mathematics

MDPI

Article

Rewarded Meta-Pruning: Meta Learning with Rewards for Channel Pruning

Athul Shibu, Abhishek Kumar, Heechul Jung and Dong-Gyu Lee *

Department of Artificial Intelligence, Kyungpook National University, Buk-gu, Daegu 41566, Republic of Korea; athulshibu@knu.ac.kr (A.S.)
* Correspondence: dglee@knu.ac.kr

Abstract: Convolutional neural networks (CNNs) have gained recognition for their remarkable performance across various tasks. However, the sheer number of parameters and the computational demands pose challenges, particularly on edge devices with limited processing power. In response to these challenges, this paper presents a novel approach aimed at enhancing the efficiency of deep learning models. Our method introduces the concept of accuracy and efficiency coefficients, offering a fine-grained control mechanism to balance the trade-off between network accuracy and computational efficiency. At our core is the Rewarded Meta-Pruning algorithm, guiding neural network training to generate pruned model weight configurations. The selection of this pruned model is based on approximations of the final model's parameters, and it is precisely controlled through a reward function. This reward function empowers us to tailor the optimization process, leading to more effective fine-tuning and improved model performance. Extensive experiments and evaluations underscore the superiority of our proposed method when compared to state-of-the-art techniques. We conducted rigorous pruning experiments on well-established architectures such as ResNet-50, MobileNetV1, and MobileNetV2. The results not only validate the efficacy of our approach but also highlight its potential to significantly advance the field of model compression and deployment on resource-constrained edge devices.

Keywords: convolutional neural networks; meta-pruning; ResNet-50; reward function; channel pruning

MSC: 68T07

Citation: Shibu, A.; Kumar, A.; Jung, H.; Lee, D.-G. Rewarded Meta-Pruning: Meta Learning with Rewards for Channel Pruning. *Mathematics* **2023**, *11*, 4849. https://doi.org/10.3390/math11234849

Academic Editor: Xinsong Yang

Received: 10 October 2023
Revised: 22 November 2023
Accepted: 27 November 2023
Published: 1 December 2023

1. Introduction

In recent years, convolutional neural networks (CNNs) have emerged as powerful tools, consistently delivering state-of-the-art performance across a spectrum of computer vision tasks [1–5]. However, harnessing the full potential of CNNs comes with two significant challenges. First, training these networks demands a substantial volume of labeled data, which can be both time-consuming and resource-intensive. Second, the computational demands for training large CNNs, especially when working with extensive datasets, impose a substantial burden on hardware resources. In response to these challenges, network pruning has gained prominence as a pivotal research area, offering solutions to streamline and expedite the deployment of large CNNs [6–9]. Network pruning techniques aim to reduce the complexity of these networks by identifying and eliminating redundant or less influential parameters, enabling more efficient models without compromising performance.

Network pruning is a critical aspect of model compression, addressing various challenges related to network structure [10], model continuity [11], and scalability [12]. In the pursuit of reducing the computational complexity and memory footprint of neural networks, there are primarily two prominent approaches: weight pruning [13–15] and channel pruning [16–19]. Weight pruning, a widely adopted technique, involves identifying and eliminating low-performing weights from the network [20]. This process entails the removal of weights with small magnitudes, a straightforward approach that is relatively

easy to implement [21]. We can effectively reduce the number of parameters by selectively pruning these connections, thus achieving model compression while preserving its overall structure and topology. Many existing deep learning frameworks often face limitations when it comes to accelerating computations involving sparse matrices. Achieving genuine compression and speedup in such scenarios often necessitates specialized software solutions [6] or dedicated hardware implementations [22] tailored to handle sparsity efficiently. Remarkably, the actual computational cost remains largely unaffected, regardless of the number of weights pruned.

Hence, an alternative approach, known as channel pruning, has gained favor in recent research [23,24]. This preference arises from the capability of channel pruning to remove entire filters, resulting in a model with structured sparsity [25]. Structured sparsity is a critical feature that enables the pruned model to fully harness the power of high-efficiency Basic Linear Algebra Subprogram (BLAS) libraries, thereby achieving more efficient acceleration [22]. This structural enhancement not only optimizes computational efficiency but also facilitates practical acceleration, making channel pruning an attractive choice for model optimization. However, the choice between weight and channel pruning is not always straightforward and often depends on the specific requirements and constraints of the task at hand. Weight pruning targets individual connections, making it well-suited for scenarios where fine-grained parameter reduction is essential. On the other hand, channel pruning operates at a coarser level by removing entire filters, offering a more substantial reduction in computational cost but potentially impacting model performance.

MetaPruning [26] stands as a noteworthy channel pruning approach, renowned for its ability to accelerate CNNs. Unlike traditional pruning methods that directly remove weights or filters from an existing network, MetaPruning introduces a distinctive approach. It focuses on generating optimized weight configurations for pruned network structures. The core idea behind MetaPruning lies in the evaluation of untrained models, using their performance to rank different network encoding vectors (NEVs). To achieve this ranking, MetaPruning leverages evolutionary algorithms inspired by the principles of natural evolution [27]. These algorithms search for the NEV that produces a model with the highest accuracy. However, MetaPruning does have a limitation; it operates within a predefined range of floating-point operations (FLOPs), restricting its ability to explore the trade-off between accuracy and computational efficiency comprehensively. Consequently, MetaPruning excels at selecting the highest accuracy within a specified FLOP range but may not fully explore the potential for striking an optimal balance between preserving accuracy and reducing FLOPs.

In this paper, we present Rewarded Meta-Pruning, an innovative approach to address the challenge of achieving high model accuracy while optimizing for network computational efficiency measured in FLOPs. Unlike traditional methods that focus solely on maximizing accuracy, Rewarded Meta-Pruning introduces a balanced objective. Our approach aims to find the highest achievable accuracy within a given FLOPs budget. In contrast to standard MetaPruning, where the reward is directly proportional to accuracy, our method rewards square of the accuracy. This adjustment results in more significant rewards for accuracy improvements, fostering rapid convergence toward higher accuracy models. Moreover, Rewarded Meta-Pruning offers enhanced control over the FLOPs of the final model. We achieve this by computing a score, referred to as the reward, which takes into account both accuracy and FLOPs. This score can be further customized to incorporate various parameters, allowing fine-tuning of the pruned model's metrics and the interaction between these parameters.

Our contribution lies in three folds:

- **Innovative Channel Pruning Method:** We introduce a novel channel pruning method, known as Rewarded Meta-Pruning. Unlike traditional pruning approaches, our method exhibits the unique capability to learn how to assign weights to pruned network channels dynamically. This adaptability enables more efficient network architectures and, consequently, superior model performance.

- **Exploring Reward Functions:** We delve into the intricacies of reward functions, emphasizing their pivotal role in channel pruning. Our research sheds light on the characteristics that define effective reward functions, providing valuable insights into the design of future pruning techniques. By doing so, we contribute not only a new method but also a deeper understanding of the underlying principles.
- **Empirical Validation:** To demonstrate the effectiveness of our proposed pruning method, we conduct a comprehensive set of experiments. These experiments involve popular pre-trained CNNs, including ResNet-50, MobileNetV1, and MobileNetV2. Our results unequivocally showcase the superiority of our method over existing techniques, underlining its practical relevance and potential impact on the field of deep learning.

2. Related Works

The Lottery Ticket Hypothesis, as introduced by Frankle and Carbin [13], suggests the existence of subnetworks within a randomly initialized dense neural network. These subnetworks, when trained independently, can achieve comparable or superior performance to the original unpruned network. This intriguing hypothesis has spurred significant research, leading to the development of various methods aimed at identifying these 'winning tickets' or 'lottery tickets'. These methods efficiently pinpoint critical subnetworks within large neural architectures, laying the foundation for improved model compression and understanding the inner workings of deep learning models.

Unstructured network pruning: Weight pruning methods play a pivotal role in model compression. Random pruning methods, such as the Lottery Ticket Hypothesis [13], sanity pruning [14], and randomized pruning strategies [15], rely on random selection and elimination of network parameters. Another category focuses on weight norms, with methods like L1 and L2 norms [28] and the geometric median of weights [29]. Beyond norm-based techniques, Molchanov et al. [30] use complex Taylor series expansions, providing an alternative perspective on weight importance. Other methods employ KL-divergence importance [31] or empirical sensitivity analysis [32]. Our work draws inspiration from and builds upon these methods to develop a novel weight pruning approach.

