mathematics

MDPI

Article

Compression of Neural Networks for Specialized Tasks via Value Locality

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

[1] Computer Science Department, Ruppin Academic Center, Emek Hefer 4025000, Israel
[2] Faculty of Electrical and Computer Engineering, The Technion—Israel Institute of Technology, Haifa 3200000, Israel; gilsho@campus.technion.ac.il
* Correspondence: freddyg@ruppin.ac.il

Abstract: Convolutional Neural Networks (CNNs) are broadly used in numerous applications such as computer vision and image classification. Although CNN models deliver state-of-the-art accuracy, they require heavy computational resources that are not always affordable or available on every platform. Limited performance, system cost, and energy consumption, such as in edge devices, argue for the optimization of computations in neural networks. Toward this end, we propose herein the value-locality-based compression (VELCRO) algorithm for neural networks. VELCRO is a method to compress general-purpose neural networks that are deployed for a small subset of focused specialized tasks. Although this study focuses on CNNs, VELCRO can be used to compress any deep neural network. VELCRO relies on the property of value locality, which suggests that activation functions exhibit values in proximity through the inference process when the network is used for specialized tasks. VELCRO consists of two stages: a preprocessing stage that identifies output elements of the activation function with a high degree of value locality, and a compression stage that replaces these elements with their corresponding average arithmetic values. As a result, VELCRO not only saves the computation of the replaced activations but also avoids processing their corresponding output feature map elements. Unlike common neural network compression algorithms, which require computationally intensive training processes, VELCRO introduces significantly fewer computational requirements. An analysis of our experiments indicates that, when CNNs are used for specialized tasks, they introduce a high degree of value locality relative to the general-purpose case. In addition, the experimental results show that without any training process, VELCRO produces a compression-saving ratio in the range 13.5–30.0% with no degradation in accuracy. Finally, the experimental results indicate that, when VELCRO is used with a relatively low compression target, it significantly improves the accuracy by 2–20% for specialized CNN tasks.

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

Citation: Gabbay, F.; Shomron, G. Compression of Neural Networks for Specialized Tasks via Value Locality. *Mathematics* **2021**, *9*, 2612. https://doi.org/10.3390/math9202612

Academic Editor: Oliviu Matei

Received: 30 September 2021
Accepted: 15 October 2021
Published: 16 October 2021

1. Introduction

Convolutional Neural Networks (CNNs) are broadly employed by numerous computer vision applications such as autonomous systems, healthcare, retail, and security. Over time, the processing requirements and complexity of CNN models have significantly increased. For example, AlexNet [1], which was introduced in 2012, has eight layers, whereas ResNet-101 [2], which was released in 2015, uses 101 layers and requires an approximately sevenfold-greater computational throughput [3]. The increasing model complexity in conjunction with large datasets used for model training has endowed CNNs with phenomenal performance for various computer vision tasks [4]. Typically, large complex networks can further extend their capacity to learn complex image features and properties. The growing model size of CNNs and the requirement of significant processing power have become major deployment challenges for migrating CNN models into mobile, Internet of Things, and edge applications. Such applications incur limited computational and memory

resources, energy constraints, and system cost and, in many cases, cannot rely on cloud computational resources due to privacy, online communication network availability, and real-time considerations.

The compression of CNN models without excessive performance loss significantly facilitates their deployment by a variety of edge systems. Such compression has the potential to reduce computational requirements, save energy, reduce memory bandwidth and storage requirements, and shorten inference time. Various techniques have been suggested to compress CNN models, one of the most common of which is pruning [5–7], which exploits the tendency to over-parameterize CNNs [8]. Pruning trades off degradation in model prediction accuracy for model size by removing weights, Output Feature Maps (OFMs), or filters that make minor or no contribution to the inference of a network. Quantization [9–12] is another common technique that attempts to further compress network size by reducing the number of bits used to represent weights, filters, and OFMs with only a minor impact on accuracy. These methods and other compression approaches are discussed in more detail in Section 2.

This paper focuses on machine learning models that are used for specialized tasks. A specialized neural network is typically a general-purpose model which has been adjusted and optimized to carry out a set of specific tasks. Specialized neural networks have recently become common not only for edge devices but also for datacenters [13–15]. Unlike general-purpose neural networks that are used for a diverse range of classification tasks, specialized neural networks are used for a small number of specific classification tasks. For example, a CNN model that is used to detect vehicles does not use its animal classification capabilities. A common usage of specialized CNN is as a fast filter in front of a heavy general-purpose CNN model. A typical example to such usage is related to offline video analytics [14], which is processed by a specialized CNN model, and only when the model has a low level of confidence are the corresponding frames sent to a general-purpose CNN. Another example is related to game scrapping, where specialized CNNs are used to classify video stream events in by scraping in-game text appearing in frames. A cascaded-CNN [16] is another approach that employs multiple specialized CNNs. The result of each specialized CNN is combined to produce a complete prediction map. The mixture-of-experts model [17] employs a combination of expert models, where each expert is a neural network specialized in specific tasks. Hierarchical classification is another example for specialized CNN usage. Since image categories are typically organized in hierarchical manner, hierarchical classification can be employed by performing a prediction starting from a super class and can only perform detailed classification within the super class. We introduce in this study the value-locality-based compression (VELCRO) algorithm. VELCRO is a method to compress deep neural networks that were originally trained for a large set of diverse classification tasks but are deployed for a smaller subset of specialized tasks. Although this work focuses on CNN models, VELCRO can be used to compress any deep neural network. The main principle of VELCRO is based on the property of value locality, which we introduce herein in the context of neural networks. This property suggests that, when the network is used for specialized tasks, a proximal range of values are produced by the activation functions in the inference process. VELCRO consists of two stages: a preprocessing stage, which identifies activation-function output elements with a high degree of value locality, and a compression stage, which replaces these activation elements with their corresponding arithmetic averages. As a result, VELCRO avoids not only the computation of these activation elements but also the convolution computation of their corresponding OFM elements. VELCRO also requires significantly fewer computational resources than common pruning techniques due to the avoidance of back propagation training. For our experimental analysis we use three CNN models: ResNet-18 [2], MobileNet V2 [18], and GoogLeNet [19] with the ILSVRC-2012 (ImageNet) [20] dataset to examine compression capabilities and model accuracy. Lastly, we implement VELCRO in hardware on a Field Programable Gate-Array (FPGA) and demonstrate the computational and energy savings.

The contributions of this paper are summarized as follows:

1. We introduce the notion of value locality in the context of deep neural networks used for specialized tasks.
2. We present the VELCRO algorithm, which exploits value locality to compress neural networks that are deployed for specialized tasks.
3. VELCRO introduces a fast compression process which solely employs statistics gathering through the inference process and avoids heavy computations involved in back-propagation training, which is usually used by traditional compression approaches such as pruning.
4. VELCRO can be used directly in conjunction with other compression methods such as pruning and quantization.
5. The results of our experiments indicate that
   a. VELCRO produces a compression-saving ratio of computations in the range 20.0–27.7% for ResNet-18, 25–30% for GoogLeNet, and 13.5–20% for MobileNet V2 with no impact on model accuracy;
   b. VELCRO significantly improves accuracy by 2–20% for specialized-task CNNs when given a relatively small compression-savings target.
6. We demonstrate the computational and energy savings of VELCRO by implementing the compression algorithm in hardware on FPGA. Our experimental results indicate a 13.5–30% reduction in energy consumption with VELCRO, which corresponds to the compression-saving ratio.

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 the conclusions and suggests future research directions.

2. Prior Works

Numerous recent studies have proposed various techniques to optimize CNN computations, reduce redundancy, and improve computational efficiency and memory storage. This section describes the following related methods: pruning, quantization, knowledge distillation, deep compression, CNN folding, ablation and CNN filters compression methods.

Pruning is one of the most common methods used for CNN optimization and was introduced in Refs. [5–7]. The concept of pruning, which is inspired by neuroscience, assumes that some network parameters are redundant and may not contribute to network performance. Various pruning techniques [5,21–25] suggest the removal of activations, weights, OFMs, or filters that make a minor or no contribution to the inference process of an already-trained network. Thereby, pruning can significantly reduce the network size and the number of computations. Traditional pruning techniques typically require fine-tuned training on the full model, which may involve significant computational overhead [26].

Pruning techniques can be classified into unstructured and structured classes. Unstructured pruning imposes no constraints on the activations or weights with respect to the network structure (i.e., individual weights or activations are removed by replacing them with zero). Structured pruning [27], in contrast, restricts the pruning process to a set of weights, channels, filters, or activations. Whereas structured pruning incurs limitations on the sparsity that can be exploited in the network due to its coarse pruning granularity, unstructured pruning uses a broader scope of the available sparsity. Conversely, unstructured pruning may involve additional overhead for representing the pruned elements and may not always fit parallel processing elements such as GPUs.

The process of pruning is typically performed by ranking the network elements in accordance with their contribution. The rank can be determined by using various functions such as the L1 or L2 norms [28–31] of weights, activations, or other metrics [32]. Activation pruning requires dynamic mechanisms to monitor activation values because activation importance may depend on the model input. For example, Ref. [33] employs reinforcement learning to prune channels, and Refs. [34,35] leverage spatial correlations of CNN OFMs to predict and prune zero-value activations. Further pruning techniques based

on weight magnitudes were recently introduced in Refs. [21,36,37], which demonstrate that computation efficiency and network scale can be improved significantly. Various gradual pruning approaches [38], given memory footprints and computational bounds, were studied by examining the accuracy and size tradeoffs. The neuron importance score propagation, introduced by Ref. [39], suggests jointly pruning neurons based on a unified goal. Other approaches such as random neuron pruning and random grouping of weight connections into hash buckets were introduced in Refs. [40,41]. Pruning based on a Taylor-expansion criterion [42] focuses on transfer learning by optimizing a network trained to a large dataset of images into a smaller and more efficient network specialized in a subset of classes. Their pruning method performs an iterative backpropagation pruning by removing feature maps with the least level of importance. Ref. [42] evaluated their pruning method by using various criteria such as weight pruning, using l2 norm, and activation pruning, using mean, variance, mutual information, and Taylor-expansion criteria. Their results indicate that the importance of OFMs decreases with layer depth and that each layer has feature maps with both high and low degrees of importance. Ref. [43] introduced pruning by compression using residual connections and limited data (CURL) for residual CNN compression when relying on small datasets that represent specialized tasks.

Quantization methods attempt to reduce the number of bits used to represent the values of weights, filters, and OFMs from 32-bit floating point to 8 bit or less with a slight degradation in model accuracy while simplifying computational complexity. Employing quantization methods that use fewer than 8 bits, however, is not trivial because quantization noise excessively degrades model accuracy. Quantization-aware training uses training processes for quantization to reduce quantization noise and recover model accuracy [44–46]. This approach can be limited when the training process cannot be used due to lack of dataset availability or lack of computational resources. Various fixed-point and vector quantization methods, introduced in Refs. [9–12], present tradeoffs between network accuracy and quantization-compression ratios. A combination of pruning and quantization was introduced in Ref. [22]. Post-training quantization methods [47–50] avoid these limitations by searching for the optimal tensor-cutting values to reduce quantization noise after the network model has been trained.

Knowledge distillation is another machine learning optimization [51,52] that transfers knowledge from a large machine learning model into a smaller compact model that mimics the original model (instead of being trained on the original dataset) to perform competitively. These systems consist of three main elements: knowledge, an algorithm for knowledge distillation, and a teacher–student model. A broad survey of knowledge distillation is available in Ref. [53].

Deep compression was introduced in Ref. [22] and consists of a three-stage pipeline: pruning, trained quantization, and Huffman coding, which operate simultaneously to optimize model size. The first stage prunes the model by learning the important connections, the second stage performs weight quantization and sharing, and the last stage uses Huffman coding. Ref. [54] extends the deep compression idea and introduces the once-for-all network, which can be installed under diverse architectural constraints and configurations, such as performance, power, and cost. The once-for-all approach introduces the progressive shrinking techniques that generalize pruning. Whereas pruning shrinks the network width, progressive shrinking operates on four dimensions: image resolution, kernel size, depth, and width, thereby achieving a higher level of flexibility.

FoldedCNN [15] is another approach to optimize CNNs for specialized-inference tasks. Unlike compression techniques, FoldedCNN does not aim at compressing the CNN model but rather attempts to increase the inference throughput and hardware utilization. The FoldedCNN approach suggests CNN model transformations to increase their arithmetic intensity when processing a large batch size without increasing processing requirements.

Additional studies have attempted to understand the internal mechanisms of CNNs and their contribution to classification tasks. From various CNN models, Refs. [55,56] created visualized images based on the OFMs of different layers and units. Their results

indicate that OFMs extract features that detect patterns, textures, shapes, concepts, and various elements related to the classified images. Ablation techniques were used by Ref. [57] to further quantify the contribution of OFM units to the classification task. Their results indicate that elements that are selective to certain classes may be excluded from the network without necessarily impacting the overall model performance. The impact on ablation of a subset of classes was further studied in Ref. [58], which found that single-OFM-unit ablation can significantly impact the model accuracy for a subset of classes, leading them to suggest different methods to measure the importance of internal OFM units to specific classification accuracy.

CNN filter compression techniques attempt to remove kernel and filters that have small contribution to the network performance. Removal of specific convolution filters based on their importance has been introduced in Ref. [59]. The authors suggest considering two consecutive network layers as a coupled function where the weights are used to compute the coupling factors. In addition, they suggest using the coupling factors to prune filter and maximize the variance of feature maps. Another study on convolution filters compression [60] has highlighted that certain feature maps inside and across CNN layers may have a different contribution to the accuracy of the inference process. The authors indicate that, first model layers typically extract semantic features while the deep layers may extract simple features. Thereby, understanding the importance of feature map can help the compression of the network. They investigate the relationship between input feature map and filter kernels and suggest Kernel Sparsity and Entropy (KSE) as a quantitative indicator for the feature map importance.

These recent studies [55–60] provide the motivation for the present study by suggesting that, when using the CNN model for specialized tasks, we eliminate unrelated computations and thereby compress the model, all with minimal impact on classification accuracy.

3. Method and Algorithm

Our proposed VELCRO compression algorithm relies on the fundamental property of value locality. We start our discussion by first presenting qualitative and quantitative aspects of value locality, following which we describe the VELCRO compression algorithm for specialized neural networks.

3.1. Value Locality of Specialized Convolutional Neural Networks

The principle of the method proposed to compress specialized CNNs is based on the property of value locality. Value locality suggests that, when a CNN model runs specialized tasks, the output values of the activation tensor is in proximity through inference of images. The rationale behind this theory relies on the assumption that the inferred images, which already have a certain level of similarity, exhibit common features such as patterns, textures, shapes, and concepts. As a result, the intermediate layers of the model produce similar values in the vicinity. Figure 1 explains the property of value locality by illustrating the activation-function output tensors in each convolution layer k and channel c. In this example, the set of elements $A^{(m)}[k][c][i][j]$ for images m = 0, 1, . . . , N − 1 in the activation tensor is populated with values in proximity through the inference between images.

For every convolution layer k, we define a variance tensor V[k], where each element V[k][c][i][j] in the variance tensor is defined as

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

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

We use the variance tensor as a measure to quantify the proximity of values for every activation tensor element A[k][c][i][j]. Thereby, a small value of V[k][c][i][j] suggests that the corresponding activation element has a high degree of value locality. The proposed compression algorithm leverages such activation elements for compression. Section 4

presents an experimental analysis of the distribution of the variance tensor for various specialized CNN models.

