




Article

Optimizing Convolutional Neural Network Architectures

Luis Balderas ^{1,2,3,4*} , Miguel Lastra ^{2,3,4,5}  and José M. Benítez ^{1,2,3,4} 

- ¹ Department of Computer Science and Artificial Intelligence, University of Granada, 18071 Granada, Spain; j.m.benitez@decsai.ugr.es
- ² Distributed Computational Intelligence and Time Series Lab, University of Granada, 18071 Granada, Spain; mlastral@ugr.es
- ³ Sport and Health University Research Institute, University of Granada, 18071 Granada, Spain
- ⁴ Andalusian Research Institute in Data Science and Computational Intelligence, University of Granada, 18071 Granada, Spain
- ⁵ Department of Software Engineering, University of Granada, 18071 Granada, Spain
- * Correspondence: luisbalru@ugr.es

Abstract: Convolutional neural networks (CNNs) are commonly employed for demanding applications, such as speech recognition, natural language processing, and computer vision. As CNN architectures become more complex, their computational demands grow, leading to substantial energy consumption and complicating their use on devices with limited resources (e.g., edge devices). Furthermore, a new line of research seeking more sustainable approaches to Artificial Intelligence development and research is increasingly drawing attention: Green AI. Motivated by an interest in optimizing Machine Learning models, in this paper, we propose Optimizing Convolutional Neural Network Architectures (OCNNA). It is a novel CNN optimization and construction method based on pruning designed to establish the importance of convolutional layers. The proposal was evaluated through a thorough empirical study including the best known datasets (CIFAR-10, CIFAR-100, and Imagenet) and CNN architectures (VGG-16, ResNet-50, DenseNet-40, and MobileNet), setting accuracy drop and the remaining parameters ratio as objective metrics to compare the performance of OCNNA with the other state-of-the-art approaches. Our method was compared with more than 20 convolutional neural network simplification algorithms, obtaining outstanding results. As a result, OCNNA is a competitive CNN construction method which could ease the deployment of neural networks on the IoT or resource-limited devices.



Citation: Balderas, L.; Lastra, M.; Benítez, J.M. Optimizing Convolutional Neural Network Architectures. *Mathematics* **2024**, *12*, 3032. <https://doi.org/10.3390/math12193032>

Academic Editor: Ioannis Tsoulos

Received: 29 August 2024

Revised: 23 September 2024

Accepted: 26 September 2024

Published: 28 September 2024



Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

Keywords: convolutional neural network simplification; neural network pruning; efficient machine learning; Green AI

MSC: 68T07

1. Introduction

Over the last years, deep neural networks (DNNs) have become the state-of-the-art technique in several challenging tasks, such as speech recognition [1], natural language processing [2], and computer vision [3]. In particular, convolutional neural networks (CNNs) have achieved remarkable success across a broad range of computer vision challenges, such as image classification [3], object detection in images [4], object detection in video [5], semantic segmentation [6], video restoration [7], and medical diagnosis [8]. The astonishing results of CNNs are associated with the huge number of annotated data and important advances in hardware. Unfortunately, CNNs usually have an immense number of parameters, incurring in high storage requirements, and significant computational and energetic costs [9]. This imposes severe restrictions on the deployment of these models on devices with limited resources, such as edge devices or mobile devices. Thus, some methods to develop smaller-size models or simplify the complexity of the available models are necessary [10].

On the other hand, these large models requiring large amounts of memory and computation time have an evident environmental impact. In 2019, researchers from the University of Massachusetts discovered that training various deep learning models, such as those involving neural architecture search, could release over 284,019 kg of carbon dioxide. This amount is roughly equivalent to five times the total lifetime emissions of the typical American vehicle, including the car's production [11,12]. Concretely, to classify a single image, the VGG-16 model [13] needs over 30 billion floating-point operations (FLOPs) and has 138 million parameters, which demand more than 500 MB of storage space [14].

In consequence, reducing the model's storage requirements and computational cost becomes critical for resource-limited devices, specially in IoT applications, embedded systems, autonomous agents, mobile devices, and edge devices [15]. Actually, the concerns relative to such a high level of energy consumption—among other resources—have lead to the surge and development of a new line of research seeking a more sustainable future for Artificial Intelligence development and deployment, named Green Artificial Intelligence [16].

Nonetheless, Ref. [17] remarks that a usual DNN property is their considerable redundancy in parameterization, which leads to the idea of reducing this redundancy by compressing the networks. However, a severe problem along with compressing the models is the loss of accuracy. In order to avoid it, there are some ways of designing efficient DNNs and generating effective solutions. For instance, one approach involves using memetic algorithms to discover an effective architecture for a given task, while another method entails employing alternative optimization algorithms to fine-tune the connection weights of a pre-defined architecture. Kernel Quantization is also used for efficient network compression [18]. Alternatively, pruning is a widely utilized technique for simplifying neural networks. Since the introduction of Optimal Brain Damage (OBD) and Optimal Brain Surgery (OBS) in 1990, pruning methods have been thoroughly researched for model compression. Over the years, numerous other approaches have been developed to create more efficient and effective neural networks of various types, including dense, convolutional, and recurrent networks [12]. The primary objective of pruning algorithms is to derive a subnetwork with significantly fewer parameters while maintaining the same level of accuracy.

In this paper, we propose a novel CNN optimization and construction technique called Optimizing Convolutional Neural Network Architectures (OCNNA) based on pruning which requires minimal tuning. OCNNA has been designed to assess the importance of convolutional layers. Since this measure is computed for every convolutional layer and unit from the model input to the output, it essentially reflects the importance of every single convolutional filter and its contribution to the information flow through the network [19]. Moreover, our method is able to sort convolutional filters within a layer by importance; as a consequence, it can be seen as a feature importance computation tool which can generate more explainable neural networks. OCNNA is easy to apply, having only one parameter (k), called the percentile of significance, which represents the proportion of filters which will be transferred to the new model based on their importance. Only the k -th percentile of filters with higher values after applying the OCNNA process will remain. The proposed OCNNA can directly be applied to trained CNNs, avoiding the training process from scratch.

We thoroughly evaluated the optimization efficacy of the OCNNA method compared with the state-of-the-art pruning techniques. Our experiments on the CIFAR-10, CIFAR-100 [20], and Imagenet [21] datasets and popular CNN architectures, such as ResNet-50, VGG-16, DenseNet40, and MobileNet, show that our algorithm leads to better performance in terms of the accuracy in prediction and the reduction in the number of parameters, following a cornerstone metric for Green AI [22] and efficient Machine Learning.

Our main contributions can be summarized as follows:

- A Green AI simplification method for CNNs called OCNNA is proposed. OCNNA measures the importance of each convolutional layer and unit from a trained model by combining well-known data science and statistical techniques, such as PCA, Frobenius norm, and Coefficient of Variation. After that, it builds a simplified model which can be even more precise than the original one and more efficient in terms of computational costs and memory footprint.
- Experiments on three benchmark datasets (CIFAR-10, CIFAR-100, and Imagenet) demonstrate that our simplification technique yields highly competitive results in terms of efficiency metrics (reduction in the number of parameters) and prediction accuracy when applied to the most widely used CNN models, such as VGG16, ResNet50, DenseNet40, and MobileNetV1.

The paper is organized as follows: Section 2 presents an overview of the latest methods for developing efficient deep neural networks. Our proposal is described in Section 3. In Section 4, our methodology is experimentally analyzed, and the results are analyzed and discussed in Section 5. Finally, Section 6 highlights the conclusions.

2. Previous work

The purpose of this section is to provide a brief overview of the main approaches in model compression: neuroevolution, neural architecture search, quantization, and pruning. Our research mainly focuses on convolutional neural network pruning.

2.1. Neuroevolution

Neuroevolution can be applied to several tasks related to efficient neural network design, such as learning neural networks (NN) building blocks, hyperparameters, or architectures. In 2002, NeuroEvolution of Augmenting Topologies (NEAT) was presented in [23], showing the effectiveness of a Genetic Algorithm (GA) in evolving NN topologies and strengthening the analogy with biological evolution. More recently, ref. [24] drew inspiration from NEAT, evolving deep neural networks by beginning with a simple neural network and gradually increasing its complexity through mutations. In [25], a more accurate approach, which consists in stacking the same layer module to make a deep neural network, like Inception, DenseNet, and ResNet, can be found [26].

