Model Reduction Using a Hybrid Approach of Genetic Algorithm and Rule-based Method

Wuqian Tang[1], Chuan-Shun Huang[1], Yung-Chih Chen[2], Yi-Ting Li[1], Shih-Chieh Chang[1], and Chun-Yao Wang[1]

[1]National Tsing Hua University, Taiwan, ROC
[2]National Taiwan University of Science and Technology, Taiwan, R.O.C.

***Abstract*—Model reduction is a technique that reduces the computational resources required to run a model (neural network) by pruning parameters or structures in the model. Most of the model reduction algorithms achieve the goals of model reduction and accuracy preserving through multiple iterations of pruning-retraining process. However, this retraining process is quite time-consuming, making the model size reduction algorithm particularly inefficient, especially when the parameters of model exceed tens of millions. In this paper, we propose a hybrid approach combining genetic algorithm (GA) and rule-based method. With the integration of GA and a rule-based method, the time cost of searching for a well-performing model can be significantly reduced. This strategy greatly reduces GA's search space and time cost. With a very limited number of retraining epochs (<10), the accuracy and pruning ratio (sparsity) of the reduced model can catch up the results of state-of-the-art. We conduct experiments on a gesture recognition model with over 30 million parameters. The experimental results show that for this model, our approach prunes 74.6% of the parameters with 3.8% accuracy drop without retraining. With only three epochs of retraining, our approach prune 93.1% of the parameters without any accuracy drop.**

I. Introduction

With the success of widespread applications of Neural Networks (NNs) in various fields, such as image recognition, speech processing, and gesture recognition, the capabilities and complexity of NNs (models) have increased significantly. Since the proposal of the backpropagation [22], the advent of Convolutional Neural Networks (CNNs) [13], and the introduction of the learning algorithm for Deep Belief Networks (DBNs) [9], NNs can tackle increasingly intricate problems.

However, as the complexity of the problems being addressed escalates, the size of the models as well as the computational requirements and energy consumption for running these models also increase considerably. Therefore, model reduction has become an increasingly crucial task. Typical objectives for model reduction are preserving the accuracy, minimizing parameter count, decreasing energy consumption in computing, and reducing model latency, enabling the reduced model to be implemented in embedded systems or resource-limited edge devices [6]. A multitude of pruning techniques have been proposed to address this challenge [5], [7], [15], [20], [27].

A common approach to model reduction typically involves one or more cycles of pruning-retraining after obtaining the initial model. Recently, many model reduction techniques have been proposed [2], [7], [16]. One-shot pruning [2], [7] is a technique that pruning and retraining are performed only once after obtaining the initial model. Subsequently, researchers proposed methods that perform multiple iterations of pruning and retraining to achieve better results [16].

Pruning is achieved by removing unimportant parameters, e.g., biases and weights, or structures, e.g., neurons and filters, in the models. In the pruning process, the main focus is on the weights rather than the biases. This is because the quantity of weights usually far exceeds that of biases, and the impact of weights on the model's complexity and performance is significantly greater than that of biases. Various pruning methods can be employed to eliminate weights in the model during the pruning phase [4], [12], [14], [16]. For instance, random pruning of weights [12], layer-wise pruning [16], and global magnitude pruning [4], [14]. Random pruning of weights is a method to eliminate a random selection of weights from the model. Layer-wise pruning is a strategy that prunes a fixed percentage of weights for each layer. In contrast, global magnitude pruning adopts a holistic view of the model and prunes a fixed percentage of weights, starting with those closest to zero in magnitude across the entire model.

On the other hand, some studies utilized genetic algorithms (GA) to explore models for achieving weights pruning [3], [19]. They used binary strings to represent models, where 0 signifies that the weight is pruned and 1 indicates the weight is retained.

After the pruning phase, retraining is a typical succeeding phase for accuracy recovery. For example, a popular method is to fine-tune the parameters by retraining for multiple epochs using the fixed learning rate [7]. Other retraining strategies include the rewind strategy [21] and the cyclic learning rate restarting strategy [12]. The former initially uses a higher fixed learning rate, then switching to a lower fixed learning rate, while the latter periodically varies the learning rates.

