applied sciences

MDPI

Article

Learning Low-Precision Structured Subnetworks Using Joint Layerwise Channel Pruning and Uniform Quantization

Xinyu Zhang [1,†], Ian Colbert [2,†] and Srinjoy Das [3,*]

1 Department of Computer Science, Rutgers University, New Brunswick, NJ 08901, USA
2 Department of Electrical and Computer Engineering, University of California, San Diego, CA 92093, USA
3 School of Mathematical and Data Sciences, West Virginia University, Morgantown, WV 26506, USA
* Correspondence: srinjoy.das@mail.wvu.edu
† These authors contributed equally to this work.

Abstract: Pruning and quantization are core techniques used to reduce the inference costs of deep neural networks. Among the state-of-the-art pruning techniques, magnitude-based pruning algorithms have demonstrated consistent success in the reduction of both weight and feature map complexity. However, we find that existing measures of neuron (or channel) importance estimation used for such pruning procedures have at least one of two limitations: (1) failure to consider the interdependence between successive layers; and/or (2) performing the estimation in a parametric setting or by using distributional assumptions on the feature maps. In this work, we demonstrate that the importance rankings of the output neurons of a given layer strongly depend on the sparsity level of the preceding layer, and therefore, naïvely estimating neuron importance to drive magnitude-based pruning will lead to suboptimal performance. Informed by this observation, we propose a purely data-driven nonparametric, magnitude-based channel pruning strategy that works in a greedy manner based on the activations of the previous sparsified layer. We demonstrate that our proposed method works effectively in combination with statistics-based quantization techniques to generate low precision structured subnetworks that can be efficiently accelerated by hardware platforms such as GPUs and FPGAs. Using our proposed algorithms, we demonstrate increased performance per memory footprint over existing solutions across a range of discriminative and generative networks.

Keywords: channel pruning; layerwise pruning; quantization; joint pruning; quantization

check for updates

Citation: Zhang, X.; Colbert, I.; Das, S. Learning Low-Precision Structured Subnetworks Using Joint Layerwise Channel Pruning and Uniform Quantization. *Appl. Sci.* **2022**, *12*, 7829. https://doi.org/10.3390/app12157829

Academic Editors: Deliang Fan and Zhezhi He

Received: 1 July 2022
Accepted: 23 July 2022
Published: 4 August 2022

1. Introduction

The performance of deep neural networks (DNNs) has been shown to scale with the size of both the training dataset and model architecture [1]; however, the resources required to deploy larger networks for inference can be prohibitive as they often exceed the compute and storage budgets of resource-constrained platforms such as mobile or edge devices [2,3]. Therefore, as the usage of deep learning has proliferated in real-time applications with tight energy consumption budgets and low latency requirements, the field of research focused on reducing inference costs while maintaining model performance has rapidly expanded in recent years. Of the many techniques studied to accomplish this task, *pruning* and *quantization* are the most widely used and are often complementary to other approaches such as network distillation [4] and neural architecture search [5,6].

Pruning (i.e., the process of removing identified redundant elements from a neural network) and quantization (i.e., the processing of reducing their precision) are often considered to be independent problems [7,8]; however, recent work has begun to study the application of both in either a joint [2,6,9,10] or unified [11,12] setting. *Unified* algorithms typically use mixed precision quantization and integrate pruning by reducing the precision of an element (or a set of elements) to 0. On the other hand, *joint* algorithms combine separate optimization objectives for pruning and quantization under one learning framework, often using the standard "prune-then-quantize" paradigm [2]. In this work, we study the joint application of pruning and quantization under this paradigm across both discriminative and generative tasks for the purpose of learning low-precision structured subnetworks that can be efficiently accelerated by highly-parallelized hardware platforms.

Motivated by our observation that data-driven measures of neuron importance strongly depend on the activation distribution of the preceding layer, we design a greedy layerwise channel pruning algorithm that is heuristically guided by nonparametric estimates. We evaluate the performance of our algorithm using various pruning schedules and alternative measures of neuron importance based on pre-existing literature and observe that our greedy layerwise algorithm yields consistent benefits. Intuitively, our results suggest that, by allowing a given layer to adjust to abrupt shifts to its input activation distribution *before* pruning, the heuristics used to rank order neurons by importance become more effective, which leads to improved network performance. Furthermore, when combined with our moving average statistics-based uniform quantization procedure, we are able to learn low-precision structured subnetworks with minimal performance degradation for both discriminative and generative tasks. Our joint pruning and quantization algorithm is visualized in Figure 1, where we depict the sequence of pruning and quantization steps used during training. As further described in Section 4, our framework uses data-driven importance measures to guide our greedy layerwise channel pruning algorithm before our quantization operator is activated to reduce the precision of both the weights $\boldsymbol{w}_i$ and activations $\boldsymbol{h}_i$ for each layer $i \in \{1, \cdots, L\}$. The pruning mask for each layer i is evaluated for t_p steps before moving to layer $i+1$. After all layers are pruned to a target sparsity, we activate the quantization operators and fine-tune for t_q steps. It is important to note that, while only the pruning mask of one layer is tuned every t_p steps, the weights in all layers are updated through gradient descent in every step, and latent operators act as identity functions until activated.

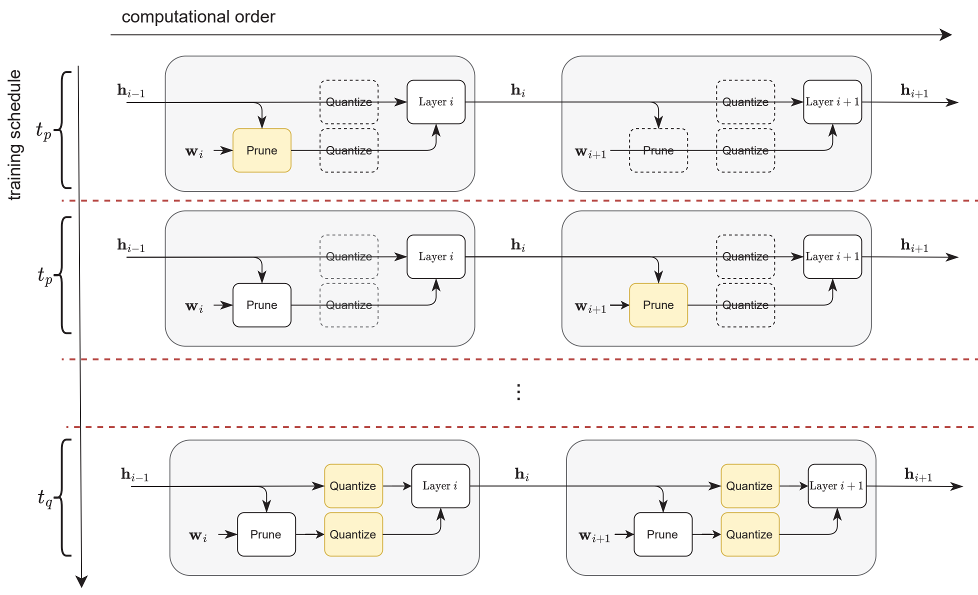


Figure 1. We introduce a joint layerwise channel pruning and uniform quantization framework built from algorithms formulated using moving average statistics. Here, yellow blocks denote operators actively being evaluated, clear blocks with dotted lines denote latent operators that have yet to be activated, and white blocks denote activated operators that have already been evaluated.

The primary contributions of our work are as follows:

1. We design a greedy layerwise channel pruning strategy using a nonparametric data-driven importance measure built without invoking any distributional assumptions.

2. We build a fully data-driven nonparametric framework to learn performant low-precision structured subnetworks by combining our layerwise channel pruning algorithm with quantization-aware training.
3. We evaluate our algorithm using alternative pruning schedules and neuron importance measures and demonstrate clear advantages over pre-existing approaches.
4. We demonstrate increased performance per memory footprint over existing solutions across a wide range of discriminative and generative computer vision tasks.

The outline of the rest of the paper is as follows. In Section 2, we review prior work in neural network pruning and quantization. In Section 3, we motivate our intuition for data-driven layerwise pruning using nonparametric measures of correlation and distance between rank orderings of neuron importance. In Section 4, we describe our layer-by-layer channel pruning and moving average statistics-based quantization-aware training algorithms. In Section 5, we evaluate the performance results of our joint channel pruning and uniform quantization framework using discriminative and generative networks. In Section 6, we conclude the paper and discuss directions for future work.

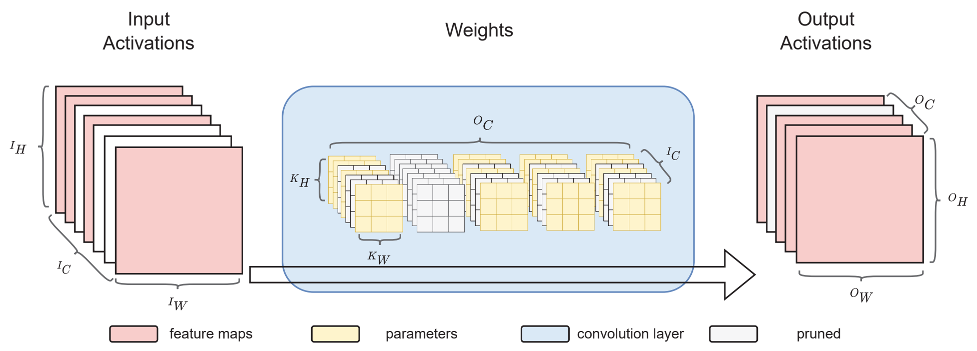
2. Background

Our work explores the joint application of channel pruning and uniform quantization on both the weights and activations of deep neural networks. Here, we motivate our selected configurations for both.

