CNN Model Optimization Using a Hybrid Approach of Genetic Algorithm-based Pruning and Retraining with Knowledge Distillation

Kuan-Ling Chou
National Tsing Hua University
Hsinchu, Taiwan
lily12457@gmail.com

Cheng-Lung Wang
National Tsing Hua University
Hsinchu, Taiwan
brother890225@gmail.com

Yung-Chih Chen
National Taiwan University of Science and Technology
Taipei, Taiwan
ycchen.ee@mail.ntust.edu.tw

Wuqian Tang
National Tsing Hua University
Hsinchu, Taiwan
wqtang@cs.nthu.edu.tw

Yi-Ting Li
National Tsing Hua University
Hsinchu, Taiwan
yitingli.yt@gmail.com

Shih-Chieh Chang
National Tsing Hua University
Hsinchu, Taiwan
scchang@cs.nthu.edu.tw

Chun-Yao Wang
National Tsing Hua University
Hsinchu, Taiwan
wcyao@cs.nthu.edu.tw

Abstract

Convolutional neural networks (CNN) have achieved significant success in various domains, e.g., image recognition, however, deploying CNN on resource-constrained devices remains challenging due to their high computation demands and large memory requirements. In this paper, we propose an approach for structured pruning of CNN using genetic algorithms, specifically focusing on channel pruning. The pruning process often results in a loss of accuracy, requiring retraining techniques to recover the network's accuracy to its original level. We integrate knowledge distillation into the retraining process to recover the model to a similar accuracy within fewer epochs. The experimental results show that our approach successfully reduces the amounts of parameters by 91.02% and computation demands by 90.96%, with a 1.82% decrease in accuracy only for a gesture recognition model with over 30 million parameters. The results demonstrate that our approach significantly reduces the computation of the model without dramatically decreasing its accuracy.

ACM Reference Format:
Kuan-Ling Chou, Cheng-Lung Wang, Yung-Chih Chen, Wuqian Tang, Yi-Ting Li, Shih-Chieh Chang, and Chun-Yao Wang. 2025. CNN Model Optimization Using a Hybrid Approach of Genetic Algorithm-based Pruning and Retraining with Knowledge Distillation. In *Great Lakes Symposium on VLSI 2025 (GLSVLSI '25), June 30–July 02, 2025, New Orleans, LA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3716368.3735193

1 Introduction

In recent years, Convolutional Neural Networks (CNN) [22] have shown remarkable success and advancements in various domains, such as image recognition, object detection, and autonomous driving. These advances improve the performance of applications in these domains. With the widespread application of deep learning in various fields, the complexity and size of the model have gradually increased, significantly raising the demand for computational resources. However, the effectiveness of CNN often comes with a requirement of extensive computational resources due to their large number of parameters and deep layers [7]. Deploying large-scale models on resource-constrained embedded or mobile devices becomes a challenge, especially for applications with strict constraints on inference time and power consumption. For instance, headsets in AR/VR application require real-time processing capabilities to ensure user experience, while mobile devices have limited battery life. Hence, it is critical to reduce model size and computation to minimize computation, power consumption, and inference time. To address this challenge, many researchers have explored various techniques to make these models smaller [2], [8], [24], [31], [38]. One of the approaches is model pruning, which reduces the model size without significantly affecting its accuracy. Han [8] introduced an unstructured pruning method, demonstrating a significant reduction in the number of parameters without dramatically degrading the accuracy. However, Liu [24] proposed a structured pruning method by pruning the entire filter in a convolutional layer, which was proved to be more hardware-friendly. He [12] further introduced a channel pruning method that reduces the computation requirement of model through an iterative pruning and retraining strategy. Additionally, Luo [26] proposed ThiNet method, which considers the importance of each channel and improves the pruning efficiency. To recover from the possible accuracy drop after pruning, retraining is a common used technique. Fine-tuning (FT) [8] is a conventional method that involves retraining the pruned network using the minimum learning rate in the training schedule of the original network. Another retraining technique, known as learning rate rewinding (LRW) [32], adopted the training schedule of the original network and set its retraining schedule based on the last few epochs. LRW demonstrates a better accuracy compared to the FT. Despite the advancements in pruning techniques, finding a good balance between model size and accuracy remains a challenge, particularly for real-time applications on resource-constrained devices. The challenge is how to reduce the model size while maintaining its accuracy.