However, it is worth noting that retraining is a very time-consuming process. Previous methods often require dozens or even hundreds of epochs in the retraining phase for recovering the accuracy of models. This phase can span days or even weeks. Thus, in this work, we propose a hybrid approach to model reduction, which integrates a genetic algorithm and a rule-based method. Our approach applies a GA iteratively during the pruning phase to generate and select superior offspring that display high quality in both accuracy and sparsity. By identifying common features among these offspring models with high quality, we are able to extract pruning rules that will be used in the pruning strategy. Following the pruning, we conduct a brief fine-tuning phase consisting of a handful of retraining epochs. The experimental results demonstrate that a model with a few of pruning-retraining cycles, typically around two or three, are sufficient to catch up the performance of leading model in terms of accuracy and pruned percentage of parameters. Not only does this approach tackle the existing challenges head-on, but it also hints at the potential of its extensibility. Our method provides a promising direction for structured pruning of models in the future, broadening the horizon for further research and application.

The main contributions of this work are threefold:

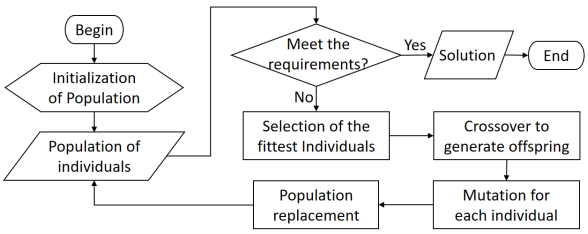

Fig. 1: Genetic Algorithm Flowchart.

- We propose a hybrid approach of combining GA and rule-based method for efficient model reduction. This is the first attempt to integrate two distinct methods for model reduction.
- As compared to the state-of-the-art, the proposed approach significantly saves the search cost to achieve a highly reduced model with a similar accuracy.
- Our approach can prune 74.6% parameters with a 3.8% accuracy drop without retraining on a 30 million-parameter gesture recognition model. With just three epochs of retraining, our approach prunes 93.1% parameters while maintaining accuracy.

The remainder of this paper is organized as follows. Section II introduces the background. Section III presents the proposed approach. Section IV shows the experimental results. Finally, we conclude this paper and present the future work in Section V.

II. Preliminaries

In this section, we provide essential background on model reduction, the fundamental process of NN pruning, and GA [10] along with multi-objective evaluation. Additionally, we introduce the proposed metric Pruning Effectiveness Per-Epoch (PEPE), which is used to evaluate the performance of different model reduction algorithms.

A. Backgrounds

GA, inspired by natural evolution, is an optimization method dating from the 1970s [10]. Starting with a population of solutions (chromosomes), its application in model reduction involves pruning models. A fitness function measures each chromosome's effectiveness. Through selection, the fittest individuals reproduce using crossover and mutation, promoting genetic diversity. Evolution concludes when specific criteria are met, presenting an optimal pruned model as the solution. Refer to Fig. 1 for a GA flowchart.

Model Reduction is categorized into two types: unstructured and structured methods. The structured method involves the removal of entire neurons, channels, filters, or even layers in the model. This method shrinks the model and elevates computational efficiency when implemented on specific architectures.

On the other hand, unstructured method targets at parameter removal. As this method results in a sparse model with many parameters of 0s, it does not remove entire structural elements as seen in the structured method. Consequently, the unstructured method allows for potentially higher pruning ratios. Once deployed on hardware optimized for sparse matrix computations, model reduction through unstructured method can lead to lower computational cost and power consumption. In pursuit of superior pruning ratios, we adopt unstructured method in this work.

Both GA and Model Reduction play crucial roles in our research. Detailed methodologies, including their nuanced applications and implementations, will be elaborated upon in Section III.

B. Multi-objective Evaluation

Evaluating the performance of a reduced model involves a multi-objective evaluation. In prior research [21], the performance of a reduced model is typically evaluated based on three objectives:

1) Accuracy Drop: This is defined as the decline in the top-1 accuracy compared to the original model. The accuracy drop can be computed using EQ(1):

$$acc_{drop} = acc_{ori} - acc_{pruned} \tag{1}$$

2) Sparsity: Also known as the pruning ratio, sparsity indicates the degree of pruning in an entire model or a specific layer. For example, if there were originally 100 parameters and 30 were pruned, then the sparsity or pruning ratio is $30/100 = 30\%$. Sparsity is computed using EQ(2):

$$sparsity = \frac{n_{pruned}}{|parameter|} \times 100\% \tag{2}$$

where n_{pruned} denotes the total number of parameters pruned from the original model or a specific layer, and $|parameter|$ represents the total number of parameters in the original model or a specific layer.