2.2. Neural Architecture Search

Recently, neural architecture search (NAS), whose main goal is to automatically design the best DNN architecture, has achieved great importance in model design. On the one hand, NAS algorithms can be divided into two categories: (1) microstructure search focuses on identifying the best operation for each layer; (2) macrostructure search aims to find the optimal number of channels or filters for each layer, or the ideal model depth [27]. Additionally, NAS algorithms can be categorized into three groups based on the optimizer used: reinforcement learning-based NAS algorithms, gradient-based NAS algorithms, and evolutionary NAS (ENAS) algorithms. In this sense, NSGA-II has been recently used for NAS creating NSGA-Net [28]. In [29], NATS-Bench is proposed, consisting in a unified benchmark for topology and size aggregating more than 10 state-of-the-art NAS algorithms.

2.3. Quantization

Quantization refers to the process of approximating a continuous signal by using a set of discrete symbols or integer values [30]. In other words, it reduces computations by reducing the precision of the data type. Advanced quantization techniques, such as asymmetric quantization [31] or calibration-based quantization, have been presented to improve accuracy. In [30], we find a complete quantization guide and recommendations.

2.4. Knowledge Distillation

Knowledge distillation, which was first defined by [32] and generalized in [33], is the process of transferring knowledge from one neural network to a different one. In [34], a student–teacher framework is presented, introducing different scenarios, such as distillation based on the number of teachers (one teacher vs. multiple teachers), distillation based on data format (data-free, with a few samples, or cross-modal distillation), or online and teacher-free distillation.

2.5. Pruning

Network pruning is one of the most effective and prevalent approaches to model compression. Pruning techniques can be classified by various aspects: structured and unstructured pruning, depending on whether the pruned network is symmetric or not [30,35]; neuron, weight, or filter pruning, depending on the network’s element which is pruned; or static and dynamic pruning [30]. While static pruning removes elements offline from the network after training and before inference, dynamic pruning determines at runtime which elements will not participate in further activity [30]. Most researchers focus on how to find unimportant filters. Magnitude-based methods [36] use the magnitude of the weights in feature maps from certain layers as a measure of importance, pruning those with lower magnitudes. Others measure the importance of a filter through their reconstruction loss (Thinet) [37] or Taylor expansion [38,39]. In [40], Average Percentage of Zeros (APoZ) is used to assess the proportion of zero activations in a neuron following ReLU mapping, thus allowing for the pruning of redundant neurons. HRank [41] understands filter pruning as an optimization problem, using the feature maps as the function which measures the importance of a filter part of the CNN. Inspired by HRank, FPWT [42] introduces a new method which transforms the feature map in the spatial domain into the frequency domain by using Wavelet Transform.

In [43], a new CNN compression technique is presented based on the filter-level pruning of redundant weights according to entropy importance criteria (FPEI) with different versions depending on the learning task and the NN. SFP [44] and FPGM, based on filter pruning via geometric median [45], use soft filter pruning; PScratch [?] proposes to prune from scratch, before training the model. In [47], a criterion for CNN pruning inspired by NN interpretability is proposed: the most relevant units are identified based on their relevance scores, which are derived from explainable AI (XAI) concepts. Ref. [48] introduces a data-driven CNN architecture determination algorithm called AutoCNN which consists of three training stages (spatial filter, classification layers, and hyperparameters). AutoCNN uses statistical parameters to decide whether to add new layers, prune redundant filters, or add new fully connected layers pruning low-information units. An iterative pruning method based on deep reinforcement learning (DRL) called Neon, formed by a DRL agent which controls the trade-off between performance and the efficiency of the pruned network, is introduced in [49]. For each hidden layer, Neon extracts the architecture-based and the layer-based feature maps which represent an observation. Then, the aforementioned hidden layer is compressed and fine-tuned. After that, a reward is calculated and used to update the deep reinforcement learning agent’s weights. This process is repeated several times for the whole neural network.

In [50], a multi-objective evolution strategy algorithm called DeepPruningES is proposed. Its final output consists of three neural networks with different trade-offs called knee (with the best trade-off between training error and the number of FLOPs), boundary-heavy (with the smallest training error), and boundary-light solutions (with the smallest number of FLOPs). In [27], a customized correlation-based filter-level pruning method for deep CNN compression called COP, which removes redundant filters through their correlations, is presented. In [51], SCWC is introduced, a shared channel weight convolution method to decrease the number of multiplications in CNNs by leveraging the distributive property, made possible through structured channel parameter sharing. In [52], a new method called CHWP for identifying the most redundant filters is proposed, taking into account the size

of filters, the difference between them, and the role of Batch Normalization layers. In [53], a training method for CNN compression is proposed. It integrates matrix factorization and regularization techniques, based on Singular Value Decomposition. Nonetheless, the method has been evaluated only on ResNet-18, ResNet-20, and ResNet-32.

In [54], a CNN pruning method called MOP-FMS is introduced, in which the pruning task is modeled as a bi-objective optimization problem based on feature map selection. The two objectives of the method are accuracy and FLOPs, which are achieved by using an ad hoc evolutionary optimization algorithm designed to perform the pruning. Ref. [55] presents a CNN pruning method, which obtains a simplified network by clustering its filters. After that, it searches the optimal compact network structure by applying a social group optimization algorithm. In [56], an evolutionary method called Bi-CNN-Pruning is proposed to prune filters and channels with the objective of preserving the performance of the original model according to different pruning criteria, such as weight magnitude or activation statistics, among others. Concretely, it maintains the important channels in an ordered way, i.e., useful filters are selected first and then the channels. ResPrune is proposed in [57]. It is a selection filter method which uses two criteria: l_2 -norm and redundancy. Unlike other methods, ResPrune does not completely omit the filters identified as irrelevant; instead, it restores them to their original values by using stochastic techniques. In [58], a pruning framework called MGPF is introduced. MGPF generates sparse models of different granularity without fine-tuning the remaining connections after pruning. CIE [59] is a cross-layer importance evaluation technique used for neural network pruning, which assesses the significance of convolutional layers based on the model's prediction accuracy. This evaluation is highly efficient in terms of time, as the process is performed only once for a given model.

2.6. Summary

As demonstrated in this section, there are numerous approaches and techniques for simplifying CNNs. Over the years, increasingly sophisticated solutions have been proposed; whether through neuroevolution, neural architecture search, quantization, knowledge distillation, or pruning, the goal remains the same: to achieve smaller models without sacrificing prediction accuracy. However, designing an efficient, versatile, and effective method is not easy. In the case of neuroevolution and neural architecture search, competitive techniques can be achieved in terms of accuracy, but the computational cost of generating them can be prohibitive. For quantization and knowledge distillation, conceptual challenges may arise, while pruning algorithms are typically designed for specific models, unable to tackle the simplification of networks of different natures or assess their performance on various benchmarks. To address all these challenges, we developed OCNNA, an efficient technique that will be evaluated on the de facto benchmarks for image classification and will simplify the most paradigmatic and general convolutional networks. It can be applied in virtually any industry or academic use case involving CNNs and image classification.

3. Proposal

In this paper, we address the challenge of finding an optimized topology for a convolutional neural network. Thus, we introduce the Optimizing Convolutional Neural Network Architectures (OCNNA) method, a new convolutional neural network construction method based on pruning. In this section, we provide a detailed description of the method.

3.1. Notation

First of all, we introduce the notation used in our proposal. Let Ω be a neural network with L convolutional layers. Let w_m^l and o_m^l be the convolutional filter and the output of the l -th layer. The subscript $m \in 1, \dots, M^l$ represents the filter index, where M^l indicates the total number of output filters in the corresponding layer. In consequence, pruning the l -th filter in layer m implies removing the corresponding w_m^l .

Principal Component Analysis (PCA) [60] is a data analysis tool applied to identify the most meaningful basis to re-express, revealing a hidden structure, a given dataset. We define $P_m^l = PCA(o_m^l)$ as the matrix result of computing PCA on the m -th filter’s output of the l -th layer.

The Frobenius norm [61] is a norm of an $m \times n$ matrix A defined as

$$\|A\|_F = \sqrt{\sum_{i=1}^m \sum_{j=1}^N |a_{ij}|^2} \tag{1}$$

It is also equal to the square root of the matrix trace of A, A^H :

$$\|A\|_F = \sqrt{Tr(AA^H)} \tag{2}$$

where A^H is the conjugate transpose [62]. Let us define $F_m^l = \|P_m^l\|_F$ as the Frobenius norm of the above PCA calculation (on the m -th filter’s output of the l -th layer).