In this paper, we propose a hybrid approach that integrates genetic algorithm with channel pruning and retraining with knowledge distillation (RKD). Channel pruning is a structured pruning method that reduces the model size by removing unimportant channels, thus reducing inference time and power consumption on edge

This work is supported in part by the National Science and Technology Council (Taiwan) under NSTC 112-2218-E-007-014, NSTC 112-2221-E-007-106-MY2, NSTC 112-2221-E-007-108, NSTC 113-2218-E-007-020, NSTC 113-2221-E-007-082-MY3, NSTC 113-2425-H-007-004, NSTC 113-2640-E-011-003 and NSTC 114-2425-H-007-002.

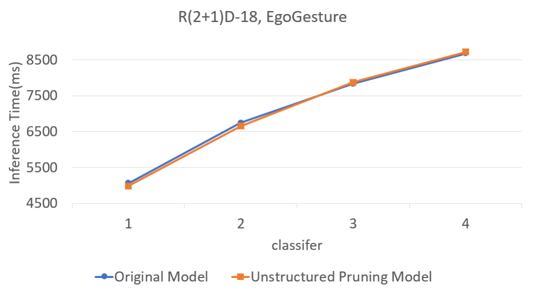

Figure 1: Average inference time of the original and unstructured pruned R(2+1)D-18 model on the EgoGesture dataset for hand gestures running on the HoloLens headset.

devices. Then, we integrate knowledge distillation into retraining process to elevate the accuracy of this pruned model. Our approach takes a well-trained model as input and returns an optimized model. We apply our approach to a gesture recognition model containing over 30 million parameters. The experimental results show that our approach can significantly optimize the model.

2 Preliminaries

2.1 Neural Network Pruning

Neural network model pruning can be categorized into two types: unstructured pruning and structured pruning. Unstructured pruning [8], [15], [34] usually sets the weights of edges with small magnitude to zero, hence effectively eliminating network computation. Unstructured pruning significantly reduces the total number of parameters within the network. However, since the positions of weights to be pruned in the unstructured pruning are arbitrary, this method requires special support in hardware to achieve model size reduction. On the other hand, structured pruning [11], [25], [36] removes the entire kernels, channels, or layers based on predetermined rules. This pruning method makes it easier to achieve model reduction because it maintains an organized network architecture, which loosens the computation requirements. Structured pruning is often preferred in practice because it is compatible with existing hardware and is easier to deploy in various applications.

Both pruning methods aim to reduce the complexity of model, but they have different impacts on hardware acceleration. For example, for the CPU on the HoloLens [27], an unstructured pruning model requires almost the same inference time as the original one, even 80% of parameters are removed, as shown in Figure 1. Therefore, a good pruning approach should focus on structured pruning, physically reducing the number of channels or filters in convolutional or fully-connected layers.

2.2 Channel Pruning

Channel pruning is a type of structured pruning that focuses on reducing the number of channels in convolutional layers. This approach aims to reduce the computation and memory usage of CNN models by eliminating less important channels. Channel pruning usually involves evaluating the importance of each channel and removing those that contribute the least to the network. The importance of a channel can be measured using various criteria, such as L1 [21], [24] or L2 norm [30], or the Taylor Expansion [28]. For example, He [12] introduced an iterative two-step algorithm to prune channels in CNN model for reducing the computational cost of neural networks. In this work, we utilize the Taylor Expansion [28] to calculate the importance of the channel.

2.3 Genetic Algorithm

Genetic algorithm (GA) is a search algorithm that imitates the principles of natural evolution. GA mimics the process of natural selection by choosing the most suitable individual from a population to generate offspring in the next generation. It employs biological-inspired mechanisms such as mutation, crossover, and selection to evolve a population of solutions toward an optimal configuration. GA has been effectively used to approximate logic synthesis problems [23]. In model pruning, the goal is to find a pruning strategy that balances the accuracy and size of model.

In this work, we apply GA to identify the most effective pruning strategy for our neural network models, leveraging their ability to explore optimal solutions, as illustrated in Figure 2. The ability of GA to explore multiple solutions and optimize multiple goals simultaneously allows it to balance the model size and accuracy. We first generate an initial population and evaluate the fitness value of each solution. Within the predefined number of generations, the solutions undergo crossover and mutation. Then, we select the solutions that perform well in the population to generate new solutions. These steps repeat until the termination condition is met, and the optimal solution is obtained.

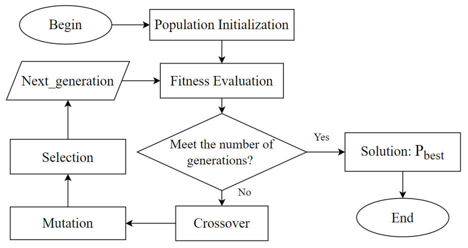

Figure 2: The flow of genetic algorithm.