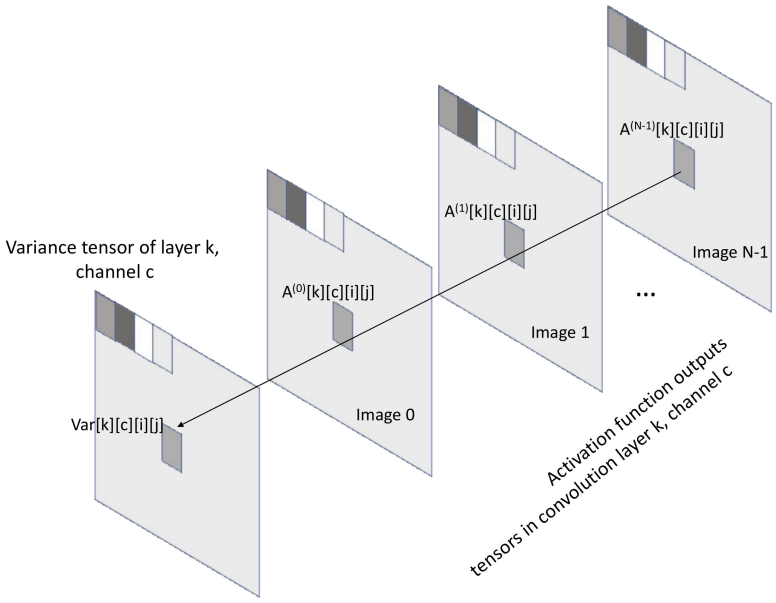


Figure 1. Value locality: The elements with coordinates i, j of the activation-function output tensor in convolutional layer k, channel c, are populated with values in proximity through the inference between images 0 to $N-1$. The variance tensor V serves to measure the degree of value locality.

3.2. VELCRO Algorithm for Specialized Neural Networks

The VELCRO algorithm consists of two stages: preprocessing and compression.

1. Preprocessing stage: In this stage, VELCRO makes an inference by applying the original CNN model to a small subset of images from the specialized task preprocessing dataset. Note that the performance of the compressed model is evaluated on a validation dataset which is distinct from the preprocessing dataset. This is discussed in detail in Section 4. During this stage, the variance tensor is calculated by using Equation (1) for each activation output in each convolution layer in the CNN model. Because the preprocessing stage of VELCRO relies only on inference, it involves a significantly smaller computational overhead with respect to traditional compression methods, which employ heavy backpropagation training processes that can last from a few hours up to hundreds of hours [61].
2. Compression stage: The compression stage uses a tuple of threshold values provided by the user as a hyperparameter for the algorithm. Each threshold element in the tuple corresponds to an individual activation function in each convolution layer. The threshold value of each layer represents the percentile of elements in the variance tensor to be compressed by the algorithm. All elements in the activation tensor with a variance within the percentile threshold are replaced by the arithmetic average constant of the elements located in the same corresponding coordinates. All other activation elements remain unchanged. Replacing activation function output elements by constants avoids not only the activation function computation but also the particular convolution computation of their related OFM elements. In fact, the compression savings of each layer is determined by the corresponding threshold, so the user can determine the overall compression-saving ratio C for the model through the threshold tuple as follows:

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

where the tuple $T = \{T_0, T_1, \ldots, T_K\}$ contains the threshold values for the activation in each convolution layer. In addition, c_k, w_k, and h_k are the number of channels, the width, and the 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.

A simple example that demonstrates the VELCRO algorithm is illustrated in Figure 2, which shows the activation output tensor in convolution layer k for a preprocessing dataset of N = 3 images. The dimensions of the activation tensor are $c_k = 1$, $w_k = 3$, and $h_k = 3$. The VELCRO preprocessing stage performs inference on the preprocessing dataset to create a variance tensor V[k] and an arithmetic average tensor B[k]. The hyperparameter threshold value for layer k is defined in this example as $T_k = 0.33$, which means that the three elements in the activation function output tensor with the lowest variance (highlighted in red) are replaced with their arithmetic average. The remaining elements remain unchanged. The outcome of the VELCRO compression stage is given by the compressed activation-function output tensor $\widetilde{A}[k]$, where the computation of three elements (highlighted in green) are replaced by the arithmetic averages.

Algorithm 1: VELCRO algorithm for specialized neural networks

Input: A CNN model M with K activation-function outputs (each in a different convolution layer), N preprocessing images, and a threshold tuple $T = \{T_0, T_1, \ldots, T_K\}$, where $\forall\, 0 \leq n < N \;\; 0 \leq T_n < 1$.

Output: A compressed CNN Model M_C.

Preprocessing stage

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, $0 \leq m < N$, where the tensors A[k] and $A^{(m)}[k]$ have dimension $c_k \times w_k \times h_k$ and c_k, w_k, and h_k are the number of channels, the width, and the height of the tensor at convolution layer k, respectively.

Step 2: For every $0 \leq k < K$, $0 \leq c < c_k$, $0 \leq i < w_k$, and $0 \leq j < h_k$:

Let tensors S and K be initialized such that S[k][c][i][j] = 0 and Q[k][c][i][j] = 0

Step 3: For each image $0 \leq m < N$:

Perform inference by model M on image m.

For every convolution layer $0 \leq k < K$:

For every $0 \leq c < c_k$, $0 \leq i < w_k$, and $0 \leq j < h_k$,

Let the tensors S and Q be

$S[k][c][i][j] = S[k][c][i][j] + A^{(m)}[k][c][i][j]$

$Q[k][c][i][j] = Q[k][c][i][j] + (A^{(m)}[k][c][i][j])^2$.

Step 4: Let B[k] be the arithmetic average tensor in convolution layer k such that each tensor element is

$B[k][c][i][j] = \frac{1}{N} S[k][c][i][j]$

For every $0 \leq c < c_k$, $0 \leq i < w_k$, and $0 \leq j < h_k$,

Step 5: Let V[k] be the variance tensor of convolution layer k such that each tensor element is

$V[k][c][i][j] = \frac{1}{N} Q[k][c][i][j] - (B[k][c][i][j])^2$

For each $0 \leq c < c_k$, $0 \leq i < w_k$, and $0 \leq j < h_k$

Compression stage:

Step 6: For each convolution layer $0 \leq k < K$:

Let p(x,Y) be the percentile function of element x in tensor Y. p returns the percentile value for x with respect to all elements in tensor Y.

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

$$\widetilde{A}[k][c][i][j] = \begin{cases} A[k][c][i][j] & p(V[k][c][i][j],\, A[k]) > T_k \\ B[k][c][i][j] & p(V[k][c][i][j],\, A[k]) \leq T_k \end{cases}$$

For each $0 \leq c < c_k$, $0 \leq i < w_k$, and $0 \leq j < h_k$

Step 7: 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 $0 \leq k < K$.

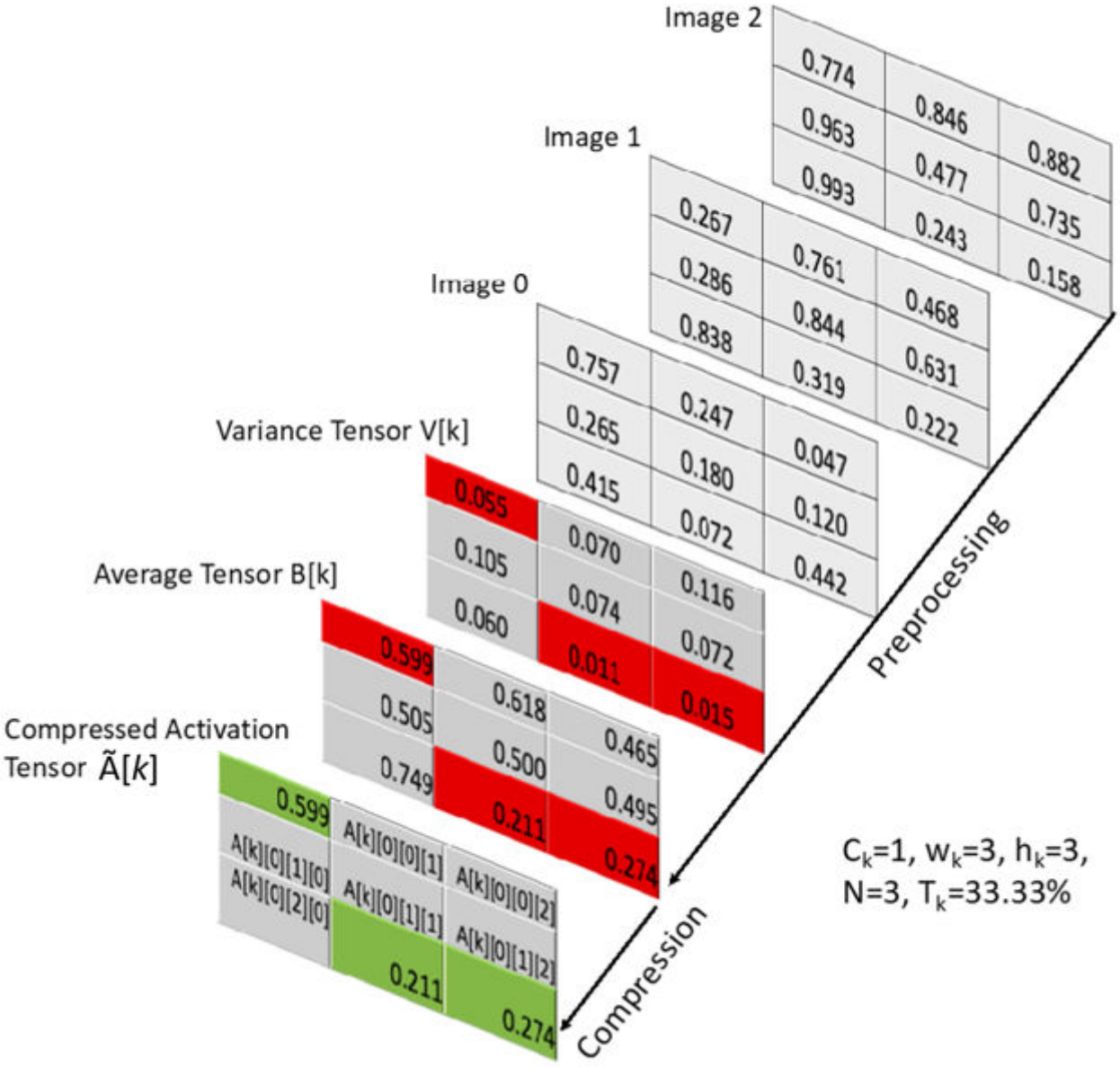


Figure 2. Example of VELCRO preprocessing and compression stages.

4. Experimental Results and Discussion

Our experimental study consists of a comprehensive analysis of both value locality and the performance of various CNN models when used for specialized tasks. In the following, we first describe the experimental environment and then introduce the value locality experimental measurements. Next, we discuss the performance of the VELCRO compression algorithm. Finally, we demonstrate the computational and energy savings of VELCRO by designing a hardware that implements the compression algorithm on FPGA.

4.1. Experimental Environment

Our experimental environment is based on PyTorch [62], the ILSVRC-2012 dataset (also known as "ImageNet") [20,60], and the ResNet-18, MobileNet V2, and GoogLeNet CNN models [18,19,55] with their PyTorch pretrained models. The VELCRO algorithm, described in Algorithm 1, has been fully implemented on the PyTorch environment. Table 1 summarizes the specialized tasks used for our experimental analysis. The experiments examine five groups of specialized tasks: the groups Cats-2, Cats-3, and Cats-4 include

two, three, and four classes from the ILSVRC-2012 dataset, respectively, and the groups Dogs and Cars include four classes each. Throughout the experimental analysis, we do not modify the first layer of the model, which is a common practice that has been used by numerous studies [46].

Table 1. Specialized tasks summary.

Specialized Tasks	ILSVRC-2012 Classes
Cats-2	Egyptian cat Persian cat
Cats-3	Egyptian cat Persian cat Cougar
Cats-4 (Cats)	Egyptian cat Persian cat Cougar Tiger cat
Dogs	English setter Siberian husky English springer Scottish deerhound
Cars	Beach wagon Cab Convertible Minivan

4.2. Experimental Analysis of Value Locality

The distribution of the variance tensor elements in each layer (skipping the first layer) is a measure to quantify the proximity of the activation-function output. Figure 3 shows the distribution of the variance tensor elements for the selected activation function outputs in the convolution layers 1, 3, 7, 10, and 14 in ResNet-18. The distribution is shown for the groups of classes Cats-2, Cats-3, and Cats-4, which include two, three, and four classes of cats from the dataset, respectively. The group "all" contains a mixture of all ILSVRC-2012 dataset classes and represents the case when the CNN model is used for general tasks. When the CNN model is used for specialized tasks (Cats-1, -2, and -3), the distribution of the variance tensor elements clearly shifts toward zero with respect to the distribution when the model is used for general tasks (all), which indicates that the CNN model produces values of closer proximity (i.e., a higher degree of value locality) for specialized tasks. Another important outcome made apparent in Figure 3 is that the three groups of specialized tasks behave similarly regardless of the number of classes. The distribution of variance tensor elements in all ReseNet-18 layers is presented in Figure A1 (Appendix A) and behaves similarly to the distribution presented herein.

Figure 4 illustrates the same experimental analysis but for the GoogLeNet CNN model for selected layers 1, 6, 12, 21, 32, 38, 47, 51, and 56. The variance tensor elements of GoogLeNet behave very similarly to those of ResNet-18. When the model is used for specialized tasks, the variance distribution shifts left with respect to the general-purpose use, indicating a higher degree of value locality. The distribution in all GoogLeNet layers is presented in Figure A2 (Appendix A).

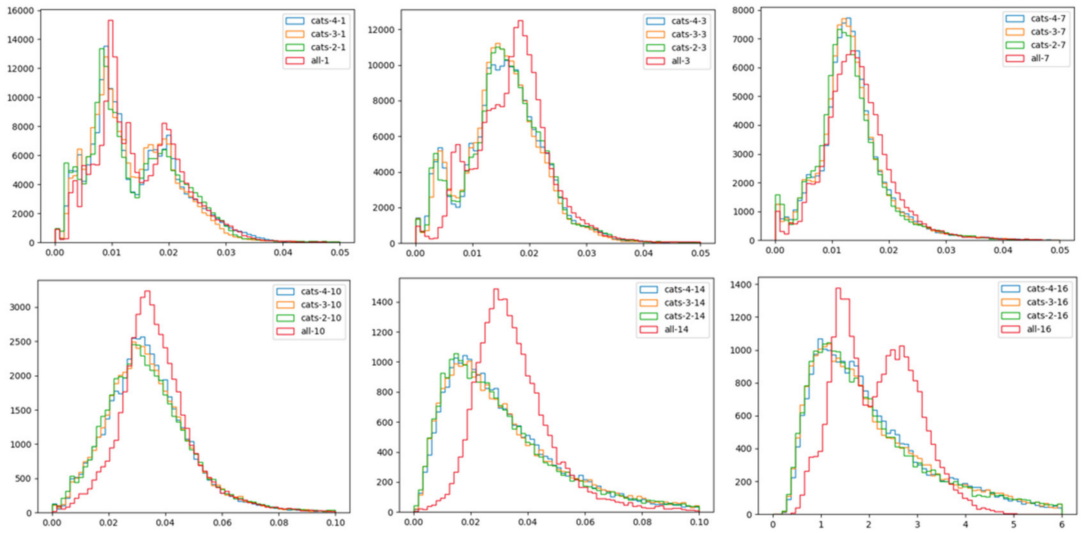


Figure 3. Distribution of ResNet-18 variance tensor elements in layers 1, 3, 7, 10, 14, and 16 for specialized tasks: all ImageNet classes, Cats-2, Cats-3, and Cats-4.