3) Search Cost: This metric signifies the total number of epochs expended to obtain the reduced model. Since the pruning-retraining process may be iteratively performed for multiple times, and each retraining step needs a different number of epochs, the search cost accounts for the total number of epochs needed in all the iterations of the pruning-retraining process, and is shown as EQ(3):

$$search\_cost = \sum_{i=1}^{N} epoch_i \tag{3}$$

In EQ(3), N represents the number of pruning-retraining iterations, and $epoch_i$ denotes the number of epochs used in the i^{th} iteration of pruning-retraining. The search cost will be considered when comparing the effectiveness of various model reduction algorithms.

C. Pruning Effectiveness Per-Epoch (PEPE)

To compare different model reduction algorithms fairly, in this work, we introduce a new metric, Pruning Effectiveness Per-Epoch (PEPE). PEPE is a metric for evaluating the effectiveness of model reduction algorithms, i.e., evaluating sparsity achieved per search_cost.

The PEPE is calculated by EQ(4):

$$PEPE = \frac{sparsity}{search\_cost} \tag{4}$$

where $search\_cost$ is computed using EQ(3). A higher value of PEPE indicates that the algorithm is more efficient for finding a pruned model.

However, it is important to note that PEPE should be evaluated under a designated acceptable accuracy drop (acc_{drop}). For instance, if we prune 99.99% of the parameters from a model without retraining ($epoch_i = 0$), the PEPE value would be infinite. However, this is not meaningful as the accuracy of the model is lower than the acceptable level. Hence, we compare different model reduction algorithms with the PEPE metric under a specified acceptable acc_{drop}, e.g., $acc_{drop} \leq 2\%$ or $acc_{drop} \leq 5\%$.

III. The Proposed Approach

In this section, we present the proposed hybrid approach, which combines a pruning strategy extraction through GA, and a rule-based method. Since the number of biases is relatively small as compared to the number of weights, e.g., less than 0.1% of the total number of parameters in the model of our experiments, we do not include the biases in the pruning step. However, the biases are included in the total number of parameters when calculating the sparsity of an entire model.

A. Genetic Algorithm Design

In the traditional GA applications for model reduction, a chromosome (also known as an individual) is a solution to a problem, which is often represented by a string. If the representation of a solution is large when the models are significantly large, e.g., containing tens of millions of parameters, the length of the string becomes excessively long, leading to substantial overhead in computation or storage. In this work, we propose an improved GA to tackle this issue. The basic principles and procedures of the GA employed in this work are described in the following paragraphs.

1) Initialization: The GA begins with a population of potential solutions, each is represented as a chromosome. Each chromosome stands for a pruned model and is encoded as a *layer-pruning vector*. A layer-pruning vector is a sequence of integers, where each integer represents the number of weights pruned in each layer of the original model. If we have a model with N layers that possess the weight attribute, we can represent the number of weights $count_i$ in each $layer_i$ as follows:

$$C = [count_1, count_2, ..., count_N] \tag{5}$$

Consequently, the chromosome can be represented as a layer-pruning vector,

$$P = [pruned_1, pruned_2, ..., pruned_N] \tag{6}$$

In EQ(6), $pruned_i$ denotes the number of weights pruned in the i^{th} layer, and $0 \leq pruned_i \leq count_i$ for each i where $1 \leq i \leq N$. For each layer's pruning, we use a magnitude pruning strategy, i.e., the smallest weights (in terms of absolute value) in the i^{th} layer are pruned. Thus, every chromosome P corresponds to a $unique$ pruned model.

In the design of GA in this work, we generate an initial population of 40 individuals, $P1 \sim P40$. The group of individuals can be represented as follows:

$$I_{init} = [P_1, P_2, ..., P_{40}] \tag{7}$$

In EQ(7), P_i refers to the layer-pruning vector of the i^{th} individual. Each P_i can be represented as $P_i = [pruned_1^i, pruned_2^i, \ldots, pruned_N^i]$. During the generation of the initial population, each $pruned_j^i$ in P_i is initialized to a random number within the interval of $[count_j \times 0.5, count_j \times 0.8]$, where $count_j$ represents the number of weights in the j^{th} layer of the original model.