2.4 Knowledge Distillation

Knowledge distillation [10], [13] is a technique in machine learning that utilizes the soft probabilities (or "logits"), produced by a larger, more complex model known as the "teacher network", to supervise and enhance the training of a smaller, simpler model called the "student network". This model compression strategy allows the student model to learn detailed information beyond what is available through labels, enhancing the learning process. Additionally, transfer learning can be integrated by pre-training the student model on a different but related task before it learns from the teacher model. This pre-training aids in developing a proper feature representation, significantly improving the student model's performance and efficiency. By transferring knowledge from the teacher model to the student model, the student model can achieve comparable performance while requiring less computational overhead and memory, making it more efficient for deployment in resource-constrained environment.

2.5 Retraining Techniques

Pruning could lead to a drop in the model's accuracy. Retraining can be used to recover the accuracy by allowing the remaining parameters to be adjusted to the pruned architecture. There are several techniques used to retrain the networks to regain accuracy.

(1) Fine-Tuning: Fine-tuning (FT) [8] involves retraining the pruned model on the original dataset with a lower learning rate. This helps regain the accuracy lost during the pruning process by allowing the remaining weights to be adjusted for compensating the removed parameters. (2) Learning Rate Rewinding: Learning Rate Rewinding (LRW) [32] involves retraining the pruned model using an initial learning rate schedule from an earlier epoch, allowing the model to retrace its training steps and regain the accuracy lost from the pruning. This method starts with a large learning rate for several epochs, then applies progressively smaller learning rates for additional epochs, which adjusts the model more effectively. Compared to fine-tuning, which uses a lower learning rate to retrain the pruned model, LRW can offer better accuracy recovery by leveraging the dynamics of the original learning rate schedule, making it a potentially superior approach.

3 Proposed Approach

3.1 Genetic Algorithm

When GA is applied to model pruning, solutions to the problem are encoded, and named chromosomes, typically represented as strings. However, when the model contains tens of millions of parameters, the representation will become large, leading to an excessive computation and storage burden. In this work, we propose an efficient GA-based method to deal with this issue. The following contents will explain the principles and steps of the GA used in this work.

3.1.1 Initialization. A CNN model is composed of multiple layers, with each layer containing a certain number of channels. For a given CNN model, suppose that the model contains N layers, the channel number distribution can be expressed as EQ(1),

$$ChannelCounts = [channel_1, channel_2, \ldots, channel_N] \tag{1}$$

where $channel_i$ is the number of channels in each $layer_i$.

In our work, each individual (chromosome) within the population is represented as a unique pruning vector. A pruning vector consists of elements, where each element represents a pruning ratio for a corresponding layer in the CNN. These ratios determine how many channels in a layer are pruned. This representation of chromosome can be expressed as EQ(2),

$$P_i = [pruned_{i,1}, pruned_{i,2}, \ldots, pruned_{i,j}, \ldots, pruned_{i,N}] \tag{2}$$

where P_i represents a pruning vector for the i^{th} individual, and $pruned_{i,j}$ representes the pruning ratio (gene) for the i^{th} individual in $layer_j$.

Example 1: Consider a CNN model with three layers, assume that each layer has 128, 64, and 256 channels, respectively. Hence, $ChannelCounts = [128, 64, 256]$. Assume that a pruning vector is represented as $[0.2, 0.32, 0.55]$, indicating that 20% of the channels in $layer_1$, 32% of the channels in $layer_2$, and 55% of the channels in $layer_3$ are pruned.

To determine the pruning ratio in the pruning vector, we combine an initial pruning ratio ($Initial\_Pruning\_Ratio_j$) with a noise factor ($NF_{i,j}$), which is used to increase the variability of the pruning ratio. The formula for computing the pruning ratio for each $layer_j$ in the i^{th} individual is as shown in EQ(3),

$$pruned_{i,j} = Initial\_Pruning\_Ratio_j + NF_{i,j} \tag{3}$$

where $pruned_{i,j}$ is the pruning ratio of $layer_j$ in the model. $NF_{i,j}$ represents a noise factor derived from a uniform distribution for the i^{th} individual in $layer_j$, which ranges [-0.2, 0.2].

We use a strategy that considers the amount of channels in each layer to determine the pruning ratios. That is, allowing layers with more channels to have a larger pruning ratio. This is because layers with more channels could tolerate more pruning without causing detrimental effects on model's accuracy. The initial pruning ratio for each layer in the individual is set using a formula in EQ(4),

$$Initial\_Pruning\_Ratio_j = \left(1 - e^{-\lambda \times \frac{channel_j}{\kappa}}\right) \tag{4}$$

where κ and λ are user-defined parameters, and $channel_j$ is the number of channels in $layer_j$.