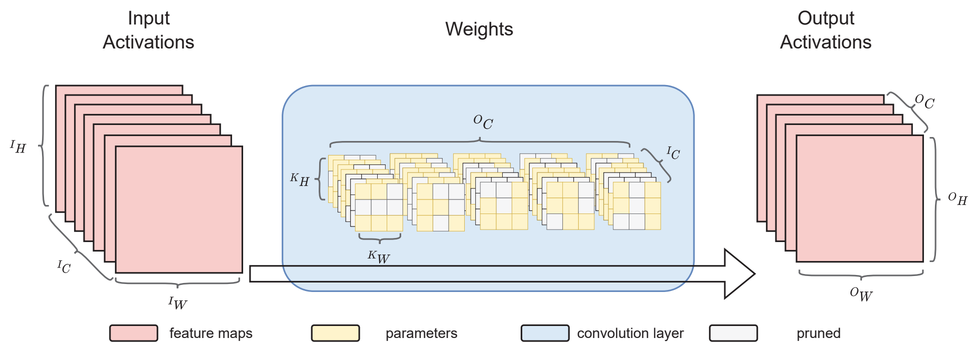
2.1. Pruning

Neural network pruning techniques aim to reduce the inference costs of overparameterized models by identifying subnetworks that minimize memory requirements while maintaining task performance. In practice, pruning is often performed by setting the values of identified elements to zero, and the proportion of zero-valued elements is referred to as *sparsity*, where higher values correspond to fewer non-zero elements. Given a target sparsity, there are a variety of criteria used to identify which elements to prune; the most important of which are the topology constraints on the resulting subnetwork, the measure used to rank elements by importance, and the schedule in which elements are pruned.

Topology constraints for pruning techniques can be divided into *structured* or *unstructured* approaches. Structured pruning techniques introduce sparsity in varying levels of granularity (e.g., entire kernels or channels), whereas unstructured pruning techniques impose no constraint on the topology of the sparsity scheme, as shown in Figure 2. Due to their inherent flexibility, unstructured pruning techniques can offer high compression rates with minimal accuracy degradation [2,13]; however, they bring little hardware efficiency due to poor data locality caused by irregular sparsity patterns, which create minimal opportunities for parallelism [14,15]. Alternatively, structured pruning techniques offer more hardware-friendly implementations, often by removing entire channels. This not only reduces the compute workload in a manner that is easily accelerated by most computing platforms [14] but also reduces energy consumption as activations dominate data transfer costs for both discriminative [16] and generative [17] models. Therefore, we focus on channel pruning in this work.



(a) Structured Pruning



(b) Unstructured Pruning

Figure 2. We depict the differences between structured (**a**) structured pruning and (**b**) unstructured pruning.

Various measures have been proposed to heuristically determine the relative importance of channels in a neural network. Weight magnitude has become the standard importance measure used to guide unstructured pruning algorithms [2,13,18], intuitively suggesting that larger magnitudes have greater influence in the network. Thus, weight magnitude can be extended to rank order channels using the ℓ_1-norm of channel weights [19]. We refer to such importance measures as *data-free*, as they do not require a data distribution to estimate neuron importance. Alternatively, purely *data-driven* approaches such as [20] prioritize removing redundancy from the information (i.e., hidden activations) propagated through the network from the input data. This class of techniques has shown tremendous promise in finding performant structured subnetworks by heuristically guiding pruning algorithms to minimize feature reconstruction error [15], remove underutilized channels [20], or minimize the mutual information between successive layers [21]. However, we find that these approaches are limited by one or two key oversights:

1. They do not consider the interdependence between successive layers in a neural network when measuring neuron importance, as in the case of [19,20,22].
2. They either use a parametric setting or invoke distributional assumptions on the activation data that may not hold true for all network architectures, as in the case of [15,21,23,24].

In this paper, using a completely nonparametric framework, we design a layerwise channel pruning algorithm using a data-driven measure of channel importance. Using targeted statistical experiments, we focus on some important differences between data-driven and data-free approaches when constructing our pruning algorithms. These experiments, further discussed in Section 3, also motivate the rationale for our layerwise approach.

2.2. Quantization

We are interested in quantization for the purpose of accelerating our structured subnetworks on mobile or edge devices; thus, we construct our "quantization" and "dequantization" operators from the standard *uniform affine* mapping from a high precision real number r to a low-precision quantized number q using a scaling factor s and zero-point z, as given by Equation (1). Although more complex non-uniform and non-linear mappings have been considered [4,25], their utility and practicality are often circumstantial as they require dequantization before performing computation in the high precision real domain and are thus far less efficient on resource-constrained hardware [26].

$$r = s \cdot (q - z) \tag{1}$$

As is standard practice, our quantizer (Equation (2)) and dequantizer (Equation (3)) are parameterized by scaling factor s and zero-point z. Here, s is a strictly positive real scaling factor and z is an integer value that maps to the real zero such that the real zero is exactly representable in the quantized domain. This ensures the numerical fidelity of common operations such as zero-padding [27,28]. Here, $\lfloor \cdot \rceil$ denotes the half-way rounding function and $\text{clip}(x; n, p) = \min(\max(x, n), p)$, where n and p are the clipping limits defined by the bitwidth b. For signed integers, $n = -2^{b-1}$ and $p = 2^{b-1} - 1$. For unsigned integers, $n = 0$ and $p = 2^b - 1$. As is standard practice, we use per-tensor scaling factors on the activations and per-channel scaling factors on only the weights [29].

$$\text{quantize}(x; s, z) := \text{clip}\left(\left\lfloor \frac{x}{s} \right\rceil + z; n, p \right) \tag{2}$$

$$\text{dequantize}(x; s, z) := s \cdot (x - z) \tag{3}$$

As reported in previous studies, eliminating zero points in the quantization mapping such that $z = 0$ reduces the computational overhead of cross-terms when executing inference using integer-only arithmetic [29,30]. This strategy is commonly referred to as *symmetric quantization*, with *asymmetric quantization* as its alternative. To demonstrate the computational overhead of cross-terms introduced by asymmetric quantization, consider the real values for input activation x, weight w, and output activation y mapping to quantized values q_x, q_w, and q_y, respectively. The arithmetic for their product, $y = x \cdot w$, becomes the following:

$$s_y(q_y - z_y) = s_x(q_x - z_x) \cdot s_w(q_w - z_w) \tag{4}$$

where s_x, s_w, and s_y represent their respective scaling factors and z_x, z_w, and z_y represent their respective zero points. This simplifies to the following:

$$q_y = \frac{s_x s_w}{s_y}(q_x q_w - q_x z_w - q_w z_x + z_x z_w) + z_y \tag{5}$$

These cross-terms often require non-trivial optimizations to remain efficient; however, by constraining the quantization scheme of all weights in a DNN to be symmetric (i.e., $z_w = 0$), we can mask the overhead of asymmetric quantization on the activations without any additional hardware or software optimizations as the arithmetic simplifies to Equation (6), where l denotes layer $l \in \{0, \cdots, L\}$ in a neural network with L sequential layers.

$$\left(q_x^{(l+1)} - z_x^{(l+1)}\right) = \frac{s_x^{(l)} s_w^{(l)}}{s_x^{(l+1)}} q_w^{(l)} \left(q_x^{(l)} - z_x^{(l)}\right) \tag{6}$$

Note that, when using a symmetric quantizer on both the inputs and outputs to the neural network (i.e., $x^{(0)}$ and $x^{(L)}$, respectively), we can freely use asymmetric quantization on the hidden layers without any overhead caused by the cross terms in Equation (5). While previous works have alluded to this optimization [26], we are not aware of any research that has explicitly exploited it as we do. In our work, we focus on uniform affine quantization, where we apply symmetric, per-channel quantization to the weights and asymmetric, per-tensor quantization to the activations.

3. Motivation

To demonstrate the interdependence between successive layers in a neural network, we inject structured sparsity into the input activations of a given layer within the network and evaluate how this leads to a shift in output channel importance rankings with respect to the dense input activations (i.e., when there is no input sparsity). To rank order output channels by importance, we use two measures: (1) the mean ℓ_1-norm of the output activations generated by each channel, and (2) the ℓ_1-norm of the learned weights for each output channel. For a given layer i with hidden activations $\boldsymbol{h}_i$ and C_i output channels, we denote the importance estimates for each channel as $\mu_{i,c}$ where $c \in \{1, \cdots, C_i\}$. In order to compare two sets of rankings, we use the following nonparametric measures:

- **Kendall's Coefficient of Rank Correlation [31]:**
  For a given layer i with C_i outputs channels, let the rank orderings of two sets of importance estimates $\boldsymbol{\mu}_i = \{\mu_{i,1}, \cdots, \mu_{i,C_i}\}$ and $\boldsymbol{\nu}_i = \{\nu_{i,1}, \cdots, \nu_{i,C_i}\}$ be given by r_{μ_i} and r_{ν_i}, respectively. The Kendall's coefficient of rank correlation measures the similarity between r_{μ_i} and r_{ν_i}. The statistic τ (referred to as Kendall's Tau) is given below in Equation (7), where $\text{sign}(x)$ is given by Equation (8). Here, $\tau = 1$ is a perfect relationship, $\tau = 0$ is no relationship at all, and $\tau = -1$ is a perfect negative relationship.

$$\tau = \frac{2}{n(n-1)} \sum_{k<j} \text{sign}(\mu_{i,k} - \mu_{i,j}) \cdot \text{sign}(\nu_{i,k} - \nu_{i,j}) \tag{7}$$

$$\text{sign}(x) = \begin{cases} +1 & \text{if x > 0} \\ 0 & \text{if x = 0} \\ -1 & \text{if x < 0} \end{cases} \tag{8}$$