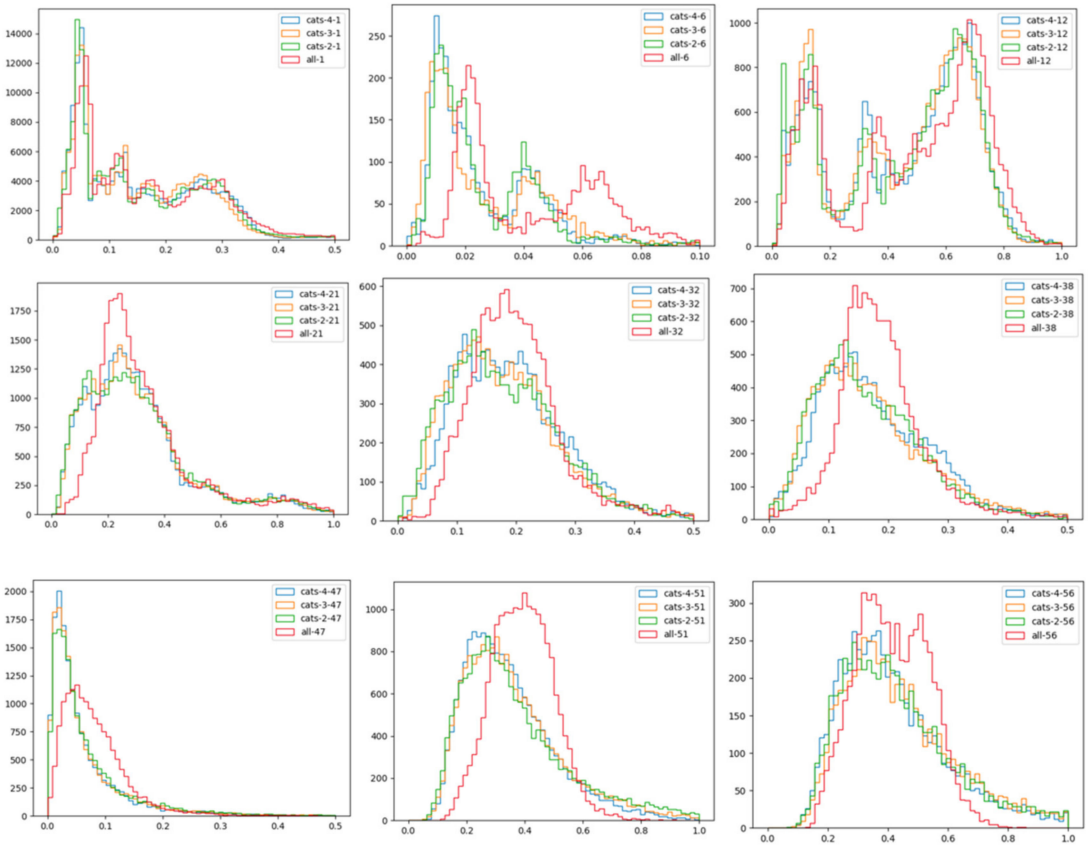


Figure 4. Distribution of GoogLeNet variance tensor elements in layers 1, 6, 12, 21, 32, 38, 47, 51, and 56 for specialized tasks: all ImageNet classes, Cats-2, Cats-3, and Cats-4.

Figure 5 presents a similar experimental analysis for MobileNet V2 layers 1, 6, 12, 19, 28, 30, and 35, and the distribution in all MobileNet V2 layers is presented in Figure A3 (Appendix A). The results indicate that a lower degree of value locality occurs relative to ResNet-18 and GoogLeNet when MobileNet V2 is used for specialized tasks. The results indicate that the shift of the variance tensor elements distribution is smaller with respect to the other CNN models. These observations reflect the highly compact nature of the

MobileNet V2 network with respect to ResNet-18 and GoogLeNet, which results in a lower potential for leveraging value locality for the former.

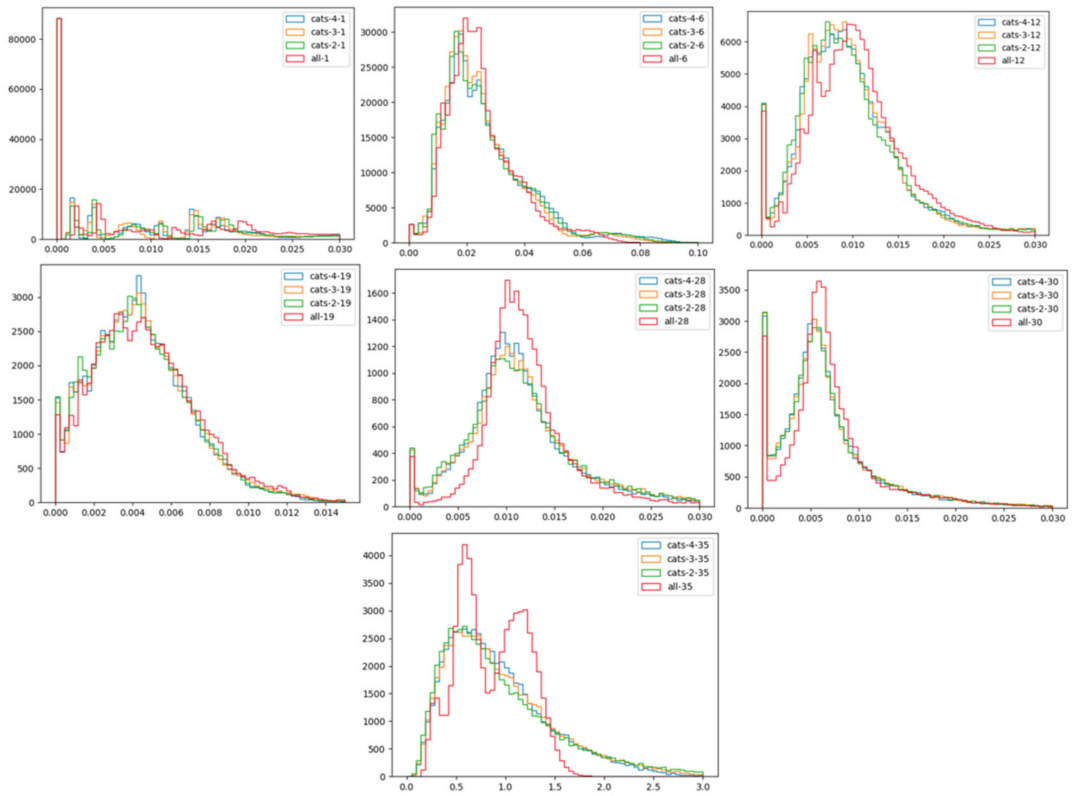


Figure 5. Distribution of MobileNet-V2 variance tensor elements in layers 1, 6, 12, 19, 28, 30, and 35 for specialized tasks: all ImageNet classes, Cats-2, Cats-3, and Cats-4.

Figures 6–8 extend our experimental analysis for additional groups of specialized tasks, Dog and Cars, each of which includes four classes from the ILSVRC-2012 dataset. Note that the Cats group corresponds to the group Cats-4. The results further confirm those shown in Figures 3–5. In all the examined CNN models and in the majority of activation-function outputs in all convolution layers, the distribution of variance tensor elements for the specialized tasks clearly shifts toward zero relative to the distribution when the model is used for general tasks (all). Like the results presented in Figure 5, we also observe that MobileNet V2 can leverage value locality but in a smaller magnitude with respect to ResNet-18 and GoogLeNet.

These experimental results support our expectations that CNN models that are used for specialized tasks exhibit a high degree of value locality. Figures A4–A6 (Appendix A) show the experimental results for all layers of all models. The complete experimental results for all layers behave similarly to the distribution presented in Figures 6–8.

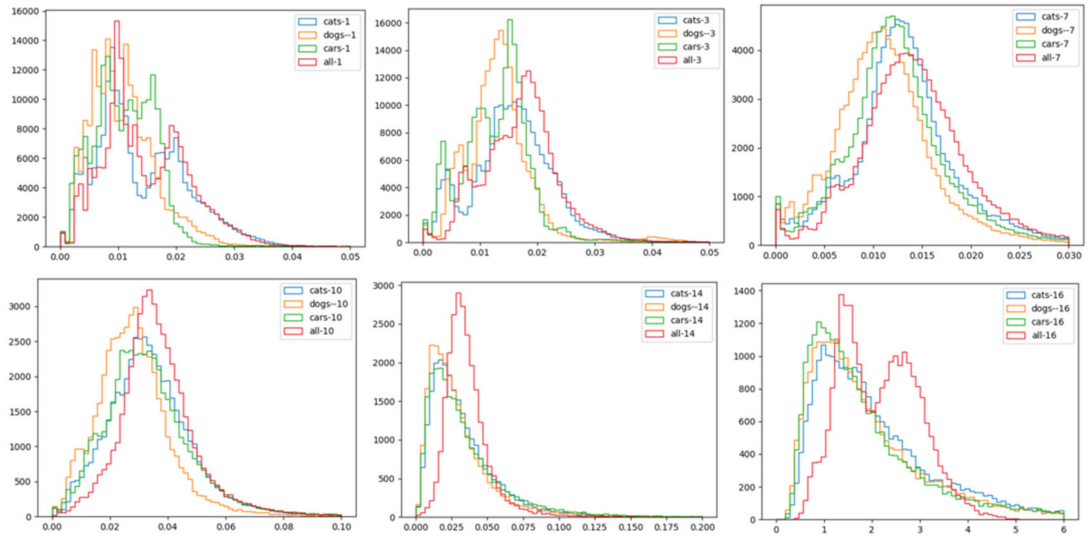


Figure 6. Distribution of ResNet-18 variance tensor elements in layers 1, 3, 7, 10, 14, and 16 for specialized tasks: Cats, Dogs, Cars, and all ImageNet classes.

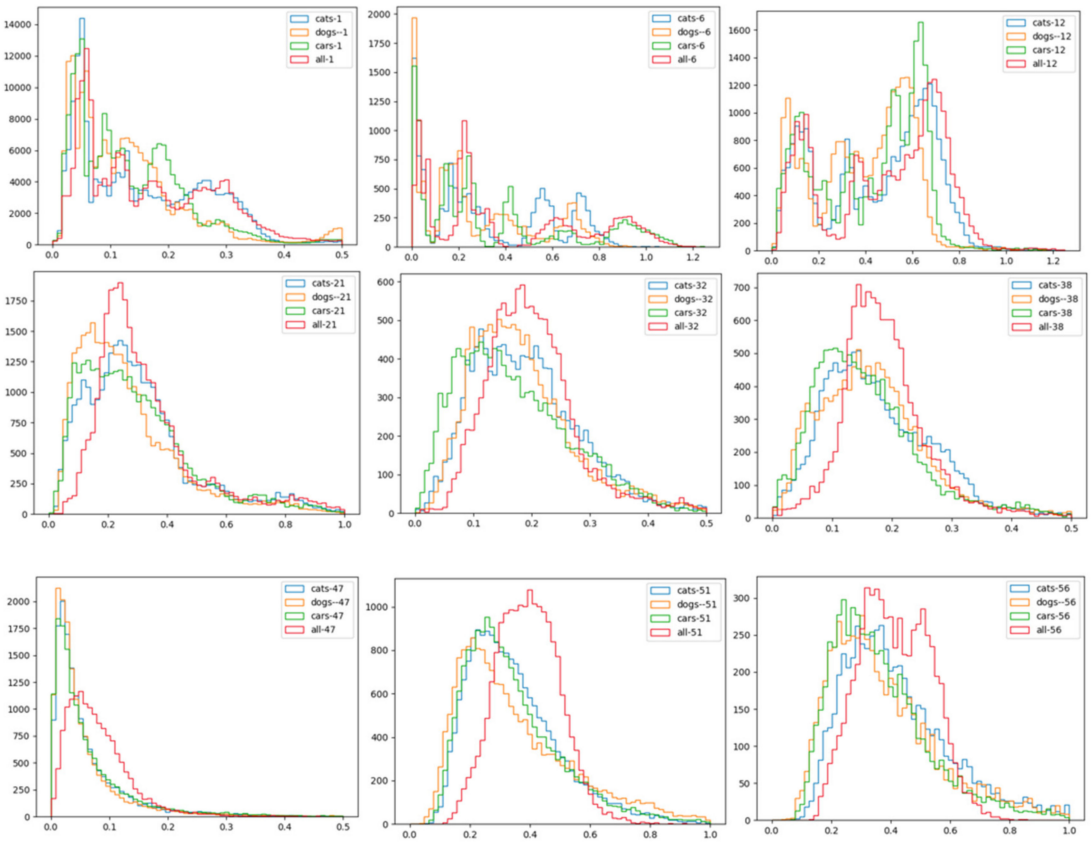


Figure 7. Distribution of GoogLeNet variance tensor elements in layers 1, 6, 12, 21, 32, 38, 47, 51, and 56 for specialized tasks: Cats, Dogs, Cars, and all ImageNet classes.

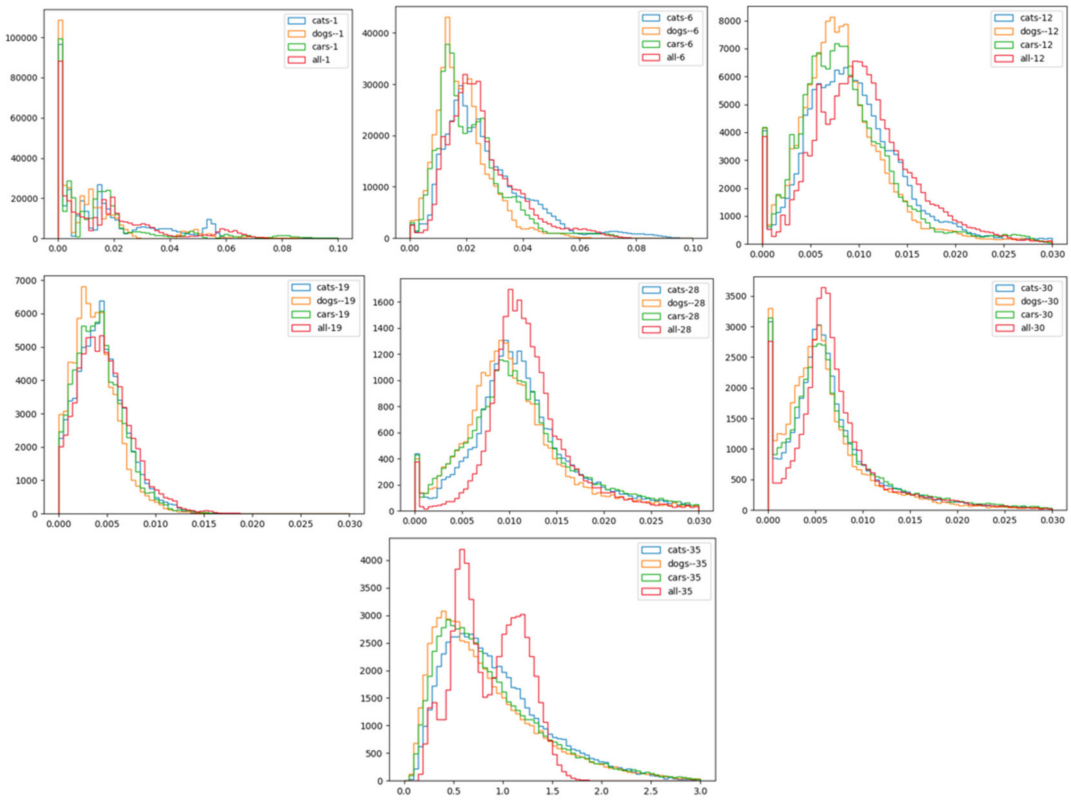
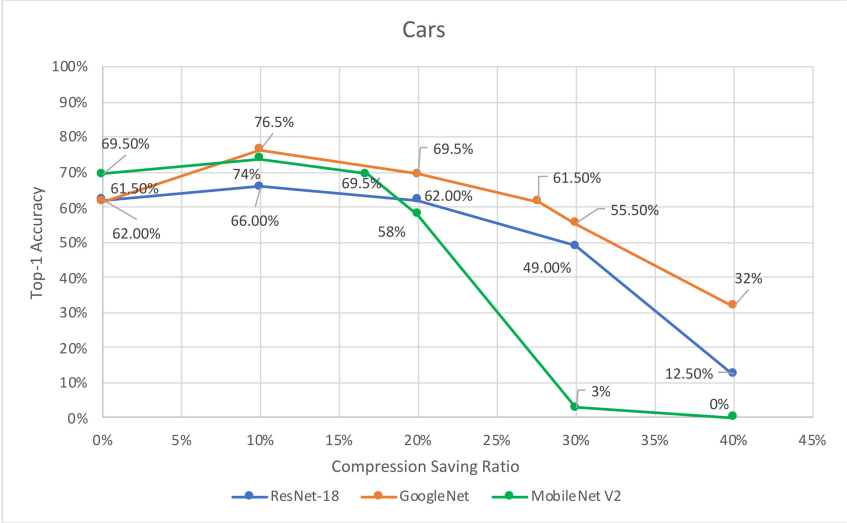


Figure 8. Distribution of MobileNet V2 variance tensor elements in layers 1, 6, 12, 19, 28, 30, and 35 for specialized tasks: Cats, Dogs, Cars, and all ImageNet classes.