For example, consider a model with $N = 3$ layers, represented by the following weights in each layer:

$$C = [100, 200, 150]$$

Two individuals, P_1 and P_2, are from the initial population I_{init}. The layer-pruning vector for the individual P_1 is:

$$P_1 = [pruned_1^1, pruned_2^1, pruned_3^1] = [60, 110, 90]$$

For the individual P_2:

$$P_2 = [pruned_1^2, pruned_2^2, pruned_3^2] = [70, 145, 105]$$

In this representation, P_1 indicates that 60 weights are pruned from the 1^{st} layer, 110 from the 2^{nd} layer, and 90 from the 3^{rd} layer. Similarly, for P_2, 70 weights are pruned from the 1^{st} layer, 145 from the 2^{nd} layer, and 105 from the 3^{rd} layer.

2) Fitness Value Calculation: The fitness function is used to calculate the fitness value of each chromosome, signifying the performance of a pruned model. We propose a metric - Pruned Weight per Accuracy Drop (PWAD), to evaluate the fitness value as EQ(8):

$$PWAD = \frac{n_{pruned}}{(acc_{ori} - acc_{pruned})} \tag{8}$$

In EQ(8), n_{pruned} denotes the total number of pruned weights, which can be calculated by accumulating the amount of pruned weights in all the layers, as shown in EQ(9):

$$n_{pruned} = \sum_{i=1}^{N} pruned_i \tag{9}$$

Additionally, in EQ(8), acc_{ori} represents the top-1 accuracy of the original model, while acc_{pruned} stands for the top-1 accuracy of the pruned model. A larger PWAD value represents better performance of the corresponding pruned model. It is worth noting that when the pruning ratio is relatively small, acc_{pruned} may even exceed acc_{ori}, resulting in a negative PWAD value. Hence, we consider a negative PWAD value is better than a positive one.

3) Selection: The goal of the selection process is to favor the chromosomes with higher fitness values (PWAD) such that their genes can be inherited to the next generation. We sort all the individuals in a descending order by PWAD and retain the top half of them. The retained individuals can be represented as $I_{curr} = [P_1, P_2, ..., P_{20}]$.

4) Crossover: The crossover operation aims to produce new offspring by swapping parts of the chromosomes from two parent chromosomes. In our algorithm, we employ a technique called "uniform crossover" [24], where two parents, denoted as P_i and P_j (with $i \neq j$), are paired randomly to generate new offspring.

For the uniform crossover, we first generate a binary string, bin, of length N. Each position k in bin, is denoted as bin_k. If bin_k is 1, the offspring's gene at position k is inherited from P_i. Conversely, if bin_k is 0, it is inherited from P_j.

For example, let's consider a model with three layers and two chromosomes:

$$P_1 = [60, 110, 90] \qquad P_2 = [70, 145, 105]$$

Assuming the generated binary string bin is [1, 0, 1], the child chromosome P_{child} will be constructed as:

$$P_{child} = [60 \text{ (from } P_1), 145 \text{ (from } P_2), 90 \text{ (from } P_1)]$$

Therefore, P_{child} will be [60, 145, 90].

This crossover process is repeated for 40 times, yielding a new population denoted as $I_{next} = [P_1, P_2, \ldots, P_{40}]$.

5) Mutation: The mutation process introduces slight alterations to certain genes in a chromosome. Its importance lies in introducing new traits to the population, ensuring diversity, and enabling a comprehensive search throughout the evolution of the algorithm. In our algorithm, each $pruned_j^i$ in the individual P_i of

I_{next} has a mutation probability of σ, which is called mutation rate.

When mutation occurs, the value of $pruned_j^i$ is randomly adjusted within the interval of $[count_j \times 0.0, count_j \times 0.9]$. This mechanism ensures that our algorithm explores the search space thoroughly and avoids getting stuck to local minima.

To illustrate the mutation process, consider a model P_1 with three layers: $P_1 = [60, 110, 90]$.

