient and accurate model, as demonstrated with the results on MNIST, CIFAR-10, and CIFAR-100 datasets.

Materials and methods

The algorithm deployed by randomly initializing N (hyperparameter) models for the first generation. Each model's hyperparameters comprised the number of convolutional layers, filter sizes, and numbers within those layers. Model architectures stayed mostly static but had to be changed when there occurred large differences in the number of convolutional layers due to complexity requirements. Every generation consisted of training and evaluating the previous set of models to determine how to generate a new set. Model architectures were developed and trained with the TensorFlow Python library, and performance was evaluated by a fitness equation. The program terminated when the convergence criteria were met. This model was tested on three different datasets accessed and downloaded directly from the TensorFlow Python Library.

Genetic Algorithm

The genetic algorithm converged to an optimal model by generating a set of models that were evaluated on a global criteria (fitness) to determine the next set of models. Generating the next set of models utilized concepts from GAs which primarily consisted of three operations: Selection, Mutation, and Crossover. Currently implemented GAs could not be used because of the incompatibility with parameterizing CNN model architecture. Hence, in order to develop a training algorithm, fine-tuned Selection, Mutation, and Crossover functions had to be developed. These functions occured one after another in that order.

Selection

The top N/3 models with the highest fitness were selected to form the basis of the next generation. By prioritizing the best-performing models, the algorithm retained promising solutions, encouraging convergence towards optimal architectures.

Mutation

To promote exploration and prevent premature convergence, mutation was also applied to selected models. X/2 parameters were randomly chosen within each model and modified, while ensuring that the new values remained within appropriate bounds. These mutations were performed randomly but with greater weight on attributes that were more similar to the original model. This is modeled in equation 1 where θ represents the parameter and σ represents the standard deviation. The weights and random values differed for each parameter.

$$mutate(\theta, \sigma, weight) = round(\theta + weight \times random_{value} \times \sigma) \quad (1)$$

This process introduced diversity into the model population, allowing the algorithm to explore a broader solution space and potentially find superior architectures.

Crossover

Finally, Crossover merged two parent models from the selected set, presenting an alternative method for exploration. While Crossover was not entirely necessary, it offered a quicker path to convergence and reduced training time while preserving additional exploration. Crossover was done by choosing a crossover point k, and using one parent's parameters before this point and the others after. Many other techniques were tried but this turned out to be the most optimal in retaining important intralayer information.

Due to the rudimentary nature of this Genetic Algorithm, hyperparameter optimization of the algorithm itself was limited. However, to ensure the algorithm's viability, it was tested off task and performed well. To name a few: crossover bounds, selection bias, and mutation draft were the most impactful parameters in determining algorithm viability. GA's are conventionally known to be stochastic and thus methods have been employed to confirm the reliability of this stochasticity (11). In the developed model, the stochasticity was confirmed and reevaluated with each successive generation due to the fitness-based selection of models. Since only the top three models were selected and maintained throughout each generation, harmful mutations and crossovers were discarded.

Fitness Equation

Two approaches were taken to determine fitness. Both approaches had to consider accuracy with respect to efficiency to accurately measure the performance of this model. Inference time and the number of parameters were both initially considered as metrics of efficiency but the relationship with the number of parameters was not linear and therefore was ruled out. Time followed a more consistent rate as the number of parameters increased and was chosen to measure efficiency.

Initially, a reward-based approach was utilized, where the reward was determined by a combination of non-linear and linear functions. For accuracy values greater than or equal to 0.5, the reward was obtained using a non-linear formula that scales the accuracy exponentially by a factor of 5. Conversely, for accuracy values less than 0.5, a linear formula was employed to compute the reward, ensuring a continuous and fair evaluation across all accuracy ranges. The linear weight had a minimum fitness of 100 while the exponential curve allowed for a maximum fitness of 1000. The fitness metric was then derived by subtracting the time taken to complete the task from the obtained reward, resulting in a comprehensive assessment that balanced accuracy and efficiency. It was represented by the two functions below where A represents accuracy and T represents the inference time for x number of images.

