ECQ^{x}: Explainability-Driven Quantization for Low-Bit and Sparse DNNs

Daniel Becking[1] Maximilian Dreyer[1] Wojciech Samek[1,2,†]
Karsten Müller[1,†] Sebastian Lapuschkin[1,†]

[1] Department of Artificial Intelligence, Fraunhofer Heinrich Hertz Institute, Berlin, Germany
[2] BIFOLD – Berlin Institute for the Foundations of Learning and Data, Berlin, Germany
[†] {wojciech.samek,karsten.mueller,sebastian.lapuschkin}@hhi.fraunhofer.de

Abstract

The remarkable success of deep neural networks (DNNs) in various applications is accompanied by a significant increase in network parameters and arithmetic operations. Such increases in memory and computational demands make deep learning prohibitive for resource-constrained hardware platforms such as mobile devices. Recent efforts aim to reduce these overheads, while preserving model performance as much as possible, and include parameter reduction techniques, parameter quantization, and lossless compression techniques.

In this chapter, we develop and describe a novel quantization paradigm for DNNs: Our method leverages concepts of explainable AI (XAI) and concepts of information theory: Instead of assigning weight values based on their distances to the quantization clusters, the assignment function additionally considers weight relevances obtained from Layer-wise Relevance Propagation (LRP) and the information content of the clusters (entropy optimization). The ultimate goal is to preserve the most relevant weights in quantization clusters of highest information content.

Experimental results show that this novel Entropy-Constrained and XAI-adjusted Quantization (ECQ^{x}) method generates ultra low-precision (2-5 bit) and simultaneously sparse neural networks while maintaining or even improving model performance. Due to reduced parameter precision and high number of zero-elements, the rendered networks are highly compressible in terms of file size, up to 103× compared to the full-precision unquantized DNN model. Our approach was evaluated on different types of models and datasets (including Google Speech Commands and CIFAR-10) and compared with previous work.

1 Introduction

Solving increasingly complex real-world problems, continuously contributes to the success of deep neural networks (DNNs)[35, 36]. DNNs have long been established in numerous machine learning tasks and for this have been significantly improved in the past decade. This is often achieved by over-parameterizing models, i.e., their performance is attributed to their growing topology, adding more layers and parameters per layer [39, 17]. Processing a very large number of parameters comes at the expense of memory and computational efficiency. The sheer size of state-of-the-art models makes it difficult to execute them on resource-constrained hardware platforms. In addition, an increasing number of parameters implies higher energy consumption and increasing run times.

Such immense storage and energy requirements however contradict the demand for efficient deep learning applications for an increasing number of hardware-constrained devices, e.g., mobile phones, wearable devices, Internet of Things, autonomous vehicles or robots. Specific restrictions of such devices include limited energy, memory, and computational budget. Beyond these, typical applications on such devices, e.g., healthcare monitoring, speech recognition, or autonomous

driving, require low latency and/or data privacy. These latter requirements are addressed by executing and running the aforementioned applications directly on the respective devices (also known as "edge computing") instead of transferring data to third-party cloud providers prior to processing.

In order to tailor deep learning to resource-constrained hardware, a large research community has emerged in recent years [10, 43]. By now, there exists a vast amount of tools to reduce the number of operations and model size, as well as tools to reduce the precision of operands and operations (bit width reduction, going from floating point to fixed point). Topics range from neural architecture search (NAS), knowledge distillation, pruning/sparsification, quantization, lossless compression and hardware design.

Beyond all, quantization and sparsification are very promising and show great improvements in terms of neural network efficiency optimization [20, 41]. Sparsification sets less important neurons or weights to zero and quantization reduces parameter's bit widths from default 32 bit float to, e.g., 4 bit integer. These two techniques enable higher computational throughput, memory reduction and skipping of arithmetic operations for zero-valued elements, just to name a few benefits. However, combining both high sparsity and low precision is challenging, especially when relying only on the weight magnitudes as a criterion for the assignment of weights to quantization clusters.

In this work, we propose a novel neural network quantization scheme to render low-bit *and* sparse DNNs. More precisely, our contributions can be summarized as follows:

1. Extending the state-of-the-art concept of entropy-constrained quantization (ECQ) to utilize concepts of XAI in the clustering assignment function.

2. Use relevances observed from Layer-wise Relevance Propagation (LRP) at the granularity of per-weight decisions to correct the magnitude-based weight assignment.

3. Obtaining state-of-the-art or better results in terms of the trade-off between efficiency and performance compared to the previous work.

The chapter is organized as follows: First, an overview of related work is given. Second, in Section 3, basic concepts of neural network quantization are explained, followed by entropy-constrained quantization. Section 4 describes the ECQ extension towards ECQ$^{\text{x}}$ as an explainability-driven approach. Here, LRP is introduced and the per-weight relevance derivation for the assignment function presented. Next, the ECQ$^{\text{x}}$ algorithm is described in detail. Section 5 presents the experimental setup and obtained results, followed by the final conclusion in Section 6.

2 Related Work

A large body of literature exists that has focused on improving DNN model efficiency. Quantization is an approach that has shown great success [13]. While most research focuses on reducing the bit width for inference, [50] and others focus on quantizing weights, gradients and activations to also accelerate backward pass and training. Quantized models often require fine-tuning or re-training to adjust model parameters and compensate for quantization-induced accuracy degradation. This is especially true for precisions < 8 bit (cf. Figure 1 in Section 3). Trained quantization is often referred to as "quantization-aware training", for which additional trainable parameters may be introduced (e.g., scaling parameters [6] or directly trained quantization levels (centroids) [51]). A precision reduction to even 1 bit was introduced by BinaryConnect [8]. However, this kind of quantization usually results in severe accuracy drops. As an extension, ternary networks allow weights to be zero, i.e., constraining them to 0 in addition to w_- and w_+, which yields results that outperform the binary counterparts [26]. In DNN quantization, most clustering approaches are based on distance measurements between the unquantized

weight distribution and the corresponding centroids. The works in [7] and [30] were pioneering in using Hessian-weighted and entropy-constrained clustering techniques. To the best of our knowledge, [32] were the first and only ones to use concepts of XAI for DNN quantization. They use DeepLIFT importance measures which are restricted to the granularity of convolutional channels, whereas our proposed ECQ$^{\text{x}}$ computes LRP relevances per weight.