For the 2^{nd} layer, we have $count_2 = 200$. If the mutation occurs for the genes in this layer, the mutated gene may become 175. Consequently, the mutated chromosome will be: $P_{mutated} = [60, 175, 90]$.

In this example, the mutation process led to the 2^{nd} layer having more weights pruned.

6) Replacement: To complete one generation cycle in the evolutionary algorithm, the old population I_{curr} is replaced by the newly generated population I_{next}. This replacement process ensures that the population is continually evolving towards better solutions by inheriting the characteristics of the newer generation. After the replacement, the algorithm will then proceed with the Fitness Value Calculation for the individuals followed by the Selection process.

In our algorithm, the GA is performed for a fixed period of time (3600 seconds), providing a balance between computational effort and solution quality. After running the genetic algorithm, the set of 10 individuals with the highest PWAD are selected, denoted as I_{best}. These individuals will serve as the basis for assisting the rule-based method, which will be presented in Section III. B.

B. Rule-based Method

To search a good solution encoded with an M-bit binary string in the solution space of GA, we face a challenge that the solution space is extremely enormous, being equal to 2^M, where M is the total number of parameters. To mitigate this, we introduce a Rule-based Method to assist the GA. Our approach aims to find common features among the high-quality offspring obtained from the GA, and reintegrate these features back into the GA. This reintegration includes changing the encoding and altering the initial population. By eliminating unnecessary search effort, we can accelerate the GA, as detailed below.

1) Deriving Common Rules: We aim to find common features among the individuals in I_{best} obtained from the GA. These common features need to be collated into rules that can be reintegrated into the GA process. Three specific rules are derived as follows:

Magnitude Pruning: we observed that the weights with smaller absolute values had a higher likelihood of being pruned in each layer of each individual in I_{best}. Thus, we integrated this rule into the GA by using the data structure of *layer-pruning vector* as mentioned in Section III. A.

Compact-Key Layers: We noticed that when some layers are very small, the pruning ratios of these layers are also very small in all P_i of I_{best}. Hence, these layers would be quite crucial such that they need to be reserved in subsequent iterations of the GA. We refer to these layers as Compact-Key Layers (CKL). In practice, a layer is considered a CKL if and only if the number of weights in the layer is less than 0.1% of the total number of parameters in the original network and the pruning ratio of the layer in each individual of I_{best} is less than 20%.

Initial Layer-Pruning Vector (ILPV): In the first iteration of the GA, each $pruned_j^i$ in the generated chromosome P_i is initialized to a value within the interval of $[count_j \times 0.5, count_j \times 0.8]$. In subsequent (non-initial) GA iterations, we average the corresponding $pruned_j^i$ in each P_i of I_{best} to determine the initial values for the new population. For randomization, we randomly increase or decrease each generated individual by 5% of the number of weights in the corresponding layer. The specific generation formula for each P_{init} is as follows:

$$P_{init} = [..., (\frac{1}{|I_{best}|} \sum_{i=1}^{|I_{best}|} pruned_j^i) + Jitter_j, ...] \quad (10)$$

In EQ(10), P_{init} is the initial layer-pruning vector for the new population. $|I_{best}|$ is the number of individuals in I_{best}, $pruned_j^i$ represents the number of pruned weights in the j^{th} layer of the i^{th} individual, and $Jitter_j$ denotes a random offset value, which is obtained using EQ(11).

$$Jitter_j = random[-count_j \times 0.05, count_j \times 0.05] \quad (11)$$

In EQ(11), $count_j$ is the total number of weights in the j^{th} layer. Additionally, after generating P_{init}, we manipulate each $pruned_i$ in P_{init} to ensure that its value falls within the interval of $[0, count_i]$. Specifically, we perform two operations: $pruned_i = max(pruned_i, 0)$ and $pruned_i = min(pruned_i, count_i)$.

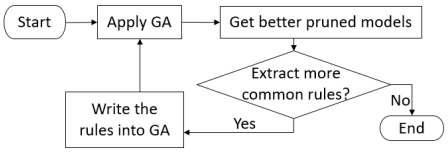

Fig. 2: The General Flowchart of Our Approach.