The Coefficient of Variation (CV) is the relationship between mean and standard deviation [63]. If D is a data distribution, with σ_D its standard deviation and μ_D its mean, CV is calculated as

$$CV_D = \frac{\sigma_D}{\mu_D} \tag{3}$$

It is attractive as a statistical tool because it permits the comparison of variables free from scale effects (dimensionless). We call $C = CV(x)$ if $x \in \mathbb{R}^n$, for a given $n \in \mathbb{N}$.

Finally, we define the percentile of significance k . Only the k -th percentile of filters with higher values in terms of importance will remain. All notations can be found in Table 1.

Table 1. Notations and definitions.

Notation	Definition
L	Number of convolutional layers
w_m^l	m -th filter from the l -th convolutional layer
o_m^l	Output of the m -th filter from the l -th layer
M^l	Number of output filters in the l -th layer
P_m^l	PCA applied to the m -th filter from the l -th layer
F_m^l	Frobenius norm of P_m^l
C	Coefficient of Variation of a vector
k -th percentile	Percentile of significance

3.2. OCNNA: The Algorithm

Since the main components of a CNN are the convolutional filters, OCNNA is designed to identify the most important ones, thus creating a new model in which the less significant convolutional units are not included. This way, OCNNA generates a more efficient model in terms of the number of parameters with minimum precision loss—as we will see in the next section. Additionally, OCNNA allows for the ordering of the convolutional filters by importance, providing a feature importance assessment method and, as a result, helping to better understand these models. Our method employs three important techniques to identify the significant filters: Principal Components Analysis (PCA), for selecting the most important features based on their hidden structure; the Frobenius norm, to summarize the PCA output information; and the Coefficient of Variation (CV), to measure the variability of Frobenius norm outputs. Figure 1 shows, as an example, the simplification process of a VGG-16 convolutional filter. As can be seen, the algorithm takes the filters from each layer (Figure 1, 1); applies PCA, Frobenius norm, and Coefficient of Variation (Figure 1, 2); and ranks the filters by importance, selecting only their top k -th percentile (Figure 1, 3). More details can be found in Algorithm 1.

Algorithm 1 OCNNA

```

1: function OCNNA(model,  $k$ ,  $D_{var}$ )
2:   compressed_model = new Model()
3:   for layer in model.Layers do
4:     if layer is Convolutional then
5:       variability = List()
6:       for filter in layer do
7:          $o = \text{model.predict}(D_{var})$ 
8:          $p = \text{PCA}(o, 95\% \text{ var})$ 
9:          $pn = \text{FrobeniusNorm}(p)$ 
10:         $c = \text{CV}(pn)$ 
11:        Append  $c$  to variability
12:      end for
13:      new_layer_index = get  $k$ -th percentile in variability
14:      Add new layer with new_layer_index filters from model
15:    else
16:      Add layer to compressed_model
17:    end if
18:  end for
19:  return compressed_model
20: end function

```

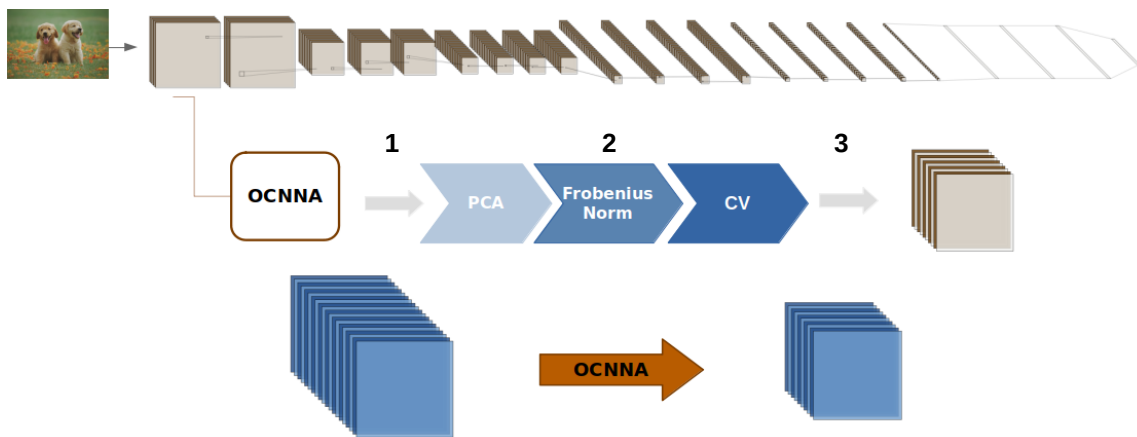


Figure 1. OCNNA applied to VGG-16. Given the output from the i -th convolutional layer, PCA, Frobenius norm, and Coefficient of Variation are applied to identify the most significant filters. The k -th percentile of filters, in terms of importance, are selected, generating a new model whose i -th convolutional layer is a optimized version of the original one. This approach is applied to every convolutional filter.

Given convolutional layer w^l , a D_{var} dataset, used exclusively for measuring the importance of filters, is evaluated by generating output o^l . o^l is formed by M^l filters. In consequence, o_m^l is the m -th filter's output from the l -th layer. By using o_l , OCNNA helps us to measure the importance of the M^l filters in the m -th layer. Concretely, we adopt a three-stage process. First, for each filter and image contained in D_{var} , we apply PCA retaining the 95% of variance.

$$P_m^l = \text{PCA}(o_m^l) \tag{4}$$

with P_m^l a matrix which contains the most meaningful features generated by the m -th filter of the l -th convolutional layer. Nonetheless, the information embedded in P_m^l is too large. In consequence, we apply the Frobenius norm to summarize this information:

$$F_m^l = ||P_m^l||_F \tag{5}$$

obtaining a vector F_m^l , in which each component is the result of the process described above applied to each image from D_{var} . Finally, we calculate the CV:

$$C_m = CV(F_m^l) \quad (6)$$

C_m is a number which summarizes the m -th filter significance within the l -th convolutional layer by measuring the variability of the process PCA and Frobenius norm for each image in D_{var} . In other words, OCNNA gives a low score of importance to a filter if for a subset of images, it generates an output whose hidden structures (PCA, 95% variance), after being summarized (Frobenius norm), have little variability (CV).

To sum up, OCNNA is able to extract insights and measure the importance of each filter of a convolutional layer, starting from hundreds of arrays which constitute the output from a dataset, called D_{var} , and generating a holistic vision summarizing the filter significance into a single number (Figure 2). As a result, OCNNA transforms the output of a convolutional layer into an array in which the i -th component represents the i -th filter's importance. Finally, using parameter k , or percentile of significance, we extract the k -th percentile of filters in terms of significance, completing the simplification process.

The larger k , the more strict the filter selection. In consequence, fewer filters will be selected and the new model will be simpler (a smaller number of parameters compared with the original one).

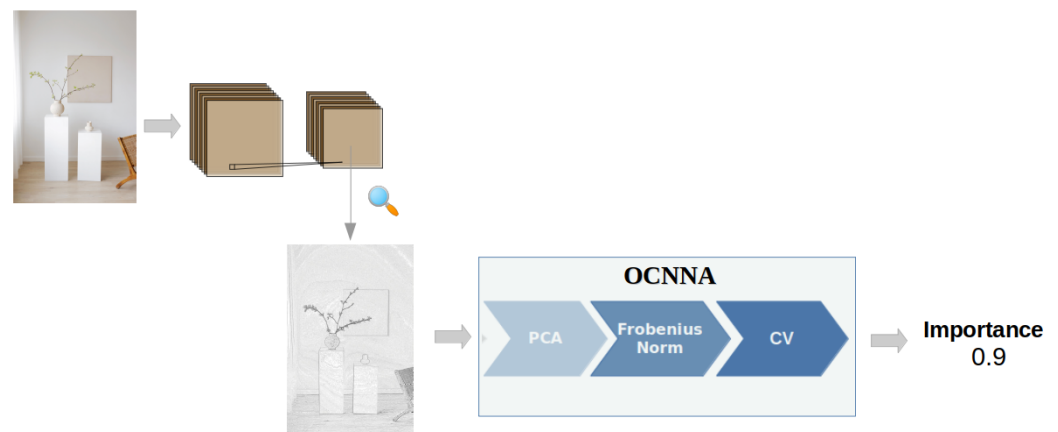


Figure 2. OCNNA provides a single number for this filter which reflects its importance. This process is iterated over all filters from a layer and their k -th percentile in terms of significance will form part of the new model.