Structured network pruning: AutoPruner [33] integrates filter selection into model training, automatically identifying and selecting unimportant filters. Sparse Structure Selection (SSS) [34] uses scaling factors to create compact models. Discriminative-aware channel pruning (DCP) [35] systematically prunes channels with genuine discriminative power, and Adaptive DCP [17] enhances pruning effectiveness with discriminative-aware loss functions. AutoML for Model Compression (AMC) [36] employs reinforcement learning for automated model compression, while HRank [37] uses feature map ranks for filter pruning. Inspired by the Lottery Ticket Hypothesis, Ye et al. [38] use a greedy search approach to identify efficient network substructures. Pruning algorithms rooted in Hebbian theory, like Fire Together Wire Together (FTWT) [18], offer a biologically inspired approach to model compression.

Meta-learning: Meta-learning involves learning algorithms from other learning algorithms [26,39,40]. It comprises three paradigms: the meta-optimizer pertains to the optimizer used in the outer loop, determining how the model adapts over time. The meta-representation focuses on acquiring and refining meta-knowledge, capturing patterns for effective generalization. The 'meta-objective' represents the ultimate task that the meta-learner aims to achieve. These paradigms form the foundation of meta-learning, offering a versatile framework for developing adaptive algorithms.

Learning to prune filters: Reinforcement Learning (RL) algorithms for network architecture optimization, such as the Try-and-Learn algorithm [6], compute rewards for individual filters, ranking and pruning them within a baseline network while maintaining desired performance. This algorithm, using the reinforce algorithm [41], demonstrates automated discovery and elimination of redundant filters, showcasing a promising avenue for efficient neural network architectures.

Neural architecture searching: Methods for optimizing neural network architectures fall into categories like RL-based approaches [42], genetic algorithms [43,44], gradient-based methods [45], parameter sharing [46], and weight prediction [47]. Each category offers unique perspectives and techniques to address the complex task of architecture optimization, highlighting the diverse landscape of this field.

3. Rewarded Meta-Pruning

The Rewarded Meta-Pruning algorithm introduces a novel approach to model compression by leveraging a reward coefficient to dynamically balance the trade-off between model accuracy and efficiency. Unlike traditional methods that search for the model with the highest accuracy within a preset range of FLOPs, our approach allows the network to autonomously learn the optimal model weights and sizes. This autonomous learning process eliminates the possibility of human error in manually setting these parameters. The primary objective of our algorithm is to maximize a reward metric, which serves as a comprehensive measure of model performance. This reward metric is designed to be positively correlated with model accuracy while negatively correlated with its computational cost (FLOPs). By optimizing this reward, our algorithm strikes an effective balance between accuracy and efficiency. Crucially, the efficiency of the pruned network hinges on the viability of the reward function, which, in turn, depends on the robustness of hyperparameters. This interplay between hyperparameters and reward function is a fundamental aspect of our approach, ensuring that the pruned models are not only accurate but also computationally efficient.

Our method unfolds in three distinct phases: training, searching, and retraining, as outlined in Algorithm 1. The first phase of the algorithm is to train a network to generate weights for a pruned model. This is done by first creating a random NEV, which is a vector of channel numbers in each layer of the network. The NEV is then used to create a pruned network, which is a network with the specified number of channels in each layer. The weight matrix of the pruned network is then generated using the PruningNet block. Finally, the pruned network is trained on a batch of input images, and the loss is calculated. The gradient of the loss is then used to update the NEV. The second phase of the algorithm is to search for the optimal NEV. This is done by using an evolutionary algorithm, which is a type of algorithm that simulates natural selection to find the best solution to a problem. The evolutionary algorithm used in Rewarded Meta-Pruning is a genetic algorithm, which uses mutation and crossover to generate new NEVs. The new NEVs are then evaluated using the reward function, which is a function that measures the trade-off between the accuracy of the network and its computing efficiency. The Top-k NEVs are then selected for the next generation of the algorithm. The third phase of the algorithm is to fine-tune the pruned network that was generated in the searching phase. This is done by training the pruned network on the entire training dataset for a fixed number of epochs. The fine-tuning process helps to improve the accuracy of the pruned network without significantly increasing its computational complexity. Through these phases, our algorithm not only reduces model complexity but also enhances its performance, making it a valuable tool in the area of model compression and optimization.

Algorithm 1 Algorithm of Rewarded Meta-Pruning

Hyperparameters: Number of training epochs *max_training*, Number of searching epochs *max_iter*, Number of fine-tuning epochs *max_tuning*
Input: *dataset*: training images that can be split into *batches*, r_i: Random integer indexed at i, w_i: Random weights indexed at i, ∇: Gradient of loss of given model
Functions: $norm(nev)$ converts *nev* to weights, $create(weights)$ creates model using *weights*, $f(model, data)$ trains model using given data, $reward(nev)$ computes reward of model created using *nev*, *mutation* and *crossover* performs evolutionary operations on list of *nev*s
Output: Pruned and trained model x

```
for i = 0 to max_training do
    for each batch in dataset do
        nev = [r_1,r_2,...,r_n]
        {w_1,w_2, ..., w_n} = norm(nev)
        x = create({w_1,w_2, ..., w_n})
        L = f(x,batch)
        x += ∇L
    end for
end for
candidate = List of n random nevs
for i = 0 to max_iter do
    rewards = [r_1,r_2,...,r_n]
    for j = 0 to n do
        r_j = reward(nev_j)
    end for
    sort candidate in descending order of rewards
    mutated = mutation(candidate[: 10])
    crossed_over = crossover(candidate[: 10])
    candidate = mutated + crossed_over
end for
{w_1,w_2, ..., w_n} = norm(candidate[0])
x = create({w_1,w_2, ..., w_n})
for i = 0 to max_tuning do
    x += ∇f(x,dataset)
end for
```

3.1. Training

Many state-of-the-art CNNs [48–50] commonly employ three primary types of layers: convolution blocks, bottlenecks, and linear layers. These architectural components collectively define the network's depth and complexity. One pivotal factor influencing their behavior is the channel scale, which determines the size of each layer. In our proposed algorithm, we introduce a novel approach to channel scale optimization. Specifically, we investigate the impact of batch normalization across different layer sizes, ranging from 10% to 100% of the size of the original architecture. By systematically exploring this spectrum of channel scales, we gain insights into how variations in layer size affect network performance and efficiency. This investigation is particularly crucial for the initial convolutional block and each type of bottleneck, as these components play a pivotal role in shaping the network's feature extraction capabilities and computational efficiency. Understanding the interplay between channel scale and layer type enables us to make informed design choices, ultimately enhancing the adaptability and effectiveness of our proposed algorithm.

Each model within our proposed algorithm is defined by a NEV, which serves as a crucial parameter for guiding the pruning process. This NEV is essentially a list of random numbers, and each number corresponds to a scale among the 31 available scales. Additionally, these NEV numbers correspond to specific Batch Normalization weights associated with each layer. The NEV also plays a role in creating linear layers at the designated

scales. To further elucidate this process, the NEV is passed through two Fully Connected (FC) layers, as illustrated in Figure 1. These FC layers are responsible for generating the weight matrix used in the model. Notably, the weights of these FC layers are trained using gradients derived from the generated weights, with respect to the weights of the FC layers themselves. This training process ensures that the model's weights are optimized to fit the specific NEV configuration. It is important to highlight that the generated weights are not arbitrary; they are uniquely mapped for each combination of output filter sizes. This uniqueness is a result of the injective nature of the function responsible for converting an NEV into weights. Consequently, for every epoch, a random NEV generates a distinct model with weights that correspond one-to-one to the NEV's sample space. To provide further context, each element C_i within the NEV C corresponds to the output channels of layer i. This mapping ensures that the model is tailored to the specific characteristics defined by the NEV, allowing for adaptability and customization during the training process. Significantly, the training spans over 1000 epochs, with each epoch featuring a unique NEV, thereby incorporating the diverse spectrum of configurations employed during the training process. This extensive training process ensures that the model is exposed to a wide range of configurations, leading to a more robust and generalizable training.

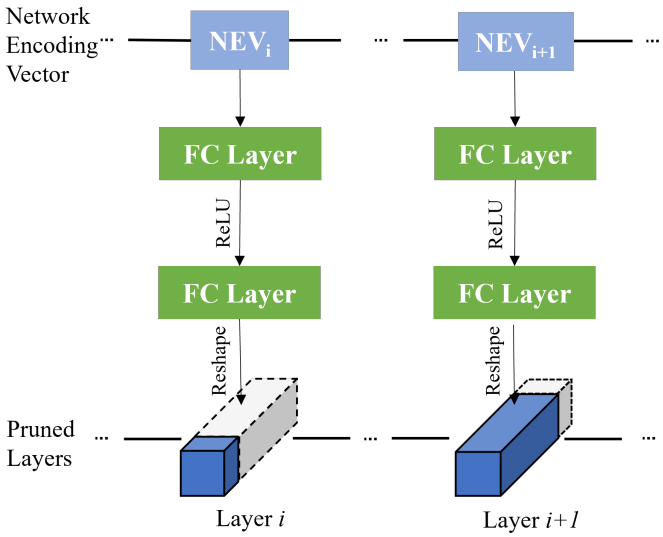

Figure 1. An illustration of stochastic training method. Weights of each layer are generated by the fully-connected layers, and the size of the layer is decided by the corresponding network encoding vector.