Combining EQ(3) and EQ(4), the pruning ratio for each $layer_j$ in the i^{th} individual is as EQ(5):

$$pruned_{i,j} = \left(1 - e^{-\lambda \times \frac{channel_j}{\kappa}}\right) + NF_{i,j} \tag{5}$$

The initial population consists of multiple individuals, each is characterized by a distinct pruning vector. The initial population can be expressed as EQ(6),

$$Init = [P_1, P_2, \ldots, P_M] \tag{6}$$

where M is the total number of individuals in the population, and each P_i represents the pruning vector for the i^{th} individual.

3.1.2 Crossover. The crossover operation is a critical step in generating new offspring. In our work, we employ two methods: Uniform Crossover [33] and Arithmetic Crossover [18]. With this idea, we expect that the offspring can inherit different characteristics from parents, which promotes genetic diversity.

In Uniform Crossover, each pruning ratio in the pruning vector is randomly chosen from one of the two parents with the same probability. For each gene in the chromosome, a random decision is made based on a predefined *selection_rate*. If the random number, $rand_C$, is less than *selection_rate*, the gene is selected from the first parent, say P_i; otherwise, it is inherited from the second parent, say P_j. The operation can be formulated as EQ(7),

$$pruned_{i,j}^{Offspring} = \begin{cases} pruned_{i,j}^{P_i}, & \text{if } rand_C < selection\_rate \\ pruned_{i,j}^{P_j}, & \text{otherwise} \end{cases} \tag{7}$$

Arithmetic Crossover uses a weighted average of the corresponding genes from two parents. The weight, r, is determined by a randomly generated ratio, ensuring that each parent contributes partial genes to the offspring. The gene of the offspring is calculated by weighting the corresponding parent genes P_i and P_j. The equation is as EQ(8).

$$pruned_{i,j}^{Offspring} = r \cdot pruned_{i,j}^{P_i} + (1 - r) \cdot pruned_{i,j}^{P_j} \tag{8}$$

Both crossover methods help maintain genetic diversity within the population, which is essential to avoid getting stuck in a local optimal point, and ensure a comprehensive exploration of the pruning space. In our work, we split our population into two groups. One uses Uniform Crossover, and the other uses Arithmetic Crossover.

3.1.3 Mutation. The mutation is an operation used to increase genetic diversity of the population and prevent convergence to a local optimal point. With the changes in the pruning ratio, mutation helps explore new regions of the solution space and aids in searching the global optimal point. In our approach, the probability of mutation is determined by a predefined parameter *mutation_rate*.

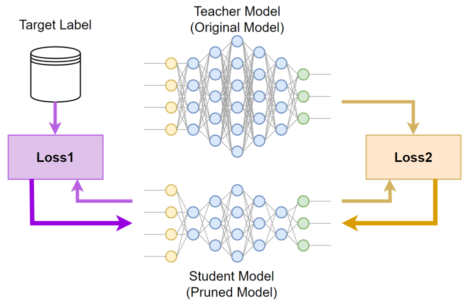

Figure 3: RKD Schematic Diagram.

For each pruning ratio in the pruning vector, the mutation is performed or not based on this probability. For each layer of each individual, we first generate a random number $rand_M$. If $rand_M$ is less than $mutation\_rate$, the $pruned_{i,j}$ for this layer is modified by EQ(9),

$$pruned'_{i,j} = pruned_{i,j} + mutation\_factor \quad \text{if } rand_M < mutation\_rate \tag{9}$$

where $pruned'_{i,j}$ and $pruned_{i,j}$ denote the mutated pruning ratio and the original pruning ratio of the $layer_j$ in the i^{th} individual, respectively. $mutation\_factor$ is a constant that determines the magnitude of change.

3.1.4 Fitness Function. The fitness function evaluates the quality of each individual within the population, aiming to optimize CNN model by balancing three key aspects: accuracy, model size, and computation. In our work, we pursue to achieve a high accuracy while minimizing the amount of operations and size of the model. The fitness value for each individual is calculated as EQ(10),

$$Fitness\ value = \alpha \cdot Acc + \frac{\beta}{\log(|Param|)} + \frac{r}{\log(|FLOP|)} \tag{10}$$

where Acc represents the accuracy of the model, $|Param|$ is the number of parameters in the model, and $|FLOP|$ [14] denotes the number of floating-point operations in the model during computation. α, β, and γ are weights used to adjust the magnitude of influences about the accuracy, the model size, and the amount of operations. The setting of these weights reflects the importances of these key aspects in the final model.