3.3. Implementation

As mentioned above, OCNNA can measure the importance of a convolutional filter by extracting hidden insights from a multidimensional array and express it through a number.

This process implies heavy computational costs. In this sense, OCNNA is designed to maximize its performance in terms of the time required to complete the simplification process by proposing a parallel computing paradigm. In other words, this entails counting the number of CPUs available and distributing the tasks associated to each filter (prediction, PCA, Frobenius norm, and CV) of the convolutional layer among them, carrying out the calculations simultaneously and, as a result, speeding-up the results. This parallelism is absolutely transparent to the user. Once all the operations are finished, a synchronization process between the different subtasks is accomplished, mapping every result (the significance of the filter) in the correct component of the vector, which represents the importance of each filter in the convolutional layer. It was implemented in Python 3.9, and Tensorflow 2.9 was used as the machine learning framework.

4. Empirical Evaluation

To assess the performance of OCNNA, we designed a thorough empirical procedure that included different well-known datasets (CIFAR-10, CIFAR-100, and Imagenet) and architectures (ResNet-50, VGG-16, DenseNet-40, and MobileNet) which represent core benchmarks extensively referenced in the literature. Moreover, OCNNA was compared with 20 state-of-the-art CNN simplification techniques, obtaining successful results. This section is structured as follows: The architectures and datasets, metrics, compared state-of-the-art approaches, and training process settings are explained in order to assure the experiments' reproducibility. Finally, the results and analysis for the CIFAR and Imagenet datasets are presented, comparing them with the other state-of-the-art techniques.

4.1. Common Architectures and Datasets

We thoroughly evaluated our CNN building and optimizing scheme. To obtain comparable results with other state-of-the-art approaches, we selected four popular CNN architectures: VGG-16 [13], ResNet-50 [64], DenseNet-40 [65], and MobileNet [66]. VGG-16 is a convolutional neural network formed by 138.4 M parameters and 16 layers. The input is an RGB image with a size of 224×224 , which is passed through a stack of 3×3 receptive field convolutional layers, with padding fixed to 1 pixel. Five max-pooling layers (pixel window of size 2×2 and stride set to 2), which follow some of the convolutional layers, are included as spatial pooling. The last stack of convolutional layers is followed by three dense layers of 4096, 4096, and 1000 channels.

ResNet-50, drawing inspiration from the VGG networks, incorporates the idea of residual learning to simplify training by reconfiguring the layers to learn residual functions relative to their inputs, rather than learning functions without references. In practice, ResNet includes shortcut connections and has lower complexity than VGG-16, given the fact that it is formed by 25.6 M parameters. DenseNet (1 M parameters and 40 layers) connects each convolutional unit as if it were a feed-forward neural network, reducing the number of parameters and diminishing problems such as the vanishing gradient. Finally, MobileNet is a light-weight neural network designed for mobile and embedded vision apps. It has 4.3 M parameters and 55 layers.

To demonstrate the versatility of our approach, we evaluated it by using a core set of widely used benchmark datasets for image classification: CIFAR-10 [20], CIFAR-100 [20], and ImageNet [21]. The CIFAR-10 dataset is an image classification problem with 10 classes formed by 60,000 images with size 32×32 (each class consists of 6000 images, with a total of 50,000 images for training and 10,000 images for testing). CIFAR-100 contains the same number of images as CIFAR-10 but 100 classes (600 images each). On the other hand, Imagenet is a dataset formed by 1431,167 annotated images (224×224) and 1000 object classes. As we have mentioned, OCNNA requires a D_{var} dataset to measure convolutional filter importance. In the case of CIFAR-10 and CIFAR-100, we selected 10% of the training images, in other words, 5000 images identically distributed by class. After completing the optimization process, we evaluated the new model's performance by using the test images. For the Imagenet dataset, we used as D_{var} the "imagenet_v2/topimages" subset and as test set "imagenet_v2/matched-frequency". Both of them have 10,000 images sampled after a decade of progress on the original ImageNet dataset, making the new test data independent of existing models and guaranteeing that the accuracy scores are not affected by adaptive overfitting [67]. These datasets can be found in [68].

4.2. Metrics

We measured the prediction performance of the optimized models with accuracy (ACC). In addition, we registered the number of parameters to assess complexity and efficiency in terms of memory requirements and runtime as it is defined for the Green AI paradigm (a lower number of parameters may lead to increased efficiency). We also recorded the remaining parameters ratio (RPR) compared with the original model for compression, as previously performed for other state-of-the-art approaches. A higher

parameter reduction ration means a smaller model size and, as a result, a less complex model. The definition of RPR is

$$RPR = 1 - \frac{NP_O - NP_S}{NP_O} \quad (7)$$

where NP_O and NP_S represent the number of parameters of the original model and the optimized one, respectively. In any case, we adjusted the metrics (and their presentation) to match those used in the literature, ensuring a fair and precise comparison between OCNNA and other approaches.

4.3. Training Process Settings

For the VGG-16, ResNet-50, MobileNet, and DenseNet models training on CIFAR-10 or CIFAR-100, we set the weight decay to 1×10^{-6} , the momentum to 0.9, the learning rate to 1×10^{-3} , and the batch size to 64. All images were augmented by horizontal and vertical flipping, zoom with range between 0.85 and 1.5, rotation range of 180, and fill mode as reflect; in other words, pixels outside the boundaries of the input image were filled according to the following mode [69]:

$$abcdcda|abcd|dcbaabcd$$

We did not train any model on Imagenet due to the existence of publicly available pre-trained models.

4.4. Results and Analysis on CIFAR Datasets

The results on the CIFAR datasets for different architectures are shown in Tables 2–5. The Dataset column shows the learning task; the Architecture column shows the neural network used in the learning task; the Base (%) column refers to the accuracy originally obtained with the aforementioned architecture for the dataset given; Acc (%) is the accuracy obtained after applying the pruning algorithm; RPR means the remaining parameters ratio, where the lower, the better; Acc. Drop is the accuracy loss after pruning, where the smaller, the better. As we have said, we used three benchmark architectures, ResNet-50, VGG-16, and DenseNet-40.

In the case of ResNet-50, OCNNA generates a compressed model where just over 45% of the parameters remain (57.12% for CIFAR-10 and 46.95% for CIFAR-100), while the accuracy loss is very small. As a result, OCNNA is an effective method for compressing CNNs; actually, it achieves the best values for both metrics among all the considered methods.

Given the fact that VGG-16 is a very complex network, it might present greater redundancy than the other architectures. In fact, we were able to reduce the model, pruning 86.68% of the parameters for CIFAR-10 and 74.03% for CIFAR-100 without any accuracy loss for CIFAR-10 (actually, an improvement of 0.42%) and keeping it nearly unchanged for CIFAR-100 (−0.44%). OCNNA achieved the best scores for both metrics in CIFAR-10 and CIFAR-100, with the exception of RPR, where it ranked third.

For DenseNet-40, OCNNA again achieved the best results in both metrics and both datasets. In addition, it improved test accuracy for CIFAR-10 (0.39%) and for CIFAR-100 (0.23%).

Finally, we can observe the same behavior for MobileNet: OCNNA ranked first in both RPR and ACC for both datasets but only in size reduction for CIFAR-10. While not compressing the most for MobileNet built for CIFAR-10, its simplified model improves test accuracy (0.65%).

Overall, OCNNA is the winner according to the results for the CIFAR-10 and CIFAR-100 datasets.

Table 2. Results of pruning ResNet-50 on CIFAR datasets. RPR means the remaining parameters ratio, where the lower, the better. Acc. Drop is the accuracy loss after pruning, where the smaller, the better. The best results are highlighted in bold.

Dataset	Base (%)	Algorithm	Acc. (%)	Acc. Drop (%)	RPR (%)
CIFAR-10	93.55	FPEI [43]	91.85	1.70	45.69
		LRP [70]	93.37	0.18	75.24
		DeepPruningES [50]	91.89	1.66	78.69
		OCNNA	93.42	0.13	42.88
CIFAR-100	73.24	FPEI	69.58	3.66	57.53
		DeepPruningES	57.81	15.43	80.91
		OCNNA	70.32	2.92	53.05

Table 3. Results of pruning VGG-16 on CIFAR datasets. RPR means the remaining parameters ratio, where the lower, the better. Acc. Drop is the accuracy loss after pruning, where the smaller, the better. The best results are highlighted in bold.