After the generation of a model, our approach initiates a training process for each batch of the training dataset using these initial weights. During this training phase, we employ the cross-entropy loss function to compute the gradients and update the model's weights iteratively. It is important to note that each model generated by our algorithm is essentially a slice or subnetwork of the complete model, and this slice is defined by the NEV that characterizes the architecture and connectivity of the subnetwork. Traditionally, validation data is employed to assess and validate the performance of a model. However, in our approach, we adopt a unique strategy. Instead of using the validation data for validation purposes, it is primarily leveraged to measure the progress of our algorithm. This distinctive approach allows us to continually refine and optimize the NEV and the associated subnetworks throughout the training process. This novel paradigm not only contributes to efficient model exploration but also aligns with our overarching objective of evolving models that are tailored to the dataset and the task at hand. By focusing on progress measurement rather than conventional validation, we can adaptively adjust the NEV and subnetworks, ultimately leading to improved model performance.

3.2. Searching

The proposed algorithm utilizes the NEV candidates to construct models by utilizing the weights obtained from the trained model. This process involves converting each NEV into a pruned model, followed by the computation of reward metric (defined in Section 3.2.2) for these models. To identify the optimal pruned model, we employ an evolutionary search algorithm inspired by the work of Mallipeddi et al. [51]. Initially, the weights used for model creation are derived from the pre-trained model. However, to eliminate any potential bias introduced by the pre-trained weights, the final model undergoes a comprehensive training process from scratch. This step ensures that the pruned model is fine-tuned to suit the specific requirements of the task at hand, promoting unbiased and optimal performance.

3.2.1. Creating Genes

In our proposed algorithm, we initiate the evolutionary search by generating a set of random candidates, referred to as 'NEVs', which serve as the starting point for the optimization process. Each NEV is essentially a gene, represented as a list of sizes that correspond to the architecture of the model. Within these lists, values are assigned to represent the weights obtained from a predefined dictionary. However, it is important to note that the performance metrics of the models created using these NEVs are inherently random due to the stochastic nature of the initialization. To exert some control over the final model's quality, we introduce arbitrary hyperparameters that help guide the optimization process. These hyperparameters play a crucial role in shaping the architecture and behavior of the evolving models. To streamline the optimization and reduce overall computation time, we incorporate an efficient strategy. Specifically, we store the FLOPs associated with each gene as the last element of the NEV. This approach allows us to quickly estimate the computational cost of each candidate without performing full model evaluations, thereby expediting the search process. As the optimization progresses, the FLOPs information is later replaced by a reward value, which serves as a more meaningful and informative metric to assess the quality of each candidate. This reward reflects the model's performance on the task at hand and guides the evolutionary search towards solutions that exhibit superior capabilities. By incorporating these elements into our algorithm, we strike a balance between randomness and control, enabling efficient exploration of the solution space and convergence towards models of higher quality.

3.2.2. Reward and Selection of NEVs

Our proposed algorithm incorporates a ranking mechanism that evaluates candidate pruned networks after each epoch using a reward function. This reward function is calculated as the weighted product of accuracy and efficiency coefficients, ensuring a balance between performance and resource utilization. The reward ($R(G_i)$) for a candidate gene (G_i) (as shown in Figure 2) is calculated using the following formula:

$$R(G_i) = \alpha(G_i, b_a) \times \psi(G_i, b_f). \tag{1}$$

The accuracy and efficiency coefficients, denoted by α and ψ , are defined as:

$$\alpha(G_i, b_a) = \left(\frac{b_a}{b_a - A(G_i)}\right)^2, \tag{2}$$

$$\psi(G_i, b_f) = \log\left(\frac{b_f}{F(G_i)}\right), \tag{3}$$

where G_i denotes a gene of index i from all candidate genes G, and A and F, represent functions that return the accuracy and FLOPs of the model created using the gene passed into them. The accuracy of a pruned network is measured using its performance on a validation dataset, while its efficiency is assessed based on the number of remaining

parameters or FLOPs. By considering both accuracy and efficiency, the reward metric guides the meta-learning process toward identifying optimal pruned networks that prioritize both performance and resource utilization.

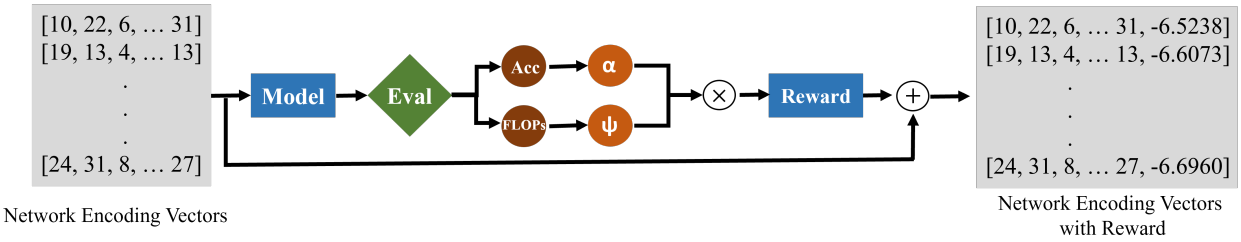

Figure 2. Computing reward for network encoding vectors. Each network encoding vector is used to generate a model using the fully-connected layers trained during the training process. The accuracy and FLOPs of these models are computed and used to compute the reward. The rewards corresponding to each network encoding vector is the appended to it and stored for future use.

The accuracy coefficient (α) exhibits an exponential increase in value as the model's accuracy improves. However, a key characteristic of α is that it approaches infinity as it approaches the base accuracy (b_a). In our approach, where the model is not fine-tuned, the accuracy typically does not converge closely enough to b_a. Consequently, this can limit the reward function's ability to reach high levels due to extremely large values of α. To address this limitation and further enhance the new model's performance, knowledge distillation techniques can be employed. This effectively sets the base accuracy to the new model's accuracy, eliminating any adverse effects stemming from Equation (2)'s asymptotic behavior. This adjustment ensures a more meaningful and tractable reward metric for guiding the pruning process. Conversely, the efficiency coefficient (ψ) demonstrates a linear decrease in value as the FLOPs increase. Notably, ψ is bounded by the FLOPs of the original, unpruned model (b_f). As a result, ψ provides a measure of pruning efficiency while remaining constrained by the base model's computational resources. Given that pruning rates are inversely proportional to FLOPs, a lower efficiency coefficient generally corresponds to a higher prune rate. In this context, the reward function exhibits a direct relationship with both accuracy and prune rate, highlighting their significance in guiding the optimization process. Therefore, the Reward metric, computed for each NEV, ranges from 0 to infinity. Specifically, the reward metric is zero when the model's FLOPs are equal to the base FLOPs (b_f), and it approaches infinity when the accuracy of the pruned model matches the base accuracy (b_a).

The proposed algorithm strives to achieve a delicate balance between accuracy and efficiency, ensuring that high accuracy is not attained at the expense of low pruning rates in the final model. This balance is achieved through a carefully designed reward computation mechanism that directly links accuracy to the reward and indirectly incorporates the influence of efficiency. The computed reward is stored as the latest element of the NEV, enabling efficient ranking of each NEV based on its performance. To effectively guide the evolutionary process, a strategy of tracking the Top-50 NEVs from each epoch is employed. From this pool, the ten best-performing NEVs are selected for further genetic operations, including mutation and crossover, to generate candidate genes for the subsequent epoch. This selection strategy ensures that both genetic diversity and the quality of candidate solutions are carefully maintained and refined throughout the optimization process. Through this iterative approach of balancing accuracy and efficiency, the algorithm achieves high accuracy while maintaining competitive pruning rates in the final model, demonstrating its effectiveness in optimizing deep neural networks for various applications.