- **Levenshtein Distance [32]:**
  For a given layer i with C_i outputs channels, let the rank orderings of two sets of importance estimates $\boldsymbol{\mu}_i = \{\mu_{i,1}, \cdots, \mu_{i,C_i}\}$ and $\boldsymbol{\nu}_i = \{\nu_{i,1}, \cdots, \nu_{i,C_i}\}$ be given by r_{μ_i} and r_{ν_i}, respectively. The Levenshtein distance between these two sequences of ranks, which we denote as $\text{lev}(r_{\mu_i}, r_{\nu_i})$, is defined as the minimum number of single element edits required to change r_{μ_i} to r_{ν_i}. The distance is formally defined using recursion as given by Equations (9) and (10). The function $\text{tail}(r)$ of an ordered set of n elements returns all but the first element of the string such that $r = \{r^{(1)}, \cdots, r^{(n)}\}$ and $\text{tail}(r) = \{r^{(2)}, \cdots, r^{(n)}\}$, where we denote element j of ranked set r_{μ_i} as $r_{\mu_i}^{(j)}$. Here, $\text{lev}(r_{\mu_i}, r_{\nu_i})$ will have a low value close to 0 if r_{μ_i} and r_{ν_i} are very similar; otherwise, it will have a high value.

$$\text{lev}(r_{\mu_i}, r_{\nu_i}) = \begin{cases} \text{length}(r_{\mu_i}) & \text{if length}(r_{\nu_i}) = 0 \\ \text{length}(r_{\nu_i}) & \text{if length}(r_{\mu_i}) = 0 \\ \text{lev}\big(\text{tail}(r_{\mu_i}), \text{tail}(r_{\nu_i})\big) & \text{if } r_{\mu_i}^{(1)} = r_{\nu_i}^{(1)} \\ 1 + f(r_{\mu_i}, r_{\nu_i}) & \text{otherwise} \end{cases} \tag{9}$$

$$f(r_{\mu_i}, r_{\nu_i}) = \min\big(\text{lev}(\text{tail}(r_{\mu_i}), r_{\nu_i}), \text{lev}(r_{\mu_i}, \text{tail}(r_{\nu_i})), \text{lev}(\text{tail}(r_{\mu_i}), \text{tail}(r_{\nu_i}))\big) \tag{10}$$

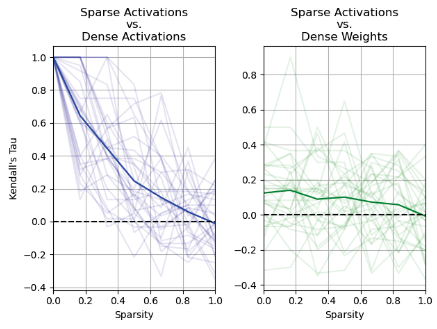
To perform our evaluation for a discriminative task, we train LeNet5 [33] models to classify MNIST [34] images. For a generative task, we train convolutional variational autoencoders (VAEs) to generate MNIST images[1]. For each case, we independently train 30 models using different random seeds. Our 30 LeNet5 models have an average test accuracy of 98.2% with a standard deviation of 0.003%, and our 30 VAEs have a mean Fréchet inception distance (FID) [35] of 9.81 with a standard deviation of 0.725. In Figure 3, we provide sampled images generated from a randomly selected VAE. These metrics and images are provided as supporting evidence that we use fully trained models in our statistical analysis, which forms the basis of our conclusions in this section.

(a) Original Images (b) Generated Images

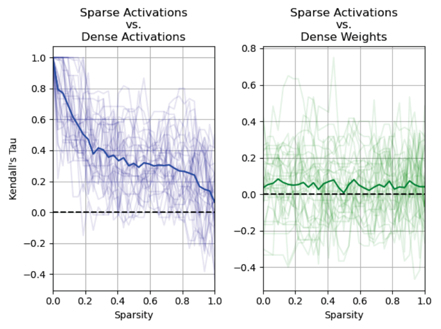
Figure 3. We provide random images generated from one of our VAEs trained on MNIST [34].

For LeNet5, we evaluate the importance of the 16 output channels of the second layer when iteratively pruning its six input channels. For the VAE, we evaluate the importance of the 16 output channels of the second layer of our decoder, which has 32 input channels. We increase the sparsity of the input channels using the rank ordering of the original unpruned network as determined by the mean ℓ_1-norm of its activations when estimated over the test dataset. We denote the importance estimate of the preceding layer as $\boldsymbol{\mu}_{i-1}$ and the respective rank ordering as $\boldsymbol{r}_{\boldsymbol{\mu}_{i-1}}$. Note that $\boldsymbol{\mu}_{i-1} = \{\mu_{i-1,1}, \cdots, \mu_{i-1,6}\}$ for LeNet5 and $\boldsymbol{\mu}_{i-1} = \{\mu_{i-1,1}, \cdots, \mu_{i-1,32}\}$ for our VAE. For each level of input sparsity, which we denote as s, we then rank order the output channels using the mean ℓ_1-norm over their respective activations also estimated over the test dataset. We denote the rank ordering of output channel activations by importance measure $\boldsymbol{\mu}_i$ for a given sparsity level (s) as $\boldsymbol{r}_{\boldsymbol{\mu}_i|s}$. Note that this sparsity level is over the input of our activation distribution, not the layer being evaluated, and the importance estimates of the evaluated layer i is $\boldsymbol{\mu}_i = \{\mu_{i,1}, \cdots, \mu_{i,16}\}$ for both LeNet5 and our VAE.

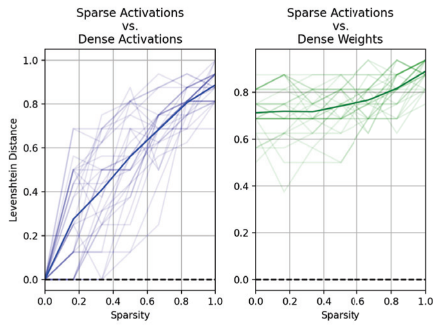
To evaluate the interdependence between successive layers, we iteratively increase the sparsity (s) of the input activations according to $\boldsymbol{\mu}_{i-1}$ for both our discriminative and generative models and measure the Kendall coefficient τ and Levenshtein distance between $\boldsymbol{r}_{\boldsymbol{\mu}_i|s}$ (i.e., the rank order of the output channels of layer i by importance measure $\boldsymbol{\mu}_i$ when input activations $\boldsymbol{h}_{i-1}$ have a sparsity of $s > 0$) and $\boldsymbol{r}_{\boldsymbol{\mu}_i}$ (i.e., the rank order of output channels of layer i by importance measure $\boldsymbol{\mu}_i$ when input activations have *no* sparsity). We visualize the results in Figure 4. For brevity in the titles of our plots, we use *sparse activations* to refer to the rank order of the output channels of layer i when using the mean ℓ_1-norm of the output channel activations as the importance measure $\boldsymbol{\mu}_i$ when input activations $\boldsymbol{h}_{i-1}$ have a sparsity of s (i.e., $\boldsymbol{r}_{\boldsymbol{\mu}_i|s}$); alternatively, we use *dense activations* when input activations $\boldsymbol{h}_{i-1}$ have no sparsity (i.e., $\boldsymbol{r}_{\boldsymbol{\mu}_i}$). We use *dense weights* to refer to the rank order of the output channels of layer i when using the ℓ_1-norm of the channel weights as the importance measure $\boldsymbol{\nu}_i$ (i.e., $\boldsymbol{r}_{\boldsymbol{\nu}_i}$). We observe that as the input activation sparsity s increases from 0 to 100%, the correlation between $\boldsymbol{r}_{\boldsymbol{\mu}_i|s}$ and $\boldsymbol{r}_{\boldsymbol{\mu}_i}$ gradually decreases and the Levenshtein distance gradually increases. We repeat this process using the ℓ_1-norm of our channel weights as our measure of importance, which we will denote as $\boldsymbol{\nu}_i$; however, the ℓ_1-norm of our channel weights is a data-free measure that is by definition invariant to this sparsity injection. Therefore, when iteratively increasing the sparsity of the input activations according to $\boldsymbol{\nu}_{i-1}$, we measure the relationship between our data-driven rankings $\boldsymbol{r}_{\boldsymbol{\mu}_i|s}$ and data-free ranking $\boldsymbol{r}_{\boldsymbol{\nu}_i}$. We observe that the correlation between $\boldsymbol{r}_{\boldsymbol{\mu}_i|s}$ and $\boldsymbol{r}_{\boldsymbol{\nu}_i}$ is much weaker, and their relationship is not as severely affected by the sparsity of the input activations.



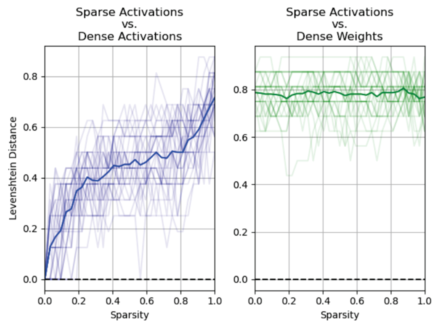
(a) LeNet5 Correlation Evaluation



(b) VAE Correlation Evaluation



(c) LeNet5 Distance Evaluation



(d) VAE Distance Evaluation

Figure 4. As discussed in Section 3, we evaluate the Kendall coefficient of rank correlation (top row) and the Levenshtein distance (bottom row) over 30 independently trained discriminative models (left column) and generative models (right column). We plot the correlation and distance between $\boldsymbol{r}_{\mu_i}$ and $\boldsymbol{r}_{\mu_i|s}$ in blue, and the correlation and distance between $\boldsymbol{r}_{\mu_i|s}$ and $\boldsymbol{r}_{\nu_i}$ in green.