Another method for reducing the memory footprint and computational cost of DNNs is sparsification. In the scope of sparsification techniques, weights with small saliency (i.e., weights which minimally affect the model's loss function) are set to zero, resulting in a sparser computational graph and higher compressible matrices. Thus, it can be interpreted as a special form of quantization, having only one quantization cluster with centroid value 0 to which part of the parameter elements are assigned to. This sparsification can be carried out as unstructured sparsification [16], where any weight in the matrix with small saliency is set to zero, independently of its position. Alternatively, a structured sparsification is applied, where an entire regular subset of parameters is set to zero, e.g., entire convolutional filters, matrix rows or columns [18]. "Pruning" is conceptually related to sparsification but actually removes the respective weights rather than setting them to zero. This has the effect of changing the number of input and output shapes of layers and weight matrices[1]. Most pruning/ sparsification approaches are magnitude-based, i.e., weight saliency is approximated by the weight values, which is straightforward. However, since the early 1990s methods that use, e.g., second-order Taylor information for weight saliency [25] have been used alongside other criteria ranging from random pruning to correlation and similarity measures (for the interested reader we recommend [20]). In [49], LRP relevances were first used for structured pruning.

Generating efficient neural network representations can also be a result of combining multiple techniques. In Deep Compression [15], a three-stage model compression pipeline is described. First, redundant connections are pruned iteratively. Next, the remaining weights are quantized. Finally, entropy coding is applied to further compress the weight matrices in a lossless manner. This three stage model is also used in the new international ISO/IEC standard on Neural Network compression and Representation (NNR) [22], where efficient data reduction, quantization and entropy coding methods are combined. For coding, the highly efficient universal entropy coder DeepCABAC [45] is used, which yields compression gains of up to 63×. Although the proposed method achieves high compression gains, the compressed representation of the DNN weights require decoding prior to performing inference. In contrast, compressed matrix formats like Compressed Sparse Row (CSR) derive a representation that enables inference directly in the compressed format [47].

Orthogonal to the previously described approaches is the research area of Neural Architecture Search (NAS)[12]. Both manual [34] and automated [42] search strategies have played an important role in optimizing DNN architectures in terms of latency, memory footprint, energy consumption, etc. Microstructural changes include, e.g., the replacement of standard convolutional layers by more efficient types like depth-wise or point-wise convolutions, layer decomposition or factorization, or kernel size reduction. The macro architecture specifies the type of modules (e.g., inverted residual), their number and connections.

Knowledge distillation (KD) [19] is another active branch of research that aims at generating efficient DNNs. The KD paradigm leverages a large teacher model that is used to train a smaller (more efficient) student model. Instead of using the "hard" class labels to train the student, the key idea of model distillation is to deploy the teacher's class probabilities, as they can contain more information about the input.

[1] In practice, pruning is often simulated by masking, instead of actually restructuring the model's architecture.

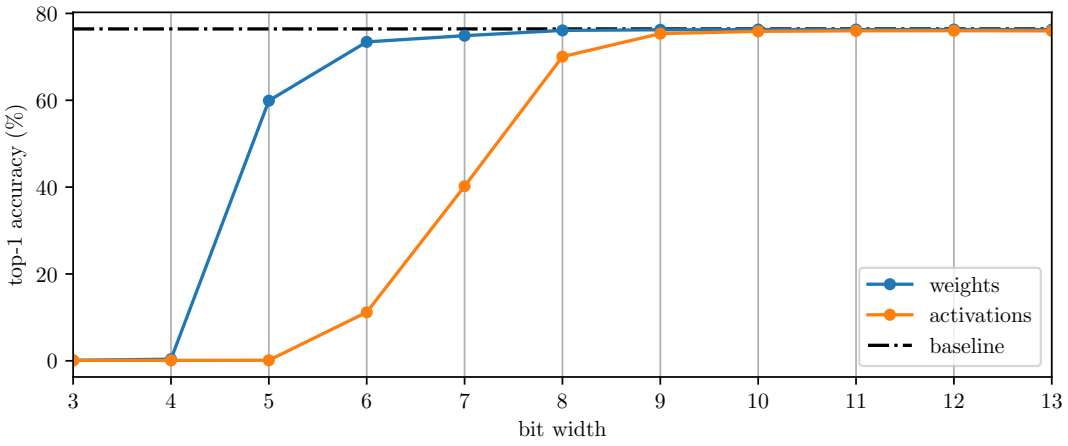

Figure 1: Difference in sensitivity between activation and weight quantization of the EfficientNet-B0 model pre-trained on ImageNet. As a quantization scheme uniform quantization without re-training was used. Activations are more sensitive to quantization since model performance drops significantly faster. Going below 8 bit is challenging and often requires (quantization-aware) re-training of the model to compensate for the quantization error. Data originates from [48].

3 Neural Network Quantization

For neural network computing, the default precision used on general hardware like GPUs or CPUs is 32 bit floating-point ("single-precision"), which causes high computational costs, power consumption, arithmetic operation latency and memory requirements [41]. Here, quantization techniques can also reduce the number of bits required to represent weight parameters and/or activations of the full-precision neural network, as they map the respective data values to a finite set of discrete quantization levels (clusters). Providing n such clusters allows to represent each data point in only $\log_2 n$ bit. However, the continuous reduction of the number of clusters generally leads to an increasingly large error and degraded performances (see the EfficientNet-B0[2] example in Figure 1).

This trade-off is a well-known problem in information theory and is addressed by rate-distortion optimization, a concept in lossy data compression. It aims to determine the minimal number of bits per data symbol (bitrate) at which the reconstruction of the compressed data does not exceed a certain level of distortion. Applying this to the domain of neural network quantization, the objective is to minimize the bitrate of the weight parameters while keeping model degradation caused by quantization below a certain threshold, i.e., the predictive performance of the model should not be affected by reduced parameter precisions. In contrast to multimedia compression approaches, e.g., for audio or video coding, the compression of DNNs has unique challenges and opportunities. Foremost, the neural network parameters to be compressed are not perceived directly by a user, as e.g., for video data. Therefore, the coding or compression error or distortion cannot be directly used as performance measure. Instead, such accuracy measurement needs to be deducted from a subsequent inference step. Then, current neural networks are highly over-parameterized [11] which allows for high errors/differences between the full-precision and the quantized parameters (while still maintaining model performance). Also, the various layer types and the location of a layer within the DNN have different impacts on the loss function, and thus different sensitivities to quantization.