3.1.5 Selection. The individual with a higher fitness value will be retained in the selection process for the next generation, ensuring that only the best-performing individuals survive in the evolution process of GA. We rank all the individuals by their fitness values and retain the top half for the next generation. The population of the retained individuals $I_{current}$ can be represented as EQ(11):

$$I_{current} = [P_1, \ldots, P_{\frac{M}{2}}] \tag{11}$$

After the process of the GA, the best-performing individual in the process is referred to as P_{best}, which will be utilized for pruning the model based on the importance of the channel.

3.2 Retraining with Knowledge Distillation (RKD)

The state-of-art retraining methods, like LRW [32] or FT [8], focus on minimizing the loss solely between the predicted outputs and the target labels. This loss, typically computed as a measure of the discrepancy between the model's predictions and the actual outcomes, often utilizes methods such as mean squared error [1] or cross-entropy [17], depending on the specific task. However, these methods do not leverage the output information from the original model during the retraining process, and lead to inefficiencies. Especially, treating all negative labels identically disregards the varying degrees of relevance, or information these labels might carry in the original model's context. To address this issue, we propose to incorporate the technique of knowledge distillation in the retraining process. Knowledge distillation utilizes the original model's output to guide the retraining of the new model, ensuring that the subtleties and insights embedded in the negative labels are not lost. This enhances the learning process by reducing the prediction error in terms of the direct loss between outputs and target labels. It also preserves and transfers the subtle distinctions learned by the original model to the retrained model.

As shown in Figure 3, we set the original model as the teacher model and the pruned model as the student model. The overall loss function is divided into two components. The first component, depicted in EQ(13) and corresponding to the right half of Figure 3, employs the Kullback-Leibler divergence (KL_div) [20], to assess and minimize the disparity between the softened output probabilities of the student and teacher models. This technique, which involves a "temperature" parameter T, enables the student model to approximate the complex behavior of the teacher model by adopting its probability distribution. The temperature T in the softmax function adjusts the sharpness of the output probability distribution, helping the student model to learn more effectively from the teacher's output. The second component, outlined in EQ(12) and corresponding to the left half of Figure 3, aligns with the approach taken by previous works such as LRW [32]. It directly measures the difference between the student model's predictions and the actual target labels using a traditional loss metric like cross-entropy, which emphasizes precise prediction accuracy. The integration of these two loss components through EQ(14), with an adjustable hyperparameter δ, allows for a balanced training regimen that not only aims for direct adherence to the training data but also the absorption of more profound, more abstract patterns from the teacher model, thereby enhancing the student model's ability.

$$\text{Loss1} = \text{Loss}(\text{Softmax}(\text{Output}_{\text{Student}}), \text{Target Label}) \tag{12}$$

$$\text{Loss2} = T^2 \times KL\_div\left(\text{Softmax}\left(\frac{\text{Output}_{\text{Student}}}{T}\right), \text{Softmax}\left(\frac{\text{Output}_{\text{Teacher}}}{T}\right)\right) \tag{13}$$

$$\text{Loss}_{\text{Total}} = \delta * \text{Loss1} + (1 - \delta) * \text{Loss2} \tag{14}$$

As shown in EQ(15), the function $T(epoch)$ describes how the temperature parameter T used in EQ(13), starts at T_0 and decreases linearly to 1 as the number of retraining epochs increases. $epoch$ represents the current epoch, and E is the total number of epochs. This dynamic adjustment of the temperature helps in controlling the smoothness of the probability outputs and facilitates effective knowledge transfer between the models.

$$T(epoch) = T_0 - \frac{T_0 - 1}{E} \cdot epoch \tag{15}$$

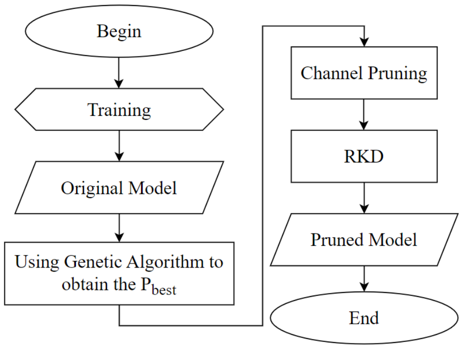

Figure 4: The overall flow of the proposed approach.

3.3 Overall Flow

Figure 4 shows the overall flow of our work. Given a well-trained original model, we conduct an iterative optimization on the original model by GA to find the P_{best}. After identifying P_{best}, we apply the channel pruning, which involves removing entire channels of neural networks based on their importance. We retrain the model with the proposed RKD approach to enhance the accuracy after pruning. Last, the pruned model will be returned.