Reward

$$R(A) = \{1000 * e^{5A} \; if \; A \geq 0.5 \; 100 + 800A \; if \; A < 0.5 \qquad (2)$$

Fitness

$$F(A,T)=R(A)-T \quad (3)$$

This initial fitness metric did not allow for the model to improve above an accuracy of ~ 65% because it began to prioritize efficiency over accuracy. The solution for this was to set a predetermined threshold, and calculate fitness on the difference in accuracies between the current and threshold. Initially, two key constants were established: 'max_reward' set at 1.0, denoting the highest attainable reward when the accuracy met or exceeded the threshold, and 'steepness' set to 10.0, regulating the rate at which the reward diminished as the accuracy deviated from the threshold. For accuracy values surpassing the threshold, the fitness was assigned the maximum reward of 1.0. However, when the accuracy fell below the threshold, the fitness was determined through a non-linear formula that penalized the model's performance. By calculating the absolute difference between the accuracy and the threshold, the penalty was inversely related to the deviation. Then, inference time was subtracted from the reward and provided a reliable fitness metric which allowed the model to converge optimally. This equation represents a revised fitness where A represents accuracy, T represents inference time on 32 images, and the threshold represents the desired accuracy.

$$F(A,T,Threshold)=1.0-\left(\frac{1.0}{1+\exp(10\ *\ (A-Threshold))}\right)-T \quad (4)$$

Convergence Criterion

Convergence was easily determined by calculating whether subsequent generations were improving or plateauing. Fitness could not be used to calculate convergence due to its extreme sensitivity to minor changes in accuracy. A new metric had to be calculated that looked at accuracy with respect to time, but in a linear manner. This metric could be more accurately used to determine convergence. Two constants were established, the variation (set at 1.25 accuracy points) needed to distinguish between a converged model and an improving model, and how many generations (set at 3) needed to be evaluated at once. During training, the five most current generations were evaluated and checked whether they fell between the variations. If so, training was completed and the final model was completed. Figure 1 shows a flowchart of the entire process.

