mathematics

MDPI

Article

Structured Compression of Convolutional Neural Networks for Specialized Tasks

Freddy Gabbay [1,*], Benjamin Salomon [1] and Gil Shomron [2]

[1] Engineering Faculty, Ruppin Academic Center, Emek Hefer 4025000, Israel
[2] NVIDIA
* Correspondence: freddyg@ruppin.ac.il

Abstract: Convolutional neural networks (CNNs) offer significant advantages when used in various image classification tasks and computer vision applications. CNNs are increasingly deployed in environments from edge and Internet of Things (IoT) devices to high-end computational infrastructures, such as supercomputers, cloud computing, and data centers. The growing amount of data and the growth in their model size and computational complexity, however, introduce major computational challenges. Such challenges present entry barriers for IoT and edge devices as well as increase the operational expenses of large-scale computing systems. Thus, it has become essential to optimize CNN algorithms. In this paper, we introduce the S-VELCRO compression algorithm, which exploits value locality to trim filters in CNN models utilized for specialized tasks. S-VELCRO uses structured compression, which can save costs and reduce overhead compared with unstructured compression. The algorithm runs in two steps: a preprocessing step identifies the filters with a high degree of value locality, and a compression step trims the selected filters. As a result, S-VELCRO reduces the computational load of the channel activation function and avoids the convolution computation of the corresponding trimmed filters. Compared with typical CNN compression algorithms that run heavy back-propagation training computations, S-VELCRO has significantly fewer computational requirements. Our experimental analysis shows that S-VELCRO achieves a compression-saving ratio between 6% and 30%, with no degradation in accuracy for ResNet-18, MobileNet-V2, and GoogLeNet when used for specialized tasks.

Keywords: machine learning; deep neural networks; convolutional neural network; structured compression

MSC: 68T07

Citation: Gabbay, F.; Salomon, B.; Shomron, G. Structured Compression of Convolutional Neural Networks for Specialized Tasks. *Mathematics* **2022**, *10*, 3679. https://doi.org/10.3390/math10193679

Academic Editor: Jonathan Blackledge

Received: 8 September 2022
Accepted: 4 October 2022
Published: 8 October 2022

1. Introduction

The usage of convolution neural networks (CNNs) is continuously growing in computer vision applications ranging from IoT and edge devices to supercomputers and cloud infrastructures. Over the years, new CNNs have been introduced that can achieve remarkable prediction accuracy for different classification tasks. The new models, however, are increasingly complex and require more and more processing throughput for both model inference and training. For example, the ResNet-101 model [1] introduced in 2015 used 101 layers and required nearly sevenfold greater computational throughput [2] than the AlexNet model [3], which was introduced in 2012 and had eight layers. In conjunction with the processing complexity, the growing number of model layers introduces a major challenge for real-time computer vision applications due to the increasing inference latency. These challenges have become major entry barriers for CNNs in IoT and edge devices because they are limited in performance, memory footprint, cost, and energy. CNNs also introduce a significant increase in power consumption and cost for high-end computer systems because they require special accelerators such as graphics processing units (GPUs)

or tensor processing units (TPUs) to handle their complex algorithms. Thus, the need to optimize CNN algorithms without excessive performance loss has become essential to enable them to fit within a system's envelope of cost, performance, memory, and energy. Different approaches, such as pruning [4–7] and quantization [8–11], have been introduced to optimize CNN models. In many cases, the optimization of these techniques involves model retraining and complex back-propagation processing. We discuss these approaches and other techniques in detail in Section 2.

In this paper, we examine the applications of CNN models used for specialized tasks. Whereas general-purpose CNNs are used for a broad range of classification tasks, specialized CNNs are optimized and tuned to handle a small set of specific tasks. Specialized CNN applications are an emerging field in machine learning for computing systems that range from edge devices to high-end large-scale systems [12–14], and several applications have been reported recently [15–17]. In Ref. [13], a specialized CNN usage is presented for offline video analytics that is performed in a hierarchical manner. A specialized lightweight CNN model is used at the first level of the classification process, and only low-confidence classifications are moved to the second-level general-purpose CNN.

Figure 1a illustrates such an application of a specialized CNN for vehicle detection on highways. In this example, the real-time video analytic application employs a compressed CNN model that is specialized in vehicle detection and is derived from a general-purpose CNN. Similar hierarchical specialized CNN approaches have been used for image classification because image datasets are typically organized in hierarchical classes. In this case, a different specialized CNN can be used for every hierarchy with the needed classification specialty. In Ref. [18], an ensemble of specialized expert models is presented, where each model is an expert in a specific classification task. Figure 1b illustrates such an ensemble of expert CNN models. The ensemble is governed by a gating network that selects one or multiple expert CNNs based on the input image. The gating network assigns a weight to every expert CNN selected, while unselected experts are assigned a weight of 0. A similar approach called cascaded-CNN is presented in Ref. [19]. In that case, each CNN is optimized for specialized tasks, and the classification of all models is combined to produce a complete prediction map. Additional examples also exist in game-scrapping applications [13].

In this paper, we extend our previous work [20] and introduce an enhanced novel algorithm for the **structured** compression of CNNs that are employed for specialized tasks. The new compression algorithm, S-VELCRO, is based on structured value-locality compression. S-VELCRO exploits the property of value locality, which was introduced in [20]. With this property, specialized task CNNs produce a proximal range of values by activating functions in the inference process. Our prior work [20] introduced the value-locality-based compression (VELCRO) algorithm, which exploits value locality for **unstructured** model compression. VELCRO trims activation function elements without a predetermined constraint related to the network structure. Such unstructured compression methods, however, may involve overhead to compute addresses and indices of compressed elements and store them in dedicated metadata storage elements. In particular, such limitations become more taxing when the unstructured compression runs on GPUs and TPUs. As illustrated in Figure 2, S-VELCRO overcomes the limitations of VELCRO by performing structured compression and trimming model filters and their corresponding activation kernels.

To do so, S-VELCRO runs in two steps: a preprocessing stage identifies the filters with a high degree of value locality, and a compression step trims the selected filters. Unlike common CNN compression algorithms, S-VELCRO does not require any back-propagation training and thus has significantly fewer computational requirements. Our experimental environment examines S-VELCRO's capabilities on three CNN models—ResNet-18 [2], MobileNet V2 [21], and GoogLeNet [22]—using the ILSVRC-2012 (ImageNet) [23] dataset. In addition, we implement S-VELCRO on a hardware platform to measure its computational and energy savings.

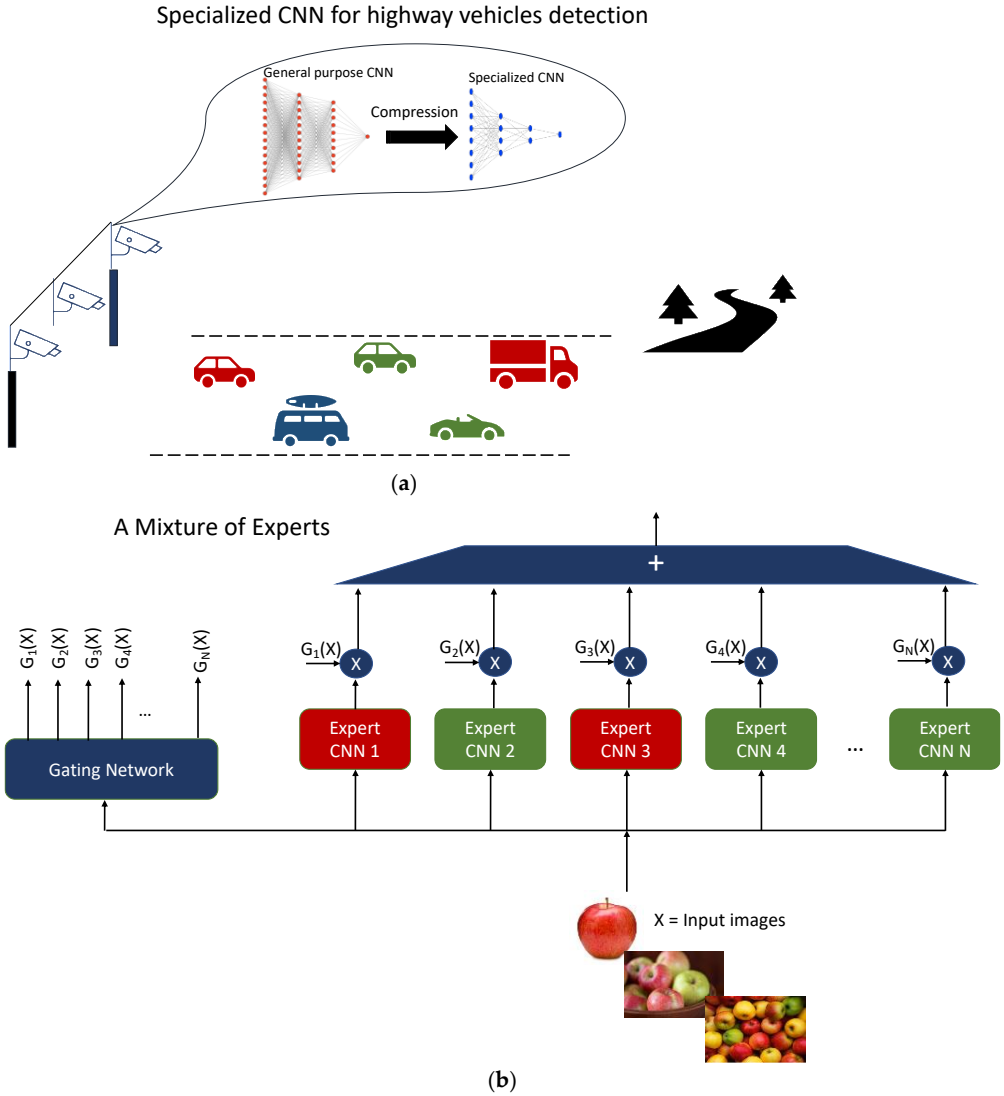


Figure 1. Illustrative examples of specialized CNNs: (**a**) specialized CNN for highway vehicle detection and (**b**) a mixture of expert CNNs.