4 Experimental Results

Our experiments were conducted using Python and the PyTorch Library [6], [29]. The computing platform is Windows 10, equipped with an Intel i5-13600 CPU, NVIDIA RTX 4090, and 64GB of RAM. We utilize ResNet-32, ResNet-56 [9], and R(2+1)D-18 [35] in our experiments. The R(2+1)D-18 model is designed for gesture recognition in a video. In our experiments, we evaluated the proposed approach by running the models for 10 to 20 generations. The parameters for GA are $\kappa = 150$ and $\lambda = 0.2$, which were used to initialize the pruning vectors for each individual in the population. The parameters $\alpha = 1$, $\beta = 4$, and $\gamma = 4$ in the fitness function. The parameters $\delta = 0.5$, $T_0 = 20$, and $E = 150$ in the proposed RKD approach.

4.1 R(2+1)D-18 on EgoGesture Dataset

For video-based applications, we employed the R(2+1)D-18 model [35], which is designed for gesture recognition tasks. This model is evaluated using the EgoGesture dataset [3], [37], which comprises 83 different hand gestures. The R(2+1)D-18 model utilized in our experiments is enhanced through a self-distillation process, resulting in a configuration with four distinct classifiers, A, B, C, and D. Each classifier corresponds to the different numbers of layers and model sizes, allowing the selection of classifiers based on the requirements in the problems. The performances of these four classifiers are different. The classifier D is the largest among them, but is with the highest accuracy. As shown in Table 1, each classifier is evaluated based on several indicators: accuracy (Acc), the number of parameters (|Param|), the number of floating-point operations (|FLOP|), and inference time (Inf. time) measured in ms. The results show that our pruned model significantly reduces |Param| and |FLOP|, with a reduction of more than 90% across all classifiers. This substantial reduction directly contributes to the reduction in computation demands, as reflected by a decrease in inference time ranging from 40% to 54%. While there is a decrease in accuracy, the accuracy drop is relatively small, which indicates that the model significantly maintains its accuracy, more than 90.00%, after pruning. In particular, the classifier D achieves a more than 90% reduction in both |Param| and |FLOP| with a 1.82% drop in accuracy only.

4.2 ResNet on CIFAR-10 and CIFAR-100

We also conducted our experiments on the ResNet-32 and ResNet-56 models [9] using the CIFAR-10 and CIFAR-100 datasets [19]. We trained our initial network using the hyperparameters specified in [16]. In Table 2, the experimental results demonstrate the efficiency of our approach. For CIFAR-10, our pruned models achieve substantial reduction in both |Param| and |FLOP| with acceptable loss in accuracy. As compared with [4], [5], the accuracies of the two pruned ResNet models are higher, and the reductions in |FLOP| are even more significant. For CIFAR-100, although the accuracy of our pruned ResNet-32 is slightly lower than that by [4], we achieve a reduction in |FLOP| by 49.37%, which is larger than 41.50% by [4]. For ResNet-56, our pruned model reduces |FLOP| by 62.23% with a 0.35% drop in accuracy only. The pruned accuracy is comparable to [5]. The significant reductions in both |Param| and |FLOP| highlight the potential of our approach in resource-constrained applications on mobile devices. Table 3 shows the results of applying different retraining approaches to ResNet models on CIFAR-10 and CIFAR-100 datasets. We evaluate the amounts of retraining epochs of our approach against LRW [32]. For CIFAR-10, the results show that our approach requires 75% epochs than LRW to achieve comparable results. Specifically, our approach achieves a 0.72% and 0.39% accuracy drop for ResNet-32 and ResNet-56, respectively, which is similar to LRW. For CIFAR-100 dataset, similar trends are observed, where our approach is consistently better than LRW in terms of maintaining accuracies. Furthermore, our approach only requires 150 epochs to achieve the similar accuracy, while LRW uses 200 epochs to recover the accuracy. This result demonstrates the effectiveness of our approach, which not only preserves the model accuracy but also accelerates the retraining process.

5 Conclusion

In this work, we propose an efficient approach to optimize CNN models for resource-constrained devices, combining GA with structured pruning and a retraining method RKD. Initially, we utilize GA to iteratively optimize the model, achieving a better pruning solution. The pruned model is then efficiently retrained with RKD, which decreases the amount of retraining epochs and accelerates the accuracy recovery. The experimental results show that our approach significantly reduces the amount of parameters and computation in the model while maintaining the accuracy of the model. Furthermore, our approach validates its performance on different neural networks and datasets, demonstrating the practicality of deploying CNN model on resource-constrained devices.

Table 1: Performance comparison of original and pruned models of the self-distilled R(2+1)D-18 model.