3.2.3. Mutation and Crossover

In our proposed algorithm, the evolutionary process involves selecting the best candidates from each epoch and subjecting them to mutation and crossover operations to

create candidates for the subsequent epoch. Mutation entails making subtle alterations to a gene by randomly modifying a fraction of its elements. Specifically, there is a 10% chance for each element in a gene to be replaced with a randomly chosen valid element. On the other hand, crossover involves combining two randomly selected genes to generate a novel gene. For each index in the gene, an element is chosen randomly from either of the two parent genes. A key observation in our approach is that channel configurations within a local region of the configuration space often exhibit similar performance metrics, as supported by prior work [52]. Consequently, the new candidates produced through mutation and crossover tend to share similar accuracy and computational complexities (FLOPs). As a result, the reward, at the very least for the best candidate, typically remains stable or experiences slight improvements over successive epochs. However, we have incorporated a mechanism to detect when the reward has plateaued for an extended period, signaling a potential entrapment in a local minimum.

The mutation operator is a pivotal component in evolutionary algorithms, playing a crucial role in the refinement of individual members within the population. Its primary objective is to enable localized exploration, thereby enhancing promising candidate individuals. This intricate process involves nuanced modifications to a candidate individual, resulting in a trial individual that exhibits subtle deviations from its parent counterpart. Through mutation, the algorithm systematically explores the solution space by introducing controlled variations, promoting diversity, and facilitating the discovery of potentially superior solutions. In this study, we leverage the *uniform mutation* technique to instill diversity among the Top-k individuals. Over a predefined number of iterations (e.g., 2.5k times, where k denotes the size of the Top-k population), we randomly select an individual ind from the Top-k population, ensuring equal opportunity for mutation across the population. For each variable in the selected individual ind, a random number u is generated from a uniform distribution in the range $[0, 1]$, representing the probability of mutating the current variable. This u is then compared to the mutation probability p_m. If $u \leq p_m$, the gene undergoes mutation. If the variable is chosen for mutation, a random integer δ is generated from a uniform distribution within the search bounds, representing the magnitude of the mutation. The selected variable is updated by replacing it with δ. Otherwise, it is copied as is from the original individual ind. Following the iteration through all individuals and their variables, the algorithm produces the mutated population $\mathbf{P}_m$, where some genes have undergone mutation based on the mutation probability p_m. This meticulous process ensures the maintenance of diversity within the population while preserving key characteristics of the top individuals.

Crossover, a pivotal operator in evolutionary algorithms, amalgamates the solution components of two or more candidate individuals, yielding novel offspring. Analogous to the mutation operator's adherence to the fundamental representation of an individual, the crossover operator requires seamless integration with the underlying solution structure. This procedural synergy ensures the retention of crucial features and attributes within the solution makeup of the population, fostering diversity and aiding convergence in the evolutionary search. The optimization framework presented here employs the *uniform crossover* operator, uniformly blending solution components from two-parent individuals selected from the Top-k population. This process iterates for a predetermined number of times, typically set to 2.5k iterations, where k represents the size of the top-k population, facilitating effective exploration of the search space. In each iteration, two individuals denoted as ind_1 and ind_2 are randomly selected from the Top-k and Top-5k populations, respectively, ensuring individuals across the population have an opportunity for crossover. For each variable within the selected individuals ind_1 and ind_2, we generate a random number u from a uniform distribution in the range $[0, 1]$, representing the probability of assigning a variable from individual ind_1 during the crossover operation. If $u \leq c_r$, the variable from ind_1 is selected and assigned to the corresponding position in the offspring population $\mathbf{P}_c(i, j)$. Otherwise, the variable from ind_2 is chosen and assigned to the corresponding position in the offspring population $\mathbf{P}_c(i, j)$. This constitutes a uniform

crossover operation, where variables from both parents are considered, and the choice is made probabilistically based on c_r. Upon completing the specified number of crossover iterations, the algorithm yields the resulting crossovered population $\mathbf{P}_c$. This process refines and enhances Top-k by generating offspring inheriting favorable traits, facilitating faster convergence to high-quality solutions.

Given that evolutionary search operates within a high-dimensional non-convex space, it is worth noting that critical points with significantly higher errors than the global minimum are more likely to be saddle points [53]. In simpler terms, the local minima encountered during the search are often close enough to the global minimum to yield satisfactory solutions. Consequently, we employ a termination criterion that allows the search to conclude when it is unlikely to substantially improve results. While increasing mutation and crossover rates might generate more diverse candidates, doing so could jeopardize the integrity of the evolutionary search process. In cases where mutation and crossover operations fail to produce a sufficient number of new candidates, the remaining candidates are created using entirely random genes, ensuring diversity within the candidate pool.

3.3. Retraining

Upon the completion of the evolutionary search phase, our algorithm identifies the best-performing gene, which exhibits the highest reward after undergoing numerous epochs of genetic searching. This top-ranked gene is selected as the initial candidate for model creation. In the majority of pruning algorithms, a pruned model undergoes a brief fine-tuning process to recover any accuracy that might have been lost during pruning, a technique well-documented in the literature [21]. However, in the case of the Rewarded Meta-Pruning algorithm, we adopt a distinctive approach. Instead of using the best Normalized Evolution Vector (NEV) solely for pruning, we leverage it as the foundation for creating an entirely new model. This model is then subjected to a comprehensive training regimen, starting from scratch. This strategy results in a notable advantage: the accuracy of our model tends to stabilize at a higher epoch during the fine-tuning process compared to conventional methods.

The rationale behind this approach is twofold. First, it ensures that the pruned model starts with a solid foundation derived from the best NEV, which can significantly reduce the need for extensive fine-tuning epochs. Second, by training the model from scratch, we allow it to adapt more effectively to the new architecture, fine-tuning its parameters in a manner that takes full advantage of the evolved structure. This innovative process contributes to the improved performance and efficiency of the Rewarded Meta-Pruning algorithm.

4. Experimental Results

In this section, we demonstrate the superiority of the Rewarded Meta-Pruning method. We compare the results obtained with other methods pruning three major networks. Lastly, we discuss the impact of various hyperparameters to understand its impact on the proposed method.

4.1. Experimental Setting

The ResNet-50 [48] network is trained for 32 epochs, while MobileNetV1 [49] and MobileNetV2 [50] are trained for 64 epochs. ResNet and MobileNetV2 are retrained after searching for 400 epochs, but MobileNetV1 only requires 320 epochs. Searching for the NEVs takes 20 epochs for all the networks. Each epoch searches 50 NEVs, searching through 1000 unique NEVs throughout the run. We employ the SGD optimizer in all datasets, where MobileNetV1 and MobileNetV2 both use the Lambda scheduler to decay the models by a γ of 0.1 every epoch from an initial learning rate of 0.2. The scheduler for ResNet, however, decreases by a factor of 0.1 at epochs 80 and 160. The pruning network of MobileNetV2 is slightly different from that used for ResNet-50 and MobileNetV1 to fit its inverted residual bottlenecks. For crossover and mutation, we set the crossover rate to 0.5 and the mutation probability to 0.1.

The experiments are conducted on three commonly used networks including ResNet-50, MobileNetV1, and MobileNetV2. The networks are trained using ImageNet [54] from scratch as described in the previous section. ImageNet consists of 1.2 M training images and 50K validation images. It also consists of 100K test images, but since the labels of the test data are not released, validation data is used for testing. The validation data have not been used in any part of the training process except to compute validation accuracy after each epoch. As a natural consequence of using evolutionary computation, Rewarded Meta-Pruning is resource-heavy in computing the pruned networks, due to the large number of mutations tested, and the corresponding models generated. But this cost is balanced out by the efficiency and accuracy of the models generated. A network need not be created each time it needs to be deployed because the knowledge distilled from it can be transferred and used in varying contexts.

4.2. Evaluation Protocol

Three metrics are used for evaluating the pruning algorithms: Top-1 and Top-5 errors and FLOPs. Accuracy is the percentage of validation data identified correctly compared to the whole dataset. Top-1 error is the inverse of accuracy, i.e., the proportion of images in which the predicted labels of the highest probability are wrong. The Top-5 error is the proportion of images with the correct label absent in the five highest probabilities of predicted labels. FLOPs measure the number of floating-point operations computed per second, reflecting the computational efficiency.

4.3. Performance on ResNet-50

ResNet-50 is a CNN with a depth of 50 layers. It was created to solve degradation in the model as deeper layers are stacked [48]. ResNet uses skip connections to identify mapping. This adds the features with their original parameters before passing them into the next layer. Identity mapping followed by linear projection is used to expand channels of the features to make it possible to be added with the original parameters.