From these experiments, we draw the following conclusions. First, data-driven channel importance rankings (e.g., the rankings of output channels by the ℓ_1-norm of activation distributions) are heavily impacted by the sparsity of the preceding input activation distribution[2]. Given this, we hypothesize that we can increase the effectiveness of data-driven pruning criterion by allowing a given layer to adjust to shifts in its input activation *before* applying pruning. Second, data-driven and data-free channel importance measurements are weakly correlated. Therefore, when used to heuristically guide channel pruning algorithms, we expect that these two importance measurements will behave differently under extreme levels of sparsity. It is important to note that we cannot conclude which ranking is necessarily "correct", as these experiments only demonstrate that they are "different". In Section 5, we empirically evaluate the performance of these neuron importance measurements using various pruning schedules to provide further insights.

4. Algorithms

For the purpose of describing our pruning and quantization algorithms, we first introduce our notation. We denote the input data to a neural network with L layers as $\boldsymbol{x}$ and, for discriminative tasks, its associated output label as $\boldsymbol{y}$. We denote the activations of the network as $\{\boldsymbol{h}_i\}_{i=1}^{L}$, where $\boldsymbol{h}_i$ denotes the activations of hidden layer i and $\boldsymbol{h}_0 = \boldsymbol{x}$ for convenience. The set of weights of each layer are denoted as $\{\boldsymbol{w}_i\}_{i=1}^{L}$, where $\boldsymbol{w}_i$ is the set of weights for layer i with C_i output channels, which we refer to as neurons. Finally, we denote the binary mask used to prune each layer as $\{\boldsymbol{m}_i\}_{i=1}^{L}$ and the importance estimates of each neuron as $\{\boldsymbol{\mu}_i\}_{i=1}^{L}$, where $m_{i,c} \in \{0,1\}$ and $\mu_{i,j} \in \mathcal{R}^+$ are, respectively, a scalar value for the binary mask and non-negative importance estimate for each neuron c in layer i, where $c \in \{1, \cdots, C_i\}$.

4.1. Greedy Layerwise Channel Pruning Using Nonparametric Statistics

When viewing the input data $\boldsymbol{x}$ as a random variable, neural networks are sometimes interpreted as Markov chains, where every hidden layer i defines the conditional probability $p(\boldsymbol{h}_i|\boldsymbol{h}_{i-1})$ [21,36]. Such a formulation motivates our observations that the effectiveness of neuron importance measurements relying on $\boldsymbol{h}_i$ is dependent on the stability of input activation distribution $\boldsymbol{h}_{i-1}$. In Section 3, we show that iteratively increasing the sparsity of the input activations of a given layer results in a monotonic decorrelation between the rank orderings of its output activations when using sparse versus dense input activations. Thus, we hypothesize that by allowing a given layer to adjust to these abrupt shifts in its input activation distribution $\boldsymbol{h}_{i-1}$ *before* pruning $\boldsymbol{h}_i$, we can increase the effectiveness of the heuristics used to rank order neurons by importance. As such, we design an iterative channel pruning algorithm that greedily traverses the topology of a feedforward neural network, starting from the first hidden layer $\boldsymbol{h}_1$ and ending at the final hidden layer $\boldsymbol{h}_L$.

Our layerwise iterative channel pruning algorithm is summarized in Algorithm 1.

Algorithm 1: Our proposed layerwise channel pruning algorithm, using per-channel ℓ_1-norm of activations to measure importance. All channel masks $\{\boldsymbol{m}_i\}_{i=1}^{L}$ are initialized to 1 and all importance measurements $\{\boldsymbol{\mu}_i\}_{i=1}^{L}$ are initialized to 0. We update learned weights of all layers in the network $\{\boldsymbol{w}_i\}_{i=1}^{L}$ at every step using backpropagation, but only update the mask $\boldsymbol{m}_i$ for layer i at step i.

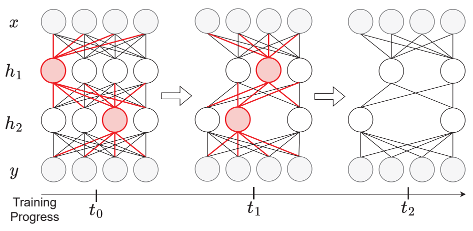
```
        $s$ := sparsity target
Input:  $t_p$ := number of evaluation steps for each layer
        $\Delta_p$ := mask update frequency
Output: $\{\boldsymbol{m}_i\}_{i=1}^{L}$

$\{\boldsymbol{m}_i\}_{i=1}^{L}$ ← initializeMasksToOne()
$\{\boldsymbol{\mu}_i\}_{i=1}^{L}$ ← initializeImportanceEstimatesToZero()
for $i \leftarrow 1$ to $L$ do
    for $t \leftarrow 1$ to $t_p$ do
        networkForwardPass()
        for $c \leftarrow 1$ to $C_i$ do
            $\mu_{i,c} \leftarrow (\mu_{i,c} \cdot (t-1) + \|\boldsymbol{h}_{i,c}\|_1)/t$
        end
        networkBackwardPass()
        every $\Delta_p$
            for $c \leftarrow 1$ to $C_i$ do
                if $m_{i,c} < quantile(\boldsymbol{\mu}_i, s)$ then
                    $m_{i,c} \leftarrow 0$
                end
            end
    end
end
```

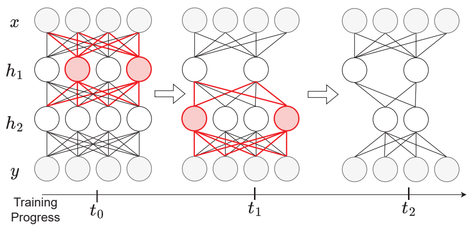
We first initialize the values of all masks $\{\boldsymbol{m}_i\}_{i=1}^{L}$ to 1 and the values of all importance measurements $\{\boldsymbol{m}_i\}_{i=1}^{L}$ to 0. Note that the mask for a given neuron $m_{i,c}$ is a binary value that prunes the neuron when $m_{i,c} = 0$, and $\mu_{i,c}$ is a non-negative value where $\mu_{i,1} > \mu_{i,2}$ is interpreted as neuron one being more important than neuron two in layer i. When pruning layer i, we estimate the importance of each neuron c using the moving average of the ℓ_1-norm of activations over channel c from $\boldsymbol{h}_i$ over t_p steps. We evaluate our pruning mask every Δ_p steps, where $\Delta_p \leq t_p$, at which point we prune neurons according to the $\boldsymbol{\mu}_i$ using a pre-defined sparsity target s_T. The entire network is pruned to the given target sparsity once all L layers are processed. As is standard practice, we do not prune visible activations (i.e., the input and output activations $\boldsymbol{x}$ and $\boldsymbol{y}$, respectively). In practice, we apply this procedure on a pre-trained network and continue to fine-tune our structured subnetwork for T more epochs after pruning. As we greedily increase the sparsity of the network layer-by-layer, we refer to this iterative pruning schedule as "layerwise" channel pruning.

Throughout our algorithm, we continue to update the weights of each layer $\{\boldsymbol{w}_i\}_{i=1}^{L}$ using gradient descent such that all weights are updated for $L \cdot t_p + T$ gradient steps.

The concept of using an iterative pruning schedule to alleviate the impact of abruptly removing neurons has been explored in prior work [11,23,37,38]; however, they iteratively introduce sparsity globally at each step. Because these pruning schedules gradually introduce sparsity according to a per-step heuristic, we refer to this class of algorithms as "stepwise" pruning. We visualize the differences in these algorithms in Figure 5. Unlike our layerwise pruning schedule, which determines $\boldsymbol{\mu}_i$ only after $\boldsymbol{h}_{i-1}$ has stabilized, stepwise pruning schedules determine $\boldsymbol{\mu}_i$ and $\boldsymbol{\mu}_{i-1}$ jointly. As discussed in Section 3, this could lead to less effective measures of neuron importance. In Section 5, we compare our layerwise pruning schedule against the standard stepwise approach.



(a) Stepwise Pruning Schedule



(b) Layerwise Pruning Schedule

Figure 5. We depict the differences between the standard stepwise pruning schedule (**a**) and our layerwise pruning schedule (**b**). While stepwise pruning algorithms iteratively increase sparsity globally in the network with each step, our layerwise pruning algorithm iteratively increases sparsity layer-by-layer with each step. Throughout the training progress (horizontal flow), gray nodes denote visible neurons that are not pruned, red nodes denote hidden neurons that have been identified to be pruned in a given step, and white nodes denote hidden neurons that remain active.

4.2. Uniform Quantization-Aware Training

To train our structured subnetworks for low-precision quantization, we use the straight-through estimator (STE) [39] to approximate the gradients of our rounding function such that $\nabla_x \lfloor x \rceil = 1$, but $\lfloor x \rceil \neq x$. To apply asymmetric quantization to our hidden activations $\{\boldsymbol{h}_i\}_{i=1}^{L}$, we adaptively fit our per-tensor scaling factor s_i and zero-point z_i. To apply symmetric quantization to our weights $\{\boldsymbol{w}_i\}_{i=1}^{L}$, we adaptively fit our per-channel scaling factor $\boldsymbol{s_i}$ where $s_{i,c}$ denotes the scaling factor for channel $c \in \{1, \cdots, C_i\}$. For each layer i, we estimate the upper and lower bounds of activation $\boldsymbol{h}_i$ and the maximum magnitude of weight $\boldsymbol{w}_i$ using moving average statistics, similar to our technique in Algorithm 1. Following the work of [28], we summarize the adaptive asymmetric and symmetric quantization algorithms used in this paper in Algorithms 2 and 3, respectively.