Dataset	Base (%)	Algorithm	Acc. (%)	Acc. Drop (%)	RPR (%)
CIFAR-10	93.70	FPGM [40]	93.00	0.7	48.97
		DeepPruningES	91.79	1.91	64.99
		ThiNet [40]	92.99	0.71	26.92
		PScratch [71]	93.02	0.68	26.96
		HRank [41]	93.43	0.27	17.10
		Slimming [36]	93.44	0.26	16.71
		COP v1 [27]	93.37	0.18	15.15
		COP v2	93.86	−0.17	13.56
		SNACS [72]	91.06	−0.17	3.84
		White-Box [73]	93.47	0.23	—
		SOKS-80% [74]	94.01	−0.31	—
		EACP(k = 80%) [55]	93.29	0.41	24.6
		MOP-FMS (80%) [54]	91.51	2.19	19.50
		Bi-CNN-Pruning [56]	93.88	−0.18	63.48
		ResPrune [57]	93.76	−0.27	—
		MGPF [58]	93.88	−0.18	9.14
		FPWT [42]	93.94	−0.24	26.70
		OCNNA	94.12	−0.42	13.32
CIFAR-100	73.51	SFP [75]	71.74	1.77	60.66
		DeepPruningES	67.06	6.45	80.07
		FPGM	72.76	1.25	48.99
		HRank [41]	72.43	1.08	44.07
		COP v1	72.63	0.88	34.81
		Slimming	72.76	0.75	33.40
		COP v2	72.99	0.52	26.16
		Bi-CNN-Pruning	72.11	1.4	78.27
		OCNNA	73.07	0.44	25.97

Table 4. Results of pruning DenseNet-40 on CIFAR datasets. RPR means the remaining parameters ratio, where the lower, the better. Acc. Drop is the accuracy loss after pruning, where the smaller, the better. The best results are highlighted in bold.

Dataset	Base (%)	Algorithm	Acc. (%)	Acc. Drop (%)	RPR (%)
CIFAR-10	76.52	HRank	75.94	0.58	58.68
		Slimming	75.90	0.62	54.28
		COP v1	75.53	0.99	56.10
		COP v2	76.03	0.49	54.08
		OCNNA	76.91	−0.39	52.96
CIFAR-100	94.84	HRank	93.68	0.60	39.00
		Slimming	94.35	0.49	34.80
		COP v1	94.19	0.65	37.66
		COP v2	94.54	0.30	34.80
		OCNNA	95.07	−0.23	34.12

Table 5. Results of pruning MobileNet on CIFAR datasets. RPR means the remaining parameters ratio, where the lower, the better. Acc. Drop is the accuracy loss after pruning, where the smaller, the better. MobileNet-0.75 means that every layer is 75% of the original one. COP-0.50 implies that 50% of filters are maintained. The best results are highlighted in bold.

Dataset	Base (%)	Algorithm	Acc. (%)	Acc. Drop (%)	RPR (%)
CIFAR-10	94.07	MobileNet-0.75	93.36	0.71	53.96
		MobileNet-0.50	92.84	1.23	25.28
		COP v1-0.50	93.59	0.48	34.06
		COP v1-0.30	92.97	1.1	25.94
		COP v2-0.50	93.89	0.18	32.72
		COP v2-0.30	93.47	0.6	25.38
		Adapt-DCP	94.57	−0.6	21.74
		MGPF	92.5	1.5	9.14
		OCNNA	94.72	−0.65	24.60
CIFAR-100	74.94	MobileNet-0.75	73.99	0.95	53.96
		MobileNet-0.50	73.20	1.74	25.28
		COP v1-0.50	73.95	0.99	42.50
		COP v1-0.30	73.45	1.49	25.35
		COP v2-0.50	74.66	0.28	40.85
		COP v2-0.30	74.01	0.93	25.42
		CIE [59]	57.53	17.41	—
		OCNNA	74.72	0.22	23.87

4.5. Results and Analysis on Imagenet Dataset

The performance of OCNNA and state-of-the-art methods for VGG-16, ResNet-50, and MobileNet on the Imagenet dataset are presented in Tables 6–8. We begin by describing the results obtained for the VGG-16 architecture (Table 6). In this case, the best performing algorithm for compression was MGPF, at the cost of a suboptimal accuracy. OCNNA ranked second in RPR.

For ResNet-50 (Table 7), more extensive experimental results can be found in the literature. To the best of our knowledge, ResPrune and SCWC ($s = 0.5$, $s = 0.4$, and $s = 0.3$) are the only methods which improve the accuracy (negative accuracy drop) but with an RPR above 65% for SCWC (no data available for ResPrune).

FPEI-R7 with DR obtains a notable RPR (37.88%), still smaller than OCNNA's (37.44%), but the accuracy drop is quite higher (1.68% vs. 0.57% for OCNNA). In [42] (FPWT), we found the best approach to compressing ResNet-50 (31.8%). OCNNA was not as effective

as FPWT in terms of RPR, but the accuracy drop for OCNNA was more than four times smaller compared with FPWT (0.57% for OCNNA vs. 2.48% for FPWT).

In real-world applications, it is essential to balance performance and compression rates based on varying computational demands, energy consumption constraints [39], and accuracy requisites.

For MobileNet (Table 8), OCNNA outperformed the different approaches of the COP method and the direct simplification of the original model.

As a final, overall conclusion, we can assert that OCNNA can obtain simplified CNNs with remarkable complexity reduction while retaining the accuracy or even improving it in some cases.

Table 6. Comparison of OCNNA and other methods on Imagenet (VGG-16). RPR means the remaining parameters ratio, where the lower, the better. Acc. is the test accuracy after pruning. Acc. Drop is the test accuracy loss after pruning, where the smaller, the better. The best results are highlighted in bold.

Base (%)	Method	Acc (%)	Acc. Drop (%)	RPR (%)
74.4% [13]	SCWC _(s=0.6) [51]	73.80	0.6	60.5
	SCWC _(s=0.5)	73.20	1.2	50.3
	SCWC _(s=0.4)	73.17	1.23	40.8
	SCWC _(s=0.3)	72.16	1.64	30.6
	SCWC _(s=0.2)	71.42	2.98	19.7
	APoZ [40]	73.09	1.31	49.0
	CP [75]	70.7	3.7	–
	ThiNet-GAP [40]	72.64	1.76	–
	SSR [71]	72.75	1.65	–
	MGPF	70.96	3.44	6.0
	OCNNA	71.54	2.86	18.9

Table 7. Comparison of OCNNA and other methods on Imagenet (ResNet-50). RPR means the remaining parameters ratio, where the lower, the better. Acc. Drop is the accuracy loss after pruning, where the smaller, the better. The best results are highlighted in bold.

Base (%)	Method	Acc. (%)	Acc. Drop (%)	RPR (%)
75.3% [64]	HRank-C1 [41]	74.13	1.17	62.99
	FPEI-R5 [43]	74.36	0.94	61.79
	FPEI-R4 with DR	74.72	0.58	44.31
	HRank-C2	71.13	4.17	53.71
	FPEI-R6	72.23	3.07	51.53
	FPEI-R7 with DR	73.62	1.68	37.88
	SFP [75]	74.18	1.12	61.74
	FPGM [40]	73.54	1.76	61.79
	PScratch [71]	74.75	0.55	49.95
	COP v2 [27]	74.97	0.33	44.79
	SCWC _(s=0.5) [51]	75.52	-0.22	77.20
	RED++ [76]	75.2	0.1	55.00
	SCWC _(s=0.4)	75.45	-0.15	72.60
	SCWC _(s=0.3)	75.35	-0.05	67.90
	SCWC _(s=0.2)	75.23	0.07	63.30
	SCWC _(s=0.1)	75.17	0.13	58.60
	ThiNet-70 [37,71]	74.03	1.27	67.10
	SSR [71]	72.47	2.83	47.80
	APoZ [40]	71.83	3.47	47.80
	LSTM-SEP [77]	74.4	0.9	57.00
	AOFP-C2 [77]	75.07	0.23	–

Table 7. Cont.