Table 1 shows the results of ResNet-50 trained using ImageNet-2012 after pruning with Rewarded Meta-Pruning and other competing methods. It can be inferred that this method has a lower error rate than every method, and this is achieved while keeping the FLOPs relatively low. The FLOPs, as compared to the baseline network [52], reduced by 52.55%, but the error has only increased by 0.84%. When compared with standard random pruning, for a similar reduction in FLOPs, there is a 0.63% lower error. The MetaPruning [26] method shows a 0.36% higher error while still using 1.34% higher FLOPs as compared to the baseline model. Adapt-DCP [17] has the closest reduction in FLOPs as compared to the baseline, but this method has a 0.61% lower error. SSS [34] and HRank [37] methods have very similar prune rates to the Rewarded Meta-Pruning, but have higher FLOPs by around 9%, and higher error by 3.94% and 0.78%, respectively.

Table 1. Benchmarking state-of-the-art channel pruning methods with ResNet-50.

Method	Top-1 Error	Top-5 Error	FLOPs
Baseline [48]	23.40%	-	4110 M
SSS [34]	28.18%	9.21%	2341 M
GAL-0.5 [55]	28.05%	9.06%	2341 M
AutoPruner [33]	25.24%	7.85%	2005 M
HRank [37]	25.02%	7.67%	2311 M
Random Pruning [52]	24.87%	7.48%	2013 M
Adapt-DCP [17]	24.85%	7.70%	1955 M
ABCPruner [56]	25.16%	-	2568 M
WhiteBox [57]	24.68%	7.57%	2228 M
MFP [25]	24.33%	-	2376 M
CLR-RNF [58]	25.15%	7.69%	2458 M
MetaPruning [26]	24.60%	-	2005 M
Rewarded Meta-Pruning	**24.24%**	**7.35%**	**1950 M**

4.4. Performance on MobileNetV2

MobileNetV2 contains depth-wise and point-wise convolution. It has an inverted residual with a linear bottleneck, which takes a low dimensional compressed representation as input and expands it to a higher dimension, then filters them with light-weight depthwise convolutions like in MobileNetV1 [50]. MobileNetV2 is an efficient network with a relatively low error and a wide variety of applications. Thus, the demonstration on MobileNetV2 is an effective way to show the performance of the pruning algorithm.

Table 2 compares the performance of the Rewarded Meta-Pruning method with the state-of-the-art methods. The Rewarded Meta-Pruning method has a lower FLOPs than any other methods while showing only 0.39% higher error than the baseline. 0.75 MobileNetV2, which is MobileNetV2 with a width 25% lower than the original, has a 1.69% higher error than this method. When compared to random pruning, which is the baseline for all pruning methods [59], this method has a 0.59% lower error. MetaPruning [26] has a 0.29% higher error despite having a 7.64% higher FLOPs. AMC [36] and Adapt-DCP [17] have 0.69% and 0.04% higher errors and FLOPs. Rewarded Meta-Pruning also outperforms Greedy selection [38] by 0.29% despite an almost similar amount of FLOPs.

Table 2. Benchmarking state-of-the-art channel pruning methods with MobileNetV2.

Method	Top-1 Error	FLOPs
Baseline [50]	28.00%	314 M
0.75 MobileNetV2 [52]	30.20%	220 M
AMC [36]	29.20%	220 M
Random Pruning [18]	29.10%	223 M
MetaPruning [26]	28.80%	227 M
Greedy Selection [38]	28.80%	201 M
Adapt-DCP [17]	28.55%	216 M
Rewarded Meta-Pruning	**28.51%**	**199 M**

4.5. Performance on MobileNetV1

MobileNetV1 has a streamlined architecture that builds lightweight deep networks using depth-wise separable convolutions. All fully connected layers are followed by batch normalization and ReLu, except the final layer, which is followed by a softmax layer for classification [49]. Unlike ResNet-50 or MobileNetV2, MobileNetV1 does not have skip connections, instead utilizing manually set global parameters to control the latency and accuracy of the final model.

In Table 3, we compare the Rewarded Meta-Pruning method with other competing pruning techniques to prune a MobileNetV1 model. It has almost regained the accuracy of the baseline network [49], with 0.2% lower accuracy and 48.15% lower FLOPs. This method clearly achieves superior results when compared to Fire-Together-Wire-Together [18] pruning method, with a 0.94% lower error and 7.03% lower FLOPs. It outperforms 0.75 MobileNet-224 [49], which is MobileNetV1 with 25% lower width, by 2% while using 5.27% lower FLOPs. While MetaPruning [26] shows a lower error than even the baseline network, our method has a lower FLOPs. However, the size of the pruned network is still 9.83% larger when compared to the proposed method. This could be due to the lack of shortcut connection in MobileNetV1 in spite of being a smaller network, leading to a large number of fully-connected layers. In terms of performance achieved for every resource, Rewarded Meta-Pruning edges out MetaPruning. It is fair to assume that the Rewarded Meta-Pruning method could have better results with more advanced reward functions. This will be validated by further research on the robustness of various hyperparameters.

Table 3. Benchmarking state-of-the-art channel pruning methods with MobileNetV1.

METHOD	TOP-1 ERROR	FLOPS
BASELINE [49]	29.40%	569 M
0.75 MOBILENET-224 [52]	31.60%	325 M
FTWT (R = 1.0) [18]	30.34%	335 M
METAPRUNING [26]	**29.10**%	324 M
REWARDED META-PRUNING	29.60%	**295 M**

4.6. Discussion

Analyzing the results, it becomes evident that the effectiveness of our proposed method is closely intertwined with the choice of reward functions. Currently, the Rewarded Meta-Pruning algorithm employs a reward function that positively correlates to model accuracy while inversely correlating with FLOPs. This design choice aims to strike a balance between achieving high accuracy and optimizing computational efficiency.

One noteworthy aspect of our reward function is its response to the accuracy of the pruned model as it approaches the baseline accuracy. In our approach, the reward increases exponentially as the pruned model's accuracy nears that of the baseline. This stands in contrast to the reward function employed in MetaPruning [26], where the reward is directly proportional to accuracy. However, it's important to note that a reward based solely on accuracy could inadvertently incentivize the generation of pruned models with unnecessarily high FLOPs. This phenomenon is exemplified in MetaPruning, where pruned models occasionally exhibit FLOPs as high as the preset maximum threshold allows, potentially negating the intended computational optimization benefits. To address this, we introduce an efficiency coefficient within our reward function, offering a nuanced control mechanism. This coefficient acts as a safeguard against the pursuit of higher rewards solely based on accuracy, ultimately guiding the algorithm towards achieving both accuracy and computational efficiency concurrently.

Our proposed method unveils a compelling interplay between the reward function and the FLOPs associated with different model configurations. As we increase the FLOPs, we observe a gradual decrease in the reward, primarily attributed to the inverse correlation between the efficiency coefficient, a crucial component of our reward function, and the FLOPs. This behavior is pivotal as it effectively controls the reward's growth rate. However, striking a balance is paramount. If the proportionality between FLOPs and the efficiency coefficient is too wide, it can excessively throttle the reward. We introduce a linearly decreasing efficiency coefficient to address this, which helps maintain a more balanced relationship between FLOPs and the reward.

Our reward function, defined in Equation (1), underscores the significance of achieving high accuracy and ensuring that FLOPs remain within acceptable bounds for model selection. As a model's accuracy improves, its probability of being selected naturally increases. However, this selection preference is conditional on the FLOPs also remaining at a sufficiently low level. This observation is evident when analyzing the gradient of the 'Green' component in Figure 3, where we observe an increase in reward as FLOPs decrease.

Nevertheless, it is crucial to note that achieving high accuracy alone is insufficient for a model to be deemed acceptable by our method. This is reflected in the distribution of the 'Red' component in Figure 3, which indicates that a model's reward increases as FLOPs decrease, but this increase is not considered acceptable until the accuracy reaches a certain threshold. Our Rewarded Meta-Pruning method emphasizes prioritizing accuracy while maintaining a fine-tuned balance with computational efficiency, ensuring that selected models strike an optimal trade-off between high performance and resource usage.

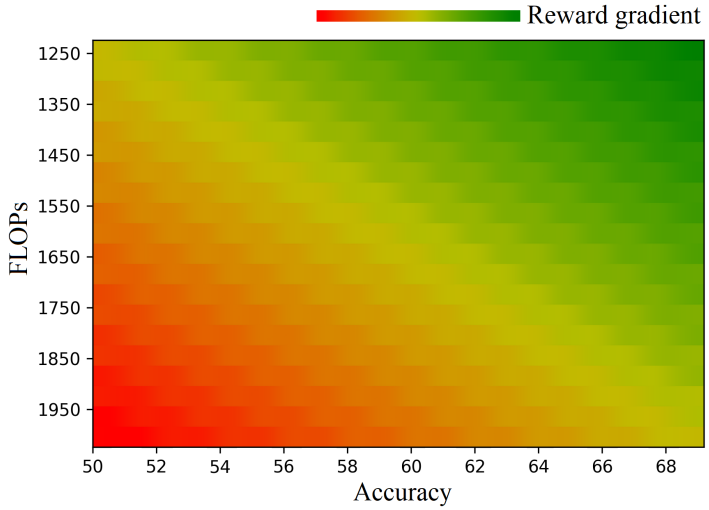

Figure 3. Reward computed at varying accuracy and FLOPs values.