Algorithm 2: Our adaptive asymmetric quantization algorithm for our activations $\boldsymbol{h}_i$ using per-tensor scaling factors. We use moving average statistics over hidden activation $\boldsymbol{h}_i$ to estimate the bounds on its dynamic range for the purpose of deriving scaling factor s_i and zero-point z_i for layer i.

```
Input: $\left(l_i^{(t)}, u_i^{(t)}\right) :=$ estimated bounds on hidden activation $\boldsymbol{h}_i$ at time step $t$
Output: Quantized activation $\hat{\boldsymbol{h}}_i$; Updated bounds $\left(l_i^{(t+1)}, u_i^{(t+1)}\right)$
1 $l_i^{(t+1)} \leftarrow (l_t \cdot t + \min(\boldsymbol{h}_i))/(t+1)$
2 $u_i^{(t+1)} \leftarrow (u_t \cdot t + \max(\boldsymbol{h}_i))/(t+1)$
3 $s_i \leftarrow (u_i^{(t+1)} - l_i^{(t+1)})/2^b$
4 $z_i \leftarrow \lfloor -l_i^{(t+1)}/s \rceil$
5 $\hat{\boldsymbol{h}}_i \leftarrow \mathrm{clip}\left(\lfloor \boldsymbol{h}_i/s_i \rceil + z_i; 0, 2^b - 1\right)$
```

Algorithm 3: Our adaptive symmetric quantization algorithm for the set of weights $\boldsymbol{w}_i$ for layer i using per-channel scaling factors. We use moving average statistics to estimate the maximum weight magnitude for each channel c to derive our per-channel scaling factors s_c.

```
Input:  $s_{i,c}^{(t)}$ := estimated scaling factor $s$ for channel $c$ of weight $\boldsymbol{w}_i$ at time $t$
Output: Quantized weight $\hat{\boldsymbol{w}}_i$; updated scaling factor $s_{i,c}^{(t+1)}$
1 for $c \leftarrow 1$ to $C_i$ do
2     $s_{i,c}^{(t+1)} \leftarrow \left( s_{i,c}^{(t)} \cdot t + \frac{\max(\boldsymbol{w}_i)}{2^{b-1}} \right) / (t+1)$
3 end
4 $\hat{\boldsymbol{w}}_i \leftarrow \text{clip}(\lfloor \boldsymbol{w}_i / s_{t+1} \rceil; -2^{b-1}, 2^{b-1} - 1)$
```

5. Experiments

In Section 3, we present our hypothesis that, by allowing a given layer to adjust to abrupt shifts to its input distribution *before* pruning its channels, the heuristics used to rank order neurons by importance become more effective. Here, we compare our greedy layerwise channel pruning algorithm against alternative pruning schedules and importance measures to demonstrate the increased performance of our approach versus existing baselines. For the purpose of enabling inference acceleration on resource-constrained hardware such as mobile and edge devices, we combine the use of our channel pruning algorithm with moving average statistics-based uniform quantization, as described in Section 4. We evaluate the performance of our algorithms using the following discriminative and generative computer vision tasks:

1. Image classification with DenseNet121 [40], and MobileNetV2 [41] on CIFAR100 [42]
2. Semantic segmentation with UNet [43], and FRRNet [44] on Cityscape [45]
3. Image style transfer with CycleGAN [46] on Cityscape

We implement our pruning algorithms using the PyTorch deep learning framework [47] and create a custom neural network pruning module to compute neuron importance estimates and derive binary masks using local buffers[3]. Example usage of the functions in this repository is provided in Appendix A. For each task, our baselines are built using the existing open-source implementations provided by the respective original authors. To the extent possible, we use the same weight initialization techniques, network architecture, and hyperparameter configurations as reported in the original papers. All models are trained through Nautilus—a distributed research compute infrastructure from the Pacific Research Platform (PRP) [48]. Single-GPU training time for each model ranges from 3 hours (e.g., MobileNetV2 on CIFAR100) to 12 hours (e.g., CycleGAN on Cityscape) depending on the complexity of the task. Furthermore, for the style transfer task, CycleGAN uses a *cycle consistency loss* that includes two mappings: (1) generator G_θ, parameterized by θ, maps input image $\boldsymbol{x}$ to the target domain $\hat{\boldsymbol{x}}$; and (2) generator F_ϕ, parameterized by ϕ, maps $\hat{\boldsymbol{x}}$ back to the original image $\boldsymbol{x}$. During training, we only prune the forward mapping into the target domain $G_\theta(\boldsymbol{x})$, as we are only interested in accelerating inference, and the reverse transform is discarded at inference time. Given an input label map, the forward mapping G_θ of our pruned and quantized CycleGAN model generates realistic 128×128 images in the first-person driving view, as shown in Figure 6.

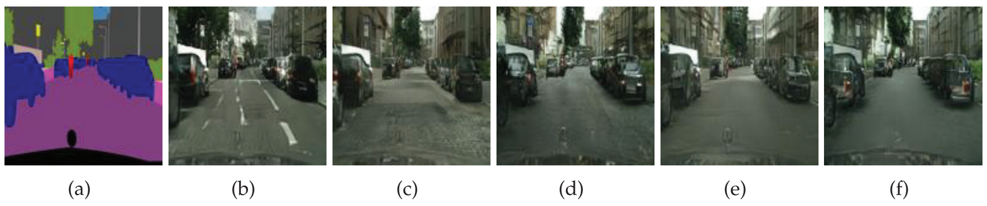


Figure 6. (**a**) Input, (**b**) Output (Baseline), (**c**) Output (P50%), (**d**) Output (P75%), (**e**) Output (P50%, W8A8), (**f**) Output (P75%, W8A8). We provide examples of images generated from CycleGAN given the same input (left). Here, P50% and P75% denote 50% and 75% channel pruning, respectively. We use "W8A8" to denote that the weights and activations have both been quantized to 8 bits.

5.1. Evaluating Pruning Schedules and Importance Measures

Here, we evaluate the importance of our layerwise iterative pruning schedule by comparing it to three alternative pruning schedules:

1. **Training from scratch.** Prior work has demonstrated that, in some cases, there is no need to implement a pruning schedule because pre-defined structured subnetwork architectures can be trained from scratch to match or surpass the performance of the original larger network [38]. As such, we evaluate our pruning algorithm against this baseline.
2. **One-shot.** We compare against the common "prune then fine-tune" strategy [20], where we train a fully connected baseline, and then prune the converged model to our target sparsity in one step before fine-tuning to heal the network.
3. **Stepwise.** Unlike one-shot pruning schedules, which jump to the target sparsity in one step, stepwise pruning schedules iteratively increase the sparsity in the network over many steps throughout training. We benchmark against the state-of-the-art iterative pruning schedule proposed by Zhu and Gupta [23].

In Table 1, we report the performance of each of these algorithms with target sparsity levels of $s_T = 50\%$ and $s_T = 75\%$ across our discriminative and generative tasks. Each entry is the averaged result over three independent runs. We use the same learning rate schedule and optimizer for each experiment to ensure an even comparison. We use top-1 accuracy to evaluate image classification models, mean intersection-over-union (mIOU) to evaluate semantic segmentation models, and Fréchet inception distance (FID) [35] to measure the difference between real images and images generated from our CycleGAN. Based on these metrics, we observe that our layerwise channel pruning algorithm performs stronger at more extreme levels of sparsity in a majority of cases.

Next, we compare the performance of data-driven and data-free neuron importance measures when used to guide our greedy layerwise channel pruning algorithm. In Table 2, we compare the performance of our algorithm when using the ℓ_1-norm of the channel activations against using the ℓ_1-norm of the channel weights. Although prior work had experimented with using the ℓ_0-norm as a data-driven importance measure [20], we do not compare against this as not all layers in our baselines are followed by a ReLU. We observe that using the ℓ_1-norm of the output activation channels yields superior results than using the ℓ_1-norm of channel weights in a majority of cases. Furthermore, we also observe that using the ℓ_1-norm of channel weights to guide our greedy layerwise pruning algorithm often yields better results than other pruning schedules reported in Table 1.

Table 1. Comparing pruning schedules across discriminative and generative tasks. With higher levels of sparsity, our greedy layerwise channel pruning algorithm performs better than existing baselines across both discriminative and generative tasks. Note that for top-1 accuracy and mIOU, higher is better, but for FID, lower is better.

		Classification (Accuracy)		**Segmentation (mIOU)**		**Style Transfer (FID)**
		DenseNet121	**MobileNetV2**	**UNet**	**FRRNet**	**CycleGAN**
Baseline	Full Model	78.70	68.31	61.69	66.32	47.17
50% Sparsity	From Scratch	76.75	65.36	58.90	61.84	50.39
	One-Shot	78.40	67.39	**60.59**	64.53	49.43
	Stepwise	77.14	**68.05**	58.13	64.90	52.83
	Layerwise (Ours)	**78.77**	67.58	60.57	**65.52**	**48.06**
75% Sparsity	From Scratch	73.07	56.66	**56.71**	59.29	59.55
	One-Shot	76.40	56.16	52.77	58.71	57.07
	Stepwise	72.55	60.40	51.05	56.59	65.67
	Layerwise (Ours)	**76.44**	**60.71**	56.48	**61.14**	**55.48**

Table 2. Comparing data-free vs. data-driven neuron importance measures across discriminative and generative tasks using our greedy layerwise channel pruning algorithm, we observe that the mean ℓ_1-norm of the output activations (i.e., "Layerwise (A)") performs better than the ℓ_1-norm of channel weights (i.e., "Layerwise (W)").