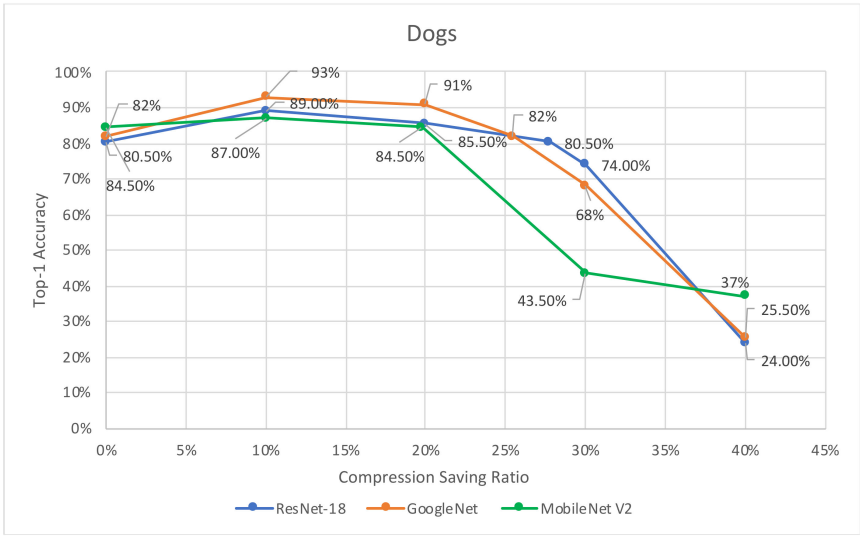
4.3. Performance of Compression Algorithm

As part of our experimental analysis, we examine the compression-saving ratio of the VELCRO algorithm on three groups of specialized tasks: cats, cars, and dogs (see Table 1). Only a very small subset (<2%) of images from the preprocessing dataset has been used for the preprocessing stage of the algorithm, while the remaining images have used for the validation of the compressed model. This approach is essential in order to perform an unbiased evaluation of the model performance and preserved the generalization property of the model. Figure 9a–c present the top-1 prediction accuracy versus the compression-saving ratio for cars, dogs, and cats, respectively. The experimental analysis is applied to the ResNet-18, GoogLeNet, and MobileNet V2 CNN models. For each compression-saving ratio, we examine different thresholds by running trial-and-error and choose those that produce the highest top-1 accuracy. Tables A1–A3 in Appendix B summarize the tuples of threshold values. Table 2 summarizes the maximum compression-saving ratio for each group of specialized tasks and each CNN model that produces the same accuracy as the original uncompressed model.

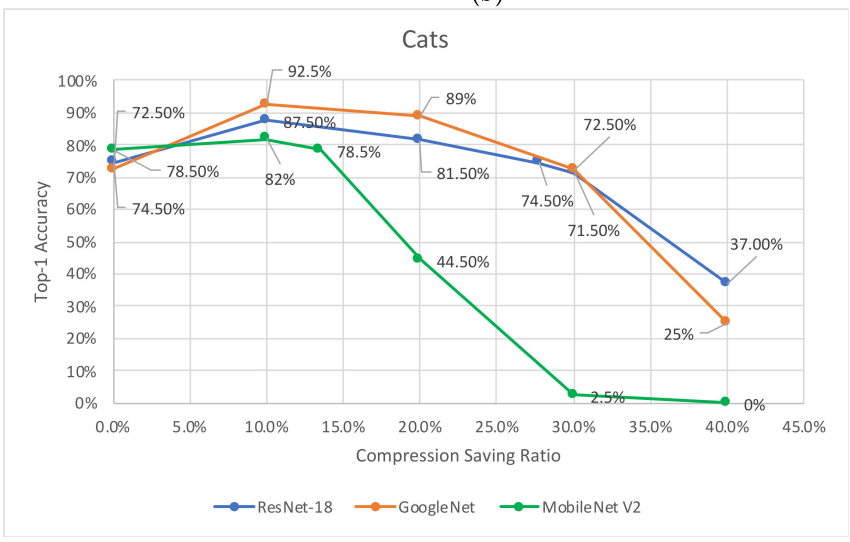
The experimental results indicate that VELCRO produces a compression-saving ratio of 20.00–27.73% in ResNet-18 and 25.46–30.00% in GoogLeNet. The higher compression-saving ratio in GoogLeNet is attributed to the fact that GoogLeNet uses significantly a greater number of parameters and thereby has higher potential to leverage value locality. This explains why GoogLeNet better leverages value locality when the network is employed for special tasks. Conversely, MobileNet V2 produces a smaller compression-saving ratio, 13.50–19.76%, for the specialized tasks examined. These results comply with our previous measurements of the distribution of the variance tensor elements, which imply that the potential of leveraging value locality in MobileNet V2 is smaller than that of the other CNNs examined. This is explained by the fact that MobileNet V2 is much more compact than the other CNNs examined and thereby has a lower potential to leverage value locality.



(**a**)



(**b**)



(**c**)

Figure 9. Accuracy for ResNet-18, GoogLeNet, and MobileNet V2 versus compression-saving ratio for specialized tasks: (**a**) Cars, (**b**) Dogs, and (**c**) Cats.

Table 2. Maximum compression-saving ratio achieved while maintaining the accuracy of the original uncompressed CNN model.

Specialized Task	ResNet-18	GoogLeNet	MobileNet V2
Cats	27.73%	30.00%	13.50%
Dogs	27.70%	25.46%	19.76%
Cars	20.00%	27.70%	16.80%

Note that VELCRO does not aim to compress the network memory footprint but rather to reduce the computational requirements. Therefore, any comparison of VELCRO to pruning approaches should consider computation aspects rather than the number of parameters in the network. Table 3 compares the VELCRO algorithm with other pruning approaches for both specialized CNNs and general-purpose ones. Although VELCRO achieves smaller computation savings, it requires significantly fewer computational resources than common pruning techniques [61] due to the avoidance of back propagation training.

Table 3. Comparison summary of VELCRO with respect to punning techniques. We also examine the output of the activation functions compressed by VELCRO.

Compression Method	Network	Specialized Task	Training Required	Computation Acceleration	Accuracy Loss
Taylor criterion [42]	AlexNet	Yes	Yes	1.9X	0.3%
CURL [43]	MobileNet V2	Yes	Yes	3X	Up to 4%
	ResNet-50	Yes	Yes	4X	Up to 2%
Deep compression [22]	Various CNN models	No	Yes	3X	None
Weights and connection learning [21]	AlexNet	No	Yes	3X	None
KSE [59]	ResNet-50	No	Yes	3.8–4.7X	0.84–0.64%
VELCRO	ResNet-18	Yes	No	1.25–1.38X	None
	GoogLeNet			1.38–1.42X	None
	MobileNet V2			1.15–1.24X	None

Table 4 presents the percent compression of activation elements with zero value out of all the compressed activation elements. The results in Table 3 correspond to the compression-saving ratios in Table 2 (i.e., when the network achieves maximum compression without losing accuracy). With ResNet-18 and GoogLeNet, the fraction of compressed zero values is in the range 0.08–0.31% and 0.56–0.64%, respectively. In contrast, MobileNet V2 produces a significantly larger fraction of compressed zero values: 10.48–14.91%, which is attributed to the fact that MobileNet V2 is a much more compact model than the other CNNs. These results indicate that VELCRO offers an extended level of compression with respect to pruning, which aims to remove weak connections of zero values.

Table 4. Compressed activation elements with zero value as a percent of all compressed activation elements.

Specialized Task	ResNet-18	GoogLeNet	MobileNet V2
Cats	0.08%	0.56%	14.91%
Dogs	0.20%	0.63%	10.48%
Cars	0.31%	0.64%	12.00%

Another important result gained from Figure 9a–c is that, when VELCRO is used with a relatively moderate compression ratio, it produces a significant increase in accuracy. The results are presented in Table 5, which summarizes the maximum top-1 accuracy achieved by VELCRO. These results are attributed to the fact that a relatively moderate level of compression helps the network leverage value locality to strengthen connections, thereby

increasing the probability of favoring the prediction of classes that are in the scope of the specialized tasks.

Table 5. The maximum top-1 accuracy increase produced by VELCRO with respect to the uncompressed model when used for specialized tasks.

Specialized Task	ResNet-18	GoogLeNet	MobileNet V2
Cats	13.00%	20.00%	3.50%
Dogs	8.50%	11.00%	2.50%
Cars	4.00%	15.00%	4.50%

4.4. Hardware Implementation

In the last part of our experimental analysis, we demonstrate the computational optimization and energy savings of VELCRO through hardware implementation on the Xilinx® Alveo™ U280 Data Center accelerator card [63]. Our hardware implementation, which is illustrated in Figure 10, consists of 16 instance modules where each is comprised of a two-dimensional convolution layer with a 64×64 input feature map (IFMAP), 3×3 filter, and a ReLU activation function. In addition, each module also includes a compression control logic which skips the compressed computations and replaces them with their corresponding arithmetic averages. Our hardware implementation was designed in Verilog and implemented using the Xilinx® Vivado™ [64] design suite.

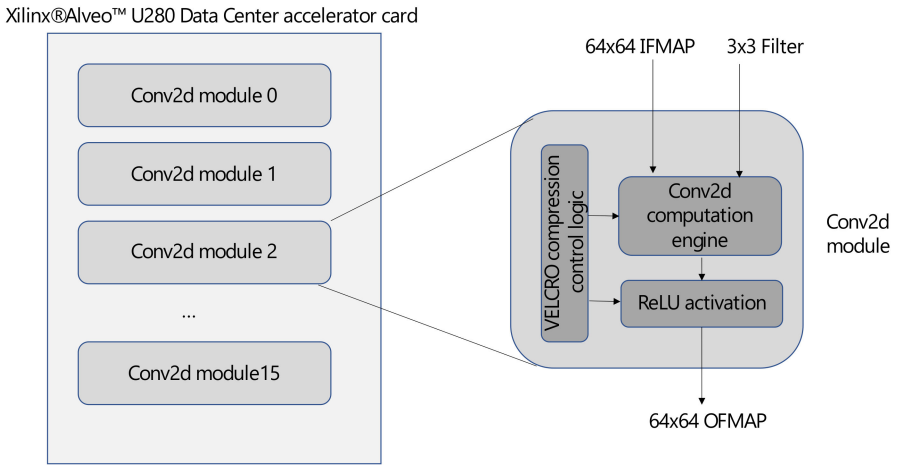


Figure 10. VELCRO compression implementation on Xilinx® Alveo™ U280 Accelerator Card.

Figure 11 presents the (normalized) throughput and energy consumption of a single module instance, denoted as conv2d, which consists of the hardware implementation of a two-dimensional convolution layer and ReLU activation. As expected, the computational throughput of the conv2d layer, which is measured as the number of conv2d operations per second, exhibits a growth rate proportional to $\frac{1}{1-C}$, where C is the compression saving ratio). In addition, it can be observed that the energy consumption related to the computation of a single conv2d layer decays linearly with the compression saving ratio. Thereby, for the compression saving results presented in Table 2, VELCRO can achieve 13.5–30% energy consumption savings while maintaining the same accuracy of the uncompressed model.

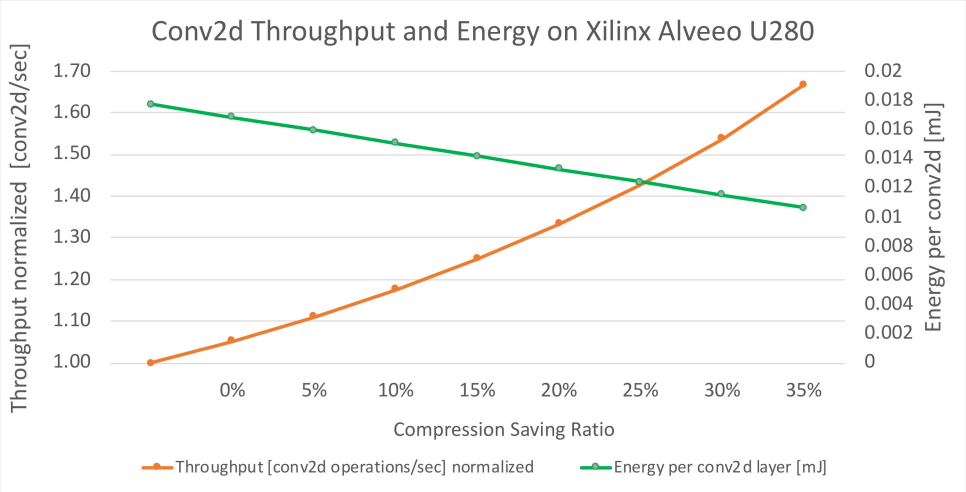


Figure 11. VELCRO throughput and energy consumption Xilinx® AlveoTM U280 Accelerator Card.

5. Conclusions

We present herein value-locality-based compression algorithm (VELCRO), wherein a compression approach is introduced for general-purpose deep neural networks deployed for a small subset of specialized tasks. We introduce the notion of value locality in the context of neural networks for specialized tasks and show that CNNs that are used for specialized tasks produce a high degree of value locality. An analysis of the experimental results indicates that VELCRO leverages value locality to compress the network and thereby saves up to 30% of the computations in ResNet-18 and GoogLeNet and up to 20% in MobileNet V2. The analysis also indicates that, for specialized tasks, VELCRO significantly improves the accuracy by 2–20% when given a relatively small compression-saving target. Finally, a major advantage of VELCRO is that it offers a fast compression process that is based on inference rather than backpropagation training, thereby liberating VELCRO from a significant computational load. We demonstrate the feasibility of VELCRO by designing the algorithm in hardware on the Xilinx® Alveo™ U280 Data Center accelerator card. Our hardware implementation indicates that VELCRO translates the computation compression into an energy consumption savings of 13.5–30%, corresponding to the compression-saving ratio.