2) Integration of Rules in GA: Having extracted the rules from the optimal individuals, we then reintegrate these rules back into the GA. These rules can help guide the GA in a more efficient and effective manner. The reintegration methods for different types of rules are as follows:

Magnitude pruning: This rule is integrated into the GA by using a new encoding method. Specifically, each individual is no longer represented as a binary string that encodes weights. Instead, it is represented as a layer-pruning vector, where the i^{th} element in the vector represents the number of weights to be pruned in the i^{th} layer.

Compact-Key Layers (CKL): This rule helps guide the mutation during the mutation process of the GA. In practice, we avoid mutating on the CKLs to ensure that these layers remain unpruned.

Initial Layer-Pruning Vector (ILPV): This rule affects the initial population during the initialization of GA. The initial layer-pruning vector for each individual in the population of the non-initial GA is set based on EQ(10).

In combining the extracted rules with the GA, we can significantly improve the efficiency in searching of high quality solutions. Although in this work, we only extract three rules, this hybrid approach can be generalized to other applications using GA. The general flowchart of our approach is as shown in Fig. 2.

C. Pruning-Retraining

We perform several iterations of pruning-retraining to get the final reduced model. According to the experimental results, two to three iterations could lead to a reduced model with good performance. The details of pruning and retraining are as follows.

TABLE I: Performance Comparsion of ResNets across Several Pruning-Retraining Iterations.

Dataset	Model	Iteration	Sparsity (%)	$\|Epoch\|$	Accuracy (%)	Acc_{drop} (%)	Runtime (s)	PEPE	
								[12]	Ours
CIFAR-10	ResNet-20	0	0	-	92.23	0	-	-	-
		1	55.01	2	90.93	1.30	80	1.00	27.51
		2	74.06	5	90.36	1.87	144	1.00	14.81
		3	81.98	9	90.44	1.79	371	1.00	9.11
CIFAR-10	ResNet-56	0	0	-	93.51	0	-	-	-
		1	55.96	2	93.16	0.35	208	1.00	27.98
		2	75.08	5	92.20	1.31	303	1.00	15.02
		3	82.94	9	91.59	1.92	914	1.00	9.22
CIFAR-10	ResNet-110	0	0	-	93.42	0	-	-	-
		1	56.18	2	93.27	0.15	330	1.00	28.09
		2	75.32	5	92.23	1.19	501	1.00	15.06
		3	83.17	9	91.82	1.60	1442	1.00	9.24

1) Pruning: We iterate the GA for two times; each run cost 3600 seconds. The first run of GA yields I_{best} and rules, which include CKL and the ILPV. After incorporating these rules back into the GA, we run the updated GA to obtain I_{best2}. From I_{best2}, we extract new rules, including CKL and ILPV. These rules serve as the strategies in the pruning phase. The CKL will not be pruned, and the remaining layers will be pruned based on their corresponding pruning ratios in ILPV.

2) Retraining: After pruning, we retrain the pruned model for some epochs with respect to the accuracy drop. The retraining strategy we used is fine-tuning [17] with a fixed learning rate [23].

D. Overall Flow of the Proposed Approach

The overall algorithm of our approach is summarized as follows:

We run GA and obtain the best individuals, I_{best}. From I_{best}, we extract common rules, including CKL and the ILPV. These rules are then incorporated back into the GA. With these rules, we run GA again and obtain new best individuals, I_{best2}. From I_{best2}, we extract new common rules, including CKL and ILPV, which will serve as our pruning strategy. This strategy includes the CKL, which will not be pruned, and ILPV, which dictates the pruning ratio for the remaining layers. After pruning, we retrain the model for 1-4 epochs depending on the accuracy drop. The retraining involves fine-tuning with a fixed learning rate. The overall flow of the proposed approach is depicted in Fig. 3.

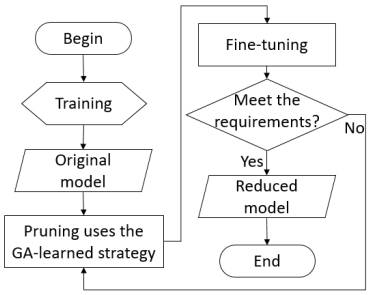

Fig. 3: The Overall Flow of the Proposed Approach.

IV. Experimental Results

We implemented the proposed approach using Python 3 language and Pytorch Library [18]. The implementation of the state-of-the-art was adapted from the works presented in [12]. The experiments were carried out on a machine running Windows 10 Professional, equipped with an Intel i7-7700K CPU (4.2 GHz and 32 GB RAM) and an NVIDIA GeForce GTX 1080 Ti GPU (1481 MHz and 11 GB VRAM).