Quantization can be further classified into uniform and non-uniform quantization. The

[2] `https://github.com/lukemelas/EfficientNet-PyTorch`, Apache License, Version 2.0 - Copyright (c) 2019 Luke Melas-Kyriazi

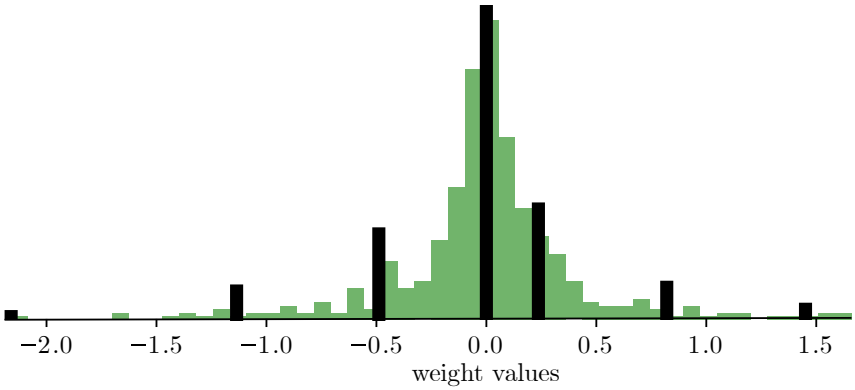

Figure 2: Quantizing a neural network's layer weights (binned weight distribution shown as green bars) to 7 discrete cluster centers (centroids). The centroids (black bars) were generated by k-means clustering and the height of each bar represents the number of layer weights which are assigned to the respective centroid.

most intuitive way to initialize centroids is by arranging them equidistantly over the range of parameter values (uniform). Other quantization schemes make use of non-uniform mapping functions, e.g., k-means clustering, which is determined by the distribution of weight values (see Figure 2). As non-uniform quantization captures the underlying distribution of parameter values better, it may achieve less distortion compared to equidistantly arranged centroids. However, non-uniform schemes are typically more difficult to deploy on hardware, e.g., they require a codebook (look-up table), whereas uniform quantization can be implemented using a single scaling factor (step size) which allows a very efficient hardware implementation with fixed-point integer logic.

3.1 Entropy-Constrained Quantization

As discussed in [47], and experimentally shown in [48], lowering the entropy of DNN weights provides benefits in terms of memory as well as computational complexity. The Entropy-Constrained Quantization (ECQ) algorithm is a clustering algorithm that also takes the entropy of the weight distributions into account. More precisely, the first-order entropy $H = -\sum_c P_c \log_2 P_c$ is used, where P_c is the ratio of the number of parameter elements in the c-th cluster to the number of all parameter elements (i.e., the source distribution). To recall, the entropy H is the theoretical limit of the average number of bits required to represent any element of the distribution [37].

Thus, ECQ assigns weight values not only based on their distances to the centroids, but also based on the information content of the clusters. Similar to other rate-distortion-optimization methods, ECQ applies Lagrange optimization:

$$\mathbf{A}^{(l)} = \underset{c}{\operatorname{argmin}}\ d(\mathbf{W}^{(l)}, w_c^{(l)}) - \lambda^{(l)} \log_2(P_c^{(l)}). \tag{1}$$

Given the full-precision weight matrix $\mathbf{W}^{(l)}$ and the centroid values $w_c^{(l)}$ of layer l, the first term in Equation (1) measures the squared distance between all weight elements and the centroids, indexed by c. The second term in Equation (1) is weighted by the scalar Lagrange parameter $\lambda^{(l)}$ and describes the entropy constraint. More precisely, the information content I is considered, i.e., $I = -\log_2(P_c^{(l)})$, where the probability $P_c^{(l)} \in [0, 1]$ defines how likely a weight element $w_{ij}^{(l)} \in \mathbf{W}^{(l)}$ is going to be assigned to centroid $w_c^{(l)}$. Data elements with a high occurrence frequency, or a high probability, contain a low information content, and vice versa. P is calculated layer-wise as $P_c^{(l)} = N_{w_c}^{(l)} / N_{\mathbf{W}}^{(l)}$, with $N_{w_c}^{(l)}$ being the number of full-precision weight

elements assigned to the cluster with centroid value $w_c^{(l)}$ (based on the squared distance), and $N_{\mathbf{W}}^{(l)}$ being the total number of parameters in $\mathbf{W}^{(l)}$. Note that $\lambda^{(l)}$ is scaled with a factor based on the number of parameters a layer has in proportion to other layers in the network to mitigate the constraint for smaller layers.

The entropy regularization term motivates sparsity and low-bit weight quantization in order to achieve smaller coded neural network representations. Based on the specific neural network coding optimization, we developed ECQ. This algorithm is based on previous work in Entropy-Constrained Trained Ternarization (EC2T) [26]. EC2T trains sparse and ternary DNNs to state-of-the-art accuracies. In our developed ECQ, we generalize the EC2T method, such that DNNs of variable bit width can be rendered. Also, ECQ does not train centroid values to facilitate integer arithmetic on general hardware. The proposed quantization-aware training algorithm includes the following steps:

1. Quantize weight parameters by applying ECQ (but keep a copy of the full-precision weights).
2. Apply Straight-Through Estimator (STE) [5]:
   (a) Compute forward and backward pass through quantized model version.
   (b) Update full-precision weights with scaled gradients obtained from quantized model.

4 Explainability-Driven Quantization

Explainable AI techniques can be applied to find relevant features in input as well as latent space. Covering large sets of data, identification of relevant and functional model substructures is thus possible. Assuming over-parameterization of DNNs, the authors of [49] exploit this for pruning (of irrelevant filters) to great effect. Their successful implementation shows the potential of applying XAI for the purpose of quantization as well, as sparsification is part of quantization, e.g., by assigning weights to the zero-cluster. Here, XAI opens up the possibility to go beyond regarding model weights as static quantities and to consider the interaction of the model with given (reference) data. This work aims to combine the two orthogonal approaches of ECQ and XAI in order to further improve sparsity and efficiency of DNNs. In the following, the LRP method is introduced, which can be applied to extract relevances of individual neurons, as well as weights.

4.1 Layer-wise Relevance Propagation

Layer-wise Relevance Propagation (LRP) [3] is an attribution method based on the conservation of flows and proportional decomposition. It explicitly is aligned to the layered structure of machine learning models. Regarding a model with n layers

$$f(x) = f_n \circ \cdots \circ f_1(x) \ , \tag{2}$$

LRP first calculates all activations during the forward pass starting with f_1 until the output layer f_n is reached. Thereafter, the prediction score $f(x)$ of any chosen model output is redistributed layer-wise as an initial quantity of relevance R_n back towards the input. During this backward pass, the redistribution process follows a conservation principle analogous to Kirchhoff's laws in electrical circuits. Specifically, all relevance that flows into a neuron is redistributed towards neurons of the layer below. In the context of neural network predictors, the whole LRP procedure can be efficiently implemented as a forward–backward pass with modified gradient computation, as demonstrated in, e.g., [33].