Author Contributions: Conceptualization, F.G.; methodology, F.G. and G.S.; software, G.S. and F.G.; validation, F.G. and G.S.; formal analysis, F.G. and G.S.; investigation, F.G. and G.S.; resources, F.G. and G.S.; data curation, F.G. and G.S.; writing—original draft preparation, F.G. and G.S.; writing—review and editing, F.G. 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 received no external funding.

Data Availability Statement: The ImageNet data sets 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.

Appendix A

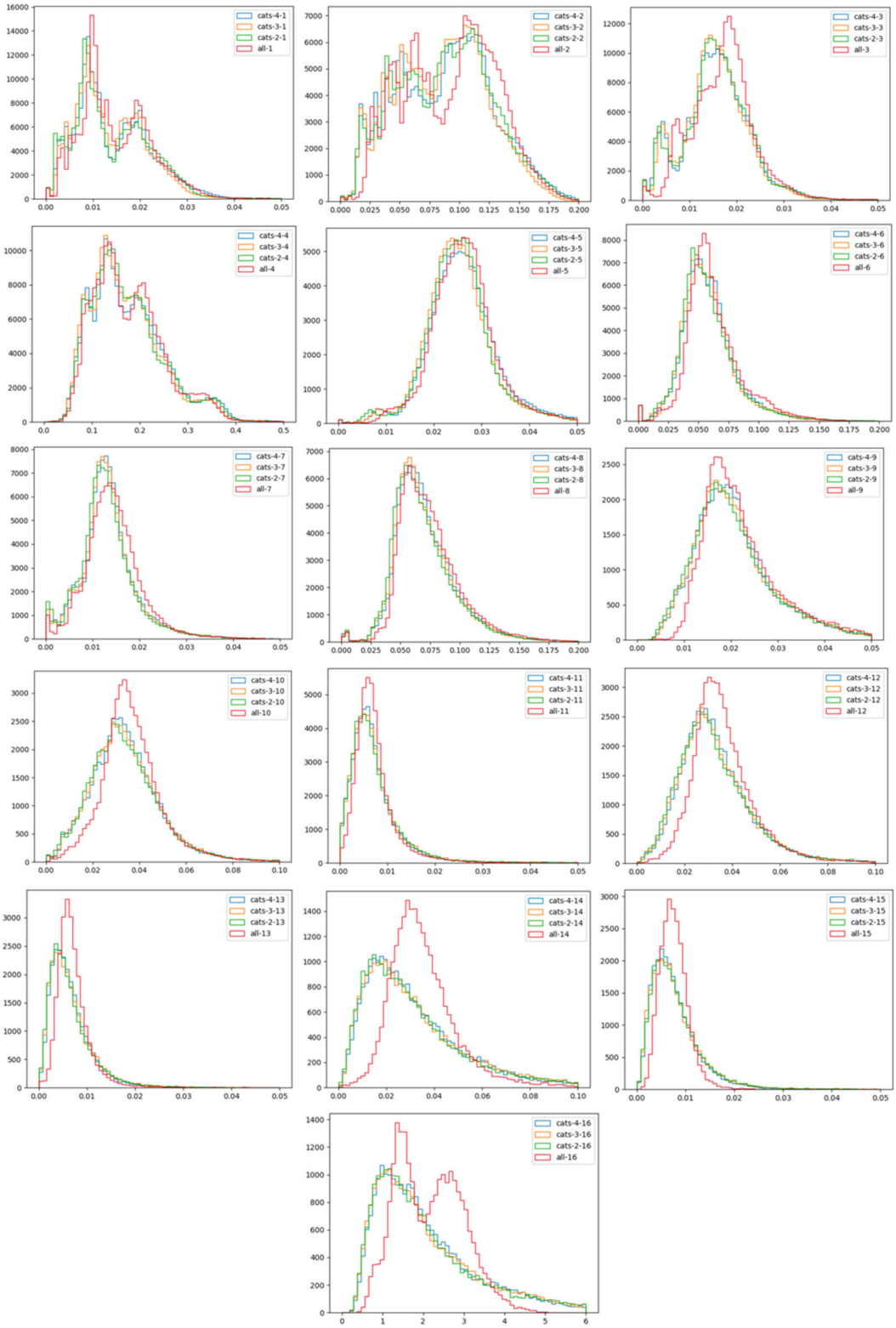


Figure A1. Distribution of ResNet-18 variance tensor elements for specialized tasks: all ImageNet classes, Cats-2, Cats-3, and Cats-4.

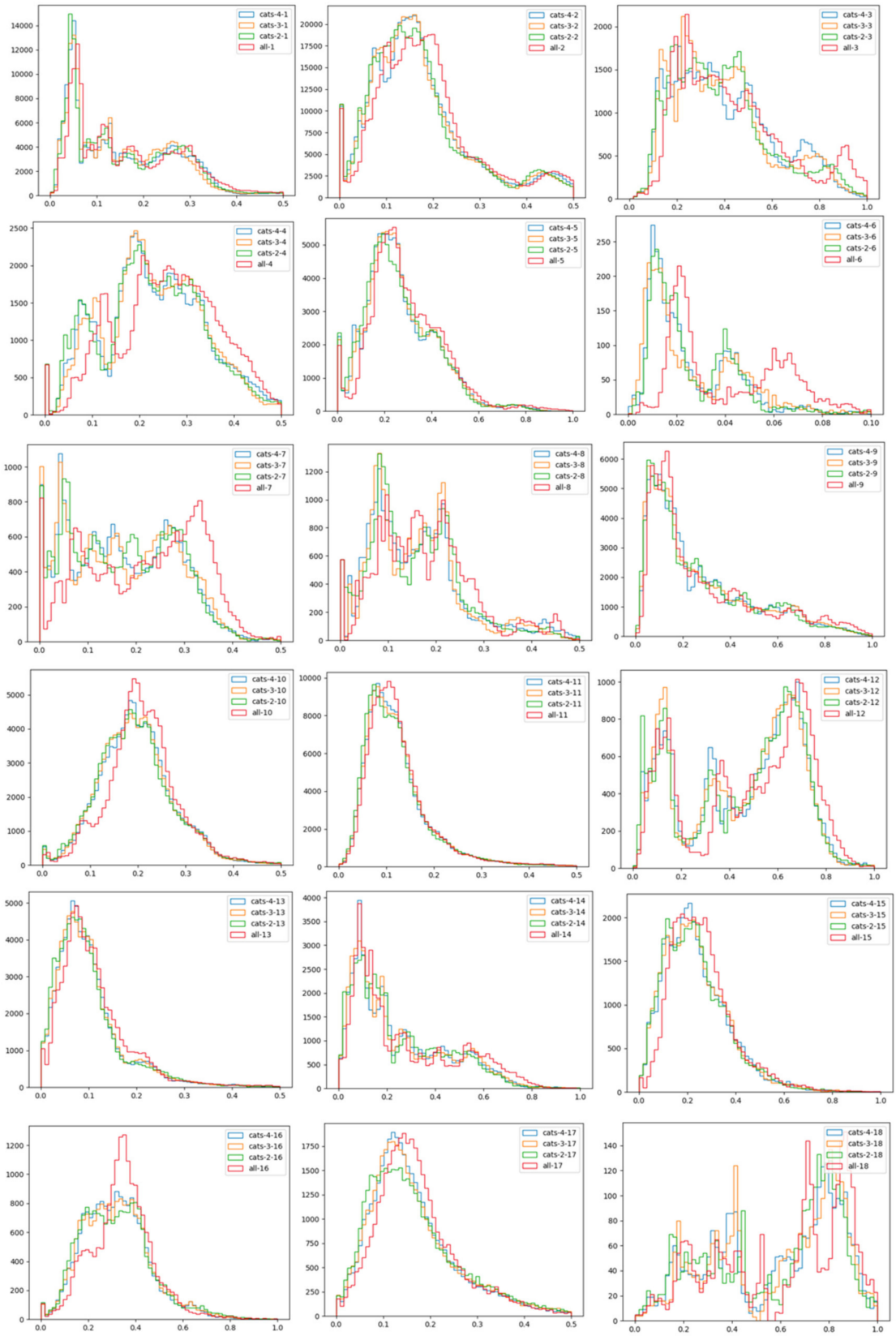


Figure A2. *Cont.*

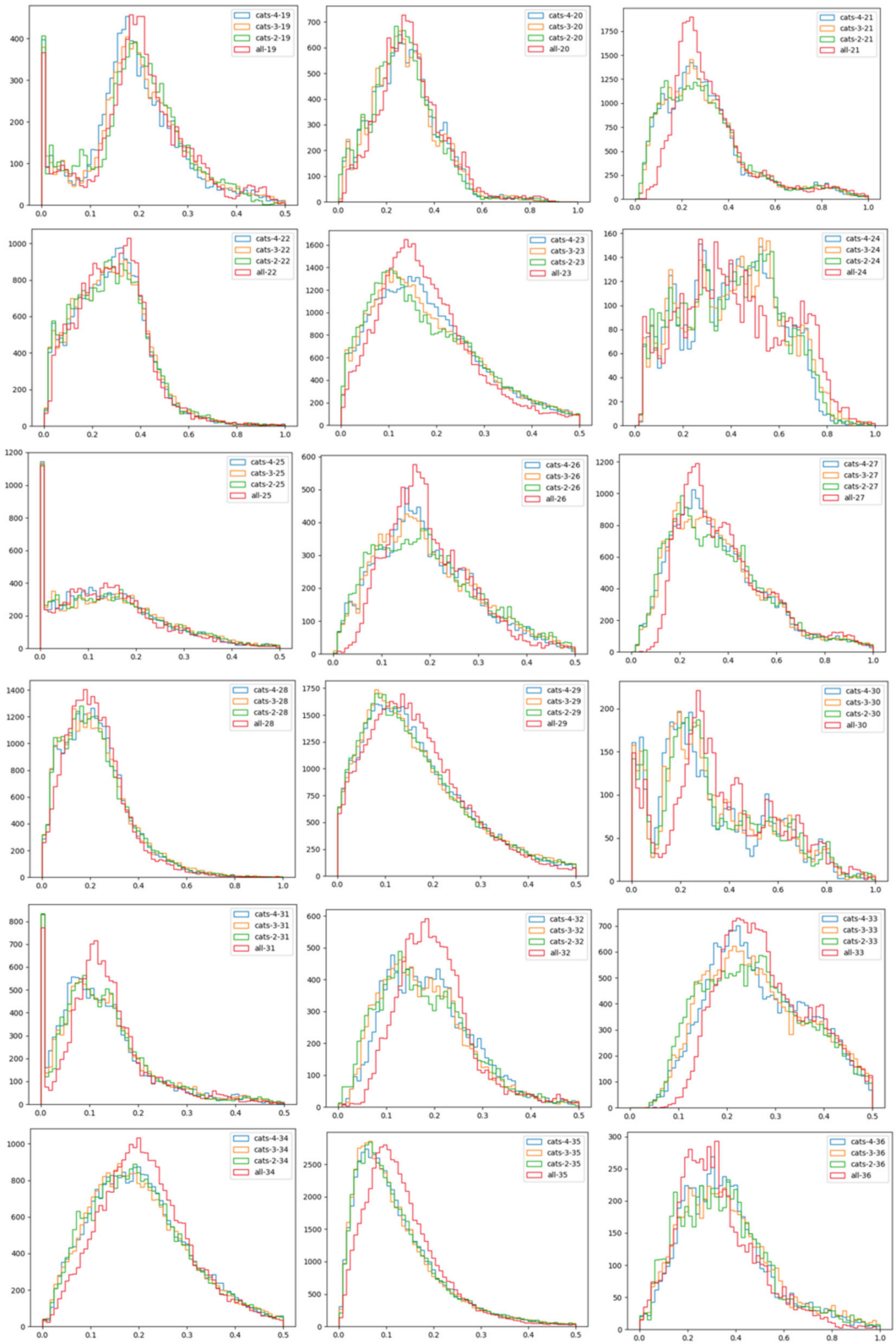


Figure A2. *Cont.*

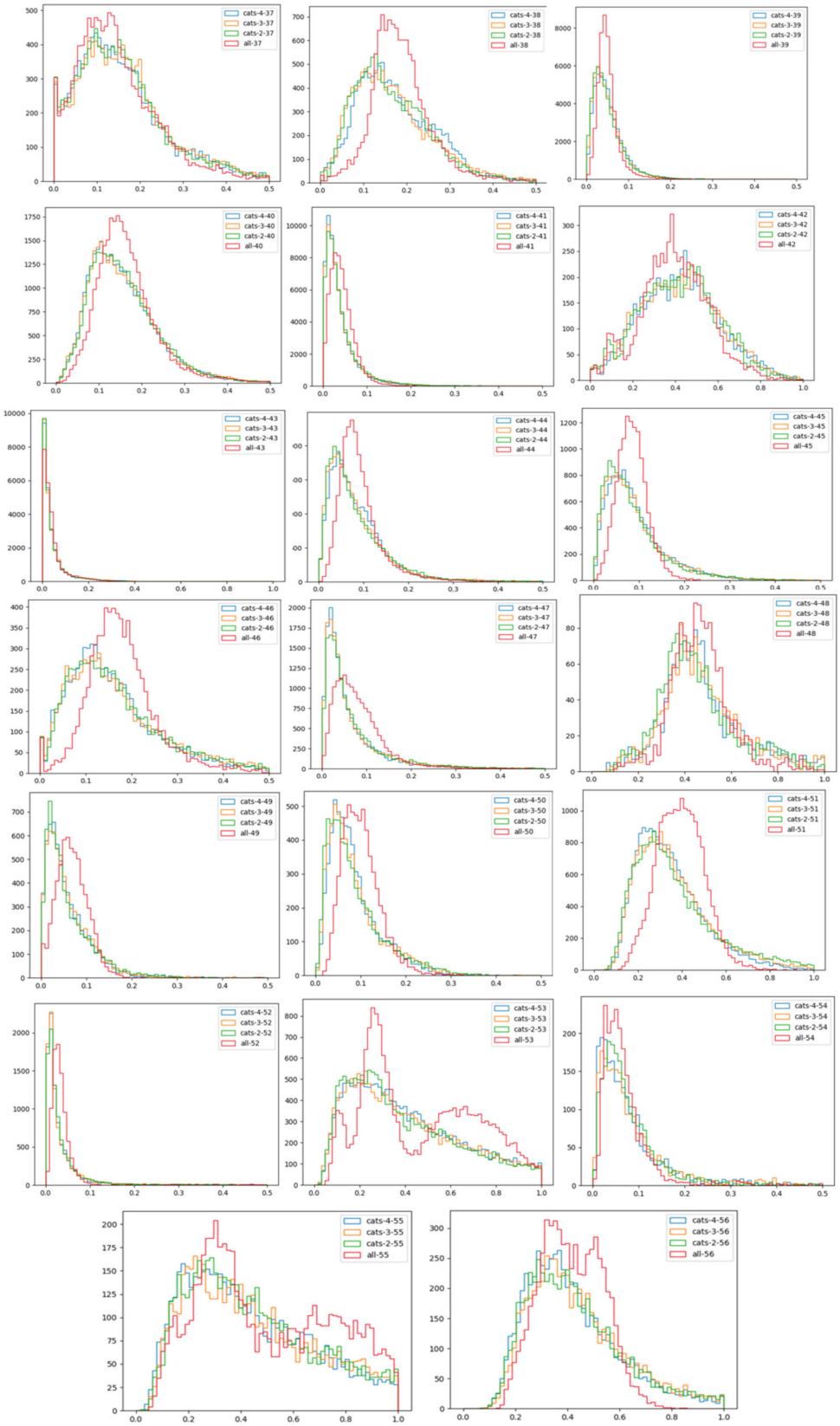


Figure A2. Distribution of GoogLeNet variance tensor elements for specialized tasks: all ImageNet classes, Cats-2, Cats-3, and Cats-4.