The experimental results are summarized in Tables I, II and III. We used a benchmark that includes a gesture recognition model sourced from R(2+1)D-18 [25], trained on the EgoGesture dataset [1], [26]. Additionally, for comparison with previous research results, our benchmark also included three ResNets [8], trained on the CIFAR-10 and CIFAR-100 dataset [11]. The hyperparameters used for training these models were adopted from the study by Renda et al. [21]. The accuracy, sparsity, and runtime presented in the tables were calculated by taking the average results of three independent runs. In our experiment, we set the mutation rate σ as 2.5%.

According to Table I, when allowing for a 2% accuracy drop, our method notably improves PEPE compared to the state-of-the-art at various sparsity levels. For instance, at sparsities over 80%, our PEPE is more than nine times of that of the state-of-the-art, suggesting our approach obtains the same pruned models with a speed up of nine.

Table I showcases the efficiency of the three ResNets after varying numbers of pruning-retraining cycles. We employ the PEPE (Pruning Effectiveness Per-Epoch) metric to benchmark these results. In [12] [21], 20% of the parameters are pruned in every iteration and then retrained for a minimum of 20 epochs. Consequently, the sparsity and search cost (retraining epochs) increase at the same rate, making the PEPE metric of EQ(4) for these state-of-the-art methods consistently equal 1.

For the state-of-the-art methods:

$$PEPE_{SOTA} = \frac{20 \times |iteration|}{20 \times |iteration|} = 1 \tag{12}$$

In Table II, the column Acc_{ori}, Acc_{pruned} denotes the top-1 accuracy of the original model and the pruned model, respectively. As can be seen from Table II, all three ResNets models can achieve over 80% sparsity with less than 2% drop in accuracy after 9 epochs of retraining. Furthermore, the gesture recognition model can reach over 93% sparsity with an accuracy drop of less than 1%. For the CIFAR-100 dataset, the performance was not as prominent as with CIFAR-10. Nevertheless, we can still achieve over 74% sparsity while maintaining an accuracy drop within 2%.

Table III summarizes the performance of the pruned gesture recognition model after different numbers of pruning-retraining iterations. The column $|Epoch|$ denotes the number of epochs used for retraining in each iteration, where the learning rate of 2.45×10^{-5} was used in our experiment. For having an acceptable accuracy $\geq$ 90%, we achieved a sparsity of 97.92% after three iterations. Furthermore, to highlight the effectiveness of our approach during the pruning phase, we show the results

TABLE II: Experimental Results with Accuracy Drop Less than 2%.

Dataset	Model	$\|Parameter\|$	Acc_{ori} (%)	Acc_{pruned} (%)	Acc_{drop} (%)	Sparsity (%)	$\|Epoch\|$	Runtime (s)
CIFAR-10	ResNet-20	269,722	92.23	90.44	1.79	81.98	9	371
	ResNet-56	853,018	93.51	91.59	1.92	82.94	9	914
	ResNet-110	1,727,962	93.42	91.82	1.60	83.17	9	1442
CIFAR-100	ResNet-56	858,868	69.66	67.72	1.94	74.06	8	366
	ResNet-110	1,733,812	71.48	69.60	1.88	74.18	8	571
EgoGesture	R(2+1)D-18	31,305,255	96.85	97.01	<0	93.07	3	521

TABLE III: Performance of Gesture Recognition Model (R(2+1)D-18) across Several Pruning-Retraining Iterations

Iteration	Sparsity (%)	$\|Epoch\|$	Accuracy (%)	Runtime (s)
0	0.00	-	96.85	-
1	74.61	1	96.68	179
2	93.07	3	97.01	521
3	97.92	6	95.52	1054
1	74.61	0	93.07	<1

of a single pruning operation without retraining in the last row of Table II. As can be seen, a high pruning ratio 74.61% was achieved while still satisfying the accuracy requirement (93.07%) for the model.

It is worth noting that the state-of-the-art method [12], was not applied on CIFAR-100 dataset. As a result, we have refrained from making a direct PEPE comparison for CIFAR-100 in Table I. However, since PEPE of the state-of-the-art remains consistently, our approach would exhibit a higher search efficiency on the CIFAR-100 dataset as well.