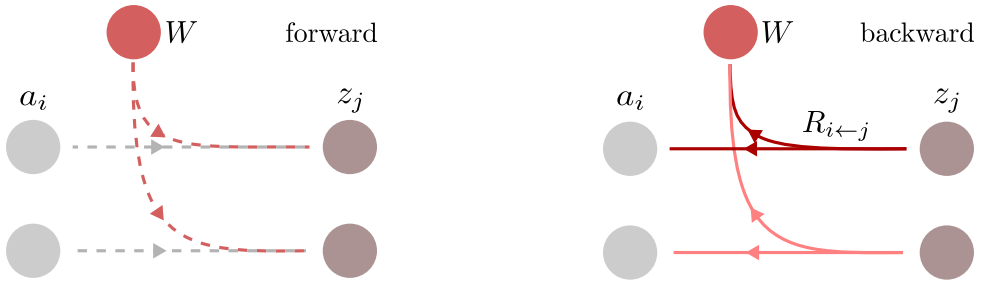

Figure 3: LRP can be utilized to calculate relevance scores for weight parameters W, which contribute to the activation of output neurons z_j during the forward pass in interaction with data-dependent inputs a_i. In the backward pass, relevance messages $R_{i\leftarrow j}$ can be aggregated at neurons / input activations a_i, but also at weights W.

Considering a layer's output neuron j, the distribution of its assigned relevance score R_j towards its lower layer input neurons i can be, in general, achieved by applying the basic decomposition rule

$$R_{i\leftarrow j} = \frac{z_{ij}}{z_j} R_j \ , \tag{3}$$

where z_{ij} describes the contribution of neuron i to the activation of neuron j [3, 27] and z_j is the aggregation of the pre-activations z_{ij} at output neuron j, i.e., $z_j = \sum_i z_{ij}$. Here, the denominator enforces the conservation principle over all i contributing to j, meaning $\sum_i R_{i\leftarrow j} = R_j$. This is achieved by ensuring the decomposition of R_j is in proportion to the relative flow of activations z_{ij}/z_j in the forward pass. The relevance of a neuron i is then simply an aggregation of all incoming relevance quantities

$$R_i = \sum_j R_{i\leftarrow j} \ . \tag{4}$$

Given the conservation of relevance in the decomposition step of Equation (3), this means that $\sum_i R_i = \sum_j R_j$ holds for consecutive neural network layers. Next to component-wise non-linearities, linearly transforming layers (e.g., dense or convolutional) are by far the most common and basic building blocks of neural networks such as VGG-16 [39] or ResNet [17]. While LRP treats the former via identity backward passes, relevance decomposition formulas can be given for the latter explicitly in terms of weights w_{ij} and input activations a_i. Let the output of a linear neuron be given as $z_j = \sum_{i,0} z_{ij} = \sum_{i,0} a_i w_{ij}$ with bias "weight" w_{0j} and respective activation $a_0 = 1$. In accordance to Equation (3), relevance is then propagated as

$$R_{i\leftarrow j} = \overbrace{\underbrace{a_i w_{ij}}_{z_{ij}} \frac{R_j}{z_j}}^{\text{explicit}} = a_i \overbrace{\underbrace{w_{ij}}_{\frac{\partial z_j}{\partial a_i}} \frac{R_j}{z_j}}^{\text{mod. grad.}} = w_{ij} \overbrace{\underbrace{a_i}_{\frac{\partial z_j}{\partial w_{ij}}} \frac{R_j}{z_j}}^{\text{mod. grad.}} \ . \tag{5}$$

Equation (5) exemplifies, that the explicit computation of the backward directed relevances $R_{i\leftarrow j}$ in linear layers can be replaced equivalently by a "(modified gradient $\times$ input)" approach. Therefore, the activation a_i or weight w_{ij} can act as the input and target wrt. which the partial derivative regarding output z_j is computed. The scaled relevance term R_j/z_j takes the role of the upstream gradient to be propagated.

At this point, LRP offers the possibility to calculate relevances not only of neurons, but also of individual weights, depending on the aggregation strategy, as illustrated in Figure 3. This can be achieved by aggregating relevances at the corresponding (gradient) targets, i.e., plugging Equation (5) into Equation (4). For a dense layer, this yields

$$R_{w_{ij}} = R_{i\leftarrow j} \tag{6}$$

with an individual weight as the aggregation target contributing (exactly) once to an output. A weight of a convolutional filter however is applied multiple times within a neural network layer. Here, we introduce a variable k signifying one such application context, e.g., one specific step in the application of a filter w in a (strided) convolution, mapping the filter's inputs i to an output j. While the relevance decomposition formula within one such context k does not change from Equation (3), we can uniquely identify its backwards distributed relevance messages as $R^k_{i\leftarrow j}$. With that, the aggregation of relevance at the convolutional filter w at a given layer is given with

$$R_{w_{ij}} = \sum_k R^k_{i\leftarrow j}, \tag{7}$$

where k iterates over all applications of this filter weight.

Note that in modern deep learning frameworks, derivatives wrt. activations or weights can be computed efficiently by leveraging the available automatic differentiation functionality (autograd) [31]. Specifying the gradient target, autograd then already merges the relevance decomposition and aggregation steps outlined above. Thus, computation of relevance scores for filter weights in convolutional layers is also appropriately supported, for Equation (3), as well as any other relevance decomposition rule which can be formulated as a modified gradient backward pass, such as Equations (8) and (9). The ability to compute the relevance of individual weights is a critical ingredient for the eXplainability-driven Entropy-Constrained Quantization strategy introduced in Section 4.2.

In the following, we will briefly introduce further LRP decomposition rules used throughout our study. In order to increase numerical stability of the basic decomposition rule in Equation (3), the LRP ε-rule introduces a small term ε in the denominator:

$$R_{i\leftarrow j} = \frac{z_{ij}}{z_j + \varepsilon \cdot \text{sign}(z_j)} R_j \ . \tag{8}$$

The term ε absorbs relevance for weak or contradictory contributions to the activation of neuron j. Note here, in order to avoid divisions by zero, the sign(z) function is defined to return 1 if $z \geq 0$ and -1 otherwise. In the case of a deep rectifier network, it can be shown [1] that the application of this rule to the whole neural network results in an explanation that is similar to (simple) (gradient $\times$ input) [38]. A common problem within deep neural networks is, that the gradient becomes increasingly noisy with network depth [33], partly a result from gradient shattering [4]. The ε parameter is able to suppress the influence of that noise given sufficient magnitude. With the aim of achieving robust decompositions, several purposed rules next to Equations (3) and (8) have been proposed in literature (see [27] for an overview).