This intricate interplay between accuracy and computational efficiency highlights the viability of our reward function in guiding the pruning process towards optimal network configurations. The reward function effectively balances the pursuit of high accuracy with the need for maintaining computational efficiency, ensuring that the selected pruned models achieve superior performance and adhere to realistic resource constraints. This demonstrates the practical applicability of our approach to real-world scenarios where both accuracy and efficiency are critical considerations.

When we compare the Rewarded Meta-Pruning method to MetaPruning, we observe a distinct difference in the reward evolution during each iteration of the search process. This discrepancy arises from the underlying reward mechanisms of the two methods. In MetaPruning, the reward is directly proportional to accuracy, a commonly used metric for model evaluation. In contrast, Rewarded Meta-Pruning introduces a novel reward calculation that is proportional to the square of accuracy. This distinction becomes evident when we examine Figure 4, which presents a visual representation of the rewards, accuracies, and computational costs (FLOPs) of the best models after each iteration of the search. Initially, both methods begin with randomly chosen models, a fundamental aspect of evolutionary searching. However, as the search progresses, notable differences emerge.

In the case of MetaPruning, we observe a trend where the FLOPs of the best model tend to increase for the most part. This can be attributed to the direct relationship between reward and accuracy, often leading to selecting models with higher computational requirements. On the other hand, the Rewarded Meta-Pruning method demonstrates a tendency to maintain lower FLOPs throughout the search process, thanks to the squared accuracy-based reward. Furthermore, while both methods exhibit accuracy improvements, MetaPruning tends to reach a saturation point relatively quickly. In contrast, Rewarded Meta-Pruning continues to show substantial accuracy gains over time, a promising characteristic for long-term model refinement.

It is worth noting that if both methods were initiated with the same batch of randomly initialized models, Rewarded Meta-Pruning would likely yield higher accuracy and lower FLOPs. This conclusion is supported by the observation that the slopes of the best accuracy and FLOPs in Rewarded Meta-Pruning are higher and lower, respectively, than those in MetaPruning. In summary, the reward mechanisms and trends in FLOPs and accuracy evolution distinguish Rewarded Meta-Pruning as a method with the potential for achieving superior accuracy while maintaining lower computational costs over extended search iterations.

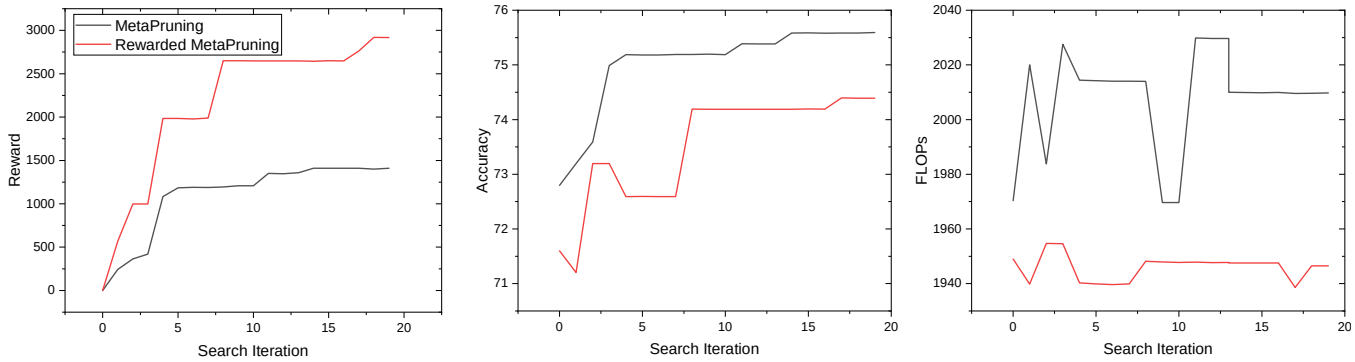

Figure 4. Rewards, accuracy, and FLOPs of the best models after each iteration of searching.

The selection process of NEVs, whether chosen randomly or after undergoing mutation or crossover operations, results in a distribution of FLOPs for the pruned models that approximates a bell curve, with values ranging between 1350 and 2100, as illustrated in Figure 5. However, it's worth noting that the distribution of FLOPs can be influenced and controlled by adjusting how random filter sizes are generated within the NEVs. Figure 5 visually represents this controllability. This observation sheds light on a critical aspect of our proposed algorithm—the adaptability and tunability of the pruned models. By modifying the generation process of random filter sizes, researchers and practitioners can intentionally tailor the FLOPs of pruned models to suit specific computational constraints or performance requirements. This level of control over FLOPs allows for a more fine-grained optimization of computational resources while maintaining or even enhancing model performance.

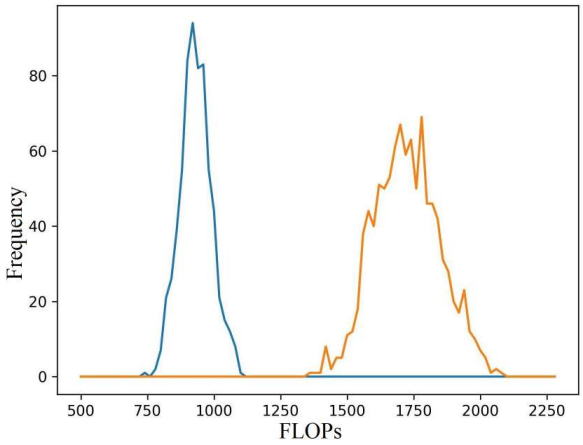

Figure 5. Distribution of FLOPs of 1000 randomly generated NEVs with varying ranges of FLOPs. Blue—750–1250 FLOPs range and Yellow—1500–2000 FLOPs range.

In our discussion, we delve into the robustness of the hyperparameters employed by Rewarded Meta-Pruning, building upon prior research such as [60]. The versatility of the reward function is a pivotal aspect to explore. By fine-tuning different hyperparameters within the reward function, we can effectively control various performance metrics of the final pruned model. For instance, we can adjust coefficients within the reward function to influence the pruning rate of the model, making it inversely proportional to the reward. Lower pruning rates inherently result in reduced FLOPs, which are directly related to hardware latency, i.e., the runtime of networks [61]. It is crucial to note that hardware latency varies across different hardware configurations, making it an important consideration in the pruning process.

Beyond FLOPs and latency, other vital metrics like energy consumption come into play. The NetAdapt framework [62], for example, leverages energy consumption as a metric by which to gauge network complexity at each stage and further prunes the network while

preserving accuracy. This multi-dimensional approach to metric selection provides a comprehensive view of model optimization, considering factors beyond mere computational efficiency. In essence, our discussion underscores the flexibility, and potential of Rewarded Meta-Pruning in adapting to diverse metrics and hardware configurations, paving the way for more refined and efficient neural network architectures.

Despite its promising results, our proposed methodology presents certain limitations that deserve careful consideration:

- Limited scope to CNN architectures: The applicability of our method is currently confined to channel pruning in convolutional neural networks (CNNs), restricting its direct application to other neural network architectures like recurrent neural networks (RNNs) and transformers. Future research efforts should focus on extending the method's applicability to a broader range of network architectures.
- Susceptibility to overfitting: When pruning a significant portion of channels, the method may tend overfitting. To mitigate this risk, we recommend incorporating regularization techniques such as early stopping or dropout to enhance the method's robustness.
- Computational overhead of meta-learning: Training a meta-learner, a crucial component of our method, can be computationally demanding. However, this cost is typically amortized across multiple pruning tasks, alleviating the overall computational burden.

Despite these limitations, we firmly believe that the advantages of our proposed methodology outweigh its drawbacks. The method consistently demonstrates impressive model size reductions without sacrificing accuracy, highlighting its efficacy. Moreover, its implementation is relatively straightforward, making it accessible to researchers and practitioners. We envision that future research will address the identified limitations and expand the method's applicability to a broader spectrum of neural network architectures.