Base (%)	Method	Acc. (%)	Acc. Drop (%)	RPR (%)
	Adapt-DCP [78]	74.44	0.86	45.10
	ResNet-50-pruned-70 [14]	75.06	0.24	70.00
	ResNet-50-pruned-50 [14]	73.99	1.31	50.00
	IoT-Qi [39]	72.92	2.38	33.70
	SNACS [72]	74.65	0.65	44.90
	White-Box [73]	74.21	1.09	–
	CHWP [52]	74.95	0.35	–
	EACP ($k = 70\%$) [55]	73.95	1.35	45.30
	ResPrune [57]	76.15	−0.85	–
	FPWT [42]	72.82	2.48	31.80
	CIE [59]	74.06	1.24	–
	OCNNA	74.73	0.57	37.44

Table 8. Results of pruning MobileNet on the Imagenet dataset. RPR means the remaining parameters ratio, where the lower, the better. Acc. Drop is the accuracy loss after pruning, where the smaller, the better. MobileNet-0.75 means that every layer is 75% of the original one. COP-0.50 implies that 50% of filters are maintained. The best results are highlighted in bold.

Dataset	Base (%)	Algorithm	Acc. (%)	Acc. Drop (%)	RPR (%)
Imagenet	69.96	MobileNet-0.75	68.01	1.95	60.94
		MobileNet-0.50	63.29	6.67	36.29
		COP v1-0.70	68.52	1.44	59.31
		COP v1-0.40	64.38	5.58	28.99
		COP v2-0.70	69.02	0.94	57.09
		COP v2-0.40	65.33	4.63	29.81
		Adapt-DCP	69.58	0.38	66.73
		OCNNA	69.75	0.21	27.22

5. Sensibility Study

As we have seen, OCNNA is a parametric algorithm designed to simplify CNN models. It can be applied to any convolutional model (e.g., ResNet or VGG networks) without any adjustment, in contrast with other state-of-the-art approaches, such as FPEI [43], which presents different versions (FPEI, FPEI-R4 with DDR, FPEI-R5, FPEI-R6, and FPEI-R7 with DR) in order to ensure quality in prediction in different situations. OCNNA counts only with one parameter, k , which represents the k -th percentile of filters with higher importance, after applying transformations such as PCA, Frobenius norm, and CV. What effect does changing the value of k have in terms of accuracy and network simplification? It is illustrated through the experiment depicted in Figure 3. In this experiment, different values of k , ranging between 10 and 75, were used, and the resulting accuracy and final number of parameters were evaluated. As the k value increases, OCNNA boosts the reduction in the number of filters which will form part of the new model. In consequence, the number of parameters substantially drops. Nonetheless, accuracy also suffers a dramatically drop as k increases. Notice the remarkable drop between $k = 40$ (74.43% in accuracy) and $k = 50$ (46.31%). In fact, 40-th percentile is the best value of k , obtaining the highest possible accuracy bearing in mind the significant reduction in parameters.

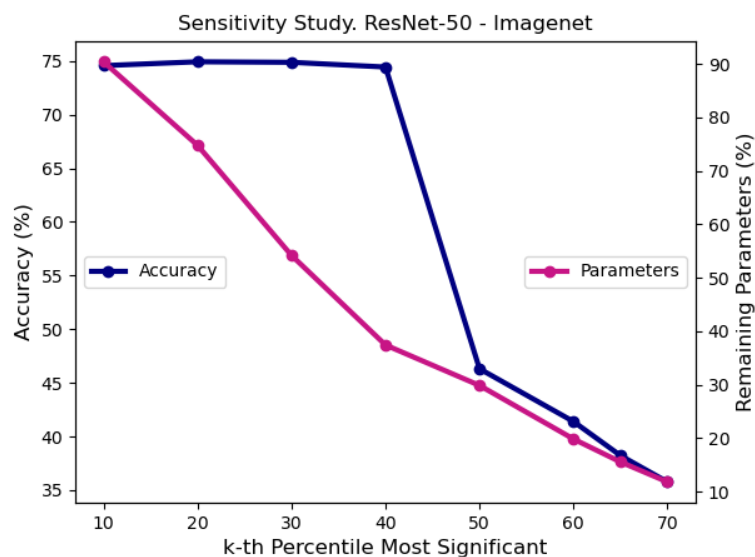


Figure 3. Sensitivity study of k percentile of significance for ResNet-50 and Imagenet dataset. The left Y-axis shows test accuracy, and the right Y-axis shows the remaining parameters ratio. The base accuracy is 75.4%. As we can see, when $k = 40$ (40-th percentile), we obtain a significant reduction in parameters (remaining 37.44%) with an accuracy drop of 0.57%.

6. Conclusions

Deep neural networks have emerged as the leading technique for tackling various challenging tasks in AI. In particular, CNNs have achieved an extraordinary success in a wide range of computer vision problems. However, these models come with high energy demands and are challenging to design efficiently.

In this paper, we introduce OCNNA, a novel CNN optimization and construction method based on pruning designed to assess the importance of convolutional layers, ordering the filters (features) by importance. Our approach enables the efficient end-to-end training and compression of CNNs. It is easy to apply and depends on a single parameter k , called percentile of significance, which represents the proportion of filters which will be transferred to the new model based on their importance. Only the k -th percentile of filters with higher values after applying the OCNNA process (PCA for feature selection, Frobenius norm for summary, and CV for measuring variability) will form part of the new optimized model.

OCNNA was evaluated through a comprehensive and detailed experimentation including the best known datasets (CIFAR-10, CIFAR-100, and Imagenet) and CNN architectures (VGG-16, ResNet-50, DenseNet-40, and MobileNet). The experimental results, based on the comparison with 20 state-of-the-art CNN simplification techniques and obtaining successful results, confirm that more efficient CNN models, following typical Green AI metrics, can be obtained with small accuracy losses. As a result, OCNNA is a competitive CNN construction method based on pruning which could ease the deployment of AI models onto edge devices (e.g., IoT devices) or other resource-limited devices.

Author Contributions: Conceptualization, L.B., M.L. and J.M.B.; methodology, L.B., M.L. and J.M.B.; software, L.B.; validation, M.L. and J.M.B.; formal analysis, L.B.; investigation, L.B.; resources, L.B.; data curation, L.B.; writing—original draft preparation, L.B.; writing—review and editing, M.L. and J.M.B.; visualization, L.B.; supervision, M.L. and J.M.B.; project administration, J.M.B.; funding acquisition, J.M.B. All authors have read and agreed to the published version of the manuscript.

Funding: This research was partially supported by the projects with references PID2020-118224RB-I00 and PID2023-151336OB-I00 granted by the Spanish Ministry of Science and Innovation, co-funded by the European Commission.

Data Availability Statement: The data used in the experimentation of this article are publicly available on the Internet. Specifically, ImageNet can be accessed at the following link: <https://www.image-net.org/challenges/LSVRC/2012/index.php>. Additionally, CIFAR-10 and CIFAR-100 can be accessed at the following link: <https://www.cs.toronto.edu/~kriz/cifar.html>.

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

Abbreviations

The following abbreviations are used in this manuscript:

CNN	convolutional neural network
OCNNA	Optimizing Convolutional Neural Network Architecture
DNN	deep neural network
FLOPs	Floating-point operations
OBD	Optimal Brain Damage
NN	neural networks
NEAT	NeuroEvolution of Augmenting Topologies
GA	Genetic Algorithm
NAS	neural architecture search
DRL	deep reinforcement learning
PCA	Principal Component Analysis
CV	Coefficient of Variation
ACC	accuracy
RPR	remaining parameters ratio