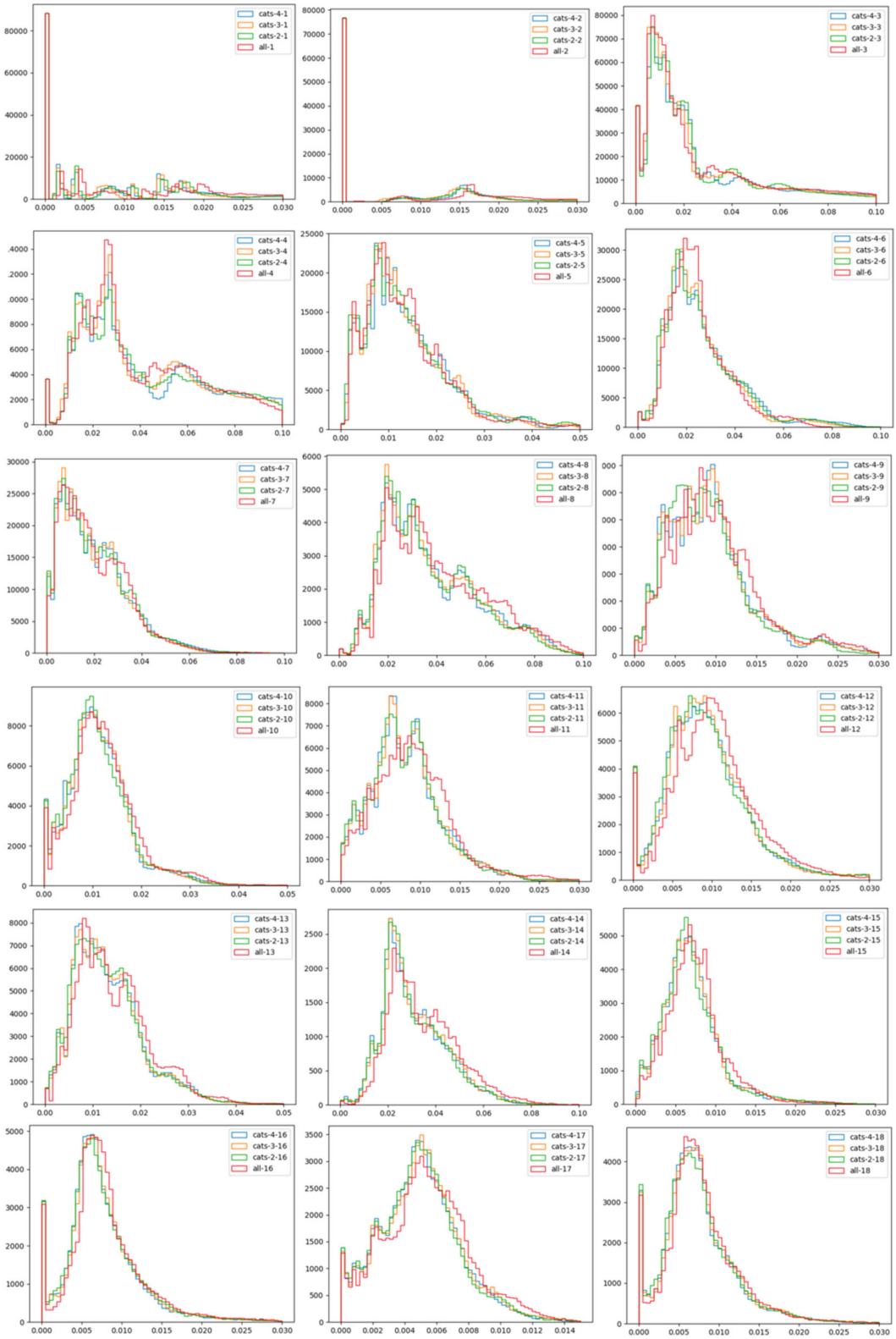


Figure A3. *Cont.*

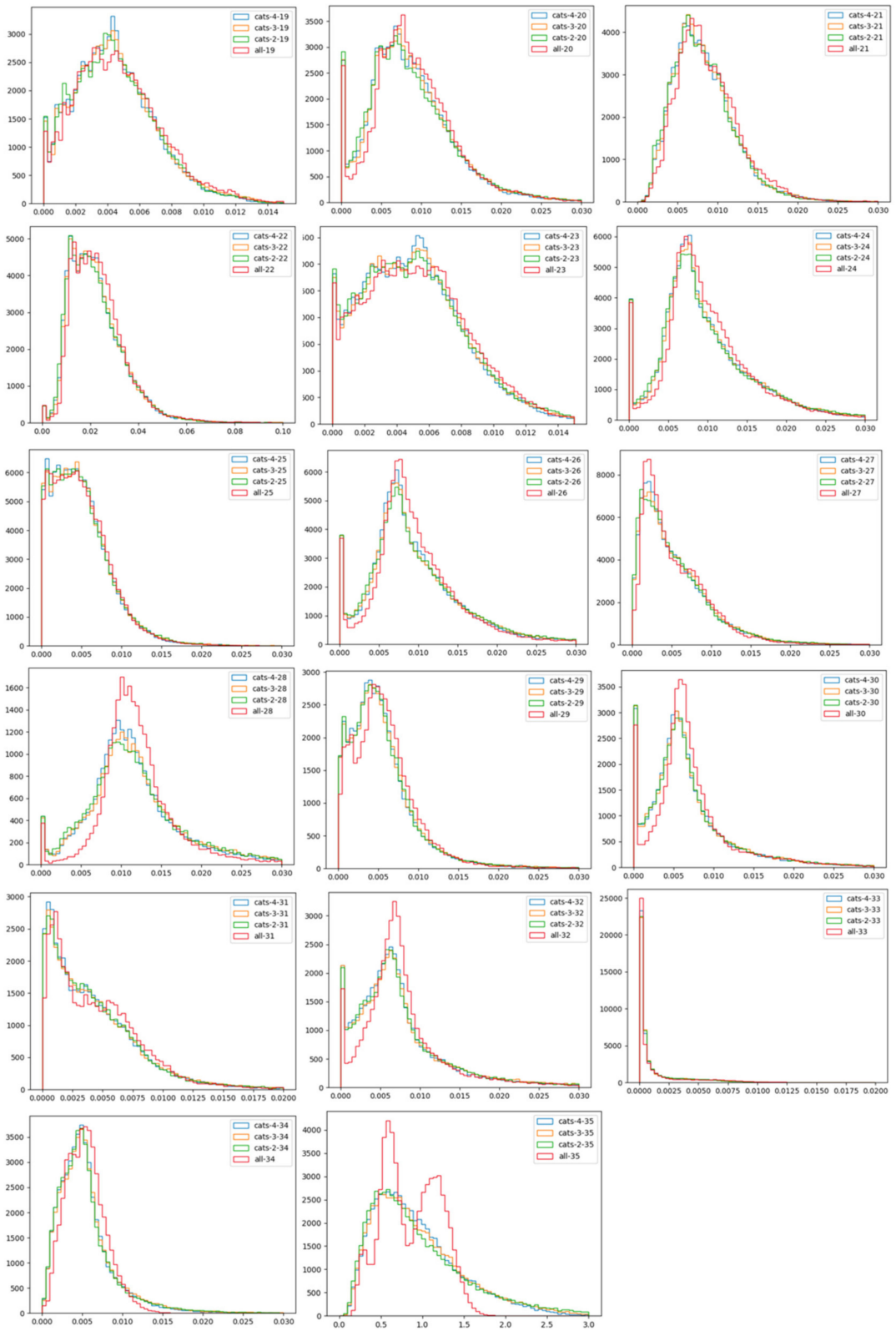


Figure A3. Distribution of MobileNet V2 variance tensor elements for specialized tasks: all ImageNet classes, Cats-2, Cats-3, and Cats-4.

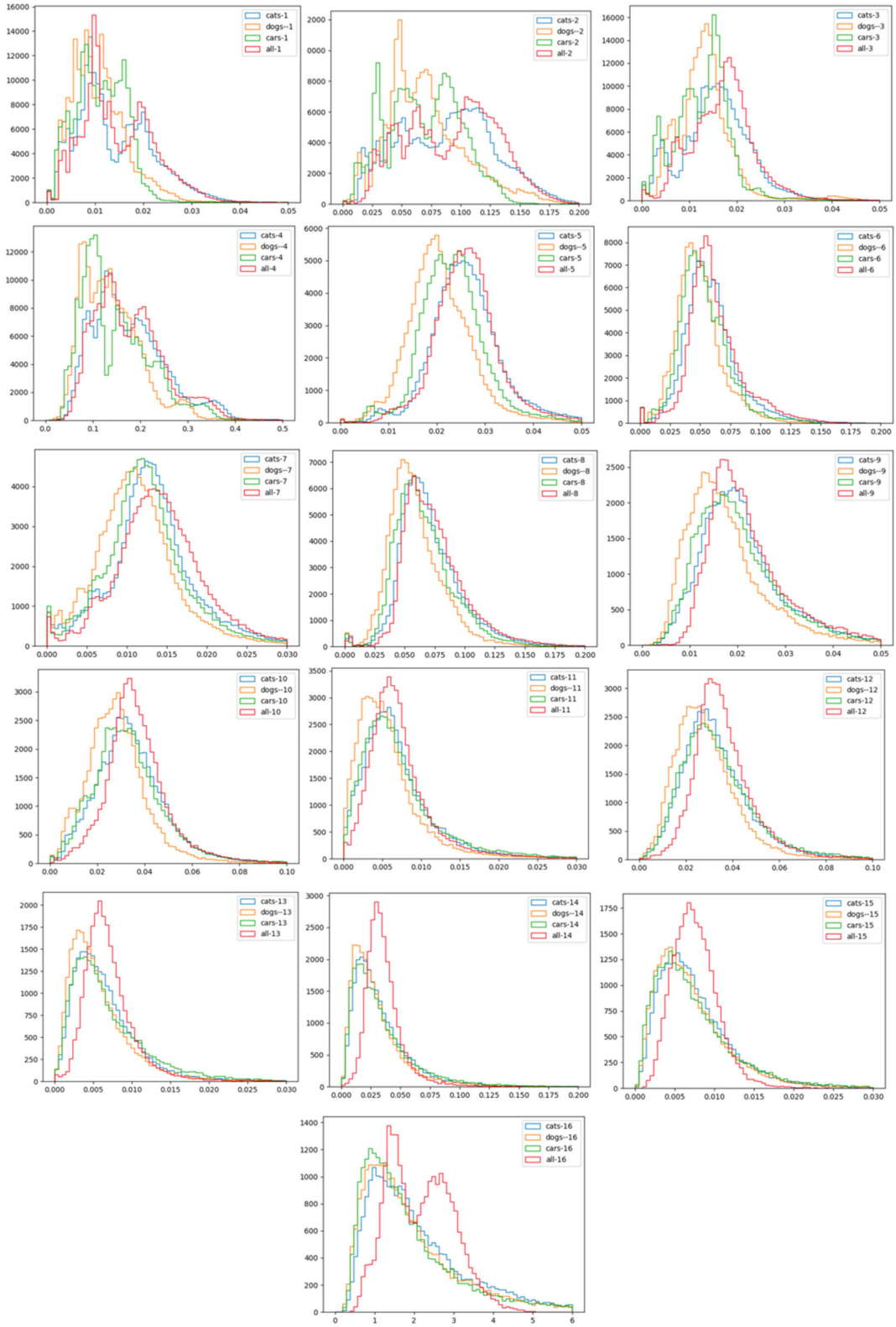


Figure A4. Distribution ResNet-18 variance tensor elements for specialized tasks: Cats, Dogs, Cars, and all ImageNet classes.