		Classification (Accuracy)		**Segmentation (mIOU)**		**Style Transfer (FID)**
		DenseNet121	**MobileNetV2**	**UNet**	**FRRNet**	**CycleGAN**
Baseline	Full Model	78.70	68.31	61.69	66.32	47.17
50% Sparsity	Layerwise (A)	**78.77**	**67.58**	**60.57**	65.52	**48.06**
	Layerwise (W)	77.88	67.56	59.41	**66.16**	52.52
75% Sparsity	Layerwise (A)	**76.44**	**60.71**	**56.48**	**61.14**	**55.48**
	Layerwise (W)	73.53	60.62	54.38	60.18	67.97

The results reported in Tables 1 and 2 suggest that we can increase the effectiveness of nonparametric data-driven pruning criteria by only pruning a given layer *after* its preceding layer has been fully pruned, which supports our motivating hypothesis. Our experiments in Section 3 show that the rank ordering of channels for a given layer i is strongly impacted by the sparsity of the input activations $\boldsymbol{h}_{i-1}$ to that layer. Equation-based pruning schedules such as [23] gradually introduce sparsity throughout the network with each step, as shown in Figure 5. Thus, at time t, they approximate the neuron importance with respect to the target sparsity s_T using the neuron importance with respect to the current sparsity s_t, where $s_t \leq s_T$ and $t \leq T$. As such, this class of algorithms uses rankings $\boldsymbol{r}_{\mu_i|s_t}$ to heuristically guide pruning. Furthermore, one-shot pruning schedules approximate the neuron importance with respect to the target sparsity s_T using estimates at step 0, when the input activations have no sparsity. As such, this class of algorithms uses rankings $\boldsymbol{r}_{\mu_i}$ to heuristically guide pruning. In contrast, our layerwise pruning schedule directly uses the neuron importance measure with respect to the target sparsity s_T, where we use $\boldsymbol{r}_{\mu_i|s_T}$ to heuristically guide pruning. As the sparsity target s_T increases to more extreme levels, there is a gradual decorrelation between $\boldsymbol{r}_{\mu_i}$ and $\boldsymbol{r}_{\mu_i|s_T}$, as shown in Section 3. Our results given in Table 1 show that directly using $\boldsymbol{r}_{\mu_i|s_T}$ within our greedy layerwise framework increases the effectiveness of data-driven rankings. We conjecture this is because our greedy layerwise channel pruning algorithm allows an unpruned layer to freely adjust to abrupt shifts in its input activation distribution caused by pruning the channels of its preceding layer.

5.2. Evaluation of Joint Pruning and Quantization

We evaluate our greedy layerwise channel pruning algorithm when jointly used with the quantization algorithms detailed in Section 4.2 to learn low-precision structured subnetworks through joint pruning and quantization. As discussed in Section 2, we apply symmetric, per-channel quantization to the weights of our deep neural networks and asymmetric, per-tensor quantization to the activations. To implement Algorithms 2 and 3, we create another custom quantization PyTorch module that accumulates the respective per-tensor or per-channel moving average statistics. As is standard practice, our modules produce a "fake quantized" output using the derived scale s and zero-point z [27,28]. We apply these modules to every hidden activation $\{\boldsymbol{h}_i\}_{i=1}^L$ and set of weights $\{\boldsymbol{w}_i\}_{i=1}^L$ in the network to prepare our models for integer-only inference [27]. We follow the standard "prune-then-quantize" training paradigm and activate the quantization operators only after the completion of layerwise pruning. Figure 1 shows both the training schedule and computational order of our joint pruning and quantization pipeline.

In Table 3, we report our joint pruning and quantization results for 50% and 75% sparsity when training 4-bit and 8-bit models. We use "W4A8" to denote quantizing weights to 4 bits and activations to 8 bits. For all networks except for CycleGAN, we quantize the weights of all layers $\{\boldsymbol{w}_i\}_{i=1}^L$ to the specified bitwidth, including the first and last layer and quantize the activations of all layers $\{\boldsymbol{h}_i\}_{i=1}^L$ to the specified bitwidth except for network input $\boldsymbol{x}$ and output $\boldsymbol{x}$. For CycleGAN, we always quantize the weights and input activation of the last layer to 8-bit regardless of global bitwidth configuration, as is standard practice [49]. For each experiment reported in Table 3, we use our greedy layerwise pruning algorithm with the ℓ_1-norm of the output channel activations as our importance measure.

Table 3. Joint channel pruning and uniform quantization to learn low-precision structured subnetworks using our algorithms depicted in Figure 1 and discussed in Section 4.

		Classification (Accuracy)		**Segmentation (mIOU)**		**Style Transfer (FID)**
		DenseNet121	**MobileNetV2**	**UNet**	**FRRNet**	**CycleGAN**
Baseline	Full Model	78.70	68.31	61.69	66.32	47.17
	W8A8	79.04	68.26	62.03	66.43	49.89
50% Sparsity	W8A8	79.01	67.58	60.44	64.81	58.12
	W4A8	78.41	64.67	60.15	63.42	64.25
	W4A4	75.99	59.25	56.12	59.04	92.71
75% Sparsity	W8A8	76.17	61.02	56.15	61.28	85.89
	W4A8	76.12	55.41	55.60	60.90	92.12
	W4A4	73.62	49.49	51.90	54.16	119.19

Finally, we apply our joint layerwise channel pruning and uniform quantization framework to train ResNet-32 [37] on the CIFAR10 dataset with sparsity ratios of 25%, 40% and 50%, and uniform bitwidths of 8 and 4. We compare against existing solutions and summarize the results in Table 4. Here, N_W and N_A denote the average number of bits used to quantize weights or activations, respectively, and s_W and s_A denote the global weight and activation sparsity, respectively. Note that, as discussed in Section 2.1, channel pruning reduces both the storage and operating memory[4] requirements of both the weights and activations, while unstructured weight pruning only reduces the memory storage requirements of the weights. Therefore, structured pruning can result in lower operating memory requirements, despite lower compression ratios with respect to unstructured pruning. To compare different approaches across these configurations, we compute the performance per operating memory size (i.e., **performance density**) for each model, as listed in the last column. For existing solutions, we summarize the results reported in prior work. For solutions that do not apply quantization to the weights or activations, we estimate their precision at 16 bits per element rather than 32 since neural networks can be quantized to 16-bit fixed-point through post-training quantization without significant accuracy degradation [50].

Table 4 shows that our method is able to achieve a higher performance density (8.92) than the highest of previous solutions (8.59) while having very competitive performance (92.53% versus 91.66% top-1 accuracy). When applying 50% pruning and 4-bit quantization, our method achieves the smallest memory footprint and highest performance density of 25.25, three times larger than the existing highest, while still maintaining sufficient accuracy (87.3%). Moreover, unlike methods such as [12,51], our framework provides direct control over the target bitwidth and sparsity, which is more useful in practical scenarios under tight design constraints, as is the case when deployed on edge GPUs or FPGAs [52–54].

Table 4. We demonstrate the superior performance per memory footprint (i.e., performance density) of our framework when compared to existing image classification solutions trained on CIFAR10. By jointly applying uniform quantization and unstructured pruning to both the weights and activations of the DNN, we achieve a higher performance density (PD) with higher compression rates than existing solutions and comparable network performance.

Method	Network	N_W	N_A	s_W	s_A	Baseline Acc	Accuracy	Weights (Mb)	Activations (Mb)	PD (Acc/Mb)
[55]	ResNet-32	8	–	77.8%	–	92.58	92.64 (+0.06)	0.83	20.19	4.41
[56]	DenseNet-76	2	–	54%	–	92.19	91.17 (−1.02)	0.68	141.26	0.64
[38]	VGG-19	–	–	95%	–	93.50	93.34 (−0.16)	16.03	19.40	2.63
	PreResNet-110	–	–	95%	–	95.04	92.35 (−2.69)	1.38	67.50	1.34
	DenseNet-100	–	–	95%	–	95.24	94.19 (−1.05)	0.97	213.37	0.44
	VGG-19	–	–	70%	70%	93.5	93.60 (+0.1)	70.70	5.31	1.23
	PreResNet-164	–	–	60%	60%	95.04	94.23 (−0.81)	16.67	40.21	1.66
	DenseNet-40	–	–	60%	60%	94.10	93.87 (−0.23)	1.60	22.97	3.82
[57]	DenseNet-40	–	–	60%	60%	94.11	93.16 (−0.95)	1.60	22.97	3.79
	ResNet-20	–	–	38%	38%	92.01	91.66 (−0.35)	2.70	7.96	8.59
	ResNet-56	–	–	45%	45%	93.04	92.26 (−0.78)	7.53	19.18	3.45
	ResNet-110	–	–	63%	63%	93.21	92.96 (−0.25)	10.25	25.12	2.63
[58]	VGG-16	–	–	78.8%	78.8%	93.40	91.50 (−1.90)	49.96	3.75	1.70
[59]	VGG16-C	–	–	95%	–	93.51	93.00 (−0.51)	11.78	17.70	3.15
	WRN-22-8	–	–	95%	–	95.74	95.07 (−0.67)	13.73	115.34	0.74
[51]	ResNet-20	1.9	–	54%	–	91.29	91.15 (−0.14)	0.24	12.85	6.97
[12]	VGG-7	4.8	5.4	–	–	93.05	93.23 (+0.18)	43.85	3.27	1.98
[60]	MobileNet	8	8	–	–	91.31	90.59 (−0.72)	25.74	13.17	2.33
[61]	ResNet-32	8	–	87.5%	–	92.58	92.57 (−0.01)	0.47	20.19	4.48
Ours	ResNet-32	8	8	25%	25%	92.58	92.53 –	2.80	7.57	**8.92**
		8	8	40%	40%		91.77 (−0.81)	2.24	6.06	**11.06**
		8	8	50%	50%		90.16 (−2.42)	1.87	5.05	**13.04**
		4	4	50%	50%		87.30 (−5.28)	0.93	2.52	**25.25**