5. Conclusions

In this work, we introduced a novel meta-pruning framework, Rewarded Meta-Pruning, which effectively addresses the challenges of network pruning, including accuracy degradation and performance bottlenecks. Our approach employs a meta-learning strategy to optimize pruning parameters using a carefully designed reward function. This reward function encourages the selection of pruning configurations that prioritize both accuracy preservation and efficiency gains. Rewarded Meta-Pruning demonstrates a superior performance compared to state-of-the-art pruning methods, achieving higher accuracy and lower FLOPs across a range of network architectures, including ResNet-50, MobileNetV1, and MobileNetV2. Our results highlight the effectiveness of our reward function in guiding the pruning process towards optimal network configurations. Furthermore, the flexibility of our reward function allows for optimization based on different objectives, such as memory usage or latency, making it adaptable to diverse application requirements. In summary, our work makes significant contributions to the field of network pruning by introducing a novel meta-learning-based approach that achieves superior performance and adaptability. Rewarded Meta-Pruning effectively addresses the trade-offs between accuracy and efficiency, covering the way for more efficient and competent pruned networks in various real-world applications. Looking ahead, future research avenues include exploring the integration of Rewarded Meta-Pruning in more complex architectures and extending its applicability to various emerging paradigms [63–66]. Our study primarily utilized accuracy and FLOPs in the reward function. The framework easily extends to include other metrics like computational efficiency, memory footprint, or latency. Exploring diverse reward metrics and task-specific functions is a promising avenue for future research. While evaluated on CNNs, our method applies to various architectures like RNNs and transformers. Investigating adaptability and architecture-specific considerations is valuable. Combining neural architecture search (NAS) and Rewarded Meta-Pruning could yield efficient pruning algorithms, offering an exciting line of future work.

Author Contributions: Conceptualization, A.S. and D.-G.L.; Methodology, A.S., A.K. and H.J.; Software, A.S.; Validation, A.S., H.J. and D.-G.L.; Formal analysis, A.K. and D.-G.L.; Investigation, A.S.; Resources, D.-G.L.; Data curation, A.S.; Writing—original draft, A.S.; Writing—review & editing, A.K., H.J. and D.-G.L.; Visualization, A.S.; Supervision, D.-G.L.; Project administration, D.-G.L.; Funding acquisition, D.-G.L. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korean Government (MSIT) (No. 2021R1C1C1012590) and (No. 2022R1A4A1023248).

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Karpathy, A.; Toderici, G.; Shetty, S.; Leung, T.; Sukthankar, R.; Fei-Fei, L. Large-scale video classification with convolutional neural networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Columbus, OH, USA, 23–28 June 2014; pp. 1725–1732.
2. Huang, G.; Liu, Z.; Van Der Maaten, L.; Weinberger, K.Q. Densely connected convolutional networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017; pp. 4700–4708.
3. Lee, D.G.; Kim, Y.K. Joint Semantic Understanding with a Multilevel Branch for Driving Perception. *Appl. Sci.* **2022**, *12*, 2877. [CrossRef]
4. Kim, Y.J.; Lee, D.G.; Lee, S.W. Three-stream fusion network for first-person interaction recognition. *Pattern Recognit.* **2020**, *103*, 107279. [CrossRef]
5. Lee, D.G.; Lee, S.W. Prediction of partially observed human activity based on pre-trained deep representation. *Pattern Recognit.* **2019**, *85*, 198–206. [CrossRef]
6. Huang, Q.; Zhou, K.; You, S.; Neumann, U. Learning to prune filters in convolutional neural networks. In Proceedings of the 2018 IEEE Winter Conference on Applications of Computer Vision (WACV), Lake Tahoe, NV, USA, 12–15 March 2018; pp. 709–718.
7. Tian, H.; Liu, B.; Yuan, X.T.; Liu, Q. Meta-Learning with Network Pruning for Overfitting Reduction. *CoRR*, 2019; *unpublished work*.
8. Lee, D.G.; Lee, S.W. Human interaction recognition framework based on interacting body part attention. *Pattern Recognit.* **2022**, *128*, 108645. [CrossRef]
9. Yamamoto, K.; Maeno, K. Pcas: Pruning channels with attention statistics for deep network compression. *arXiv* **2018**, arXiv:1806.05382.
10. Li, H.; Kadav, A.; Durdanovic, I.; Samet, H.; Graf, H.P. Pruning filters for efficient convnets. *arXiv* **2016**, arXiv:1608.08710.
11. Louizos, C.; Welling, M.; Kingma, D.P. Learning sparse neural networks through L_0 regularization. *arXiv* **2017**, arXiv:1712.01312.
12. Liu, Z.; Sun, M.; Zhou, T.; Huang, G.; Darrell, T. Rethinking the value of network pruning. *arXiv* **2018**, arXiv:1810.05270.
13. Frankle, J.; Carbin, M. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv* **2018**, arXiv:1803.03635.
14. Su, J.; Chen, Y.; Cai, T.; Wu, T.; Gao, R.; Wang, L.; Lee, J.D. Sanity-checking pruning methods: Random tickets can win the jackpot. In Proceedings of the Advances in Neural Information Processing Systems, Vancouver, BC, Canada, 6–12 December 2020; Volume 33, pp. 20390–20401.
15. Bouchard-Côté, A.; Petrov, S.; Klein, D. Randomized pruning: Efficiently calculating expectations in large dynamic programs. In Proceedings of the Advances in Neural Information Processing Systems, Vancouver, BC, Canada, 7–10 December 2009; Volume 22.
16. Deng, J.; Guo, J.; Xue, N.; Zafeiriou, S. Arcface: Additive angular margin loss for deep face recognition. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Long Beach, CA, USA, 15–20 June 2019; pp. 4690–4699.
17. Liu, J.; Zhuang, B.; Zhuang, Z.; Guo, Y.; Huang, J.; Zhu, J.; Tan, M. Discrimination-aware network pruning for deep model compression. *IEEE Trans. Pattern Anal. Mach. Intell.* **2021**, *44*, 4035–4051. [CrossRef]
18. Elkerdawy, S.; Elhoushi, M.; Zhang, H.; Ray, N. Fire Together Wire Together: A Dynamic Pruning Approach with Self-Supervised Mask Prediction. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, New Orleans, LA, USA, 18–24 June 2022; pp. 12454–12463.
19. Shibu, A.; Lee, D.G. EvolveNet: Evolving Networks by Learning Scale of Depth and Width. *Mathematics* **2023**, *11*, 3611. [CrossRef]
20. Han, S.; Mao, H.; Dally, W.J. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv* **2015**, arXiv:1510.00149.
21. Han, S.; Pool, J.; Tran, J.; Dally, W. Learning both weights and connections for efficient neural network. In Proceedings of the Advances in Neural Information Processing Systems, Montreal, QC, Canada, 7–12 December 2015; Volume 28.
22. Han, S.; Liu, X.; Mao, H.; Pu, J.; Pedram, A.; Horowitz, M.A.; Dally, W.J. EIE: Efficient inference engine on compressed deep neural network. *ACM SIGARCH Comput. Archit. News* **2016**, *44*, 243–254. [CrossRef]
23. Kruschke, J.K.; Movellan, J.R. Benefits of gain: Speeded learning and minimal hidden layers in back-propagation networks. *IEEE Trans. Syst. Man Cybern.* **1991**, *21*, 273–280. [CrossRef]
24. Liu, Z.; Li, J.; Shen, Z.; Huang, G.; Yan, S.; Zhang, C. Learning efficient convolutional networks through network slimming. In Proceedings of the IEEE International Conference on Computer Vision, Venice, Italy, 22–29 October 2017; pp. 2736–2744.