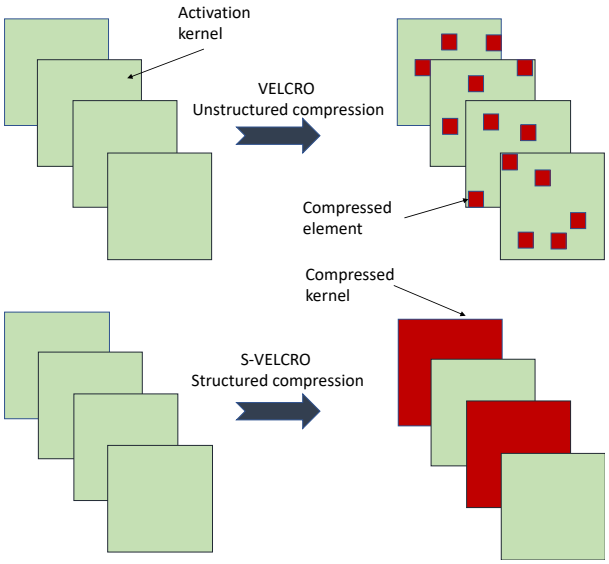


Figure 2. Structured compression versus unstructured compression.

The contributions of this paper are summarized as follows:

1. We present a novel algorithm, S-VELCRO, that exploits value locality to trim filters in CNN models utilized for specialized tasks. S-VELCRO offers **structured** compression that can save costs and overhead over unstructured compression: (i) addressing overhead [24] for computing addresses and indices of compressed elements and (ii) storing them in dedicated metadata.
2. S-VELCRO presents a fast compression process that solely uses statistics gathering through the inference process and avoids the heavy computations required for the back-propagation training used by traditional compression approaches, such as pruning.
3. The results of our experiments indicate that S-VELCRO produces a compression-saving ratio of computations in the range of 24.60–27.84% for ResNet-18, 6.07–11.27% for GoogLeNet, and 13.35–17.33% for MobileNet V2, with no impact on model accuracy. In addition, our experimental analysis indicates that S-VELCRO can save 37.32–44.92% of the filter memory footprint for ResNet-18, 15.59–22.37% for MobileNet V2, and 10.01–15.94% for GoogLeNet.
4. We demonstrate the energy savings of S-VELCRO by implementing the compression algorithm on a hardware platform using a field-programmable gate array (FPGA) for ResNet-18. Our experimental results indicate a 24–27% reduction in energy consumption with S-VELCRO.

The remainder of this paper is organized as follows: Section 2 reviews previous work. Section 3 introduces the proposed method and algorithm. Section 4 presents the experimental results. Finally, Section 5 summarizes this study's conclusions.

2. Prior Work

CNN models typically require large amounts of storage and have high computational costs. Therefore, numerous model compression and acceleration methods have been proposed in recent years [25,26]. Approaches seek to optimize CNN computations by reducing redundancy, computational complexity, and memory storage. Such savings are critical for some real-time applications and in the deployment of CNN models on portable devices. In this section, we describe several related methods: model pruning and quantization, deep compression (a combination of pruning, quantization, and Huffman coding), knowledge distillation, CNN design for specialized inference tasks, and filter compression methods.

2.1. Pruning

Model pruning [4–6] is an effective approach to reducing model complexity and addressing the over-fitting problem. Taking inspiration from neuroscience studies, pruning tries to eliminate unimportant or redundant CNN parameters that are not sensitive to the performance of the model. Pruning techniques have been extensively studied [4,27–31], and the core idea is to remove redundant weights, neurons, kernels, or filters with minimum accuracy loss at the inference stage of a trained network. Therefore, the use of pruning can result in a smaller network and fewer computations. Traditional pruning algorithms consist of training a large model, pruning, and then fine-tuning the pruned model, which may incur high computational complexity [32].

CNN pruning techniques can be classified as unstructured or structured. In unconstructed pruning, elements such as weights and neurons are removed without consideration of the network structure. Typically, less important weights or activations are replaced by zeros independently. In structured pruning [24], the pruning process is restricted by sparsity constraints and removes structured parts (e.g., filters or layers). Unstructured pruning may fully exploit redundancy in the network, but it may require a special format to represent the pruned elements, and speedup may require a particular software/hardware accelerator for sparse matrix multiplications. Structured pruning may limit the redundancy that

can be exploited in the network, but the speedup is typically well-supported by various off-the-shelf libraries.

Pruning redundant, noninformative elements is usually done in accordance with their importance and contribution to the network's accuracy. The process of pruning is based on ranking the network elements according to various metrics—e.g., the L1 or L2 norms [5,33–36] of weights, activations, and filters or the saliency of two weight sets [37]. Dynamic mechanisms are required to prune activations because their importance may depend on the model's input. Reinforcement learning is used to prune channels in Ref. [38], and spatial correlations of CNN output feature maps (OFMs) are used to predict and prune zero-valued activations in [39,40]. Model size reduction and inference time speed-up have also been demonstrated by various pruning techniques based on weight magnitudes [27,41,42].

Gradual pruning methods [43] attempt to arrive at an accurate model given a resource-constrained environment (e.g., a model's memory footprint). In Ref. [44], the authors propose neuron importance score propagation (NISP) to jointly prune neurons in the entire network based on a unified goal. In Ref. [45], the authors prune neurons randomly, and random grouping of connection weights into hash buckets was proposed in Ref. [46]. Ref [47] proposed a new iterative pruning scheme based on Taylor expansion while focusing on transfer learning. Their results indicate that CNNs can be pruned by iteratively removing the least important OFMs and that a Taylor expansion-based criterion demonstrates improvement over other criteria such as weight pruning, using l2 norm, and activation pruning, using mean, variance, and mutual information. In that study, large trained CNNs were adapted to efficient smaller networks for specialized tasks (specialized in a subset of classes). The authors observed that every layer has both high-and low-degree important OFMs and that the median importance of OFMs tends to decrease with later depth [47]. In Ref. [48], the authors proposed the CURL method (compression using residual connections and limited data), which consists of compression of residual blocks and a label refinement strategy for small datasets.

While pruning techniques attempt to eliminate redundant network parameters, quantization approaches, which are described in the next subsection, attempt to employ an efficient data representation for the model parameters.

2.2. Quantization

Quantization methods compress the network by reducing the number of bits used to represent each weight, filter, and feature map. Typically, CNNs apply 32-bit floating-point precision. Several works [8–11] introduced fixed-point and vector quantization methods with a trade-off between accuracy and compression. In Ref. [8], Vanhouchke et al. demonstrated that 8-bit quantization of the parameters can result in significant compression without significant loss of accuracy. However, quantization methods that use fewer than 8 bits tend to significantly decrease the model accuracy.

Quantization schemes for weights and activations during training [49–51] try to reduce quantization errors under low precision while achieving accuracy comparable to full precision networks. This approach may be limited by a lack of data or computational resources. Aggressive quantization generally causes a significant loss of accuracy. Post-training quantization methods [52–55] circumvent this limitation. For example, the authors in Ref. [53] proposed a minimum square error problem for weights, and sparsity-aware quantization was introduced in Ref. [55].

2.3. Advanced Compression Methods

While pruning and quantization attempt to remove redundancy in the network structure and its data representation, advanced compression methods have been proposed to further optimize CNNs by adopting different approaches that are described in this subsection.

Deep compression, a combination of pruning, trained quantization, and Huffman coding, was proposed in Ref. [28]. This method includes pruning that removes redundant units or channels while weight and activation quantization occurs simultaneously.

Knowledge distillation [56–58] effectively trains a small (student) model from a large (teacher) model without a significant loss of accuracy. The student model should mimic the teacher model, and the problem is determining how to distill the knowledge from a larger CNN into a small network. Knowledge distillation systems consist of three main elements: knowledge, an algorithm for knowledge distillation, and a teacher-student architecture. A comprehensive survey of knowledge distillation is available in Ref. [58].

Training one large network and specializing it without additional training for efficient deployment was proposed in Ref. [59]. That work introduced the once-for-all (OFA) approach that supports diverse architectural constraints (e.g., power, cost, latency, and performance). A progressive shrinking algorithm reduces the model size by operating across four dimensions (image resolution, depth, width, and kernel size). Many specialized subnetworks can be easily obtained from the OFA network with accuracy similar to training them independently [47].

FoldedCNN [14] is another approach to CNN design for specialized inference tasks. FoldedCNN increases inference throughput and hardware utilization of specialized CNNs beyond increased batch size. This approach does not compress CNN models but rather increases arithmetic intensity, which boosts utilization and throughput when it runs on certain accelerators.