6. Conclusions and Future Work

In this work, we demonstrate that, when using data-driven measures of neuron importance, the rank orderings of the output channels of a given layer strongly depend on the sparsity level of the preceding layer. In Section 3, we show that as we increase the sparsity of the input activations for a given layer, the correlation trends toward zero when comparing: (1) the rank order of its output channels using input activations with no sparsity; versus (2) the rank order of its output channels when the input activations have sparsity $s > 0$. Informed by this observation, we propose a data-driven nonparametric, magnitude-based channel pruning algorithm that works in a greedy manner based on the activations of the previous sparsified layer, as detailed in Section 4. For the purpose of learning low-precision structured subnetworks, we demonstrate that our proposed algorithm works effectively in combination with statistics-based uniform quantization. As demonstrated in Section 5, our joint layerwise channel pruning and uniform quantization framework yields increased performance per memory footprint over existing solutions across a range of discriminative and generative networks. The resulting neural networks can be efficiently accelerated by resource-constrained hardware platforms such as edge GPUs and FPGAs. In future work, we aim to focus more closely on applying our framework to generative models and explore advanced quantization-aware training techniques to work jointly with our nonparametric greedy layerwise channel pruning algorithm.

Author Contributions: Software and experiments—X.Z.; writing, formal analysis, and visualization—X.Z., I.C. and S.D. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Publicly available datasets were analyzed in this study. These data can be found in [42,45]. Pre-trained models and configuration files are provided at: https://github.com/mlzxy/mdpi2022 (accessed on 30 June 2022).

Acknowledgments: This work was supported in part by NSF awards CNS1730158, ACI-1540112, ACI-1541349, OAC-1826967, the University of California Office of the President, and the California Institute for Telecommunications and Information Technology's Qualcomm Institute (Calit2-QI).

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

Appendix A. Software Library

The code for each algorithm introduced in this paper can be found in our Python library, `qsparse`, at https://github.com/mlzxy/qsparse (accessed on 30 June 2022). In Listing A1, we provide an example of how to use our algorithms for quantization and pruning as PyTorch modules [47].

Listing A1. Examples of our software interface for quantization and pruning on activations.

```python
import torch.nn as nn
from qsparse import prune, quantize

# activation pruning and quantization
nn.Sequential(
    nn.Conv2d(10, 30, 3),
    prune(sparsity=0.5),  # pruning module
    quantize(bits=8)      # quantization module
)
```

The majority of existing model compression Python libraries provide either quantization [62–64] or pruning [18] support. In contrast, the simplicity of our framework enables an efficient software interface that supports a wide range of neural network architectures for both pruning and quantization. To improve the library flexibility, we introduce a technique that directly transforms the weight attribute of the input layer into a pruned or quantized version at runtime, as shown in Listing A2. Thus, our library is layer-agnostic and can work with any PyTorch module as long as their parameters can be accessed from their weight attribute, as is standard practice [47].

Listing A2. An illustration of the technique we use to create weight-quantized modules without modifying the internal implementation of specific modules. By injecting the `weight` property, we are able to automatically apply quantization or pruning at each access of `weight`. By chaining transformations for both `prune` and `quantize`, we create modules with weights that are jointly pruned and quantized.

```
def create_general_quantized_module(mod: nn.Module):
    OriginClass = mod.__class__

    class Wrapper(OriginClass):
        @property
        def weight(self):
            return quantize(getattr(OriginClass, "weight").__get__(self))

    Wrapper.__name__ = OriginClass.__name__
    mod.__class__ = Wrapper
    return mod
```

Based on our custom modules and transformation technique, we provide the network conversion function, as shown in Listing A3, in which we traverse the entire network and automatically modify the computational graph to the aim of pruning and quantization. For each network in our experiments, we apply different conversions on top of the baseline network to the requirements of evaluation.

Listing A3. An example usage of our software interface for converting the computational graph of a pre-defined full-precision network to a quantized one by injecting the `quantize` module after each `ReLU` and transforming each `Conv2d` to its weight-quantized version without the need to modify the implementation of the original network.

```
import torch.nn as nn
from qsparse import quantize, convert

quantized_net = convert(
    pre_defined_net,
    quantize(bits=8),
    activation_layers=[nn.ReLU, ],
    weight_layers=[nn.Conv2d]
)
```

Notes

[1] We use a network architecture comparable to LeNet for the purpose of a reasonably symmetric evaluation across discriminative and generative tasks. Our encoder uses two convolution layers, each followed by a ReLU and max pooling layer. Our decoder uses three deconvolution layers, each followed by a ReLU except for the final deconvolution layer, which is followed by a sigmoid.

[2] We repeated these experiments using the ℓ_0-norm of the output activations and saw very similar results.

[3] The code for the algorithms discussed in this paper can be found at https://github.com/mlzxy/mdpi202 (accessed on 30 June 2022).

[4] We define operating memory as the aggregate hardware storage area used for weights and activations of the network during inference, all of which are required to be kept so they can be readily accessed.

References

1. Hestness, J.; Narang, S.; Ardalani, N.; Diamos, G.; Jun, H.; Kianinejad, H.; Patwary, M.; Ali, M.; Yang, Y.; Zhou, Y. Deep learning scaling is predictable, empirically. *arXiv* **2017**, arXiv:1712.00409.
2. Han, S.; Mao, H.; Dally, W.J. Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding. In the Proceedings of the International Conference on Learning Representations, San Juan, Puerto Rico, 2–4 May 2016.
3. Gale, T.; Elsen, E.; Hooker, S. The state of sparsity in deep neural networks. *arXiv* **2019**, arXiv:1902.09574.

4. Polino, A.; Pascanu, R.; Alistarh, D. Model compression via distillation and quantization. In Proceedings of the International Conference on Learning Representations, Vancouver, BC, Canada, 30 April–3 May 2018.
5. Dong, X.; Yang, Y. Network pruning via transformable architecture search. *arXiv* **2019**, arXiv:1905.09717.
6. Wang, T.; Wang, K.; Cai, H.; Lin, J.; Liu, Z.; Wang, H.; Lin, Y.; Han, S. Apq: Joint search for network architecture, pruning and quantization policy. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Seattle, WA, USA, 14–19 June 2020; pp. 2078–2087.
7. Paupamah, K.; James, S.; Klein, R. Quantisation and pruning for neural network compression and regularisation. In Proceedings of the 2020 International SAUPEC/RobMech/PRASA Conference, Cape Town, South Africa, 29–31 January 2020; pp. 1–6.
8. Liang, T.; Glossner, J.; Wang, L.; Shi, S.; Zhang, X. Pruning and quantization for deep neural network acceleration: A survey. *Neurocomputing* **2021**, *461*, 370–403. [CrossRef]
9. Zhao, Y.; Gao, X.; Bates, D.; Mullins, R.; Xu, C.Z. Focused quantization for sparse CNNs. *arXiv* **2019**, arXiv:1905.09717.
10. Yu, P.H.; Wu, S.S.; Klopp, J.P.; Chen, L.G.; Chien, S.Y. Joint Pruning & Quantization for Extremely Sparse Neural Networks. *arXiv* **2020**, arXiv:2010.01892.
11. Colbert, I.; Kreutz-Delgado, K.; Das, S. AX-DBN: An approximate computing framework for the design of low-power discriminative deep belief networks. In Proceedings of the 2019 International Joint Conference on Neural Networks (IJCNN), Budapest, Hungary, 14–19 July 2019; pp. 1–9.
12. Van Baalen, M.; Louizos, C.; Nagel, M.; Amjad, R.A.; Wang, Y.; Blankevoort, T.; Welling, M. Bayesian bits: Unifying quantization and pruning. *Adv. Neural Inf. Process. Syst.* **2020**, *33*, 5741–5752.
13. Frankle, J.; Carbin, M. The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks. In Proceedings of the International Conference on Learning Representations, New Orleans, LA, USA, 6–9 May 2019.
14. Mao, H.; Han, S.; Pool, J.; Li, W.; Liu, X.; Wang, Y.; Dally, W.J. Exploring the Granularity of Sparsity in Convolutional Neural Networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops, Honolulu, HI, USA, 21–26 July 2017.
15. He, Y.; Zhang, X.; Sun, J. Channel pruning for accelerating very deep neural networks. In Proceedings of the IEEE International Conference on Computer Vision, Honolulu, HI, USA, 21–26 July 2017; pp. 1389–1397.
16. Jha, N.K.; Mittal, S.; Avancha, S. Data-type aware arithmetic intensity for deep neural networks. *Energy* **2021**, *120*, x109.
17. Colbert, I.; Kreutz-Delgado, K.; Das, S. An Energy-Efficient Edge Computing Paradigm for Convolution-Based Image Upsampling. *IEEE Access* **2021**, *9*, 147967–147984. [CrossRef]
18. Blalock, D.; Gonzalez Ortiz, J.J.; Frankle, J.; Guttag, J. What is the state of neural network pruning? *Proc. Mach. Learn. Syst.* **2020**, *2*, 129–146.
19. Chen, T.; Chen, X.; Ma, X.; Wang, Y.; Wang, Z. Coarsening the Granularity: Towards Structurally Sparse Lottery Tickets. In Proceedings of the International Conference on Machine Learning, Baltimore, MA, USA, 17–23 July 2022.
20. Hu, H.; Peng, R.; Tai, Y.W.; Tang, C.K. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *arXiv* **2016**, arXiv:1607.03250
21. Dai, B.; Zhu, C.; Guo, B.; Wipf, D. Compressing neural networks using the variational information bottleneck. In Proceedings of the International Conference on Machine Learning. PMLR, 2018, Stockholm, Sweden, 10–15 July 2018; pp. 1135–1144.
22. Molchanov, P.; Mallya, A.; Tyree, S.; Frosio, I.; Kautz, J. Importance estimation for neural network pruning. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Long Beach, CA, USA, 15–20 June 2019; pp. 11264–11272.
23. Zhu, M.; Gupta, S. To Prune, or Not to Prune: Exploring the Efficacy of Pruning for Model Compression. In Proceedings of the International Conference on Learning Representations, Vancouver, BC, Canada, 30 April–3 May 2018.
24. Luo, J.H.; Wu, J.; Lin, W. Thinet: A filter level pruning method for deep neural network compression. In Proceedings of the IEEE International Conference on Computer Vision, Honolulu, HI, USA, 21–26 July 2017; pp. 5058–5066.
25. Wang, Y.; Lu, Y.; Blankevoort, T. Differentiable joint pruning and quantization for hardware efficiency. In Proceedings of the European Conference on Computer Vision, Glasgow, UK, 23–28 August 2020; Springer: Cham, Switzerland, 2020; pp. 259–277.
26. Wu, H.; Judd, P.; Zhang, X.; Isaev, M.; Micikevicius, P. Integer Quantization for Deep Learning Inference: Principles and Empirical Evaluation. *arXiv* **2020**, arXiv:2004.09602.
27. Jacob, B.; Kligys, S.; Chen, B.; Zhu, M.; Tang, M.; Howard, A.; Adam, H.; Kalenichenko, D. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–23 June 2018; pp. 2704–2713.
28. Krishnamoorthi, R. Quantizing deep convolutional networks for efficient inference: A whitepaper. *arXiv* **2018**, arXiv:1806.08342.
29. Gholami, A.; Kim, S.; Dong, Z.; Yao, Z.; Mahoney, M.W.; Keutzer, K. A survey of quantization methods for efficient neural network inference. *arXiv* **2021**, arXiv:1806.08342.
30. Jain, S.; Gural, A.; Wu, M.; Dick, C. Trained quantization thresholds for accurate and efficient fixed-point inference of deep neural networks. *Proc. Mach. Learn. Syst.* **2020**, *2*, 112–128.
31. Knight, W.R. A computer method for calculating Kendall's tau with ungrouped data. *J. Am. Stat. Assoc.* **1966**, *61*, 436–439. [CrossRef]
32. Levenshtein, V.I. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet Physics—Doklady*; 1966; Volume 10, pp. 707–710. Available online: https://nymity.ch/sybilhunting/pdf/Levenshtein1966a.pdf (accessed on 30 June 2022).