References

- Graves, A.; Schmidhuber, J. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Netw.* **2005**, *18*, 602–610. [[CrossRef](#)] [[PubMed](#)]
- Collobert, R.; Weston, J.; Bottou, L.; Karlen, M.; Kavukcuoglu, K.; Kuksa, P. Natural Language Processing (almost) from Scratch. *arXiv* **2011**, arXiv:1103.0398.
- Krizhevsky, A.; Sutskever, I.; Hinton, G.E. ImageNet Classification with Deep Convolutional Neural Networks. In Proceedings of the 26th Annual Conference on Neural Information Processing Systems 2012, Lake Tahoe, NV, USA, 3–6 December 2012; Volume 25.
- Zhao, H.; Yang, G.; Wang, D.; Lu, H. Deep mutualearning for visual object tracking. *Pattern Recognit.* **2021**, *112*, 107796. [[CrossRef](#)]
- Olmos, R.; Tabik, S.; Herrera, F. Automatic handgun detection alarm in videos using deepearning. *Neurocomputing* **2018**, *275*, 66–72. [[CrossRef](#)]
- Yu, Q.; Gao, Y.; Zheng, Y.; Zhu, J.; Dai, Y.; Shi, Y. Crossover-Net: Leveraging vertical-horizontal crossover relation for robust medical image segmentation. *Pattern Recognit.* **2021**, *113*, 107756. [[CrossRef](#)]
- Guo, J.; Chao, H. Building an end-to-end spatial-temporal convolutional network for video super-resolution. In Proceedings of the 31th AAAI Conference on Artificial Intelligence, San Francisco, CA, USA, 4–9 February 2017; pp. 4053–4060.
- Cai, L.; Gao, J.; Zhao, D. A review of the application of deepearning in medical image classification and segmentation. *Ann. Transl. Med.* **2020**, *8*, 713. [[CrossRef](#)]
- Han, S.; Pool, J.; Tran, J.; Dally, W.J. Learning both Weights and Connections for Efficient Neural Networks. *arXiv* **2015**, arXiv:1506.0262.
- Khouas, A.R.; Bouadjenek, M.R.; Hacid, H.; Aryal, S. Training Machine Learning Models at the Edge: A Survey. *arXiv* **2024**, arXiv:2403.02619.
- Strubell, E.; Ganesh, A.; McCallum, A. Energy and Policy Considerations for Deep Learning in NLP. *arXiv* **2019**, arXiv:1906.02243.
- Balderas, L.; Lastra, M.; Benítez, J.M. Optimizing dense feed-forward neural networks. *Neural Netw.* **2024**, *171*, 229–241. [[CrossRef](#)]
- Simonyan, K.; Zisserman, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv* **2015**, arXiv:1409.1556.
- Alqahtani, A.; Xie, X.; Jones, M.W.; Essa, E. Pruning CNN filters via quantifying the importance of deep visual representations. *Comput. Vis. Image Underst.* **2021**, *208–209*, 103220. [[CrossRef](#)]
- Tu, Y.; Lin, Y. Deep Neural Network Compression Technique Towards Efficient Digital Signal Modulation Recognition in Edge Device. *IEEE Access* **2019**, *7*, 58113–58119. [[CrossRef](#)]
- Bolón-Canedo, V.; Morán-Fernández, L.; Cancela, B.; Alonso-Betanzos, A. A review of green artificial intelligence: Towards a more sustainable future. *Neurocomputing* **2024**, *599*, 128096. [[CrossRef](#)]
- Denton, E.; Zaremba, W.; Bruna, J.; LeCun, Y.; Fergus, R. Exploiting linear structure within convolutional networks for efficient evaluation. In Proceedings of the Annual Conference on Neural Information Processing Systems 2014, Montreal, QC, Canada, 8–13 December 2014; pp. 1269–1277.

18. Yu, Z.; Shi, Y. Kernel Quantization for Efficient Network Compression. *IEEE Access* **2022**, *10*, 4063–4071. [[CrossRef](#)]
19. Gong, M.; Gao, Y.; Wu, Y.; Zhang, Y.; Qin, A.K.; Ong, Y.S. Heterogeneous Multi-Party Learning With Data-Driven Network Sampling. *IEEE Trans. Pattern Anal. Mach. Intell.* **2023**, *45*, 13328–13343. [[CrossRef](#)] [[PubMed](#)]
20. Krizhevsky, A. Learning Multiple Layers of Features from Tiny Images. 2009. Available online: <https://www.cs.toronto.edu/~kriz/cifar.html> (accessed on 15 May 2023).
21. Russakovsky, O.; Deng, J.; Su, H.; Krause, J.; Satheesh, S.; Ma, S.; Huang, Z.; Karpathy, A.; Khosla, A.; Bernstein, M.; et al. ImageNet large scale visual recognition challenge. *Int. J. Comput. Vis.* **2015**, *115*, 211–252. [[CrossRef](#)]
22. Schwartz, R.; Dodge, J.; Smith, N.A.; Etzioni, O. Green AI. *Commun. ACM* **2020**, *63*, 54–63. [[CrossRef](#)]
23. Stanley, K.O.; Miikkulainen, R. Evolving Neural Networks through Augmenting Topologies. *Evol. Comput.* **2002**, *10*, 99–127. [[CrossRef](#)]
24. 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 34th International Conference on Machine Learning, ICML'17, Sydney, Australia, 6–11 August 2017; Volume 70, pp. 2902–2911.
25. Real, E.; Aggarwal, A.; Huang, Y.; Le, Q.V. Regularized evolution for image classifier architecture search. *Proc. Conf. AAAI Artif. Intell.* **2019**, *33*, 4780–4789. [[CrossRef](#)]
26. Stanley, K.O.; Clune, J.; Lehman, J.; Miikkulainen, R. Designing neural networks through neuroevolution. *Nat. Mach. Intell.* **2019**, *1*, 24–35. [[CrossRef](#)]
27. Wang, W.; Yu, Z.; Fu, C.; Cai, D.; He, X. COP: Customized correlation-based Filter level pruning method for deep CNN compression. *Neurocomputing* **2021**, *464*, 533–545. [[CrossRef](#)]
28. Lu, Z.; Whalen, I.; Boddeti, V.; Dhebar, Y.; Deb, K.; Goodman, E.; Banzhaf, W. NSGA-Net: Neural Architecture Search Using Multi-Objective Genetic Algorithm. In Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '19, New York, NY, USA, 13–17 July 2019; pp. 419–427. [[CrossRef](#)]
29. Dong, X.; Liu, L.; Musial, K.; Gabrys, B. NATS-Bench: Benchmarking NAS Algorithms for Architecture Topology and Size. *IEEE Trans. Pattern Anal. Mach. Intell.* **2022**, *44*, 3634–3646. [[CrossRef](#)] [[PubMed](#)]
30. Liang, T.; Glossner, J.; Wang, L.; Shi, S.; Zhang, X. Pruning and quantization for deep neural network acceleration: A survey. *Neurocomputing* **2021**, *461*, 370–403. [[CrossRef](#)]
31. Jacob, B.; Kligys, S.; Chen, B.; Zhu, M.; Tang, M.; Howard, A.; Adam, H.; Kalenichenko, D. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. In Proceedings of the 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 8–23 June 2018; pp. 2704–2713. [[CrossRef](#)]
32. Bucilua, C.; Caruana, R.; Niculescu-Mizil, A. Model compression. In Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Philadelphia, PA, USA, 20–23 August 2006.
33. Hinton, G.; Vinyals, O.; Dean, J. Distilling the Knowledge in a Neural Network. *arXiv* **2015**, arXiv:1503.02531.
34. Wang, L.; Yoon, K. Knowledge Distillation and Student-Teacher Learning for Visual Intelligence: A Review and New Outlooks. *IEEE Trans. Pattern Anal. Mach. Intell.* **2022**, *44*, 3048–3068. [[CrossRef](#)]
35. Liu, Z.; Sun, M.; Zhou, T.; Huang, G.; Darrell, T. Rethinking the Value of Network Pruning. *arXiv* **2019**, arXiv:1810.05270.
36. Liu, Z.; Li, J.; Shen, Z.; Huang, G.; Yan, S.; Zhang, C. Learning Efficient Convolutional Networks through Network Slimming. In Proceedings of the 2017 IEEE International Conference on Computer Vision (ICCV), Venice, Italy, 22–29 October 2017; pp. 2755–2763. [[CrossRef](#)]
37. Luo, J.H.; Zhang, H.; Zhou, H.Y.; Xie, C.W.; Wu, J.; Lin, W. ThiNet: Pruning CNN Filters for a Thinner Net. *IEEE Trans. Pattern Anal. Mach. Intell.* **2019**, *41*, 2525–2538. [[CrossRef](#)]
38. Molchanov, P.; Tyree, S.; Karras, T.; Aila, T.; Kautz, J. Pruning Convolutional Neural Networks for Resource Efficient Inference. *arXiv* **2017**, arXiv:1611.06440.
39. Qi, C.; Shen, S.; Li, R.; Zhao, Z.; Liu, Q.; Liang, J.; Zhang, H. An efficient pruning scheme of deep neural networks for Internet of Things applications. *EURASIP J. Adv. Signal Process.* **2021**, *2021*, 31. [[CrossRef](#)]
40. Hu, H.; Peng, R.; Tai, Y.W.; Tang, C.K. Network Trimming: A Data-Driven Neuron Pruning Approach towards Efficient Deep Architectures. *arXiv* **2016**, arXiv:1607.03250.
41. Lin, M.; Ji, R.; Wang, Y.; Zhang, Y.; Zhang, B.; Tian, Y.; Shao, L. HRank: Filter Pruning using High-Rank Feature Map. *arXiv* **2020**, arXiv:2002.10179.
42. Liu, Y.; Fan, K.; Zhou, W. FPWT: Filter pruning via wavelet transform for CNNs. *Neural Netw.* **2024**, *179*, 106577. [[CrossRef](#)] [[PubMed](#)]
43. Wang, J.; Jiang, T.; Cui, Z.; Cao, Z. Filter pruning with a feature map entropy importance criterion for convolution neural networks compressing. *Neurocomputing* **2021**, *461*, 41–54. [[CrossRef](#)]
44. He, Y.; Kang, G.; Dong, X.; Fu, Y.; Yang, Y. Soft filter pruning for accelerating deep convolutional neural networks. *arXiv* **2018**, arXiv:1808.06866.
45. 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 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Long Beach, CA, USA, 15–20 June 2019; pp. 4335–4344. [[CrossRef](#)]
46. Wang, Y.; Zhang, X.; Xie, L.; Zhou, J.; Su, H.; Zhang, B.; Hu, X. Pruning from Scratch. *arXiv* **2019**, arXiv:1909.12579. [[CrossRef](#)]