2.4. CNN Insights

Several CNN insight explorations have extended the understanding of CNNs' internal mechanisms and their relation to feature extraction. In Ref. [60], a visualization technique has been introduced that provides insight into the internal mechanisms of CNNs and can be used to select effective architectures. An additional simpler visualization method to estimate the receptive fields of units in each layer has also been suggested by Ref. [61]. Their results reveal that OFMs have interpretable patterns and extract features at several levels of abstraction (edges, textures, shapes, concepts, etc.). Another study [62] used ablation analysis and the addition of noise to quantify the contribution of OFM units and their role in the network's output. Their experiments suggest that highly class-selective elements may degrade network performance, so their removal may not impact overall performance.

Further study of the importance of individual units in CNNs found that, while removal of individual OFMs may not significantly decrease the overall model accuracy, it can significantly impact the accuracy of specific classes [63]. Thus, the ablation experiments in Ref. [63] demonstrated that individual units specialize in subsets of classes, and different methods were proposed to measure the contribution of individual OFMs to classification accuracy.

CNN filter compression techniques attempt to remove kernels and filters that correspond to unimportant weights. In Ref. [64], the authors proposed removing filters based on their importance. The coupling factors between consecutive layers are computed and used to remove unimportant pathways in the networks. These factors are used to maximize the variance of feature maps and to preserve the most relevant filters. Another study on CNN filter compression [65] showed that information richness and sparsity can be used to determine the importance of feature maps. The relationship between input feature maps and 2D kernels was examined, and the kernel sparsity and entropy (KSE) indicator was proposed for measuring the feature map importance. The authors proposed compressing CNNs by reducing the number of kernels based on the KSE indicator [65]. A common thread is the observation that feature maps may contribute differently to the accuracy. Hence, being able to quantify their importance is helpful for network compression. Additionally, the first model layers extract simple features such as edges, corners, and simple textures, and the last layers have higher representations (e.g., recognize objects).

The recent studies presented in Refs. [60–65] have provided the motivation for the VELCRO algorithm for neural networks [20]. The core idea is that we can remove unimportant computations when using the CNN mode for specialized tasks. Thus, compression and speedup are achieved with minimal accuracy degradation. VELCRO identifies output elements of the activation function with a high degree of value locality and replaces these elements with their corresponding average arithmetic values. Thus, it reduces computational load and performs unstructured compression of the network while avoiding a highly complex training process.

3. Method and Algorithm

The S-VELCRO compression algorithm introduced in this study leverages the property of value locality, which was introduced in Ref. [20]. That study showed that when CNNs are used for specialized tasks, the output values of the activation tensor are in the proximity of the inferred inputs. In addition, the authors used the variances tensor to quantify value locality, which is illustrated in Figure 3 and Equation (1). Figure 3 shows the activation function output tensors in a layer k and channel c for the CNN model. The set of elements $A^{(m)}[k][c][i][j]$ in the activation tensor represent the values calculated by the CNN model using the specialize set of images m = 0, 1, . . . , N−1. The variance tensor V[k] for layer k is defined for each element V[k][c][i][j] as

$$\begin{aligned} V[k][c][i][j] &= \mathrm{Var}(A[k][c][i][j]) = E(A[k][c][i][j]^2) - E(A[k][c][i][j])^2 \\ &= \tfrac{1}{N}\sum_{m=0}^{N-1} A^{(m)}[k][c][i][j]^2 - (\tfrac{1}{N}\sum_{m=0}^{N-1} A^{(m)}[k][c][i][j])^2, \end{aligned} \tag{1}$$

where c is the channel index and i and j are the element coordinates.

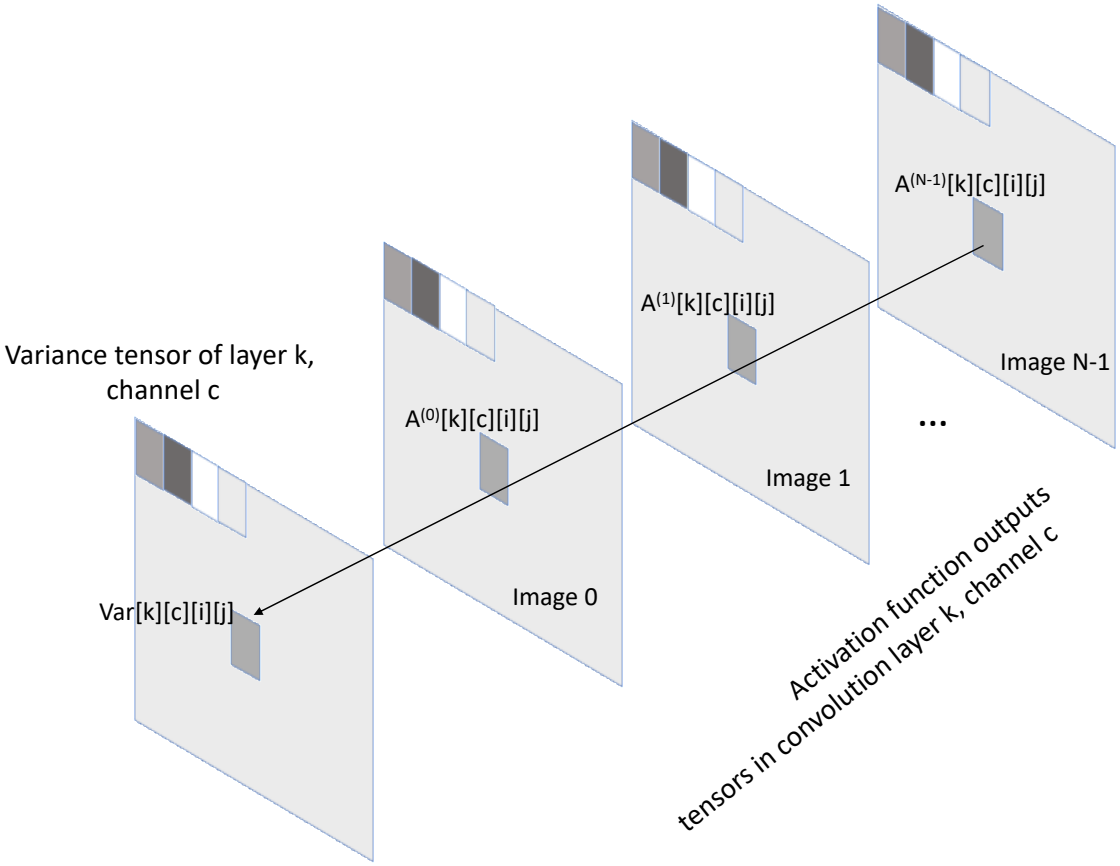


Figure 3. Value locality measured by the variance tensor V.

The variance tensor is a fundamental metric used by the S-VELCRO algorithm to leverage value locality through the structured compression process of specialized task CNNs. The S-VELCRO algorithm trims convolution filters that produce activation kernels with a high degree of value locality in CNN models for specialized tasks and thereby eliminates the need to compute the kernel that corresponds to the trimmed filter. The S-VELCRO is run in two stages:

1. Preprocessing stage: The S-VELCRO algorithm applies the uncompressed CNN model to calculate inference for a small group of images from the specialized task preprocessing dataset. Note that the dataset used for the preprocessing stage is distinct from the validation dataset. The variance tensor is calculated using Equation (1) for each convolution layer. Because the preprocessing stage relies only on inference, it requires significantly smaller computational overhead than common approaches, which employ a lengthy back-propagation training step [66].
2. Compression stage: The compression stage is provided with a hyperparameter tuple $T = \{T_0, T_1, T_2, \dots\}$, where each element in the tuple corresponds to the number of channels to be compressed in the corresponding convolution layer. For example, T_k represents the number of channels that will be compressed in convolution layer k. The compression stage processes every convolution layer separately. First, it calculates the rank of every channel by summing all variance elements in the variance tensor with indices that correspond to the channel indices. Next, it selects the T_k channels with the smallest rank to be compressed in convolution layer k. All compressed channels in the activation output function are replaced by the arithmetic average constant of the elements located at the same corresponding coordinates. All other activation elements remain unchanged. This channel compression process avoids both the channel activation function computation and the convolution of the corresponding trimmed filters. The hyperparameter tuple, T, determines the compression savings of each layer as well as the overall compression-saving ratio C for the model:

$$C = 1 - \frac{\text{Compressed model computations}}{\text{Original model computations}} = \sum_{k=0}^{K-1} T_k \times F_k \times w_k \times h_k, \quad (2)$$

where the tuple $T = \{T_0, T_1, \dots, T_K\}$ represents the number of channels to be compressed, F_k is the filter size in layer k, and w_k and h_k are the channel width and height of the activation function output tensor for convolution layer k, respectively.

The complete and formal definition of the algorithm is given in Algorithm 1. An example of the compression algorithm is depicted in Figure 4. Figure 4a illustrates the calculation of the variance tensor in the preprocessing stage for N = 3 images, and the dimensions of the activation tensor are $c_k = 3$, $w_k = 3$, and $h_k = 3$, where c_k represents the number of channels in layer k. The S-VELCRO preprocessing stage performs inference on the preprocessing dataset to create a variance tensor V[k]. Figure 4b illustrates the compression stage. The hyperparameter T_k for layer k is defined in this example as $T_k = 1$, which means that only one channel will be compressed. As illustrated in Figure 4b, the ranks r[0], r[1], and r[2] are calculated for every channel, and the channel with the smallest rank is compressed. In the illustrated example, channel 2 has the lowest rank, and its activation function outputs are replaced with their arithmetic average. The remaining elements remain unchanged with their original model activation function. The outcome of the S-VELCRO compression stage is given by the compressed activation-function output tensor $\widetilde{A}[k]$, where the computation of channel 2 (highlighted in green) is replaced by the arithmetic averages.