Model	Orignial Model				Ours							
	Acc (%)	|Param| (M)	|FLOP| (G)	Inf. time (ms)	Acc (%)	Acc↓ (%)	|Param| (M)	|Param|↓ (%)	|FLOP| (G)	|FLOP|↓ (%)	Inf. time (ms)	Inf. time ↓ (%)
A	85.09	1.17	29.28	2.4	77.12	7.97	0.11	90.22	2.66	90.92	1.1	54.17
B	91.34	2.71	34.32	2.8	89.01	2.33	0.25	90.71	3.11	90.94	1.4	50.00
C	92.12	8.73	38.35	3.2	90.14	1.98	0.79	90.98	3.47	90.96	1.9	40.63
D	92.24	31.34	40.52	3.6	90.42	**1.82**	2.81	**91.02**	3.66	**90.96**	2.1	41.67

Table 2: Comparison of pruned ResNet on CIFAR-10 and CIFAR-100.

Dataset	Model	Approach	Ori. Acc (%)	New Acc (%)	Acc↓ (%)	|Param|↓ (%)	|FLOP|↓ (%)
CIFAR-10	ResNet-32	[5]	92.63	91.02	1.61	-	41.50
		[4]	92.63	92.14	0.49	-	53.20
		Ours	92.89	**92.17**	0.72	50.43	**58.14**
	ResNet-56	[5]	93.59	91.67	1.92	-	54.60
		Ours	93.22	**92.83**	0.39	63.53	**63.85**
CIFAR-100	ResNet-32	[4]	70.28	69.44	0.84	-	41.50
		Ours	69.94	**68.88**	1.06	43.06	**49.37**
	ResNet-56	[5]	71.07	70.73	0.34	-	52.60
		Ours	71.04	**70.69**	0.35	59.93	**62.23**

Table 3: Comparison of ResNet-32, ResNet-56 using different retraining approaches.

Dataset	Model	Ori. Acc (%)	|Param|↓ (%)	|FLOP|↓ (%)	Approach	New Acc (%)	Acc↓ (%)	Epochs
CIFAR-10	ResNet-32	92.89	50.43	58.14	LRW[32]	92.02	0.87	200
					Ours	92.17	0.72	150
	ResNet-56	93.22	63.53	63.85	LRW[32]	92.88	0.34	200
					Ours	92.83	0.39	150
CIFAR-100	ResNet-32	69.94	43.06	49.37	LRW[32]	68.77	1.17	200
					Ours	**68.88**	1.06	150
	ResNet-56	71.04	59.93	62.23	LRW[32]	70.60	0.44	200
					Ours	**70.69**	0.35	150

References

[1] Christopher M Bishop and Nasser M Nasrabadi. 2006. *Pattern recognition and machine learning*. Vol. 4. Springer.

[2] Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag. 2020. What is the state of neural network pruning? *Proc. machine learning and systems* 2 (2020), 129–146.

[3] Congqi Cao, Yifan Zhang, Yi Wu, Hanqing Lu, and Jian Cheng. 2017. Egocentric gesture recognition using recurrent 3d convolutional neural networks with spatiotemporal transformer modules. In *Proc. ICCV*. 3763–3771.

[4] Hanjing Cheng, Zidong Wang, Lifeng Ma, Xiaohui Liu, and Zhihui Wei. 2021. Multi-task pruning via filter index sharing: A many-objective optimization approach. *Cognitive Computation* 13, 4 (2021), 1070–1084.

[5] Hanjing Cheng, Zidong Wang, Lifeng Ma, Zhihui Wei, Fawaz E Alsaadi, and Xiaohui Liu. 2023. Differentiable channel pruning guided via attention mechanism: a novel neural network pruning approach. *Complex & Intelligent Systems* 9, 5 (2023), 5611–5624.

[6] Gongfan Fang, Xinyin Ma, Mingli Song, Michael Bi Mi, and Xinchao Wang. 2023. Depgraph: Towards any structural pruning. In *Proc. CVPR*. 16091–16101.

[7] Song Han. 2017. *Efficient methods and hardware for deep learning*. Ph. D. Dissertation. Stanford University.

[8] Song Han, Jeff Pool, John Tran, and William J. Dally. 2015. Learning both Weights and Connections for Efficient Neural Networks. In *Proc. NeurIPS*.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proc. ICCV*. 770–778.

[10] Ruifei He, Shuyang Sun, Jihan Yang, Song Bai, and Xiaojuan Qi. 2022. Knowledge distillation as efficient pre-training: Faster convergence, higher data-efficiency, and better transferability. In *Proc. the IEEE/CVF conference on computer vision and pattern recognition*. 9161–9171.