One particular rule choice, which reduces the problem of gradient shattering and which has been shown to work well in practice, is the $\alpha\beta$-rule [3, 28]

$$R_{i\leftarrow j} = \left(\alpha \frac{(z_{ij})^+}{(z_j)^+} - \beta \frac{(z_{ij})^-}{(z_j)^-} \right) R_j \ , \tag{9}$$

where $(\cdot)^+$ and $(\cdot)^-$ denote the positive and negative parts of the variables z_{ij} and z_j, respectively. Further, the parameters α and β are chosen subject to the constraints $\alpha - \beta = 1$ and $\beta \geq 0$ (i.e., $\alpha \geq 1$) in order to propagate relevance conservatively throughout the network. Setting $\alpha = 1$, the relevance flow is computed only with respect to the positive contributions $(z_{ij})^+$ in the forward pass. When alternatively parameterizing with, e.g., $\alpha = 2$ and $\beta = 1$, which is a common choice in literature, negative contributions are included as well, while favoring positive contributions.

Recent works recommend a composite strategy of decomposition rule assignments mapping multiple rules purposedly to different parts of the network [27, 23]. This leads to an increased quality of relevance attributions for the intention of explaining prediction outcomes. In the following, a composite strategy consisting of the ε-rule for dense layers and the $\alpha\beta$-rule with

$\beta = 1$ for convolutional layers is used. Regarding LRP-based pruning, Yeom et al. [49] utilize the $\alpha\beta$-rule (9) with $\beta = 0$ for convolutional as well as dense layers. However, using $\beta = 0$, subparts of the network that contributed solely negatively, might receive no relevance. In our case of quantization, all individual weights have to be considered. Thus, the $\alpha\beta$-rule with $\beta = 1$ is used for convolutional layers, because it also includes negative contributions in the relevance distribution process and reduces gradient shattering. The LRP implementation is based on the software package Zennit [2], which offers a flexible integration of composite strategies and readily enables extensions required for the computation of relevance scores for weights.

4.2 eXplainability-driven Entropy-Constrained Quantization

For our novel eXplainability-driven Entropy-Constrained Quantization (ECQ$^{\text{x}}$), we modify the ECQ assignment function to optimally re-assign the weight clustering based on LRP relevances in order to achieve higher performance measures and compression efficiency. The rationale behind using LRP to optimize the ECQ quantization algorithm is two-fold:

Assignment correction: In the quantization process, the entropy regularization term encourages weight assignments to more populated clusters in order to minimize the overall entropy. Since weights are usually normally distributed around zero, the entropy term also strongly encourages sparsity. In practice, this quantization scheme works well rendering sparse and low-bit neural networks for various machine learning tasks and network architectures [48, 26, 46].

From a scientific point of view, however, one might wonder why the shift of numerous weights from their nearest-neighbor clusters to a more distant cluster does not lead to greater model degradation, especially when assigned to zero. The quantization-aware re-training and fine-tuning can, up to a certain extent, compensate for this shift. Here, the LRP-generated relevances show potential to further improve quantization in two ways: 1) by re-adding "highly relevant" weights (i.e., preventing their assignment to zero if they have a high relevance), and 2) by assigning additional, "irrelevant" weights to zero (i.e., preventing their distance- and entropy-based assignment to a non-zero centroid).

We evaluated the discrepancy between weight relevance and magnitude in a correlation analysis depicted in Figure 4. Here, all weight values w_{ij} are plotted against their associated relevance $R_{w_{ij}}$ for the input layer (left) and output layer (right) of the full-precision model MLP_GSC (which will be introduced in Section 5.1). In addition, histograms of both parameters are shown above and to the right of each relevance-weight-chart in Figure 4 to better visualize the correlation between w_{ij} and $R_{w_{ij}}$. In particular, a weight of high magnitude is not necessarily also a relevant weight. And in contrast, there are also weights of small or medium magnitude that have a high relevance and thus should not be omitted in the quantization process. This phenomenon is especially true for layers closer to the input. The outcome of this analysis strongly motivates the use of LRP relevances for the weight assignment correction process of low-bit and sparse ECQ$^{\text{x}}$.

Regularizing effect for training: Since the previously described re-adding (which is also referred to as "regrowth" in literature) and removing of weights due to LRP depends on the propagated input data, weight relevances can change from data batch to data batch. In our quantization-aware training, we apply the STE, and thus the re-assignment of weights, after each forward-backward pass.

The regularizing effect which occurs due to dynamic re-adding and removing weights is probably related to the generalization effect which random Dropout [40] has on neural networks. However, as elaborated in the extensive survey by Hoefler et al. [20], in terms of dynamic sparsification, re-adding ("drop in") the best weights is as crucial as removing ("drop out") the right ones. Instead of randomly dropping weights, the work in [9] shows that re-adding weights based on largest gradients is related to Hebbian learning and biologically more plausible. LRP

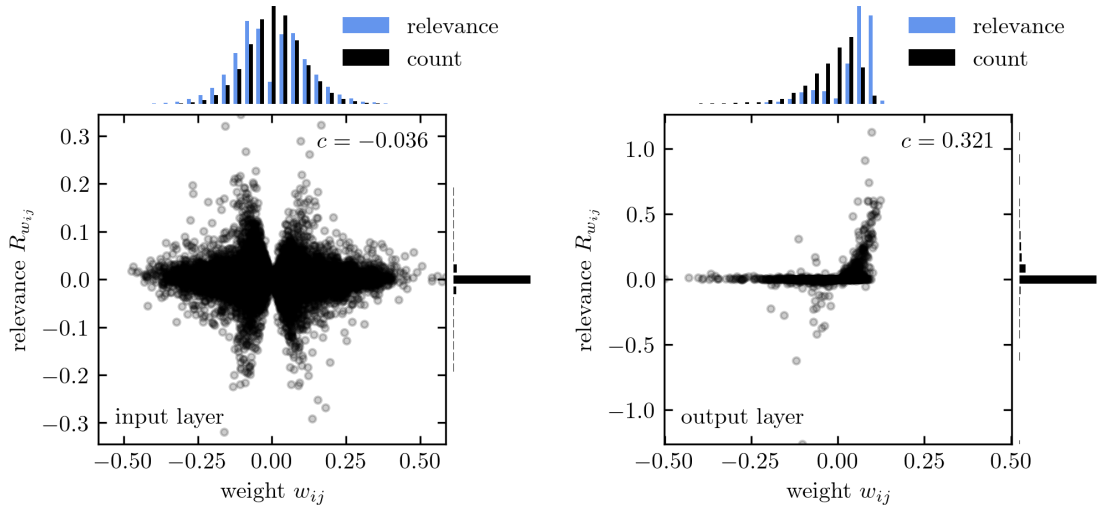

Figure 4: Weight relevance $R_{w_{ij}}$ vs. weight value w_{ij} for the input layer (left) and output layer (right) of the full-precision MLP_GSC model (introduced in Section 5.1). The black histograms to the top and right of each panel display the distributions of weights (top) and relevances (right). The blue histograms further show the amount of relevance (blue) of each weight histogram bin. All relevances are collected over the validation set with equally weighted samples (i.e., by choosing $R_n = 1$). The value c measures the Pearsson correlation coefficient between weights and relevances.

relevances go beyond the gradient criterion, which is why we consider it a suitable candidate.