Algorithm 1: S-VELCRO algorithm for specialized CNNs.

Input: A CNN model M with K activation-function outputs (each in a different convolution layer), N preprocessing images, and a channel trimming hyperparameter tuple $T = \{T_0, T_1, \ldots, T_K\}$, where $\forall\ 0 \le k < N\ 0 \le T_k < c_k$ and c_k is the number of channels in convolution layer k.

Output: A compressed CNN model M_C.

Preprocessing stage (variance tensor calculation):

Step 1: Let A(k) be the activation-function output tensor in convolution layer k, and let $A^{(m)}(k)$ be the corresponding activation-tensor values at the inference of image m, $0 \le m < N$, where the tensors A[k] and $A^{(m)}[k]$ have the same dimensions and c_k, w_k, and h_k are the number of channels, the width, and the height of the tensors at convolution layer k, respectively, for $0 \le k < K$.

Step 2: Let model M perform inference for each image m such that $0 \le m < N$, and let V[k] be the variance tensor and S[k] be the average tensor of convolution layer k ($0 \le k < K$) such that each element in the tensors V is defined as described in Equation (1), and S is defined as

$$S[k][c][i][j] = \frac{1}{N}\sum_{m=0}^{N-1} A(m)[k][c][i][j]$$

$$V[k][c][i][j] = \frac{1}{N}\sum_{m=0}^{N-1} (A(m)[k][c][i][j])^2 - \left(\frac{1}{N}\sum_{m=0}^{N-1} A(m)[k][c][i][j]\right)^2$$

for each $0 \le c < c_k$, $0 \le i < w_k$ and $0 \le j < h_k$.

Compression stage (filter trimming):

Step 3: For each convolution layer $0 \le k < K$ and for each $0 \le c < c_k$, let $r[c]_k$ be the rank of channel c in convolution layer k:

$$r[c]_k = \sum_{i=0}^{w_k}\sum_{j=0}^{h_k} V[k][c][i][j].$$

Let r_k be the rank tuple of convolution layer k:

$$r_k = \{r[0]_k, r[1]_k, \ldots, r[c-1]_k\} \text{ for } 0 \le k < K.$$

Let $\widetilde{r}_k$ be the sorted tuple of r_k in a monotone increasing order:
$\widetilde{r}_k = \{\widetilde{r}[0]_k, \widetilde{r}[1]_k, \ldots, \widetilde{r}[c-1]_k\}$ for $0 \le k < K$ such that

$$\widetilde{r}[0]_k \le \widetilde{r}[1] \le, \ldots, \le \widetilde{r}[c-1].$$

Let $\bar{r}_k = \begin{cases} \{\widetilde{r}[0]_k, \widetilde{r}[1]_k, \ldots, \widetilde{r}[T_k-1]_k\}, & T_k > 0 \\ \varnothing, & T_k = 0 \end{cases}$
for $0 \le k < K$ such that $\bar{r}_k \subseteq \widetilde{r}_k$.

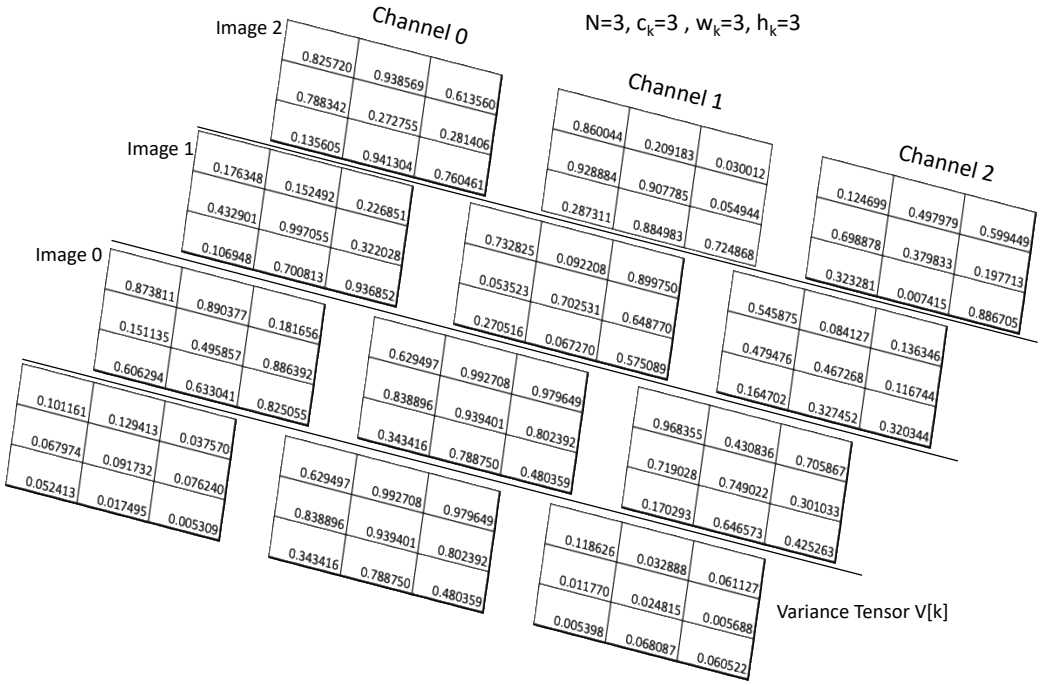
Let $I_k = \begin{cases} \{I[0]_k, I[1]_k, \ldots, I[T_k-1]_k\}, & T_k > 0 \\ \varnothing, & T_k = 0 \end{cases}$ for $0 \le k < K$
where $I[l]_k$ is the corresponding index of $\widetilde{r}[l]_k$ in the tuple
$r_k = \{r[0]_k, r[1]_k, \ldots, r[c-1]_k\}$ for $0 \le k < K$ and $0 \le l < T_k$.
That is, $r[I[l]_k] = \widetilde{r}[l]_k$.

Let the tensor $\widetilde{A}[k]$ be

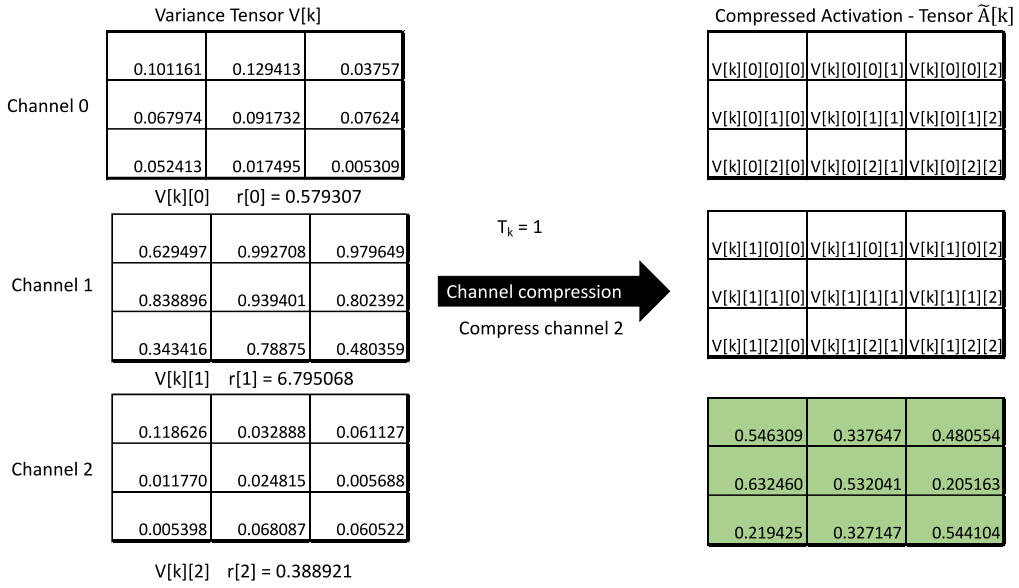
$$\widetilde{A}[k][c][i][j] = \begin{cases} A[k][c][i][j] & c \notin I_k \\ S[k][c][i][j] & c \in I_k \end{cases}$$

for each $0 \le c < c_k$, $0 \le i < w_k$, and $0 \le j < h_k$.

Step 4: Let the compressed CNN model M_C be such that every activation function output tensor A[k] is replaced with $\widetilde{A}[k]$ for every convolution layer $0 \le k < K$.



(**a**)



(**b**)

Figure 4. Example of S-VELCRO (**a**) preprocessing and (**b**) compression stages.

During the compression phase, the threshold values of the tuples T that produce the optimal compression saving ratio need to be found. As part of this study, we also introduce a novel automatic hyperparameters tuning process that avoids complex manual tuning. The user provides a target prediction accuracy, and the hyperparameter tuning algorithm searches for the optimal T that meets the target accuracy. Our hyperparameter tuning approach incrementally modifies each element in the tuple T as long as it does not decrease the overall prediction accuracy of the CNN model. This process is repeated for every element in every tuple. After reaching the last element in the tuple, the entire tuning process is repeated, beginning with the first element in the tuple, until the target prediction accuracy has been achieved. If the target prediction accuracy cannot be achieved, the hyperparameter tuning process terminates with the achievable prediction accuracy closest to the target.