[11] Yang He and Lingao Xiao. 2023. Structured pruning for deep convolutional neural networks: A survey. *IEEE transactions on pattern analysis and machine intelligence* (2023).

[12] Yihui He, Xiangyu Zhang, and Jian Sun. 2017. Channel pruning for accelerating very deep neural networks. In *Proc. CVPR*. 1389–1397.

[13] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. In *Proc. NeurIPS*.

[14] Mark Horowitz. 2014. 1.1 computing's energy problem (and what we can do about it). In *Proc. ISSCC*. 10–14.

[15] Chuan-Shun Huang, Wuqian Tang, Yung-Chih Chen, Yi-Ting Li, Shih-Chieh Chang, and Chun-Yao Wang. 2024. An Efficient Approach to Iterative Network Pruning. In *Proc. VLSI-TSA*.

[16] Y. Idelbayev. 2021. Proper ResNet implementation for CIFAR10/CIFAR100 in PyTorch. https://github.com/akamaster/pytorch_resnet_cifar10.

[17] Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani, et al. 2013. *An introduction to statistical learning*. Vol. 112. Springer.

[18] Onur Köksoy and Tankut Yalcinoz. 2008. Robust design using Pareto type optimization: a genetic algorithm with arithmetic crossover. *Computers & Industrial Engineering* 55, 1 (2008), 208–218.

[19] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images.

[20] Solomon Kullback and Richard A Leibler. 1951. On information and sufficiency. *The Annals of Mathematical Statistics* 22, 1 (1951), 79–86.

[21] Aakash Kumar, Ali Muhammad Shaikh, Yun Li, Hazrat Bilal, and Baoqun Yin. 2021. Pruning filters with L1-norm and capped L1-norm for CNN compression. *Applied Intelligence* 51 (2021), 1152–1160.

[22] Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. the IEEE* 86, 11 (1998), 2278–2324.

[23] Chun-Ting Lee, Yi-Ting Li, Yung-Chih Chen, and Chun-Yao Wang. 2023. Approximate logic synthesis by genetic algorithm with an error rate guarantee. In *Proc. ASPDAC*. 146–151.

[24] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. 2017. Learning efficient convolutional networks through network slimming. In *Proc. ICCV*. 2736–2744.

[25] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. 2018. Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270* (2018).

[26] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. 2017. Thinet: A filter level pruning method for deep neural network compression. In *Proc. ICCV*. 5058–5066.

[27] Microsoft Corporation. 2024. HoloLens. https://www.microsoft.com/en-us/hololens.

[28] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. 2016. Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440* (2016).

[29] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. In *Proc. NeurIPS*.

[30] Svetlana Pavlitska, Oliver Bagge, Federico Peccia, Toghrul Mammadov, and J Marius Zöllner. 2024. Iterative Filter Pruning for Concatenation-based CNN Architectures. *arXiv preprint arXiv:2405.03715* (2024).

[31] Russell Reed. 1993. Pruning algorithms-a survey. *IEEE transactions on Neural Networks* 4, 5 (1993), 740–747.

[32] Alex Renda, Jonathan Frankle, and Michael Carbin. 2020. Comparing Rewinding and Fine-tuning in Neural Network Pruning. In *Proc. ICLR*.

[33] Gilbert Syswerda et al. 1989. Uniform crossover in genetic algorithms.. In *Proc. ICGA*, Vol. 3.

[34] Wuqian Tang, Chuan-Shun Huang, Yung-Chih Chen, Yi-Ting Li, Shih-Chieh Chang, and Chun-Yao Wang. 2024. Model Reduction Using a Hybrid Approach of Genetic Algorithm and Rule-Based Method. In *Proc. IEEE 37th International System-on-Chip Conference*.

[35] Du Tran, Heng Wang, Lorenzo Torresani, Jamie Ray, Yann LeCun, and Manohar Paluri. 2018. A closer look at spatiotemporal convolutions for action recognition. In *Proc. ICCV*. 6450–6459.

[36] Zi Wang, Chengcheng Li, and Xiangyang Wang. 2021. Convolutional neural network pruning with structural redundancy reduction. In *Proc. the IEEE/CVF conference on computer vision and pattern recognition*. 14913–14922.

[37] Yifan Zhang, Congqi Cao, Jian Cheng, and Hanqing Lu. 2018. EgoGesture: A new dataset and benchmark for egocentric hand gesture recognition. *IEEE Transactions on Multimedia* 20, 5 (2018), 1038–1050.

[38] Michael Zhu and Suyog Gupta. 2017. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878* (2017).