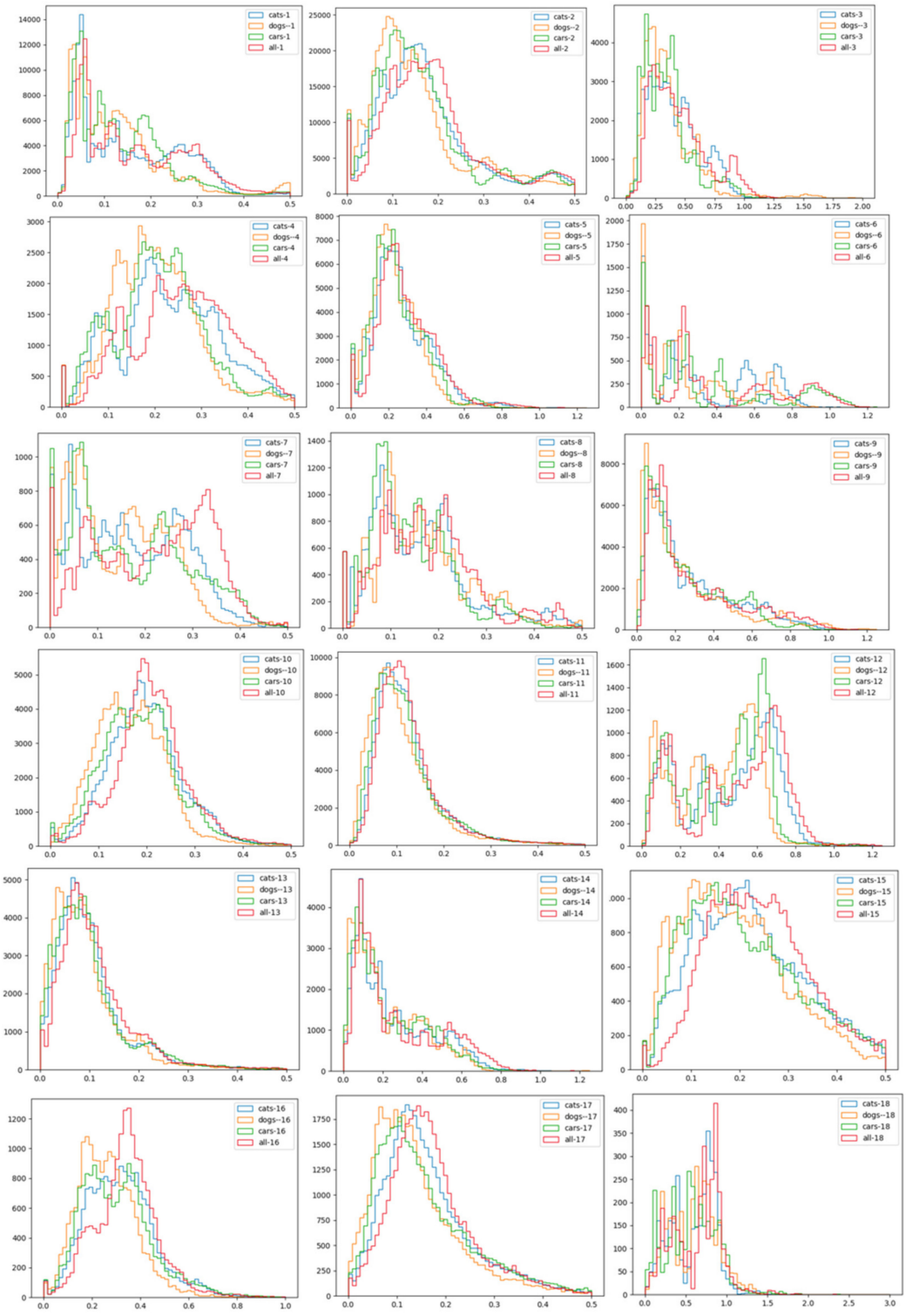


Figure A5. *Cont.*

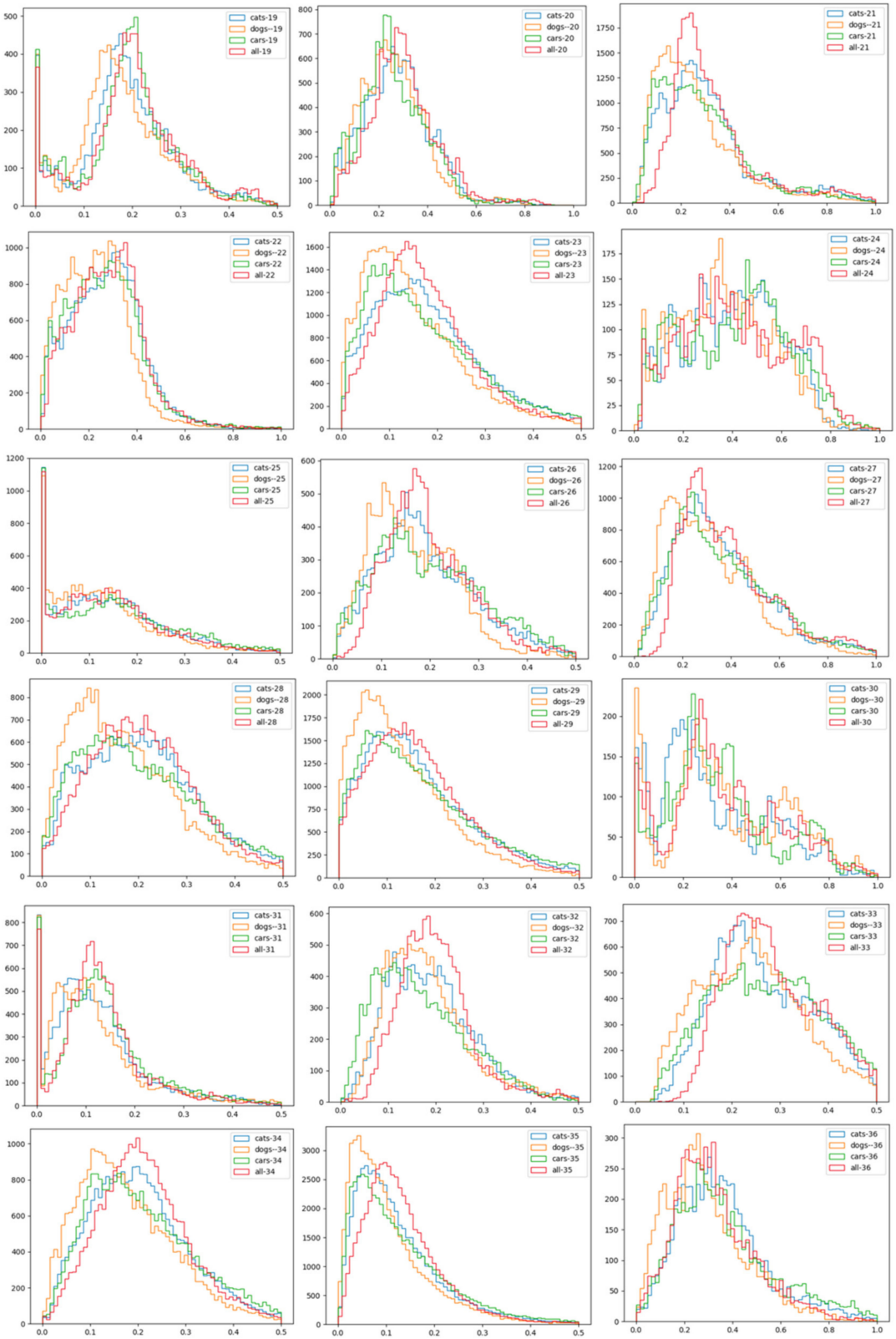


Figure A5. *Cont.*

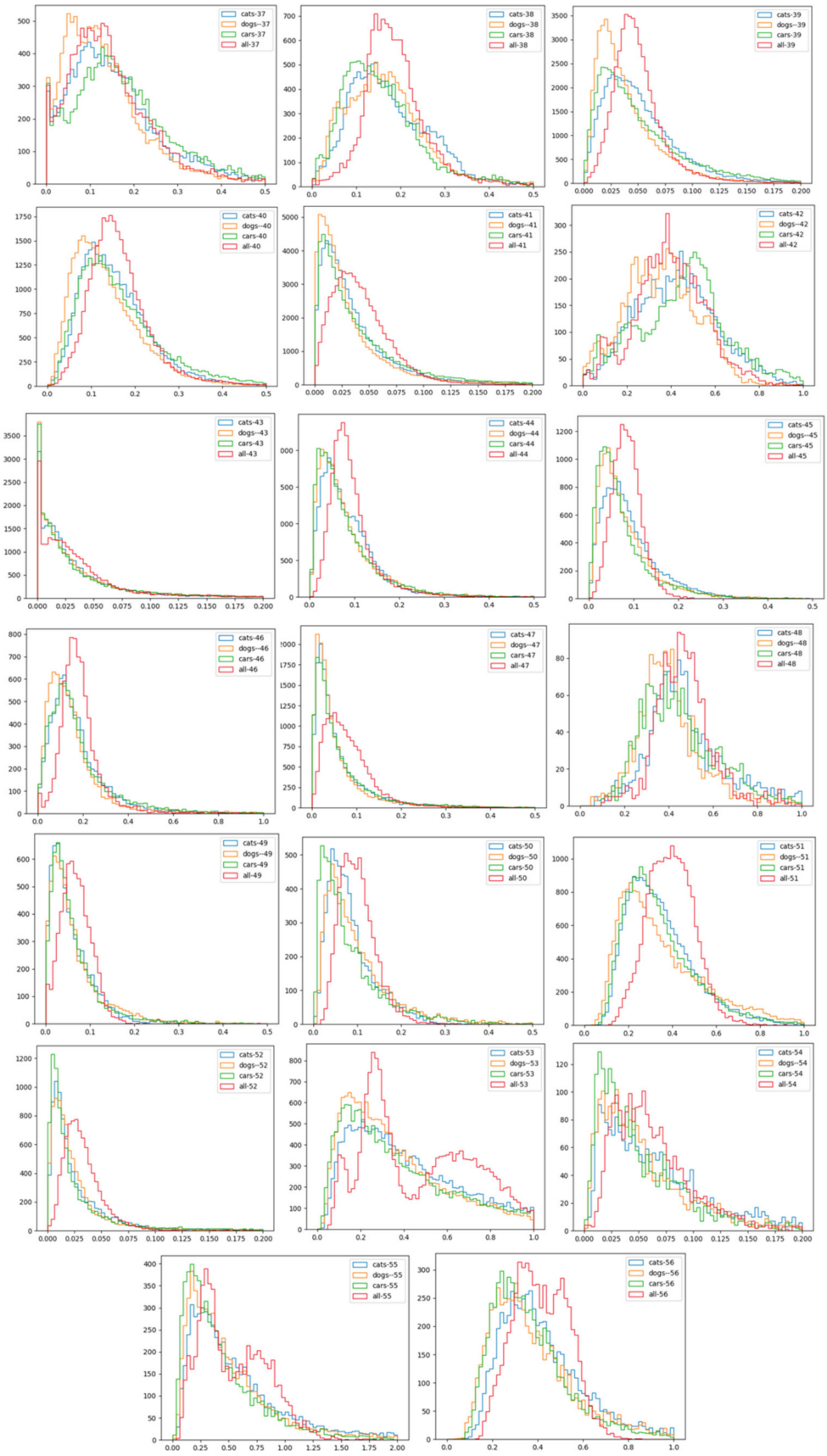


Figure A5. Distribution of GoogLeNet variance tensor elements for specialized tasks: Cats, Dogs, Cars, and all ImageNet classes.

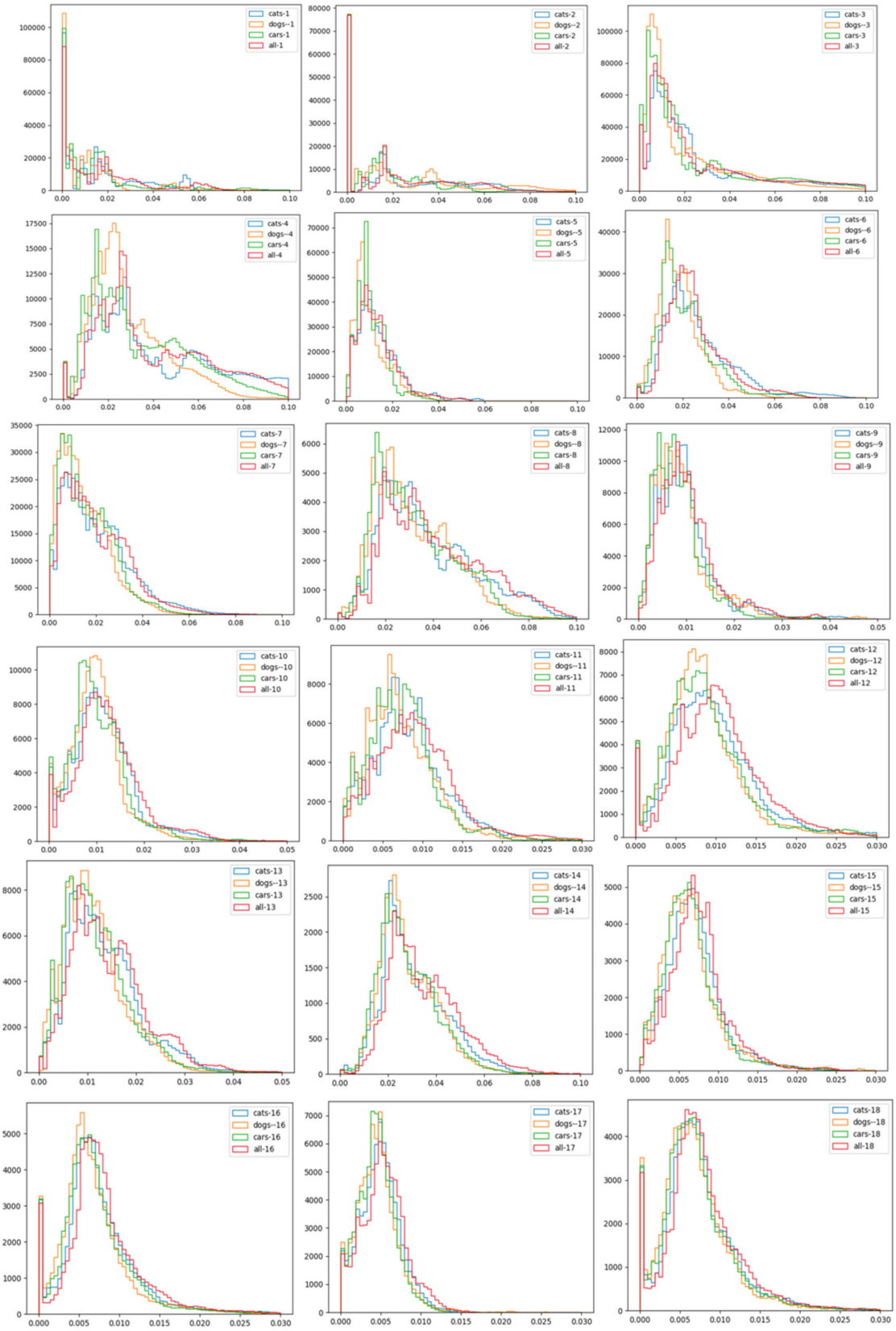


Figure A6. *Cont.*

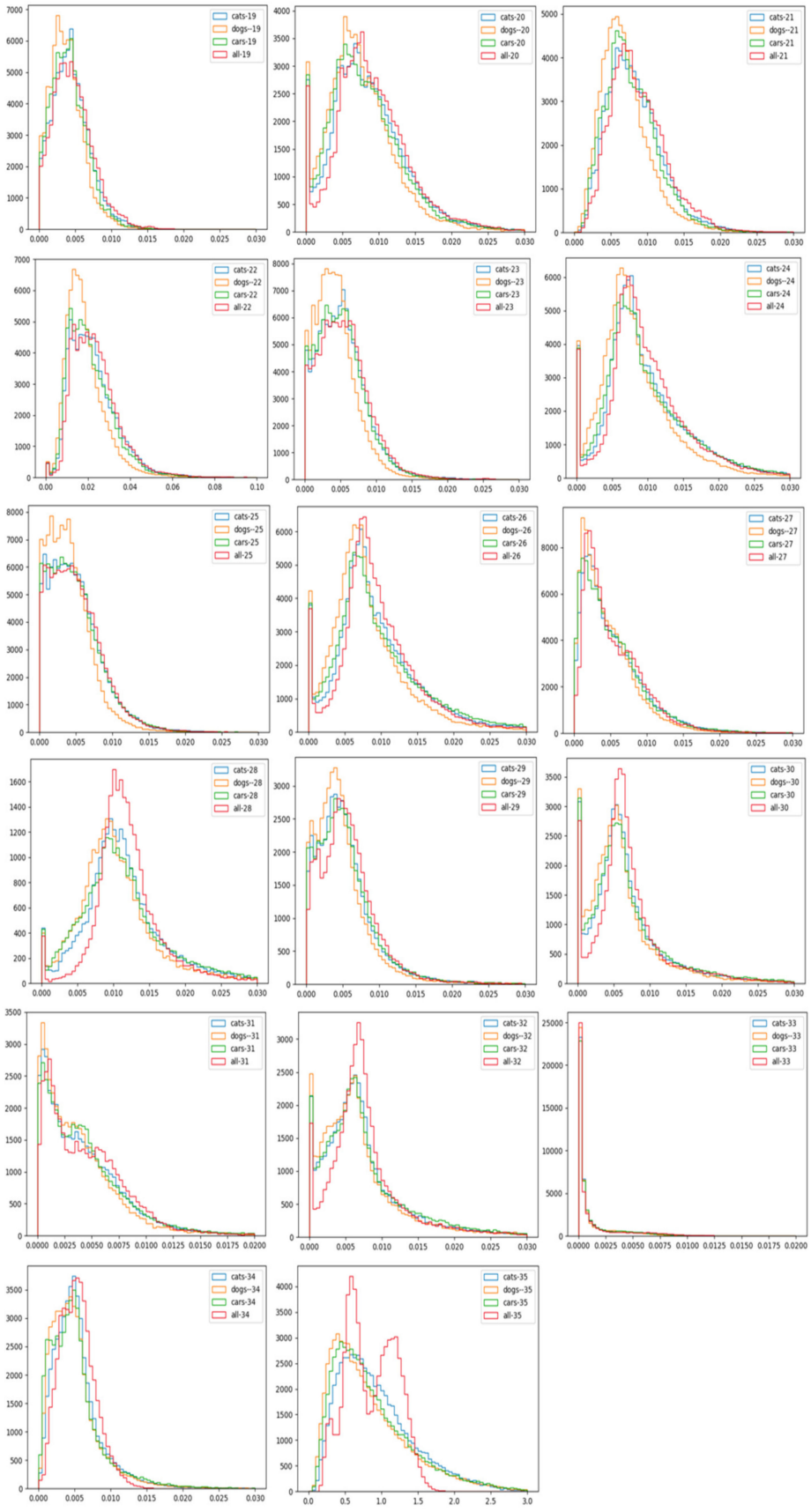


Figure A6. Distribution of MobileNet V2 variance tensor elements for specialized tasks: Cats, Dogs, Cars, and all ImageNet classes.

Appendix B

Table A1. Threshold tuple for Cats.