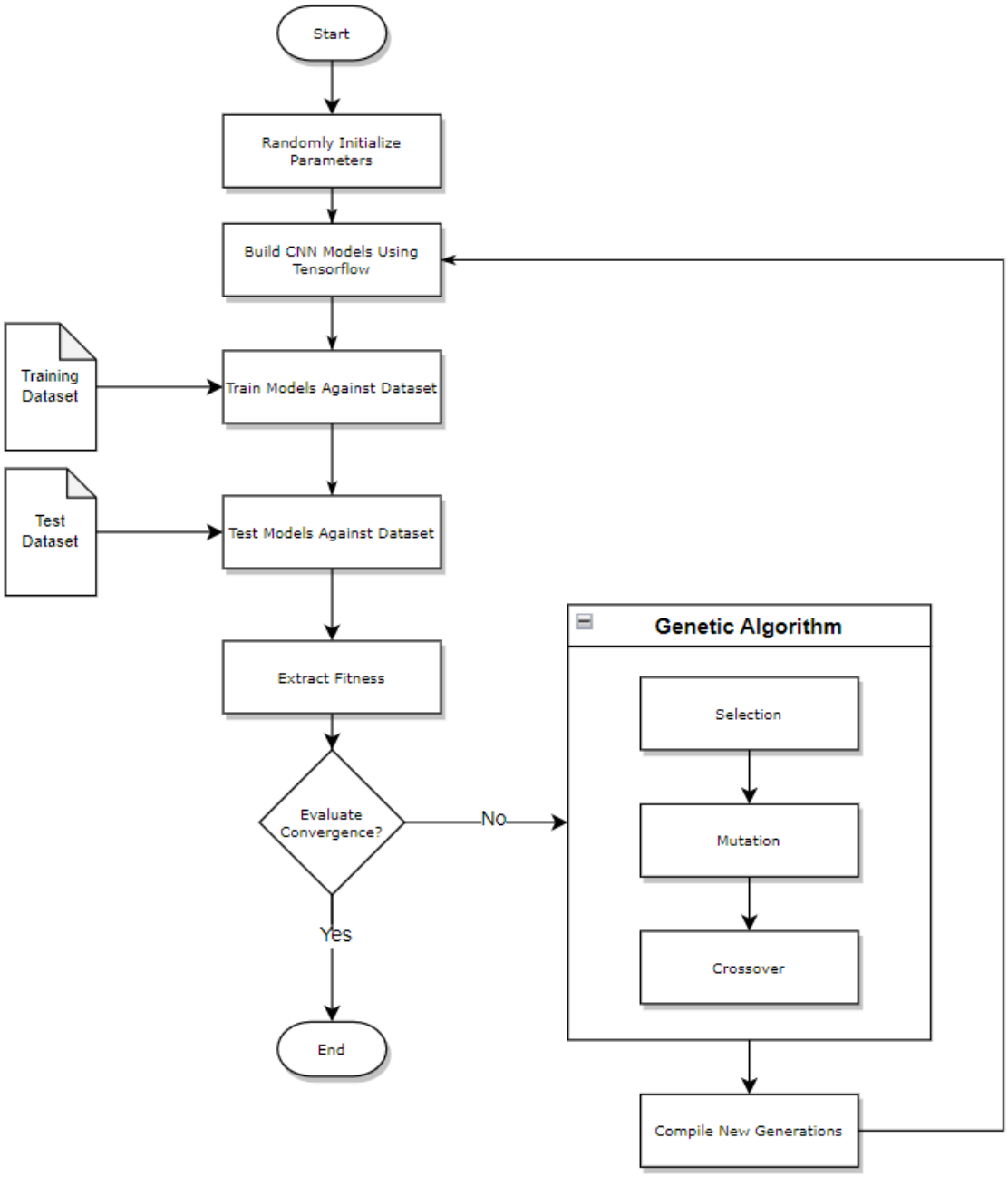

Figure 1. Flow Chart demonstrating the incorporation of the Genetic Algorithm into the CNN

Results

The model was tested on three image classification datasets, MNIST, CIFAR-10, and CIFAR 100, respectively ordered in increasing difficulty. The fitness metric displayed in Figure 2 represents a combination of accuracy and efficiency, where efficiency was measured by the inference time. As demonstrated, the

model improved as the number of generations progressed, eventually converging and terminating. Figure 2 only displays an average of the top three models per generation to account for variations in lower fitness models that would skew the data. Each dataset had its own number of generations because more complex models took longer to converge (12). It is important to note that model accuracy was significantly influenced by the number of epochs used during training. However, results showed that any number of epochs exceeding eight yielded similar outcomes.

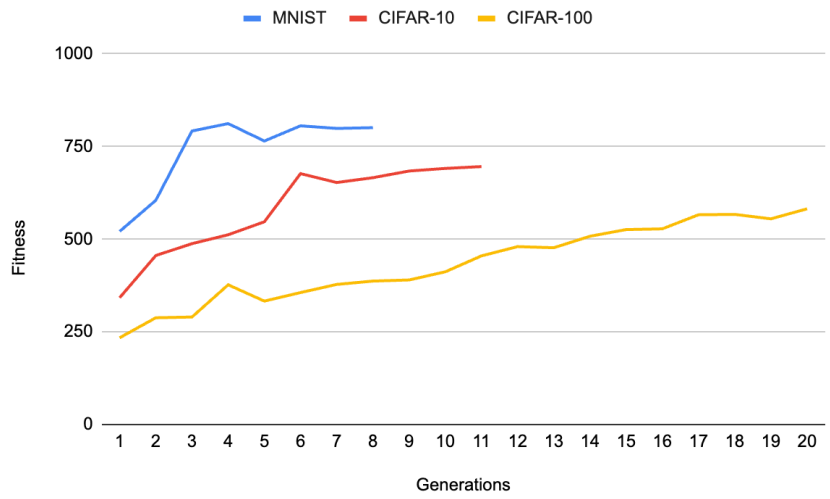

Figure 2. Fitness of MNIST, CIFAR-10 and CIFAR-100 after training over certain number of generations. Line graph showing fitness metric determined by a combination of accuracy and time, using TensorFlow Python library.

Figures 3 and 4 are an example of how the model performed on the CIFAR 10 dataset. Since reaching the threshold accuracy was assigned a much greater weight than minimizing training time, the model quickly reached the threshold. After, it slowly began to increase fitness by reducing training time while very minutely affecting accuracy, demonstrating the effectiveness of Genetic Algorithms for removing unnecessary filters while retaining important ones.