25. He, Y.; Liu, P.; Zhu, L.; Yang, Y. Filter pruning by switching to neighboring CNNs with good attributes. *IEEE Trans. Neural Netw. Learn. Syst.* **2022**, *34*, 8044–8056. [CrossRef]
26. Liu, Z.; Mu, H.; Zhang, X.; Guo, Z.; Yang, X.; Cheng, K.T.; Sun, J. Metapruning: Meta learning for automatic neural network channel pruning. In Proceedings of the IEEE/CVF International Conference on Computer Vision, Seoul, Republic of Korea, 27 October–2 November 2019; pp. 3296–3305.
27. Kumar, A.; Misra, R.K.; Singh, D.; Mishra, S.; Das, S. The spherical search algorithm for bound-constrained global optimization problems. *Appl. Soft Comput.* **2019**, *85*, 105734. [CrossRef]
28. Ye, J.; Lu, X.; Lin, Z.; Wang, J.Z. Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers. *arXiv* **2018**, arXiv:1802.00124.
29. He, Y.; Liu, P.; Wang, Z.; Hu, Z.; Yang, Y. Filter pruning via geometric median for deep convolutional neural networks acceleration. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Long Beach, CA, USA, 15–20 June 2019; pp. 4340–4349.
30. Molchanov, P.; Mallya, A.; Tyree, S.; Frosio, I.; Kautz, J. Importance estimation for neural network pruning. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Long Beach, CA, USA, 15–20 June 2019; pp. 11264–11272.
31. Luo, J.H.; Wu, J. Neural network pruning with residual-connections and limited-data. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Seattle, WA, USA, 13–19 June 2020; pp. 1458–1467.
32. Liebenwein, L.; Baykal, C.; Lang, H.; Feldman, D.; Rus, D. Provable filter pruning for efficient neural networks. *arXiv* **2019**, arXiv:1911.07412.
33. Luo, J.H.; Wu, J. Autopruner: An end-to-end trainable filter pruning method for efficient deep model inference. *Pattern Recognit.* **2020**, *107*, 107461. [CrossRef]
34. Huang, Z.; Wang, N. Data-driven sparse structure selection for deep neural networks. In Proceedings of the European Conference on Computer Vision (ECCV), Munich, Germany, 8–14 September 2018; pp. 304–320.
35. Zhuang, Z.; Tan, M.; Zhuang, B.; Liu, J.; Guo, Y.; Wu, Q.; Huang, J.; Zhu, J. Discrimination-aware Channel Pruning for Deep Neural Networks. In Proceedings of the Advances in Neural Information Processing Systems, Montreal, QC, Canada, 3–8 December 2018; Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2018; Volume 31.
36. He, Y.; Lin, J.; Liu, Z.; Wang, H.; Li, L.J.; Han, S. Amc: Automl for model compression and acceleration on mobile devices. In Proceedings of the European Conference on Computer Vision (ECCV), Munich, Germany, 8–14 September 2018; pp. 784–800.
37. Lin, M.; Ji, R.; Wang, Y.; Zhang, Y.; Zhang, B.; Tian, Y.; Shao, L. Hrank: Filter pruning using high-rank feature map. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Seattle, WA, USA, 13–19 June 2020; pp. 1529–1538.
38. Ye, M.; Gong, C.; Nie, L.; Zhou, D.; Klivans, A.; Liu, Q. Good subnetworks provably exist: Pruning via greedy forward selection. In Proceedings of the International Conference on Machine Learning, PMLR, Virtual, 13–18 July 2020; pp. 10820–10830.
39. Vanschoren, J. Meta-learning: A survey. *arXiv* **2018**, arXiv:1810.03548.
40. Finn, C.; Abbeel, P.; Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks. In Proceedings of the International Conference on Machine Learning, PMLR, Sydney, Australia, 6–11 August 2017; pp. 1126–1135.
41. Williams, R.J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.* **1992**, *8*, 229–256. [CrossRef]
42. Zoph, B.; Le, Q.V. Neural architecture search with reinforcement learning. *arXiv* **2016**, arXiv:1611.01578.
43. Xie, L.; Yuille, A. Genetic cnn. In Proceedings of the IEEE International Conference on Computer Vision, Venice, Italy, 22–29 October 2017; pp. 1379–1388.
44. Real, E.; Moore, S.; Selle, A.; Saxena, S.; Suematsu, Y.L.; Tan, J.; Le, Q.V.; Kurakin, A. Large-scale evolution of image classifiers. In Proceedings of the International Conference on Machine Learning, PMLR, Sydney, Australia, 6–11 August 2017; pp. 2902–2911.
45. Tancik, M.; Mildenhall, B.; Wang, T.; Schmidt, D.; Srinivasan, P.P.; Barron, J.T.; Ng, R. Learned initializations for optimizing coordinate-based neural representations. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Nashville, TN, USA, 20–25 June 2021; pp. 2846–2855.
46. Cai, H.; Zhu, L.; Han, S. Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv* **2018**, arXiv:1812.00332.
47. Brock, A.; Lim, T.; Ritchie, J.M.; Weston, N. Smash: One-shot model architecture search through hypernetworks. *arXiv* **2017**, arXiv:1708.05344.
48. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 770–778.
49. Howard, A.G.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; Andreetto, M.; Adam, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv* **2017**, arXiv:1704.04861.
50. Sandler, M.; Howard, A.; Zhu, M.; Zhmoginov, A.; Chen, L.C. Mobilenetv2: Inverted residuals and linear bottlenecks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–23 June 2018; pp. 4510–4520.
51. Mallipeddi, R.; Suganthan, P.N.; Pan, Q.K.; Tasgetiren, M.F. Differential evolution algorithm with ensemble of parameters and mutation strategies. *Appl. Soft Comput.* **2011**, *11*, 1679–1696. [CrossRef]
52. Li, Y.; Adamczewski, K.; Li, W.; Gu, S.; Timofte, R.; Van Gool, L. Revisiting Random Channel Pruning for Neural Network Compression. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, New Orleans, LA, USA, 18–24 June 2022; pp. 191–201.

53. Dauphin, Y.N.; Pascanu, R.; Gulcehre, C.; Cho, K.; Ganguli, S.; Bengio, Y. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In Proceedings of the Advances in Neural Information Processing Systems, Montreal, QC, Canada, 8–13 December 2014; Volume 27.
54. Deng, J.; Dong, W.; Socher, R.; Li, L.J.; Li, K.; Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In Proceedings of the 2009 IEEE Conference on Computer Vision and Pattern Recognition, Miami, FL, USA, 20–25 June 2009; pp. 248–255.
55. Lin, S.; Ji, R.; Yan, C.; Zhang, B.; Cao, L.; Ye, Q.; Huang, F.; Doermann, D. Towards optimal structured cnn pruning via generative adversarial learning. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Long Beach, CA, USA, 15–20 June 2019; pp. 2790–2799.
56. Lin, M.; Ji, R.; Zhang, Y.; Zhang, B.; Wu, Y.; Tian, Y. Channel pruning via automatic structure search. *arXiv* **2020**, arXiv:2001.08565.
57. Zhang, Y.; Lin, M.; Lin, C.W.; Chen, J.; Wu, Y.; Tian, Y.; Ji, R. Carrying out CNN channel pruning in a white box. *IEEE Trans. Neural Netw. Learn. Syst.* **2022**, *34*, 7946–7955. [CrossRef] [PubMed]
58. Lin, M.; Cao, L.; Zhang, Y.; Shao, L.; Lin, C.W.; Ji, R. Pruning networks with cross-layer ranking & k-reciprocal nearest filters. *IEEE Trans. Neural Netw. Learn. Syst.* **2022**, *34*, 9139–9148.
59. Blalock, D.; Gonzalez Ortiz, J.J.; Frankle, J.; Guttag, J. What is the state of neural network pruning? *Proc. Mach. Learn. Syst.* **2020**, *2*, 129–146.
60. He, Y.; Ding, Y.; Liu, P.; Zhu, L.; Zhang, H.; Yang, Y. Learning filter pruning criteria for deep convolutional neural networks acceleration. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Seattle, WA, USA, 13–19 June 2020; pp. 2009–2018.
61. Dong, J.D.; Cheng, A.C.; Juan, D.C.; Wei, W.; Sun, M. Dpp-net: Device-aware progressive search for pareto-optimal neural architectures. In Proceedings of the European Conference on Computer Vision (ECCV), Munich, Germany, 8–14 September 2018; pp. 517–531.
62. Yang, T.J.; Howard, A.; Chen, B.; Zhang, X.; Go, A.; Sandler, M.; Sze, V.; Adam, H. Netadapt: Platform-aware neural network adaptation for mobile applications. In Proceedings of the European Conference on Computer Vision (ECCV), Munich, Germany, 8–14 September 2018; pp. 285–300.
63. Xiao, J.; Zhong, S.; Wen, S. Unified analysis on the global dissipativity and stability of fractional-order multidimension-valued memristive neural networks with time delay. *IEEE Trans. Neural Netw. Learn. Syst.* **2021**, *33*, 5656–5665. [CrossRef]
64. Xiao, J.; Guo, X.; Li, Y.; Wen, S.; Shi, K.; Tang, Y. Extended analysis on the global Mittag-Leffler synchronization problem for fractional-order octonion-valued BAM neural networks. *Neural Netw.* **2022**, *154*, 491–507. [CrossRef] [PubMed]
65. Xiao, J.; Guo, X.; Li, Y.; Wen, S. Further Research on the Problems of Synchronization for Fractional-Order BAM Neural Networks in Octonion-Valued Domain. *Neural Process. Lett.* **2023**, *55*, 11173–11208. [CrossRef]
66. Xiao, J.; Li, Y. Novel synchronization conditions for the unified system of multi-dimension-valued neural networks. *Mathematics* **2022**, *10*, 3031. [CrossRef]