Compression Saving Ratio	ResNet-18 Threshold Tuple	GoogLeNet Threshold Tuple	MobileNet V2 Threshold Tuple
10%	(3, 3, 4, 4, 10, 10, 10, 10, 10, 10, 10, 10, 70, 10, 80, 90)	(8, 7, 7, 7, 7, 7, 8, 7, 7, 7, 7, 0, 0, 0, 0, 0, 0, 0, 0, 5, 6, 5, 5, 5, 5, 5, 5, 14, 14, 5, 5, 5, 8, 5, 5, 5,15, 15, 15, 36, 37, 18, 20, 34, 34, 34, 25, 40, 34, 34, 90, 90, 96, 90, 92, 92)	(22, 21, 10, 11, 11, 11, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 40, 40, 40, 40, 40, 40, 90)
13.5%	n/a	n/a	(40, 40, 7, 2, 5, 2, 1, 11, 2, 10, 10, 15, 10, 15, 20, 10, 10, 15, 2, 5, 5, 10, 10, 10, 10, 15, 7, 5, 40, 42, 40, 40, 40, 40, 90)
20%	(17, 16, 17, 10, 10, 20, 20, 20, 20, 20, 30, 27, 70, 10, 80, 90)	(17, 15, 15, 15, 15, 15, 17, 15, 15, 15, 15, 19, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 22, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 17, 37, 37, 37, 36, 37, 37, 40, 34, 34, 34, 34, 40, 34, 34, 90, 90, 96, 90, 92, 92)	(35, 35, 18, 17, 17, 20, 8, 8, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 4, 16, 0, 16, 3, 16, 16, 16, 23, 16, 42, 40, 40, 42, 40, 45, 90)
27.3%	(34, 36, 30, 15, 10, 25, 20, 20, 20, 20, 30, 27, 70, 10, 80, 90)	n/a	n/a
30%	(40, 40, 36, 15, 12, 25, 20, 20, 20, 20, 30, 27, 70, 10, 80, 90)	(34, 24, 24, 24, 24, 24, 27, 24, 24, 24, 24, 29, 25, 24, 25, 24, 24, 25, 26, 24, 25, 24, 33, 25, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 25, 24, 27, 56, 56, 54, 52, 56, 54, 60, 52, 52, 52, 52, 60, 52, 52, 90, 90, 96, 90, 92, 92)	(38, 38, 33, 32, 32, 31, 20, 20, 25, 31, 31, 26, 26, 26, 26, 26, 26, 26, 4, 28, 0, 27, 6, 26, 26, 25, 24, 17, 42, 40, 40, 42, 40, 45, 90)
40%	(70, 61, 60, 15, 12, 25, 20, 20, 20, 20, 30, 27, 70, 10, 80, 90)	(46, 32, 32, 32, 32, 32, 36, 32, 32, 32, 32, 36, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 44, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 36, 75, 75, 75, 75, 75, 75, 80, 75, 75, 75, 75, 80, 75, 75, 92, 92, 96, 92, 94, 94)	(66, 66, 35, 17, 32, 20, 17, 38, 13, 52, 52, 64, 44, 58, 58, 42, 42, 54, 13, 33, 33, 36, 36, 36, 36, 54, 26, 21, 80, 80, 80, 80, 80, 80, 90)

Table A2. Threshold tuple for Dogs.

Compression Saving Ratio	ResNet-18 Threshold Tuple	GoogLeNet Threshold Tuple	MobileNet V2 Threshold Tuple
10%	(8, 8, 8, 3, 8, 9, 9, 9, 9, 9, 9, 17, 17, 6, 60, 80)	(7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 10, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 27, 16, 72, 72, 90, 92, 90, 91)	(20, 20, 10, 10, 10, 10, 5, 0, 10, 0, 10, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 40, 40, 40, 40, 40, 40, 90)
19.76%	n/a	n/a	(29, 29, 16, 11, 15, 18, 13, 13, 36, 35, 43, 21, 13, 5, 16, 30, 23, 8, 5, 6, 0, 1, 7, 17, 10, 10, 5, 13, 40, 47, 41, 40, 42, 42, 90)
20%	(21, 21, 20, 6, 10, 21, 20, 20, 20, 18, 33, 24, 30, 6, 71, 80)	(15, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 20, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 55, 32, 72, 72, 90, 92, 90, 91)	n/a

Table A2. *Cont.*

Compression Saving Ratio	ResNet-18 Threshold Tuple	GoogLeNet Threshold Tuple	MobileNet V2 Threshold Tuple
25.46%	n/a	(20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 22, 22, 20, 20, 20, 22, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 22, 20, 20, 20, 20, 20, 20, 20, 20, 25, 50, 52, 50, 50, 50, 52, 50, 52, 50, 50, 55, 55, 68, 42, 92, 90, 90, 92, 90, 91)	n/a
27.7%	(44, 28, 38, 12, 12, 21, 20, 20, 20, 20, 33, 24, 32, 10, 71, 87)	n/a	n/a
30%	(49, 33, 44, 13, 12, 21, 20, 20, 20, 20, 33, 24, 32, 12, 72, 90)	(25, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 26, 24, 24, 24, 26, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 26, 24, 24, 24, 24, 24, 24, 24, 30, 60, 60, 60, 60, 60, 60, 60, 60, 60, 65, 65, 68, 50, 92, 90, 90, 92, 90, 91)	(45, 45, 25, 27, 23, 28, 19, 19, 54, 53, 64, 31, 19, 7, 24, 45, 34, 12, 7, 9, 0, 1, 10, 25, 15, 15, 7, 20, 60, 70, 61, 60, 62, 62, 90)
40%	(50, 50, 50, 50, 40, 21, 20, 20, 20, 20, 33, 24, 32, 10, 70, 90)	(36, 36, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 38, 36, 36, 36, 38, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 45, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 65, 65, 68, 50, 92, 90, 90, 92, 90, 91)	(67, 67, 34, 37, 31, 38, 28, 28, 73, 71, 64, 42, 27, 9, 32, 60, 45, 16, 9, 12, 0, 1, 13, 33, 20, 20, 9, 26, 65, 75, 65, 65, 65, 70, 90)

Table A3. Threshold tuple for Cars.

Compression Saving Ratio	ResNet-18 Threshold Tuple	GoogLeNet Threshold Tuple	MobileNet V2 Threshold Tuple
10%	(3, 5, 5, 10, 10, 10, 10, 10, 10, 10, 10, 12, 30, 13, 50, 80)	(6, 9, 9, 9, 9, 9, 15, 7, 7, 7, 7, 7, 10, 7, 8, 7, 10, 7, 7, 7, 8, 8, 8, 10, 8, 8, 8, 8, 10, 8, 10, 12, 9, 7, 7, 2, 8, 10, 7, 7, 5, 12, 10, 30, 1, 27, 27, 30, 1, 32, 27, 23, 40, 90, 90, 95, 90)	(20, 20, 20, 10, 10, 10, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 20, 20, 30, 75)
16.8%	n/a	n/a	(30, 31, 20, 10, 15, 15, 11, 10, 10, 21, 16, 11, 1, 0, 26, 17, 21, 20, 2, 10, 0, 2, 8, 5, 5, 20, 4, 5, 2, 22, 20, 23, 20, 30, 75)
20%	(21, 21, 20, 20, 20, 20, 10, 20, 20, 10, 10, 12, 30, 13, 50, 80)	(15, 20, 20, 20, 20, 20, 30, 14, 14, 14, 14, 14, 20, 14, 16, 14, 20, 14, 14, 14, 17, 17, 17, 20, 17, 17, 17, 20, 17, 20, 24, 18, 14, 14, 4, 17, 20, 14, 14, 10, 24, 20, 60, 3, 55, 55, 60, 2, 65, 55, 50, 90, 90, 90, 95, 90)	(35, 35, 25, 12, 19, 19, 14, 13, 13, 26, 24, 13, 1, 0, 30, 17, 21, 20, 2, 12, 0, 2, 9, 6, 6, 23, 5, 6, 2, 22, 20, 23, 20, 30, 75)
27.70%	n/a	(20, 30, 30, 30, 30, 30, 45, 20, 20, 20, 20, 20, 30, 20, 25, 20, 30, 20, 20, 20, 25, 25, 25, 30, 25, 25, 25, 30, 25, 30, 35, 25, 20, 20, 7, 25, 30, 20, 30, 15, 35, 20, 60, 3, 55, 55, 60, 2, 65, 55, 50, 90, 90, 90, 95, 90)	
30%	(35, 30, 30, 30, 30, 30, 30, 30, 30, 10, 12, 12, 30, 30, 50, 85)	(24, 32, 32, 32, 32, 32, 49, 22, 22, 22, 22, 22, 32, 22, 27, 22, 33, 22, 22, 22, 28, 28, 27, 33, 28, 28, 27, 33, 27, 33, 38, 27, 22, 22, 7, 27, 32, 22, 33, 16, 38, 20, 66, 3, 60, 60, 66, 2, 71, 60, 55, 90, 90, 90, 94, 89)	(52, 52, 42, 20, 29, 29, 21, 19, 19, 39, 36, 19, 1, 0, 45, 25, 31, 30, 3, 18, 0, 2, 13, 9, 9, 34, 7,8, 3, 22, 20, 23, 20, 30, 75)

Table A3. *Cont.*

Compression Saving Ratio	ResNet-18 Threshold Tuple	GoogLeNet Threshold Tuple	MobileNet V2 Threshold Tuple
40%	(50, 40, 40, 40, 40, 40, 40, 40, 40, 10, 20, 20, 35, 35, 80, 90)	(36, 44, 44, 44, 44, 44, 55, 32, 32, 32, 30, 30, 43, 30, 37, 30, 44, 30, 30, 30, 40, 40, 38, 44, 36, 36, 36, 44, 30, 44, 52, 37, 30, 30, 12, 40, 44, 40, 45, 18, 55, 30, 67, 10, 65, 65, 66, 60, 71, 60, 55, 90, 90, 90, 94, 89)	(67, 67, 34, 37, 31, 38, 28, 28, 73, 71, 64, 42, 27, 9, 32, 60, 45, 16, 9, 12, 0, 1, 13, 33, 20, 20, 9, 26, 65, 75, 65, 65, 65, 70, 90)

References

1. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. ImageNet Classification with Deep Convolutional Neural Networks. *Commun. ACM* **2017**, *60*, 84–90. [CrossRef]
2. 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.
3. Bianco, S.; Cadene, R.; Celona, L.; Napoletano, P. Benchmark Analysis of Representative Deep Neural Network Architectures. *IEEE Access* **2018**, *6*, 64270–64277. [CrossRef]
4. 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]
5. Reed, R. Pruning algorithms—A survey. *IEEE Trans. Neural Netw.* **1993**, *4*, 740–747. [CrossRef]
6. 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.
7. 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.
8. Zhang, C.; Bengio, S.; Hardt, M.; Recht, B.; Vinyals, O. Understanding Deep Learning Requires Rethinking Generalization. *arXiv* **2016**, arXiv:1611.03530.
9. 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.
10. Gong, Y.; Liu, L.; Yang, M.; Bourdev, L. Compressing deep convolutional networks using vector quantization. *arXiv* **2014**, arXiv:1412.6115.
11. 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.
12. Lin, Z.; Courbariaux, M.; Memisevic, R.; Bengio, Y. Neural networks with few multiplications. *arXiv* **2015**, arXiv:1510.03009.
13. 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.
14. 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]
15. 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 Proceedings of the 38th International Conference on Machine Learning, PMLR 139, Long Beach, CA, USA, 18–24 July 2021.
16. 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.
17. 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.
18. 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.
19. 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.
20. 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.
21. Han, S.; Pool, J.; Tran, J.; Dally, W.J. Learning both weights and connections for efficient neural networks. *arXiv* **2015**, arXiv:1506.02626.
22. 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.

23. Castellano, G.; Fanelli, A.M.; Pelillo, M. An iterative pruning algorithm for feedforward neural networks. *IEEE Trans. Neural Netw.* **1997**, *8*, 519–531. [CrossRef] [PubMed]
24. Collins, M.D.; Kohli, P. Memory bounded deep convolutional networks. *arXiv* **2014**, arXiv:1412.1442.
25. Stepniewski, S.W.; Keane, A.J. Pruning backpropagation neural networks using modern stochastic optimisation techniques. *Neural. Comput. Appl.* **1997**, *5*, 76–98. [CrossRef]
26. Liu, Z.; Sun, M.; Zhou, T.; Huang, G.; Darrell, T. Rethinking the Value of Network Pruning. *arXiv* **2018**, arXiv:1810.05270.
27. Anwar, S.; Hwang, K.; Sung, W. Structured pruning of deep convolutional neural networks. *ACM J. Emerg. Technol. Comput. Syst.* **2017**, *13*, 1–18. [CrossRef]
28. 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.
29. 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.
30. 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.
31. Li, H.; Kadav, A.; Durdanovic, I.; Samet, H.; Graf, H.P. Pruning Filters for Efficient ConvNets. *arXiv* **2016**, arXiv:1608.08710.
32. 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: Guildford, UK, 2015.
33. 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.
34. Shomron, G.; Weiser, U. Spatial Correlation and Value Prediction in Convolutional Neural Networks. *IEEE Comput. Arch. Lett.* **2019**, *18*, 10–13. [CrossRef]
35. 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.
36. See, A.; Luong, M.-T.; Manning, C.D. Compression of Neural Machine Translation Models via Pruning. *arXiv* **2016**, arXiv:1606.09274.
37. Narang, S.; Elsen, E.; Diamos, G.; Sengupta, S. Exploring Sparsity in Recurrent Neural Networks. *arXiv* **2017**, arXiv:1704.05119.
38. Zhu, M.; Gupta, S. To prune, or not to prune: Exploring the efficacy of pruning for model compression. *arXiv* **2017**, arXiv:1710.01878.
39. 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.
40. 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.
41. 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.
42. Molchanov, P.; Tyree, S.; Karras, T.; Aila, T.; Kautz, J. Pruning Convolutional Neural Networks for Resource Efficient Inference. *arXiv* **2016**, arXiv:1611.06440.
43. 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.
44. 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.
45. 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.
46. 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.
47. Banner, R.; Nahshan, Y.; Hoffer, E.; Soudry, D. Post-training 4-bit quantization of convolution networks for rapid-deployment. *arXiv* **2018**, arXiv:1810.05723.
48. 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.
49. 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.
50. Shomron, G.; Gabbay, F.; Kurzum, S.; Weiser, U. Post-Training Sparsity-Aware Quantization. *arXiv* **2021**, arXiv:2105.11010.
51. 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.
52. Hinton, G.; Vinyals, O.; Dean, J. Distilling the Knowledge in a Neural Network. *arXiv* **2015**, arXiv:1503.02531.
53. Gou, J.; Yu, B.; Maybank, S.J.; Tao, D. Knowledge Distillation: A Survey. *Int. J. Comput. Vis.* **2021**, *129*, 1789–1819. [CrossRef]

54. 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.
55. Zeiler, M.D.; Fergus, R. Visualizing and Understanding Convolutional Networks. In *Computer Vision—ECCV 2014*; Springer International Publishing: Cham, Switzerland, 2014; pp. 818–833.
56. Zhou, B.; Khosla, A.; Lapedriza, A.; Oliva, A.; Torralba, A. Object Detectors Emerge in Deep Scene CNNs. *arXiv* **2014**, arXiv:1412.6856.
57. 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.
58. Zhou, B.; Sun, Y.; Bau, D.; Torralba, A. Revisiting the Importance of Individual Units in CNNs via Ablation. *arXiv* **2018**, arXiv:1806.02891.
59. 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.
60. 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.
61. 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.
62. 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.
63. 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).
64. 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).