In the last part of this section, we summarize the fundamental differences between VELCRO and S-VELCRO:

1. While VELCRO employs unstructured compression, which can potentially compress any activation output element, S-VELSCRO performs structured compression of filters and their corresponding kernels.
2. VELCRO and S-VELCRO use different hyperparameter mechanisms for the compression process. While VELCRO employs a hyperparameter tuple that represents the percentile of elements in the variance tensor to be compressed in every convolution layer, S-VELCRO uses a hyperparameter tuple that denotes the number of filters to be trimmed in every layer.
3. VELCRO and S-VELCRO use fundamentally different approaches for the compression process. VELCRO compresses all elements in the activation tensor with a variance within the percentile threshold and replaces them with the arithmetic average. S-VELCRO compresses kernels and their corresponding filter using the channel rank such that all compressed channels in the activation output are replaced by the arithmetic average kernel.

4. Experimental Results and Discussion

Our experimental analysis examines the S-VELCRO algorithm using various CNN models and different specialized tasks. The first subsection describes the experimental environment. Next, we present the compression ratios S-VELCRO achieved for different prediction accuracy targets, the distribution of filters removed in each model layer, and the overall saving in filter memory footprint size. Lastly, we show the energy saving for S-VELCRO by demonstrating a hardware implementation on an FPGA board.

4.1. Experimental Environment

The experimental environment used for our study is based on PyTorch [67]; the ResNet-18, MobileNet V2, and GoogLeNet CNN models [21,22,60] (with their PyTorch pre-trained models); and the ILSVRC-2012 dataset (also known as ImageNet) [23,65]. The S-VELCRO algorithm has been fully implemented in the PyTorch environment. Table 1 lists the three groups of specialized tasks used for our experiments: cats, dogs, and cars. Each group includes four classes of images from the ILSVRC-2012 dataset. Throughout the experimental analysis, we do not modify the first layer of the model, which is a common practice used in other studies [51].

Table 1. Specialized task summary.

Specialized Tasks	ILSVRC-2012 Classes
Dogs	English springer English setter Scottish deerhound Siberian husky
Cats	Cougar Tiger cat Persian cat Egyptian cat
Cars	Cab Beach wagon Minivan Convertible

4.2. Compression Algorithm Performance

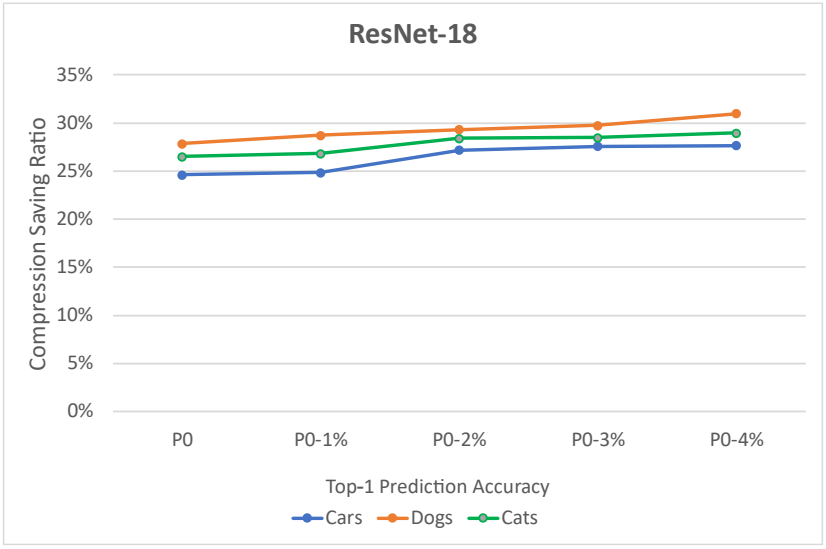
In this section, we present our experimental results for the compression-saving ratio achieved by S-VELCRO on the three groups of specialized tasks (see Table 1). For the preprocessing step, we used a small subset (<2%) of images from the preprocessing dataset. The validation step used the remaining images. It should be noted that the validation process was performed using images not used by the model in the compression step. This is essential to performing an unbiased evaluation of the model performance and preserving the generalization property of the model. Figure 5 presents the compression-saving ratios versus top-1 prediction accuracy of the three CNN models for cars, dogs, and cats, respectively. P0 denotes the prediction accuracy achieved by the compressed model with no degradation relative to the uncompressed model. P0 is summarized in Table 2 for every CNN and specialized task. P0-n% represents a degradation of n% relative to P0.

Table 2. Prediction accuracy (P0) achieved by the compressed model with no degradation relative to the uncompressed model.

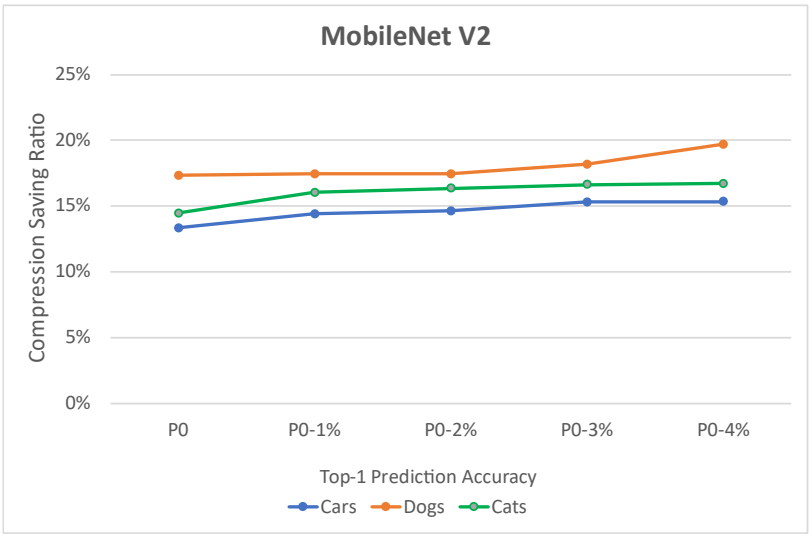
CNN	Specialized Tasks	P0 (%)
ResNet-18	Dogs	80.5
	Cats	74.5
	Cars	62.0
MobileNet V2	Dogs	84.5
	Cats	78.5
	Cars	69.5
GoogleNet	Dogs	82.0
	Cats	72.5
	Cars	61.5

We examined the compression-saving ratios for five top-1 prediction accuracy targets: P0, P0-1%, P0-2%, P0-3%, and P0-4%. For each target, we examined different thresholds via trial and error and chose those that produce the highest compression-saving ratio. The compression-saving ratios S-VELCRO achieved for ResNet-18, with no degradation in the prediction accuracy compared with the uncompressed model, were 24.60%, 27.84%, and 26.48% for cars, dogs, and cats, respectively. For MobileNet V2, S-VELCRO achieved 13.35%, 17.33%, and 14.45% for cars, dogs, and cats, respectively, for P0 target prediction accuracy. Lastly, for GoogleNet, S-VELCRO achieved 11.64%, 6.07%, and 11.27% for cars, dogs, and cats, respectively. Note that when compromising the top-1 prediction accuracy in the range of 1–4%, the compression-saving ratio increases up to approximately 3%, 2.5%, and 4% for ResNet-18, MobileNet V2, and GoogLeNet, respectively.

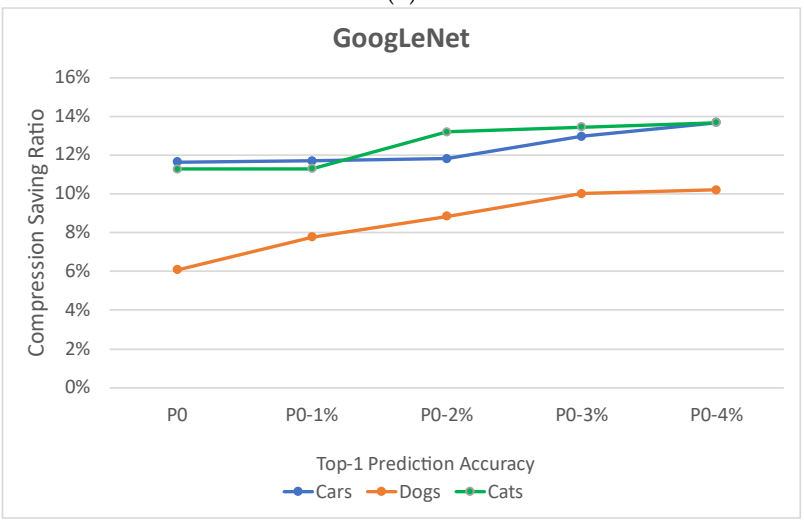
Table 3 compares the S-VELCRO algorithm with other pruning and compression approaches for both specialized and general-purpose CNNs. First, it can be observed that S-VELCRO outperformed VELCRO for ResNet-18, achieved similar computation accelerations for MobileNet V2, and underperformed VELCRO for GoogLeNet. However, when comparing S-VELCRO to VELCRO, we should keep in mind that since VELCRO employs unstructured compression, it incurs overhead to compute addresses and indices of compressed elements and store them in dedicated metadata storage elements. Such limitations, which are out of the scope of this study, become more evident when the unstructured compression method runs on GPUs and TPUs. When comparing S-VELCRO to the pruning methods described in Table 3, it should be noted that although S-VELCRO achieves smaller computational savings, it requires significantly fewer computational resources than common pruning techniques [66] because it avoids back-propagation training.