47. Yeom, S.K.; Seegerer, P.; Lapuschkin, S.; Binder, A.; Wiedemann, S.; Müller, K.R.; Samek, W. Pruning by explaining: A novel criterion for deep neural network pruning. *Pattern Recognit.* **2021**, *115*, 107899. [[CrossRef](#)]
48. Aradhya, A.M.; Ashfahani, A.; Angelina, F.; Pratama, M.; de Mello, R.F.; Sundaram, S. Autonomous CNN (AutoCNN): A data-driven approach to network architecture determination. *Inf. Sci.* **2022**, *607*, 638–653. [[CrossRef](#)]
49. Hirsch, L.; Katz, G. Multi-objective pruning of dense neural networks using deep reinforcement learning. *Inf. Sci.* **2022**, *610*, 381–400. [[CrossRef](#)]
50. Fernandes, F.E., Jr.; Yen, G.G. Pruning Deep Convolutional Neural Networks Architectures with Evolution Strategy. *Inf. Sci.* **2021**, *552*, 29–47. [[CrossRef](#)]
51. Li, G.; Zhang, M.; Wang, J.; Weng, D.; Corporaal, H. SCWC: Structured channel weight sharing to compress convolutional neural networks. *Inf. Sci.* **2022**, *587*, 82–96. [[CrossRef](#)]
52. Geng, X.; Gao, J.; Zhang, Y.; Xu, D. Complex hybrid weighted pruning method for accelerating convolutional neural networks. *Sci. Rep.* **2024**, *14*, 5570. [[CrossRef](#)]
53. Sharma, M.; Heard, J.; Saber, E.; Markopoulos, P.P. Convolutional Neural Network Compression via Dynamic Parameter Rank Pruning. *arXiv* **2024**, arXiv:2401.08014.
54. Jiang, P.; Xue, Y.; Neri, F. Convolutional neural network pruning based on multi-objective feature map selection for image classification. *Appl. Soft Comput.* **2023**, *139*, 110229. [[CrossRef](#)]
55. Liu, Y.; Wu, D.; Zhou, W.; Fan, K.; Zhou, Z. EACP: An effective automatic channel pruning for neural networks. *Neurocomputing* **2023**, *526*, 131–142. [[CrossRef](#)]
56. Louati, H.; Louati, A.; Bechikh, S.; Kariri, E. Joint filter and channel pruning of convolutional neural networks as a bi-level optimization problem. *Memetic Comput.* **2024**, *16*, 71–90. [[CrossRef](#)]
57. Jayasimhan, A.; Pabitha, P. ResPrune: An energy-efficient restorative filter pruning method using stochastic optimization for accelerating CNN. *Pattern Recognit.* **2024**, *155*, 110671. [[CrossRef](#)]
58. Zhang, P.; Tian, C.; Zhao, L.; Duan, Z. A multi-granularity CNN pruning framework via deformable soft mask with joint training. *Neurocomputing* **2024**, *572*, 127189. [[CrossRef](#)]
59. Lian, Y.; Peng, P.; Jiang, K.; Xu, W. Cross-layer importance evaluation for neural network pruning. *Neural Netw.* **2024**, *179*, 106496. [[CrossRef](#)]
60. Kurita, T. Principal component analysis (PCA). In *Computer Vision: A Reference Guide*; Springer: Cham, Switzerland, 2019; pp. 1–4.
61. Golub, G.; Van Loan, C. *Matrix Computations*, 3rd, ed.; JHU Press: Baltimore, MD, USA, 1996.
62. Frobenius Norm—From Wolfram MathWorld. Available online: <https://mathworld.wolfram.com/FrobeniusNorm.html> (accessed on 25 April 2023).
63. Everitt, B.; Skrondal, A. *The Cambridge Dictionary of Statistics B.S. Everitt Aut; A. Skrondal Aut*; Cambridge University Press: Cambridge, UK, 2011.
64. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. In Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 27–30 June 2016.
65. Huang, G.; Liu, Z.; van der Maaten, L.; Weinberger, K.Q. Densely Connected Convolutional Networks. *arXiv* **2018**, arXiv:1608.06993.
66. 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.
67. Recht, B.; Roelofs, R.; Schmidt, L.; Shankar, V. Do ImageNet Classifiers Generalize to ImageNet? In Proceedings of the 36th International Conference on Machine Learning, Long Beach, CA, USA, 9–15 June 2019; Volume 97, pp. 5389–5400.
68. imagenet_v2 | TensorFlow Datasets—tensorflow.org. 2022. Available online: https://www.tensorflow.org/datasets/catalog/imagenet_v2 (accessed on 10 May 2023).
69. Tensorflow. *Image Data Generator v2.14.1*; Tensorflow. 2022. Available online: https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator (accessed on 20 May 2023).
70. Guillemot, M.; Heusele, C.; Korichi, R.; Schnebert, S.; Chen, L. Breaking Batch Normalization for better explainability of Deep Neural Networks through Layer-wise Relevance Propagation **2020**. [[arXiv:cs.LG/2002.11018](#)].
71. Lin, S.; Ji, R.; Li, Y.; Deng, C.; Li, X. Toward Compact ConvNets via Structure-Sparsity Regularized Filter Pruning. *IEEE Trans. Neural Netw. Learn. Syst.* **2020**, *31*, 574–588. [[CrossRef](#)] [[PubMed](#)]
72. Ravi Ganesh, M.; Blanchard, D.; Corso, J.J.; Sekeh, S.Y. Slimming Neural Netw. Using Adaptive Connectivity Scores. *IEEE Trans. Neural Netw. Learn. Syst.* **2024**, *35*, 3794–3808. [[CrossRef](#)] [[PubMed](#)]
73. 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.* **2023**, *34*, 7946–7955. [[CrossRef](#)]
74. Liu, G.; Zhang, K.; Lv, M. SOKS: Automatic Searching of the Optimal Kernel Shapes for Stripe-Wise Network Pruning. *IEEE Trans. Neural Netw. Learn. Syst.* **2023**, *34*, 9912–9924. [[CrossRef](#)]
75. He, Y.; Zhang, X.; Sun, J. Channel Pruning for Accelerating Very Deep Neural Networks. *arXiv* **2017**, arXiv:1707.06168.
76. Yvinec, E.; Dapogny, A.; Cord, M.; Bailly, K. RED++ : Data-Free Pruning of Deep Neural Networks via Input Splitting and Output Merging. *IEEE Trans. Pattern Anal. Mach. Intell.* **2023**, *45*, 3664–3676. [[CrossRef](#)]

-
77. Ding, G.; Zhang, S.; Jia, Z.; Zhong, J.; Han, J. Where to Prune: Using LSTM to Guide Data-Dependent Soft Pruning. *IEEE Trans. Image Process.* **2021**, *30*, 293–304. [[CrossRef](#)]
 78. 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](#)]

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.