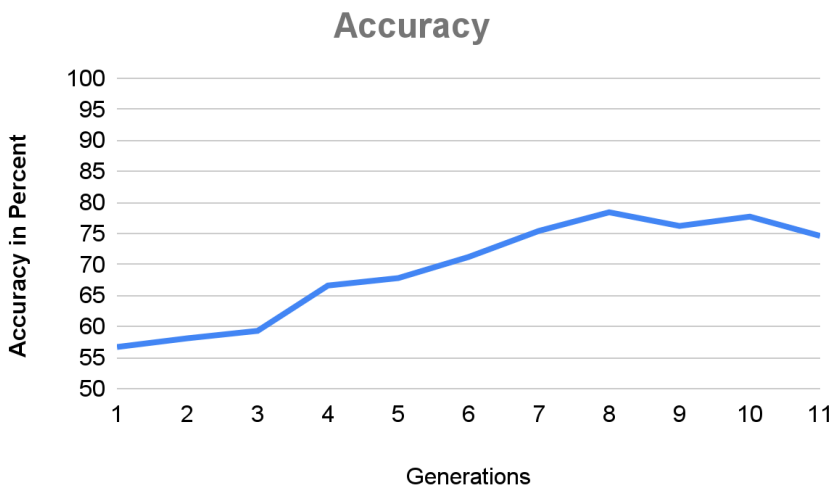

Figure 3. Progression of Accuracy of model on CIFAR-10 dataset. Line graph showing progression of accuracy. Model was trained and tested using genetic algorithm with TensorFlow Python library. Percent of test data set accurately identified.

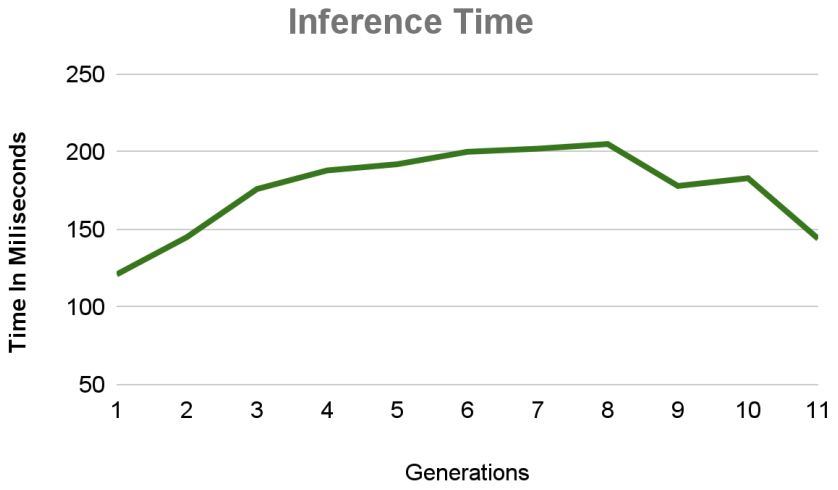

Figure 4. Progression of Inference time of model on CIFAR-10 Dataset. Line graph showing progression of inference time. Model was trained and tested using genetic algorithm with TensorFlow Python library. Inference time for model to run iteration over dataset. Batch size 32.

As demonstrated by all three datasets (Figure 5), the number of convolutional layers increased with model complexity because the images became more intricate and harder to classify (13). This increase cannot be attributed to an increase in size of the images because MNIST has a 28 by 28 image size while both CIFAR 10 and CIFAR 100 have 32 by 32 image sizes. Additionally, kernel sizes continually displayed an upward trend throughout all of the models and aligned with previous predictions (Figure 6). CNN filters are utilized to simplify the input image, and by having small sizes in the beginning and gradually increasing them, the model can capture intricate features in the data and then look at the image more holistically (14). In contrast, the number of filters in every layer displayed a slightly downward trend before eventually plateauing. The exact reason for this is unknown but it may be because when the filters are capturing more intricate features, the layer requires more filters to simplify it but after the filters become larger, less passes over the image are required.