In order to embed LRP relevances in the assignment function (1), we update the cost for the zero centroid ($c = 0$) by extending it as

$$\rho \; \mathbf{R}_{W^{(l)}} \cdot \Big(d(\mathbf{W}^{(l)}, w^{(l)}_{c=0}) - \lambda^{(l)} \; \log_2(P^{(l)}_{c=0}) \Big) \tag{10}$$

with relevance matrix $\mathbf{R}_{W^{(l)}}$ containing all weight relevances $R_{w_{ij}}$ of layer l with row/input index i and column/output index j, as specified in Equation (7). The full assignment is thus described by:

$$\mathbf{A}^{(l)}_{\mathrm{x}}(\mathbf{W}^{(l)}) = \underset{c}{\operatorname{argmin}} \begin{cases} \rho \; \mathbf{R}_{W^{(l)}} \cdot \Big(d(\mathbf{W}^{(l)}, w^{(l)}_{c=0}) - \lambda^{(l)} \; \log_2(P^{(l)}_{c=0}) \Big) & \text{, if } c = 0 \\ d(\mathbf{W}^{(l)}, w^{(l)}_{c}) - \lambda^{(l)} \; \log_2(P^{(l)}_{c}) & \text{, if } c \neq 0 \end{cases} \tag{11}$$

where ρ is a normalizing scaling factor, which also takes relevances of the previous data batches into account (momentum). The term $\rho \; R_W$ increases the assignment cost of the zero cluster for relevant weights and decreases it for irrelevant weights.

Figure 5 shows an example of one ECQ$^{\mathrm{x}}$ iteration that includes the following steps: 1) ECQ$^{\mathrm{x}}$ computes the forward-backward pass through the quantized model, derives gradients and computes LRP relevances. 2) LRP relevances are then scaled by factor ρ, and 3) gradients are scaled by non-zero centroid values. 4) The scaled gradients are then applied to the full-precision background model, 5) This model is updated with ADAM and the weights are assigned to their nearest-neighbor cluster centroids. 6) Finally, the assignment cost for each weight to each centroid is calculated using the λ-scaled information content of clusters (i.e., $I_{-\text{ (blue)}} \approx 1.7$, $I_{0\text{ (green)}} = 1.0$ and $I_{+\text{ (purple)}} \approx 2.4$ in this example) and ρ-scaled relevances. Here, relevances above the threshold (i.e., mean $\bar{\mathbf{R}}_W \approx 0.3$) increase the cost for the zero cluster assignment, while relevances below (highlighted in red) decrease it. Each weight is assigned such that the

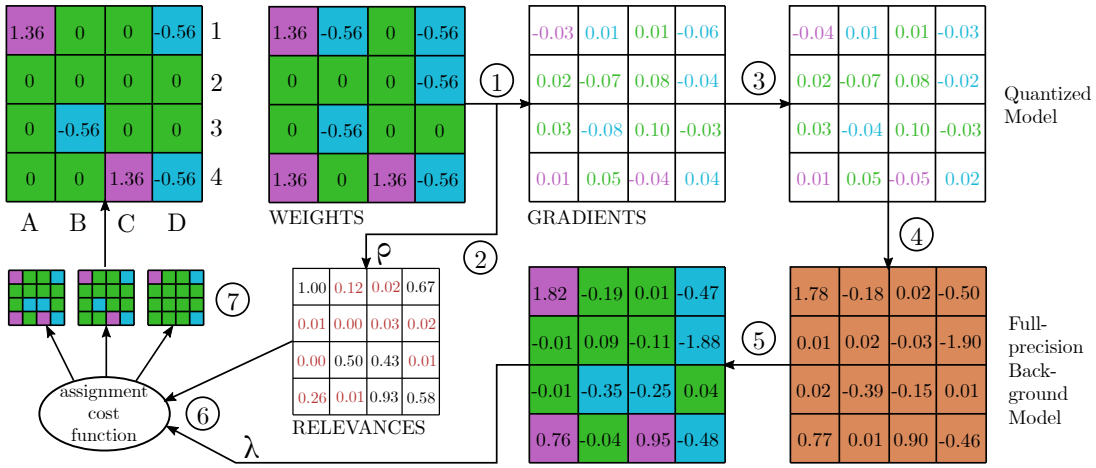

Figure 5: Exemplary ECQ$^{\text{x}}$ weight update. For simplicity, 3 centroids are used (i.e., symmetric 2 bit case). The process involves the following steps: 1) Derive gradients and LRP relevances from forward-backward pass. 2) LRP relevance scaling. 3) Gradients scaling. 4) Gradient attachment to full precision background model. 5) Background model update and nearest-neighbor clustering. 6) Computing of the assignment cost for each weight using the λ-scaled information content of clusters and the ρ-scaled relevances. Assign each weight by minimizing the cost. 7) Choosing an appropriate candidate (of various λ and ρ settings).

cost function is minimized (cf. Equation (11)). 7) Depending on the intensity of the entropy and relevance constraints (controlled by λ and ρ), different assignment candidates can be rendered to fit a specific deep learning task. In the example shown in Figure 5, an exemplary candidate grid was selected, which is depicted at the top left of the Figure. The weight at grid coordinate $D2$, for example, was assigned to the zero cluster due to its irrelevance and the weight at $C3$ due to the entropy constraint.

In the case of dense or convolutional layers, LRP relevances can be computed efficiently using the autograd functionality, as mentioned in Section 4.1. For a classification task, it is sensible to use the target class score as a starting point for the LRP backward pass. This way, the relevance of a neuron or weight describes its contribution to the target class prediction. Since the output is propagated throughout the network, all relevance is proportional to the output score. Consequently, relevances of each sample in a training batch are, in general, weighted differently according to their respective model output, or prediction confidence. However, with the aim of suppressing relevances for inaccurate predictions, it is sensible to weigh samples according to the model output, because a low output score usually corresponds to an unconfident decision of the model.