These experimental results in Table I, II and III strongly indicate the superior efficiency of our model reduction strategy, especially when a minor accuracy drop for the pruned models is acceptable.

V. Conclusion and Future Work

In this work, we propose a hybrid approach for model reduction, which uses a genetic algorithm and a rule-based method to determine the pruning strategy during the pruning-training cycles. Our approach demonstrates a higher efficiency than the state-of-the-art methods for obtaining pruned models with similar performance on several ResNet models. Although our approach is currently applied to unstructured pruning, it could be further extended to structured pruning. This will be our future work.

References

[1] C. Cao, Y. Zhang, Y. Wu, H. Lu, and J. Cheng, "Egocentric gesture recognition using recurrent 3d convolutional neural networks with spatiotemporal transformer modules," in *Proc. of the IEEE International Conference on Computer Vision*, 2017, pp. 3763–3771.

[2] T. Chen, B. Ji, T. Ding, B. Fang, G. Wang, Z. Zhu, L. Liang, Y. Shi, S. Yi, and X. Tu, "Only train once: A one-shot neural network training and pruning framework," *Advances in Neural Information Processing Systems*, vol. 34, pp. 19 637–19 651, 2021.

[3] H. Chiroma, S. Abdulkareem, A. Abubakar, and T. Herawan, "Neural networks optimization through genetic algorithm searches: A review," *Appl. Math. Inf. Sci*, vol. 11, no. 6, pp. 1543–1564, 2017.

[4] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," *arXiv preprint arXiv:1803.03635*, 2018.

[5] T. Gale, E. Elsen, and S. Hooker, "The state of sparsity in deep neural networks," *arXiv preprint arXiv:1902.09574*, 2019.

[6] S. Han, "Efficient methods and hardware for deep learning," Ph.D. dissertation, Stanford University, 2017.

[7] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," *Advances in Neural Information Processing Systems*, vol. 28, 2015.

[8] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[9] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural Computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[10] J. H. Holland, *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence*. University of Michigan Press, 1975.

[11] A. Krizhevsky, G. Hinton, *et al.*, *Learning multiple layers of features from tiny images*. Toronto, ON, Canada, 2009.

[12] D. H. Le and B.-S. Hua, "Network pruning that matters: A case study on retraining variants," *arXiv preprint arXiv:2105.03193*, 2021.

[13] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[14] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," *arXiv preprint arXiv:1608.08710*, 2016.

[15] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming," in *Proc. of the IEEE International Conference on Computer Vision*, 2017, pp. 2736–2744.

[16] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, "Rethinking the value of network pruning," *arXiv preprint arXiv:1810.05270*, 2018.

[17] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[18] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[19] J. Poyatos, D. Molina, A. D. Martinez, J. Del Ser, and F. Herrera, "Evoprunedeeptl: An evolutionary pruning model for transfer learning based deep neural networks," *Neural Networks*, vol. 158, pp. 59–82, 2023.

[20] R. Reed, "Pruning algorithms-a survey," *IEEE Transactions on Neural Networks*, vol. 4, no. 5, pp. 740–747, 1993.

[21] A. Renda, J. Frankle, and M. Carbin, "Comparing rewinding and fine-tuning in neural network pruning," *arXiv preprint arXiv:2003.02389*, 2020.

[22] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[23] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[24] G. Syswerda *et al.*, "Uniform crossover in genetic algorithms," in *Proc. of International Conference on Genetic Algorithms*, vol. 3, 1989, pp. 2–9.

[25] D. Tran, H. Wang, L. Torresani, J. Ray, Y. LeCun, and M. Paluri, "A closer look at spatiotemporal convolutions for action recognition," in *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 6450–6459.

[26] Y. Zhang, C. Cao, J. Cheng, and H. Lu, "Egogesture: A new dataset and benchmark for egocentric hand gesture recognition," *IEEE Transactions on Multimedia*, vol. 20, no. 5, pp. 1038–1050, 2018.

[27] M. Zhu and S. Gupta, "To prune, or not to prune: Exploring the efficacy of pruning for model compression," *arXiv preprint arXiv:1710.01878*, 2017.