(**a**)



(**b**)



(**c**)

Figure 5. Compression-saving ratios for (**a**) ResNet-18, (**b**) MobileNet V2, and (**c**) GoogLeNet versus prediction accuracy for specialized tasks.

Table 3. Comparison of S-VELCRO with prior compression techniques.

Compression Method	Network	Specialized Task	Training Required	Computation Acceleration	Accuracy Loss
Taylor criterion [47]	AlexNet	Yes	Yes	1.9×	0.3%
CURL [48]	MobileNet V2	Yes	Yes	3×	Up to 4%
	ResNet-50	Yes	Yes	4×	Up to 2%
Deep compression [28]	Various CNN models	No	Yes	3×	None
Weights and connection learning [27]	AlexNet	No	Yes	3×	None
KSE [64]	ResNet-50	No	Yes	3.8–4.7×	0.84–0.64%
VELCRO [20]	ResNet-18	Yes	No	1.25–1.38×	None
	GoogleNet			1.38–1.42×	None
	MobileNet V2			1.15–1.24×	None
S-VELCRO	ResNet-18	Yes	No	1.33–1.39×	None
	GoogleNet			1.06–1.14×	None
	MobileNet V2			1.12–1.21×	None

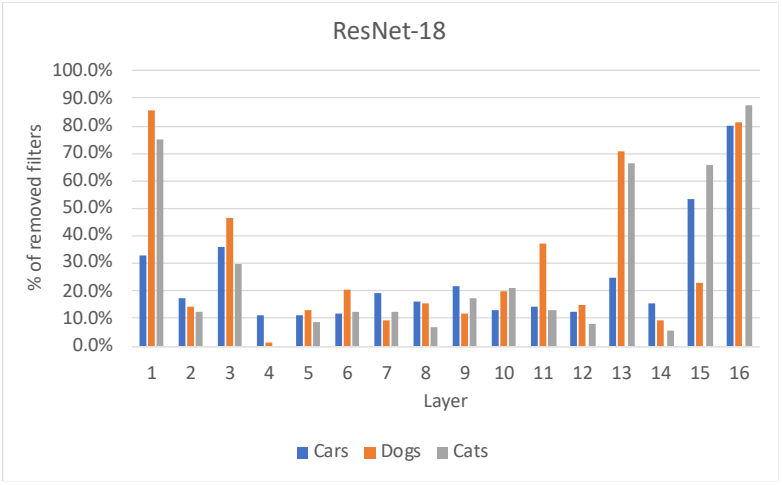
4.3. Filter Memory Footprint Size

Our next experimental analysis examines the savings in filter memory footprint size. The results summarized in Table 4 indicate that the savings vary from 37.32 to 44.92% for ResNet-18, whereas the savings achieved for MobileNet V2 and GoogLeNet are 15.59–22.37% and 10.01–19.41%, respectively.

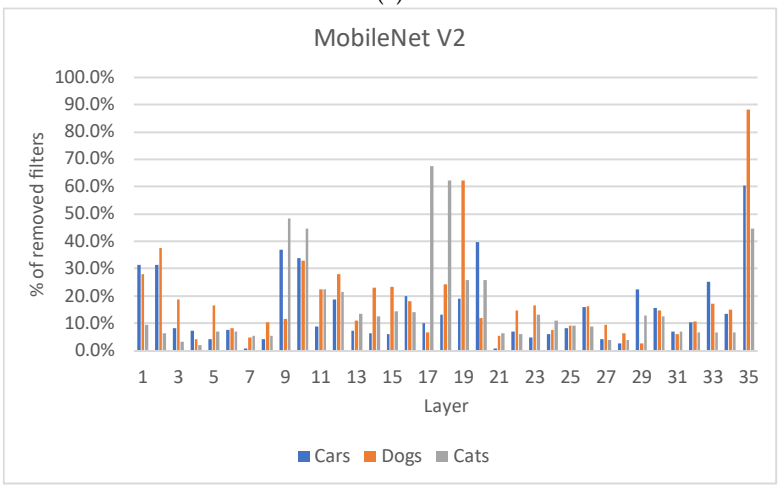
Table 4. Filter Memory Footprint Saving.

CNN	Specialized Tasks	Filter Memory Footprint Saving (%)
ResNet-18	Dogs	37.32
	Cats	44.92
	Cars	38.30
MobileNet V2	Dogs	22.37
	Cats	15.59
	Cars	19.31
GoogleNet	Dogs	10.01
	Cats	19.41
	Cars	15.94

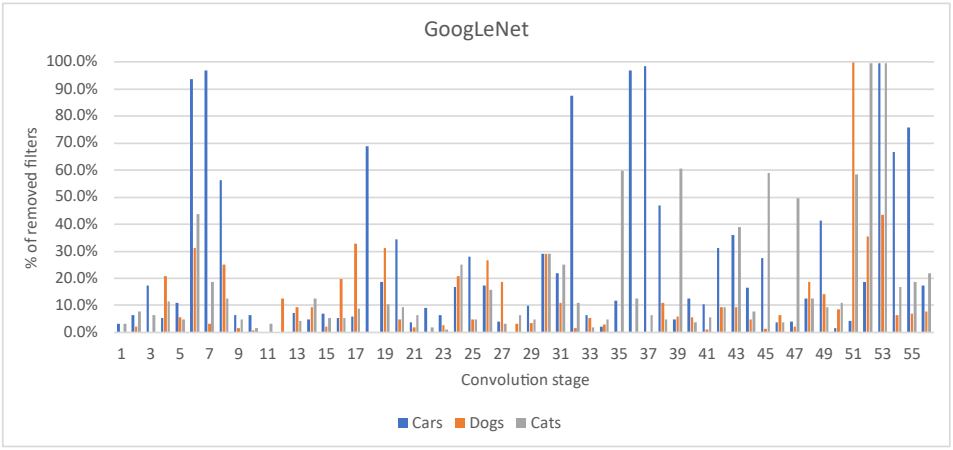
Figure 6 presents histograms of the percentages of the filters removed by S-VELCRO for every layer in the CNNs. For ResNet-18, the first layers 1 and 3 and the deep layers 13, 15, and 16 of the model have the highest percentages of removed filters. For MobileNet V2, the overall filter memory footprint saving is significantly smaller than for ResNet-18. These observations reflect the highly compact nature of the MobileNet V2 network with respect to ResNet-18 and GoogLeNet. Figure 6c illustrates the percentages of removed filters in every convolution stage located through all model layers (convolution and inception) in GoogLeNet. Our results indicate that for the majority of the convolution stages, the trimmed filter percentages are lower than 10%. At the same time, several convolution stages exhibit a high percentage of trimmed filters. For example, in convolution stage 53 for cars and cats, nearly 100% of the filters are trimmed. These observations also support the low compression-saving ratio measurements presented by GoogLeNet.



(**a**)



(**b**)



(**c**)

Figure 6. Percentages of filters removed by S-VELCRO in every mode layer for (**a**) ResNet-18, (**b**) MobileNet V2, and (**c**) GoogLeNet.

4.4. Hardware Implementation

We demonstrated S-VELCRO energy savings on a Xilinx Alveo U280 Data Center accelerator FPGA card [68] (Figure 7). When implementing the ResNet-18 CNN model on the FPGA acceleration card, we measured the energy consumption of the original model and the model compressed by S-VELCRO. Our hardware implementation was designed in Verilog and implemented using the Xilinx Vivado [69] design suite.

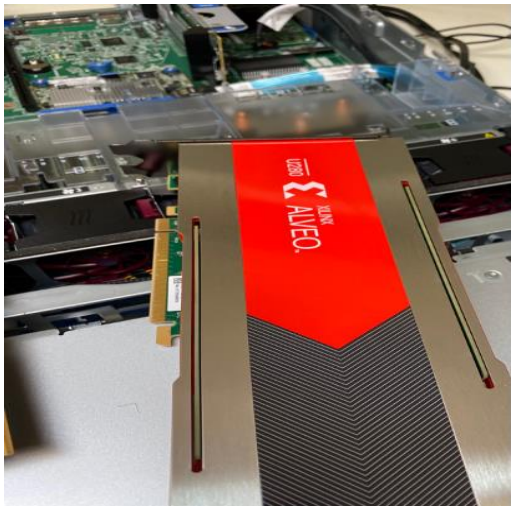


Figure 7. Xilinx Alveo U280 accelerator card.

Our energy consumption measurements are illustrated in Figure 8 for a single inference operation. When the model was compressed with no degradation in the prediction accuracy (P0), the energy consumption was reduced by approximately 24–27%. When the model was further compressed, although the prediction accuracy was compromised in the range of 1–4%, the additional energy saving was negligible.