After the relevance calculation of a whole data batch, the relevance scores $\mathbf{R}_{W^{(l)}}$ are transformed to their absolute value and normalized, such that $\mathbf{R}_{W^{(l)}} \in [0, 1]$. Even though negative contributions work against an output, they might still be relevant to the network functionality, and their influence is thus considered instead of omitted. On one hand, they can lead to positive contributions for other classes. On the other, they can be relevant to balancing neuron activations throughout the network.

The relevance matrices $\mathbf{R}_{W^{(l)}}$ resulting from LRP are usually sparse, as can be seen in the weight histograms of Figure 4. In order to control the effect of LRP in the assignment function, the relevances are exponentially transformed by β, applying a similar effect as for gamma correction in image processing:

$$\mathbf{R}'_{W^{(l)}} = (\mathbf{R}_{W^{(l)}})^{\beta}$$

with $\beta \in [0, 1]$. Here, the parameter β is initially chosen such that the mean relevance $\hat{R}_{W^{(l)}}$

does not change the assignment, e.g., $\rho\left(\hat{R}_{W^{(l)}}\right)^{\beta} = 1$ or $\beta = -\frac{\ln \rho}{\ln \hat{R}_{W^{(l)}}}$. In order to further control the sparsity of a layer, the target sparsity p is introduced. If the assignment increases a layer's sparsity by more than the target sparsity p, parameter β is accordingly minimized. Thus, in ECQ$^{\text{x}}$, LRP relevances are directly included in the assignment function and their effect can be controlled by parameter p. An experimental validation of the developed ECQ$^{\text{x}}$ method, including state-of-the-art comparison and parameter variation tests, is given in the following section.

5 Experiments

In the experiments, we evaluate our novel quantization method ECQ$^{\text{x}}$ using two widely used neural network architectures, namely a convolutional neural network (CNN) and a multilayer perceptron (MLP). More precisely, we deploy VGG16 for the task of small-scale image classification (CIFAR-10) and an MLP with 5 hidden layers and ReLU non-linearities solving the task of keyword spotting in audio data (Google Speech Commands).

In the first subsection, the experimental setup and test conditions are described, while the results are shown and discussed in the second subsection. In particular, results for ECQ$^{\text{x}}$ hyperparameter variation are shown, followed by a comparison against classical ECQ and results for bit width variation. Finally, overall results for ECQ$^{\text{x}}$ for different accuracy and compression measurements are shown and discussed.

5.1 Experimental Setup

All experiments were conducted using the PyTorch deep learning framework, version 1.7.1 with torchvision 0.8.2 and torchaudio 0.7.2 extensions. As a hardware platform we used Tesla V100 GPUs with CUDA version 10.2. The quantization-aware training of ECQ$^{\text{x}}$ was executed for 20 epochs in all experiments. As an optimizer we used ADAM with an initial learning rate of 0.0001. In the scope of the training procedure, we consider *all* convolutional and fully-connected layers of the neural networks for quantization, including the input and output layers. Note that numerous approaches in related works keep the input and/or output layers in full-precision (32 bit float), which may compensate for the model degradation caused by quantization, but is usually difficult to bring into application and incurs significant overhead in terms of energy consumption.

5.1.1 Google Speech Commands

The Google Speech Commands (GSC [44]) dataset consists of 105,829 utterances of 35 words recorded from 2,618 speakers. The standard is to discriminate ten words "Yes", "No", "Up", "Down", "Left", "Right", "On", "Off", "Stop", and "Go", and adding two additional labels, one for "Unknown Words", and another for "Silence" (no speech detected). Following the official Tensorflow example code for training[3], we implemented the corresponding data augmentation with PyTorch's torchaudio package. It includes randomly adding background noise with a probability of 80% and time shifting the audio $[-100, 100]$ ms with a probability of 50%. To generate features, the audio is transformed to MFCC fingerprints (Mel Frequency Cepstral Coefficients). We use 15 bins and a window length of 2000 ms. To solve GSC, we deploy a MLP (which we name *MLP_GSC* in the following) consisting of an input layer, five hidden layers and an output layer featuring 512, 512, 256, 256, 128, 128 and 12 output features, respectively. The MLP_GSC was pre-trained for 100 epochs using stochastic gradient descent (SGD) optimization with a momentum of 0.9, an initial learning rate of 0.01 and a cosine annealing learning rate schedule.

[3] `https://github.com/tensorflow/tensorflow/tree/master/tensorflow/examples/speech_commands`

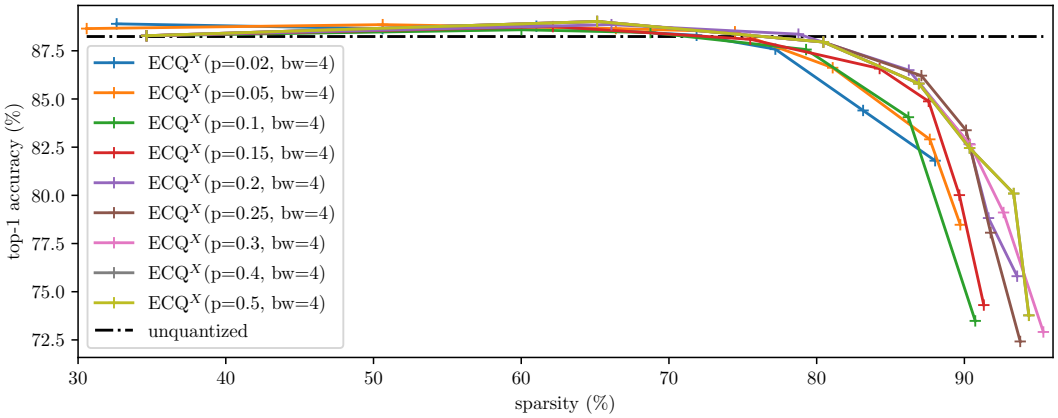

Figure 6: Hyperparameter p controls the LRP-introduced sparsity.