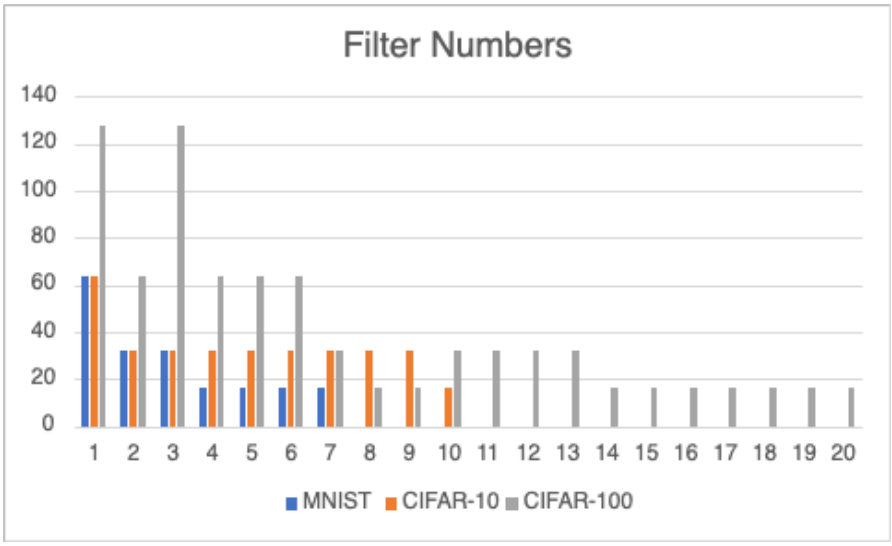

Figure 5. Converged model Filter Numbers for each dataset. Bar Graph representing number of filters for every layer. Model was trained and tested using TensorFlow Python Library. Number of filters are constrained to (16, 32, 64, 124).

As the threshold decreased, an increase in efficiency could be observed. Intuitively, if the model was aiming for a lower accuracy, it would be able to prioritize time and sacrifice accuracy more abundantly. However, for complex datasets such as CIFAR-10 and 100, accuracies above approximately 80 percent could not be achieved due to a rudimentary architecture consisting solely of convolutional layers, max pooling layers, and dense layers (15).

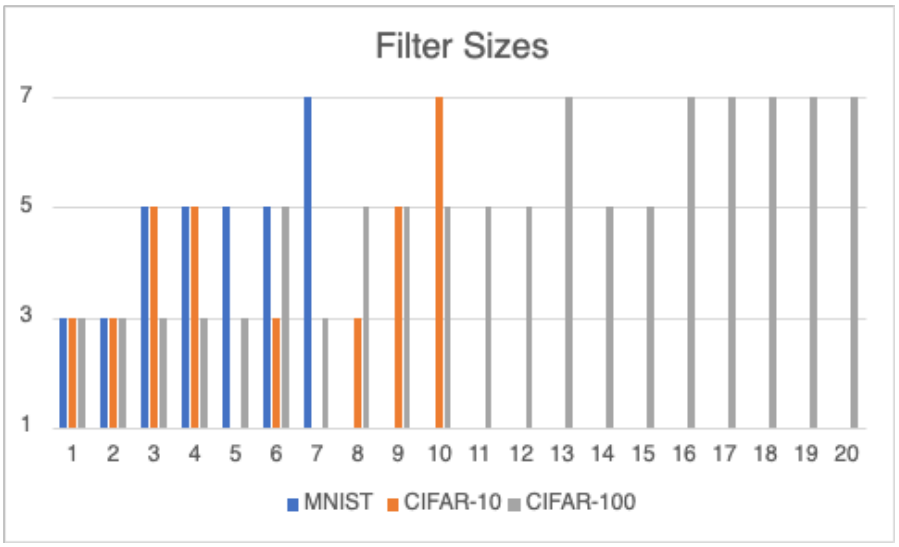

Figure 6. Converged models filter sizes for each dataset. Bar graph showing terminated models filter sizes per layer. Trained and tested using TensorFlow Python library. Filter Sizes constrained to (3, 5, 7) due to feasibility.

Table 1. First generation of the Genetic Algorithm on the MNIST dataset

Operation	Parent 1	Parent 2	Parent 3	Parent 4	Parent 5
Selection	Conv2D(3, 16) Conv2D(5, 32) Conv2D(3, 16)	Conv2D(5, 16) Conv2D(7, 128)	Conv2D(7, 16) Conv2D(5, 128) Conv2D(3, 64)	Conv2D(5, 16) Conv2D(3, 128)	Conv2D(7, 128) Conv2D(5, 16) Conv2D(5, 32)
Mutation	Conv2D(7, 32) Conv2D(3, 64) Conv2D(5, 128)	Conv2D(3, 64) Conv2D(7, 64)	Conv2D(7, 16) Conv2D(5, 128) Conv2D(3, 64)	Conv2D(3, 16) Conv2D(3, 128	Conv2D(7, 128) Conv2D(5, 16) Conv2D(5, 32)
Crossover	Conv2D(7, 16) Conv2D(5, 128) Conv2D(3, 64)	Conv2D(7, 32) Conv2D(3, 64) Conv2D(5, 128)	Conv2D(3, 16) Conv2D(3, 128	Conv2D(3, 16) Conv2D(3, 128	Conv2D(7, 16) Conv2D(5, 128) Conv2D(3, 64)