33. LeCun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. Gradient-based learning applied to document recognition. *Proc. IEEE* **1998**, *86*, 2278–2324. [CrossRef]
34. LeCun, Y. The MNIST Database of Handwritten Digits. 1998. Available online: http://yann.lecun.com/exdb/mnist/ (accessed on 30 June 2022).
35. Heusel, M.; Ramsauer, H.; Unterthiner, T.; Nessler, B.; Hochreiter, S. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Adv. Neural Inf. Process. Syst.* **2017**, *30.*
36. Tishby, N.; Zaslavsky, N. Deep learning and the information bottleneck principle. In Proceedings of the 2015 IEEE information theory workshop (itw), Jerusalem, Israel, 26 April–1 May 2015; pp. 1–5.
37. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–20 June 2016; pp. 770–778.
38. Liu, Z.; Sun, M.; Zhou, T.; Huang, G.; Darrell, T. Rethinking the Value of Network Pruning. In Proceedings of the International Conference on Learning Representations, New Orleans, LA, USA, 6–9 May 2019.
39. Bengio, Y.; Léonard, N.; Courville, A. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv* **2013**, arXiv:1308.3432.
40. Huang, G.; Liu, Z.; Van Der Maaten, L.; Weinberger, K.Q. Densely connected convolutional networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops, Honolulu, HI, USA, 21–26 July 2017; pp. 4700–4708.
41. Sandler, M.; Howard, A.; Zhu, M.; Zhmoginov, A.; Chen, L.C. Mobilenetv2: Inverted residuals and linear bottlenecks. In Proceedings of the International Conference on Machine Learning, PMLR, 2018, Stockholm, Sweden, 10–15 July 2018; pp. 4510–4520.
42. Krizhevsky, A.; Hinton, G. Learning Multiple Layers of Features from Tiny Images. 2009. Available online: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf (accessed on 30 June 2022).
43. Ronneberger, O.; Fischer, P.; Brox, T. U-net: Convolutional networks for biomedical image segmentation. In Proceedings of the International Conference on Medical image computing and computer-assisted intervention, Munich, Germany, 5–9 October 2015; Springer: Berlin/Heidelberg, Germany, 2015; pp. 234–241.
44. Pohlen, T.; Hermans, A.; Mathias, M.; Leibe, B. Full-resolution residual networks for semantic segmentation in street scenes. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops, Honolulu, HI, USA, 21—26 July 2017; pp. 4151–4160.
45. Cordts, M.; Omran, M.; Ramos, S.; Rehfeld, T.; Enzweiler, M.; Benenson, R.; Franke, U.; Roth, S.; Schiele, B. The Cityscapes Dataset for Semantic Urban Scene Understanding. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 27–30 June 2016.
46. Zhu, J.Y.; Park, T.; Isola, P.; Efros, A.A. Unpaired image-to-image translation using cycle-consistent adversarial networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops, Honolulu, HI, USA, 21–26 July 2017; pp. 2223–2232.
47. 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.
48. Nautilus. 2022. Available online: https://ucsd-prp.gitlab.io/ (accessed on 30 June 2022).
49. Thomas, M.M.; Vaidyanathan, K.; Liktor, G.; Forbes, A.G. A reduced-precision network for image reconstruction. *ACM Trans. Graph. Tog* **2020**, *39*, 1–12. [CrossRef]
50. Rezk, N.M.; Nordström, T.; Ul-Abdin, Z. Shrink and Eliminate: A Study of Post-Training Quantization and Repeated Operations Elimination in RNN Models. *Information* **2022**, *13*, 176. [CrossRef]
51. Yang, H.; Gui, S.; Zhu, Y.; Liu, J. Automatic neural network compression by sparsity-quantization joint learning: A constrained optimization-based approach. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Seattle, WA, USA, 13–19 June 2020; pp. 2178–2188.
52. Zhang, X. *A Design Methodology for Efficient Implementation of Deconvolutional Neural Networks on an FPGA*; University of California: San Diego, CA, USA, 2017.
53. Biookaghazadeh, S.; Zhao, M.; Ren, F. Are {FPGAs} Suitable for Edge Computing? In Proceedings of the USENIX Workshop on Hot Topics in Edge Computing (HotEdge 18): Boston, MA, USA , 10 July 2018
54. Colbert, I.; Daly, J.; Kreutz-Delgado, K.; Das, S. A competitive edge: Can FPGAs beat GPUs at DCNN inference acceleration in resource-limited edge computing applications? *arXiv* **2021**, arXiv:2102.00294.
55. Choi, Y.; El-Khamy, M.; Lee, J. Towards the Limit of Network Quantization. In Proceedings of the International Conference on Learning Representations, Toulon, France, 24–26 April 2017.
56. Achterhold, J.; Koehler, J.M.; Schmeink, A.; Genewein, T. Variational network quantization. In Proceedings of the International Conference on Learning Representations, Vancouver, BC, Canada, 30 April–3 May 2018.
57. Zhao, C.; Ni, B.; Zhang, J.; Zhao, Q.; Zhang, W.; Tian, Q. Variational convolutional neural network pruning. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Long Beach, CA, USA, 15–20 June 2019; pp. 2780–2789.
58. Xiao, X.; Wang, Z. Autoprune: Automatic network pruning by regularizing auxiliary parameters. *Adv. Neural Inf. Process. Syst. (NeurIPS 2019)* **2019**, *32.*
59. Dettmers, T.; Zettlemoyer, L. Sparse networks from scratch: Faster training without losing performance. *arXiv* **2019**, arXiv:1907.04840.

60. Paupamah, K.; James, S.; Klein, R. Quantisation and pruning for neural network compression and regularisation. In Proceedings of the 2020 International SAUPEC/RobMech/PRASA Conference, Cape Town, South Africa, 29–31 January 2020; pp. 1–6.
61. Choi, Y.; El-Khamy, M.; Lee, J. Universal deep neural network compression. *IEEE J. Sel. Top. Signal Process.* **2020**, *14*, 715–726. [CrossRef]
62. Pappalardo, A. Xilinx/Brevitas. Available online: https://zenodo.org/record/5779154#.YujQyepBxPY (accessed on 30 June 2022).
63. torch.nn.qat—PyTorch 1.9.0 Documentation. 2021. Available online: https://pytorch.org/docs/stable/torch.nn.qat.html (accessed on 30 June 2022).
64. Coelho C.N., Jr.; Kuusela, A.; Zhuang, H.; Aarrestad, T.; Loncar, V.; Ngadiuba, J.; Pierini, M.; Summers, S. Ultra low-latency, low-area inference accelerators using heterogeneous deep quantization with QKeras and hls4ml. *arXiv* **2020**, arXiv:2006.10159