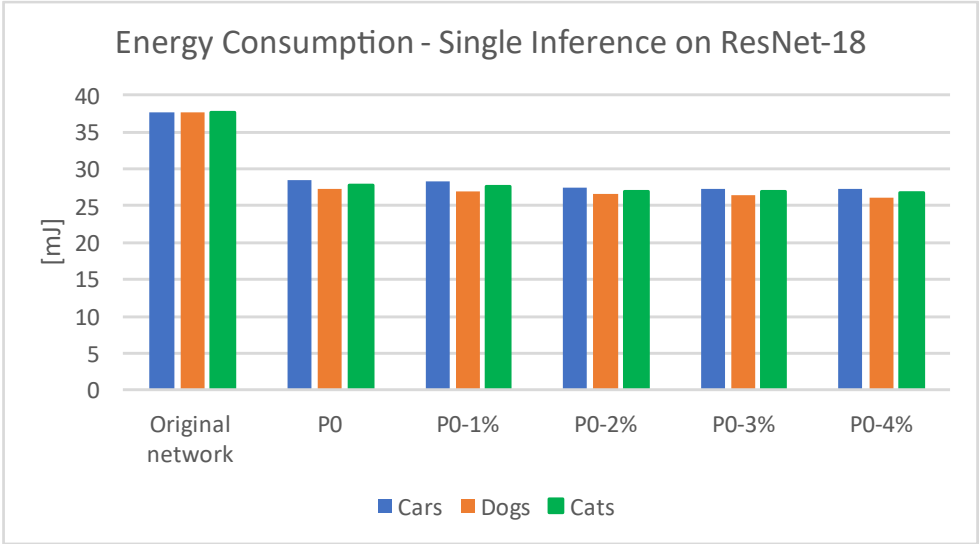


Figure 8. S-VELCRO energy consumption using a Xilinx Alveo U280 accelerator card.

5. Conclusions

This study presents S-VELCRO, a new compression algorithm that exploits value locality to perform structured compression on CNN models used for specialized tasks. S-VELCRO eliminates the overhead of unstructured compression by computing address indices and managing the metadata associated with the compressed elements. In addition, S-VELCRO offers a fast compression process because it avoids back-propagation training, which involves a heavy computational load. Our experimental analysis indicates that S-VELCRO produces a compression-saving ratio of up to 27.84%, 11.27%, and 17.33% for ResNet-18, GoogLeNet, and MobileNet V2, respectively. In addition, S-VELCRO can save up to 44.92%, 22.37%, and 19.41% of the filter memory footprint for ResNet-18, MobileNet v2, and GoogLeNet, respectively. Lastly, we demonstrated S-VELCRO energy savings using an FPGA board, showing that it can save up to 27% of the hardware energy consumption.

Among the important advantages offered by this study, S-VELCRO has several limitations:

1. S-VELCRO cannot perform unstructured compression, and thereby, the potential compression saving ratio is smaller than unstructured compression techniques.
2. Other CNN compression techniques, such as pruning, may potentially reduce the degree of value locality and, as a result, may also affect the efficiency of S-VELCRO.
3. S-VELCRO is aimed at compression-specialized CNNs. When the number of tasks increases, a CNN model becomes more general-purpose, and thereby, the compression saving ratio achieved by S-VELCRO in this case may be limited.

Future studies are highly encouraged to explore further enhancements of specialized CNN compression and energy saving. Some of the suggested directions are:

1. Combining S-VELCRO with various quantization techniques.
2. Applying adaptive data representation ([55]), which can exploit sparsity in conjunction with S-VELCRO.
3. Further enhancing S-VELCRO to exploit value locality relations in between adjacent network layers.

Such directions can potentially simplify CNN computations and introduce additional energy savings.

Author Contributions: Conceptualization, F.G.; methodology, F.G. and G.S.; software, G.S. and F.G.; validation, F.G., B.S. and G.S.; formal analysis, F.G., B.S. and G.S.; investigation, F.G., B.S. and G.S.; resources, F.G. and G.S.; data curation, F.G. and G.S.; writing—original draft preparation, F.G., B.S. and G.S.; writing—review and editing, F.G., B.S. and G.S.; visualization, F.G. and G.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research was not supported by external funding.

Data Availability Statement: The ImageNet datasets used in our experiments are publicly available at https://image-net.org (accessed on 11 March 2021).

Conflicts of Interest: The authors declare no conflict of interest. The author's (Gil Shomron) contributions to this paper were made outside of and unrelated to employment at NVIDIA.

References

1. 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.
2. Bianco, S.; Cadene, R.; Celona, L.; Napoletano, P. Benchmark Analysis of Representative Deep Neural Network Architectures. *IEEE Access* **2018**, *6*, 64270–64277. [CrossRef]
3. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. ImageNet Classification with Deep Convolutional Neural Networks. *Commun. ACM* **2017**, *60*, 84–90. [CrossRef]
4. Reed, R. Pruning algorithms–A survey. *IEEE Trans. Neural Netw.* **1993**, *4*, 740–747. [CrossRef]
5. LeCun, Y.; Denker, J.S.; Solla, S.; Howard, R.E.; Jackel, L.D. Optimal brain damage. In *Advances in Neural Information Processing Systems (NIPS 1989)*; Touretzky, D., Ed.; Morgan Kaufmann: Denver, CO, USA, 1990; Volume 2.
6. Hassibi, B.; Stork, D.G.; Wolff, G.J. Optimal Brain Surgeon and general network pruning. In Proceedings of the IEEE International Conference on Neural Networks, San Francisco, CA, USA, 28 March–1 April 1993; Volume 1, pp. 293–299.
7. Zhang, C.; Bengio, S.; Hardt, M.; Recht, B.; Vinyals, O. Understanding Deep Learning Requires Rethinking Generalization. *arXiv* **2016**, arXiv:1611.03530. [CrossRef]
8. Vanhoucke, V.; Senior, A.; Mao, M.Z. Improving the speed of neural networks on CPUs. In *Deep Learning and Unsupervised Feature Learning Workshop*; NIPS: Granada, Spain, 2011.
9. Gong, Y.; Liu, L.; Yang, M.; Bourdev, L. Compressing deep convolutional networks using vector quantization. *arXiv* **2014**, arXiv:1412.6115.
10. Courbariaux, M.; Bengio, Y.; David, J.-P. BinaryConnect: Training Deep Neural Networks with binary weights during propagations. In Proceedings of the 28th International Conference on Neural Information Processing Systems, Bali, Indonesia, 8–12 December 2015.
11. Lin, Z.; Courbariaux, M.; Memisevic, R.; Bengio, Y. Neural networks with few multiplications. *arXiv* **2015**, arXiv:1510.03009.
12. Shen, H.; Han, S.; Philipose, M.; Krishnamurthy, A. Fast Video Classification via Adaptive Cascading of Deep Models. In Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 21–26 July 2017.
13. Kang, D.; Emmons, J.; Abuzaid, F.; Bailis, P.; Zaharia, M. NoScope: Optimizing Neural Network Queries over Video at Scale. *Proc. VLDB Endow.* **2017**, *10*, 1586–1597. [CrossRef]