Table 1 demonstrates how the first generation of the Genetic Algorithm worked on the MNIST dataset. Each generation composed of a Selection, Mutation, and Crossover function. This representation is extremely limited due to the inaccurate representation of the immensity of the models and their layers. Solely for demonstrating how the algorithm works. Each parent consists of a set number of

convolutional layers which are each deducted by Conv2D (Filter Size, Filter #).

Discussion

The results demonstrated the effectiveness of this approach in achieving resource-efficient models while maintaining competitive accuracy. The results found a significant correlation between model complexity and the number of convolutional layers required for optimal performance. As the complexity of the dataset increased, the model tended to benefit from a deeper architecture with more convolutional layers. This finding aligned with the intuition that deeper architectures are better equipped to capture intricate features and patterns present in complex datasets, such as CIFAR-100. The dynamic nature of the proposed approach allowed the model to autonomously determine the most suitable depth for different datasets, leading to a more efficient allocation of computational resources. Additionally, it was observed that as the depth of the model increased, the filter size tended to increase as well. Larger filter sizes in deeper layers enabled the CNN to capture broader spatial information and higher-level features. This adaptation was crucial for ensuring that the model retained the ability to extract relevant information from the input data as it progressed through the layers. The GA-based optimization process effectively identified this trend, allowing for a decrease in parameters while maintaining accuracy. Another noteworthy finding is that as the model complexity increased, the number of filters within each layer tended to decrease. This trend could be attributed to the need for more specialized and discriminative features in complex datasets. By reducing the number of filters in deeper layers, the model could focus computational resources on learning more abstract and high-level representations. The proposed approach captured this dynamic adaptation, showcasing its ability to optimize the filter numbers for various levels of dataset complexity. These trends are useful extractions for development of future CNN's. However, this model was still necessary because the rate at which these trends occur is unknown and differs from dataset to dataset. The results of this model could not be compared to a baseline because of a basic architectural structure and an inability to exceed accuracies of 80 percent on harder datasets. However, this model does have its advantages due to the streamlined process for simpler datasets. For more complex architectures, convolutional layers will always be present and so will the filter sizes and numbers, allowing for fine-tuned versions of this research to be implemented. While the extensive increase in training time may seem daunting, the decrease in inference time significantly offsets training time. However, it depends on the extent that the model is used. If the model is used extensively the increase in training time will prove to be rewarding and vice versa. This paper did not measure if the CNN performed better in terms of execution time or efficiency by the use of the GA. Hence, there can be no assurance that using GA to optimize CNN hyperparameters actually results in an increase in CNN efficiency and/or a decrease in execution time.

Conclusion

In conclusion, this study introduced an approach to optimize Convolutional Neural

Network (CNN) hyperparameters using Genetic Algorithms (GAs). Results found a significant correlation between model complexity and the optimal number of convolutional layers, filter sizes, and filter numbers. The proposed method dynamically adapted the network architecture, demonstrating a delicate balance between accuracy and efficiency. The adaptability of the algorithm in autonomously determining the most suitable parameters for different datasets highlighted its potential in resource-constrained environments. Successful application on MNIST, CIFAR-10, and CIFAR-100 datasets demonstrated the approach's efficacy in achieving resource-efficient models while maintaining competitive accuracy. Overall, this research provided another perspective to CNN optimization, offering insights into evolving optimal architectures for diverse datasets.

Acknowledgement

I would like to express my gratitude to my mentor, Charanraj Thimmisetty, for his valuable support throughout the course of this research. His mentorship has been instrumental in shaping the direction and success of this research, and I am truly thankful for the opportunity to work under his guidance.