14. Kosaian, J.; Phanishayee, A.; Philipose, M.; Dey, D.; Vinayek, R. Boosting the Throughput and Accelerator Utilization of Specialized CNN Inference Beyond Increasing Batch Size. In Proceedings of the 38th International Conference on Machine Learning, PMLR 139, Long Beach, CA, USA, 18–24 July 2021.
15. Wu, Y.; Guo, H.; Chakraborty, C.; Khosravi, M.; Berretti, S.; Wan, S. Edge Computing Driven Low-Light Image Dynamic Enhancement for Object Detection. *IEEE Trans. Netw. Sci. Eng.* **2022**. [CrossRef]
16. Bai, Y.; Liu, J.; Lou, Y.; Wang, C.; Duan, L.-Y. Disentangled Feature Learning Network and a Comprehensive Benchmark for Vehicle Re-Identification. *IEEE Trans. Pattern Anal. Mach. Intell.* **2022**, *44*, 6854–6871. [CrossRef]
17. Chen, C.; Jiang, J.; Zhou, Y.; Lv, N.; Liang, X.; Wan, S. An edge intelligence empowered flooding process prediction using Internet of things in smart city. *J. Parallel Distrib. Comput.* **2022**, *165*, 66–78. [CrossRef]
18. Shazeer, N.; Mirhoseini, A.; Maziarz, K.; Davis, A.; Le, Q.V.; Hinton, G.E.; Dean, J. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv* **2017**, arXiv:1701.06538.
19. Violaand, P.; Jones, M. Rapid object detection using a boosted cascade of simple features. In Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001, Kauai, HI, USA, 8–14 December 2001; Volume 1, pp. I-511–I-518.
20. Gabbay, F.; Shomron, G. Compression of Neural Networks for Specialized Tasks via Value Locality. *Mathematics* **2021**, *9*, 2612. [CrossRef]
21. 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.
22. Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; Rabinovich, A. Going Deeper with Convolutions. In Proceedings of the 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Boston, MA, USA, 7–12 June 2015.
23. 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, 22–24 June 2009.
24. Anwar, S.; Hwang, K.; Sung, W. Structured pruning of deep convolutional neural networks. *ACM J. Emerg. Technol. Comput. Syst.* **2017**, *13*, 1–18. [CrossRef]
25. Cheng, Y.; Wang, D.; Zhou, P.; Zhang, T. A Survey of Model Compression and Acceleration for Deep Neural Networks. *arXiv* **2020**, arXiv:1710.09282.
26. Alqahtani, A.; Xie, X.; Jones, M.W. Literature Review of Deep Network Compression. *Informatics* **2021**, *8*, 77. [CrossRef]
27. Han, S.; Pool, J.; Tran, J.; Dally, W.J. Learning both weights and connections for efficient neural networks. *arXiv* **2015**, arXiv:1506.02626.
28. 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.
29. Castellano, G.; Fanelli, A.M.; Pelillo, M. An iterative pruning algorithm for feedforward neural networks. *IEEE Trans. Neural. Netw.* **1997**, *8*, 519–531. [CrossRef]
30. Collins, M.D.; Kohli, P. Memory bounded deep convolutional networks. *arXiv* **2014**, arXiv:1412.1442.
31. Stepniewski, S.W.; Keane, A.J. Pruning backpropagation neural networks using modern stochastic optimisation techniques. *Neural Comput. Appl.* **1997**, *5*, 76–98. [CrossRef]
32. Liu, Z.; Sun, M.; Zhou, T.; Huang, G.; Darrell, T. Rethinking the Value of Network Pruning. *arXiv* **2018**, arXiv:1810.05270.
33. Lebedev, V.; Lempitsky, V. Fast ConvNets using group-wise brain damage. In Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 26 June–1 July 2016.
34. Zhou, H.; Alvarez, J.M.; Porikli, F. Less is more: Towards compact CNNs. In *Computer Vision—ECCV 2016*; Springer International Publishing: Cham, Switzerland, 2016; pp. 662–677. ISBN 9783319464923.
35. Wen, W.; Wu, C.; Wang, Y.; Chen, Y.; Li, H. Learning structured sparsity in Deep Neural Networks. *Adv. Neural Inf. Process. Syst.* **2016**, *29*, 2074–2082.
36. Li, H.; Kadav, A.; Durdanovic, I.; Samet, H.; Graf, H.P. Pruning Filters for Efficient ConvNets. *arXiv* **2016**, arXiv:1608.08710.
37. Srinivas, S.; Babu, R.V. Data-Free Parameter Pruning for Deep Neural Networks. In Proceedings of the British Machine Vision Conference 2015, Swansea, UK, 7–10 September 2015; British Machine Vision Association, 2015.
38. Rao, Y.; Lu, J.; Lin, J.; Zhou, J. Runtime Neural Pruning. In Proceedings of the Advances in Neural Information Processing Systems, Long Beach, CA, USA, 4–9 December 2017; pp. 2181–2191.
39. Shomron, G.; Weiser, U. Spatial Correlation and Value Prediction in Convolutional Neural Networks. *IEEE Comput. Arch. Lett.* **2019**, *18*, 10–13. [CrossRef]
40. Shomron, G.; Banner, R.; Shkolnik, M.; Weiser, U. Thanks for Nothing: Predicting Zero-Valued Activations with Lightweight Convolutional Neural Networks. In *Computer Vision—ECCV 2020*; Springer International Publishing: Cham, Switzerland, 2020; pp. 234–250.
41. See, A.; Luong, M.-T.; Manning, C.D. Compression of Neural Machine Translation Models via Pruning. *arXiv* **2016**, arXiv:1606.09274.
42. Narang, S.; Elsen, E.; Diamos, G.; Sengupta, S. Exploring Sparsity in Recurrent Neural Networks. *arXiv* **2017**, arXiv:1704.05119.
43. Zhu, M.; Gupta, S. To prune, or not to prune: Exploring the efficacy of pruning for model compression. *arXiv* **2017**, arXiv:1710.01878.

44. Yu, R.; Li, A.; Chen, C.-F.; Lai, J.-H.; Morariu, V.I.; Han, X.; Gao, M.; Lin, C.-Y.; Davis, L.S. NISP: Pruning Networks Using Neuron Importance Score Propagation. In Proceedings of the 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–22 June 2018.
45. Cireşan, D.C.; Meier, U.; Masci, J.; Gambardella, L.M.; Schmidhuber, J. High-Performance Neural Networks for Visual Object Classification. *arXiv* **2011**, arXiv:1102.0183.
46. Chen, W.; Wilson, J.T.; Tyree, S.; Weinberger, K.Q.; Chen, Y. Compressing Neural Networks with the Hashing Trick. In Proceedings of the 32nd International Conference on Machine Learning, Lille, France, 6–11 July 2015.
47. Molchanov, P.; Tyree, S.; Karras, T.; Aila, T.; Kautz, J. Pruning Convolutional Neural Networks for Resource Efficient Inference. *arXiv* **2016**, arXiv:1611.06440.
48. Luo, J.-H.; Wu, J. Neural Network Pruning with Residual-Connections and Limited-Data. In Proceedings of the 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Seattle, WA, USA, 14–19 June 2020.
49. Choi, J.; Wang, Z.; Venkataramani, S.; Chuang, P.I.-J.; Srinivasan, V.; Gopalakrishnan, K. PACT: Parameterized Clipping acTivation for quantized neural networks. *arXiv* **2018**, arXiv:1805.06085.
50. Park, E.; Yoo, S.; Vajda, P. Value-aware quantization for training and inference of neural networks. In *Computer Vision—ECCV 2018*; Springer International Publishing: Cham, Switzerland, 2018; pp. 608–624. ISBN 9783030012243.
51. Zhou, S.; Wu, Y.; Ni, Z.; Zhou, X.; Wen, H.; Zou, Y. DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv* **2016**, arXiv:1606.06160.
52. Banner, R.; Nahshan, Y.; Hoffer, E.; Soudry, D. Post-training 4-bit quantization of convolution networks for rapid-deployment. *arXiv* **2018**, arXiv:1810.05723.
53. Choukroun, Y.; Kravchik, E.; Yang, F.; Kisilev, P. Low-bit quantization of neural networks for efficient inference. In Proceedings of the 2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW), Seoul, Korea, 27–28 October 2019.
54. Fang, J.; Shafiee, A.; Abdel-Aziz, H.; Thorsley, D.; Georgiadis, G.; Hassoun, J.H. Post-training piecewise linear quantization for deep neural networks. In *Computer Vision—ECCV 2020*; Springer International Publishing: Cham, Switzerland, 2020; pp. 69–86. ISBN 9783030585358.
55. Shomron, G.; Gabbay, F.; Kurzum, S.; Weiser, U. Post-Training Sparsity-Aware Quantization. *arXiv* **2021**, arXiv:2105.11010.
56. Buciluǎ, C.; Caruana, R.; Niculescu-Mizil, A. Model Compression. In Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining—KDD'06, Philadelphia, PA, USA, 20–23 August 2006; ACM Press: New York, NY, USA.
57. Hinton, G.; Vinyals, O.; Dean, J. Distilling the Knowledge in a Neural Network. *arXiv* **2015**, arXiv:1503.02531.
58. Gou, J.; Yu, B.; Maybank, S.J.; Tao, D. Knowledge Distillation: A Survey. *Int. J. Comput. Vis.* **2021**, *129*, 1789–1819. [CrossRef]
59. Cai, H.; Gan, C.; Wang, T.; Zhang, Z.; Han, S. Once-for-All: Train One Network and Specialize It for Efficient Deployment. *arXiv* **2019**, arXiv:1908.09791.
60. Zeiler, M.D.; Fergus, R. Visualizing and Understanding Convolutional Networks. In *Computer Vision—ECCV 2014*; Springer International Publishing: Cham, Switzerland, 2014; pp. 818–833.
61. Zhou, B.; Khosla, A.; Lapedriza, A.; Oliva, A.; Torralba, A. Object Detectors Emerge in Deep Scene CNNs. *arXiv* **2014**, arXiv:1412.6856.
62. Morcos, A.S.; Barrett, D.G.T.; Rabinowitz, N.C.; Botvinick, M. On the Importance of Single Directions for Generalization. *arXiv* **2018**, arXiv:1803.06959.
63. Zhou, B.; Sun, Y.; Bau, D.; Torralba, A. Revisiting the Importance of Individual Units in CNNs via Ablation. *arXiv* **2018**, arXiv:1806.02891.
64. Boone-Sifuentes, T.; Robles-Kelly, A.; Nazari, A. Max-Variance Convolutional Neural Network Model Compression. In Proceedings of the 2020 Digital Image Computing: Techniques and Applications (DICTA), Melbourne, Australia, 29 November–2 December 2020; pp. 1–6.
65. Li, Y.; Lin, S.; Zhang, B.; Liu, J.; Doermann, D.; Wu, Y.; Huang, F.; Ji, R. Exploiting kernel sparsity and entropy for interpretable CNN compression. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, California, CA, USA, 16–20 June 2019; pp. 2800–2809.
66. Wang, Y.; Zhang, X.; Xie, L.; Zhou, J.; Su, H.; Zhang, B.; Hu, X. Pruning from Scratch. In Proceedings of the AAAI Conference on Artificial Intelligence, New York, NY, USA, 7–12 February 2020; Volume 34, pp. 12273–12280.
67. Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. *Adv. Neural Inf. Process. Syst.* **2019**, *32*, 8026–8037.
68. Xilinx. Breathe New Life into Your Data Center with Alveo Adaptable Accelerator Cards. Xilinx White Paper, WP499 (v1.0). Available online: https://www.xilinx.com/support/documentation/white_papers/wp499-alveo-intro.pdf (accessed on 19 November 2018).
69. Xilinx. Vivado Design Suite. Xilinx White Paper, WP416 (v1.1). Available online: https://www.xilinx.com/support/documentation/white_papers/wp416-Vivado-Design-Suite.pdf (accessed on 22 June 2012).