References

1. Babaqi, T., Jaradat, M., Yildirim, A.E., Al-Nimer, S.H., Won, D. Eye Disease Classification Using Deep Learning Techniques. arXiv: Computer Vision and Pattern Recognition, 2307.10501v1, 2023. https://doi.org/10.48550/arXiv.2307.10501

2. Krizhevsky, A., Sutskever, I., Hinton, G.E. Imagenet classification with deep convolutional neural networks. In Advances in Neural Information Processing Systems (NIPS), 2012. https://doi.org/10.1145/3065386

3. Simonyan, K., Zisserman, A. Very deep convolutional networks for large-scale image recognition. arXiv: Computer Vision and Pattern Recognition, 1409.1556, 2014. https://doi.org/10.48550/arXiv.1409.1556

4. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D, et al. Going deeper with convolutions. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015. https://doi.org/10.48550/arXiv.1409.4842

5. Denil, M., Shakibi, B., Dinh, L., de Freitas, N., Ranzato M. Predicting parameters in deep learning. In Advances in Neural Information Processing Systems, 2148–2156, 2013. https://doi.org/10.48550/arXiv.1306.0543

6. He, K., Zhang, X., Ren, S., Sun, J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In International Conference on Computer Vision (ICCV), 2015. https://doi.org/10.48550/arXiv.1502.01852

7. Pinto, N., Doukhan, D., DiCarlo, J.J., Cox, D.D. A high-throughput screening approach to discovering good forms of biologically inspired visual representation. PLoS Comput Biol, 5 (11): e1000579, 2009. http://hdl.handle.net/1721.1/52429

8. Coates, A., Lee, H., Ng, A. An analysis of single-layer networks in unsupervised feature learning. NIPS Deep Learning and Unsupervised Feature Learning Workshop, 2010. https://www.researchgate.net/publication/220321031_An_Analysis_of_Single-Layer_Networks_in_Unsupervised_Feature_Learning

9. Coates, A., Ng, A.Y. The importance of encoding versus training with sparse coding and vector quantization. In Proceedings of the Twenty-eighth International Conference on Machine Learning (ICML11), 2010. https://www.researchgate.net/publication/221344737_The_Importance_of_Encoding_Versus_Training_with_Sparse_Coding_and_Vector_Quantization

10. Huang, Q., Zhou, K., You, S., Neumann, U. Learning to Prune Filters in Convolutional Neural Networks. arXiv: Computer Vision and Pattern Recognition, 1801.07365v1, 2018. https://www.cosenza.eu/papers/HartensteinPARS20.pdf

11. Hinz, T., Navarro-Guerrero, N., Magg, S., Wermter, S. Speeding up the Hyperparameter Optimization of Deep Convolutional Neural Networks. International Journal of Computational Intelligence and Applications, 17(02), arXiv:1807.07362, 2018. https://doi.org/10.1142/S1469026818500086

12. Wen, W., Xu, C., Wu, C., Wang, Y., Chen, Y., Li, H. Coordinating Filters for Faster Deep Neural Networks. arXiv: Computer Vision and Pattern Recognition, 1703.09746v3, 2017. https://doi.org/10.48550/arXiv.1703.09746

13. Bergstra, J., Bardenet, R., Bengio, Y., Kégl, B. Algorithms for Hyper-Parameter Optimization. In Proceedings of the Neural Information Processing Systems (NeurIPS), 2011. https://www.researchgate.net/publication/216816964_Algorithms_for_Hyper-Parameter_Optimization

14. Alam, T., Qamar, S., Dixit, A., Benaida, M. Genetic Algorithm: Reviews, Implementations, and Applications. International Journal of Engineering Pedagogy (iJEP), 2020.

https://www.researchgate.net/publication/343424351_Genetic_Algorithm_Reviews_Implementations_and_Applications

15. Thakallapelli, A., Ghosh, S., Kamalasadan, S. Real-time frequency based reduced order modeling of large power grid. Power and Energy Society General Meeting, 2016. https://doi.org/10.1109/PESGM.2016.7741877

Appendix

Algorithm

Initialize filter size buffer with n size
Initialize filter number buffer with n size
Initialize n random filter sizes and numbers
Initialize Fitness/Model buffer
While not converged do:
    For 1, 2 … n in models:
        Create CNN using TensorFlow with n sub i parameters (filter size and numbers)
        Accuracy, Time = TensorFlow Train CNN
        Evaluate model using Fitness metric
        Store Fitness/Model pair in buffer
    END FOR
    Selection - Select the top n/3 models
    Store selected models into corresponding filter size and number buffers
    Mutation - Randomly mutate selected models by changing parameters/2 values
    Store mutated models into different filter size and number buffers
    Crossover - Crossover values from selected models to create new offspring
    Store crossover models into different filter size and number buffers
    Evaluate convergence