5.1.2 CIFAR-10

The CIFAR-10 [24] dataset consists of natural images with a resolution of 32×32 pixels. It contains 10 classes, with 6,000 images per class. Data is split to 50,000 training and 10,000 test images. We use standard data pre-processing, i.e., normalization, random horizontal flipping and cropping. To solve the task, we deploy a VGG16 from the torchvision model zoo[4]. The VGG16 classifier is adapted from 1,000 ImageNet classes to ten CIFAR classes by replacing its three fully-connected layers (with dimensions [25,088, 4,096], [4,096, 4,096], [4,096, 1,000]) by two ([512, 512], [512, 10]), as a consequence of CIFAR's smaller image size. We also implemented a VGG16 supporting batch normalization (VGG16_bn from torchvision). The VGGs were transfer-learned for 60 epochs using ADAM optimization and an initial learning rate of 0.0005.

5.2 ECQ^x Results

In this subsection, we compare ECQ^x to state-of-the-art ECQ quantization, analysing accuracy preservation vs. sparsity increase. Furthermore, we investigate ECQ^x compressibility, behavior on BatchNorm layers, and an appropriate choice of hyperparameters.

5.2.1 ECQ^x Hyperparameter Variation

In ECQ^x, two important hyperparameters, λ and p, influence the performance and thus are optimized for the comparative experiments described below. λ increases the intensity of the entropy constraint and thus distributes the working points of each trial over a range of sparsities (see Figure 6). The p hyperparameter defines an upper bound for the per-layer percentage of zero values, allowing a maximum amount of p additional sparsity, on top of the λ-introduced sparsity. It thus implicitly controls the intensity of the LRP constraint.

Figure 6 shows results using several p values for the 4 bit ($bw = 4$) quantization of the MLP_GSC model. Note, that the variation of bit width bw is discussed below the comparative results. For smaller p, less sparse models are rendered with higher top-1 accuracies in the low-sparsity regime (e.g., $p = 0.02$ or $p = 0.05$ between 30-50% total network sparsity). In the regime of higher sparsity, larger values of p show a better sparsity-accuracy trade-off. Note, that larger p do not only set more weights to zero but also re-add relevant weights (regrowth). For $p = 0.4$ and $p = 0.5$, both lines are congruent since no layer is achieving more than 40% additional LRP-introduced sparsity with the initial β value (cf. Section 4.2).

[4] `https://pytorch.org/vision/stable/models.html`

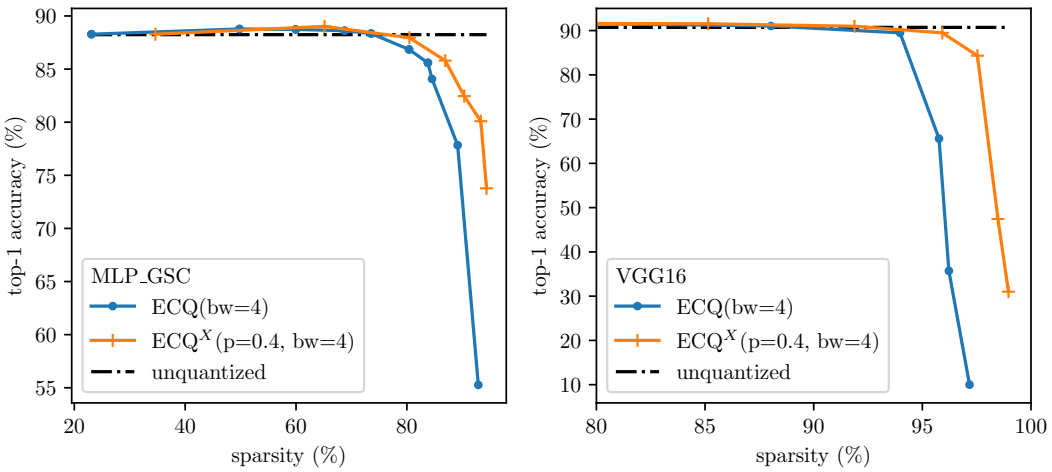

Figure 7: Resulting model performances, when applying ECQ vs. ECQ$^{\text{x}}$ 4 bit quantization on MLP_GSC (left) and VGG16 (right). Each point corresponds to a model rendered with a specific λ which is a regulator for the entropy constraint and thus incrementally enhances sparsity. Abbreviations in the legend labels refer to bit width (bw) and target sparsity (p), which is defined in 4.2.

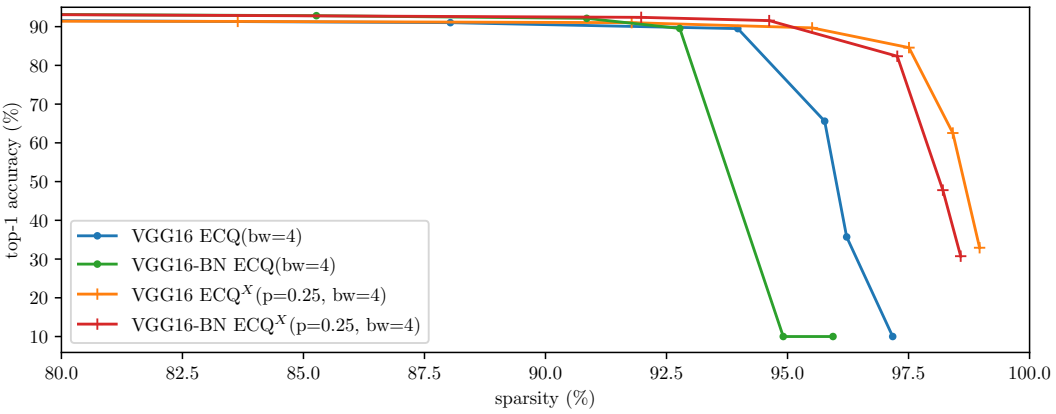

Figure 8: Resulting model performances, when applying ECQ vs. ECQ$^{\text{x}}$ 4 bit quantization on VGG16 and VGG16 with BatchNorm modules.

5.2.2 ECQ$^{\text{x}}$ vs. ECQ Analysis

As shown in Figure 7, the LRP-driven ECQ$^{\text{x}}$ approach renders models with higher performance and simultaneously higher efficiency. In this comparison, efficiency is determined in terms of sparsity, which can be exploited to compress the model more or to skip arithmetic operations with zero values. Both methods achieve a quantization to 4 bit integer without any performance degradation of the model. Performance is even slightly increased due to quantization when compared to the unquantized baseline. In the regime of high sparsity, model accuracy of the previous state-of-the-art (ECQ) drops significantly faster compared to the LRP-adjusted quantization scheme.

Regarding the handling of BatchNorm modules for LRP, it is proposed in literature to merge the BatchNorm layer parameters with the preceding linear layer [14] into a single linear transformation. This canonization process is sensible, because it reduces the number of computational steps in the backward pass while maintaining functional equivalence between the original and the canonized model in the forward pass.

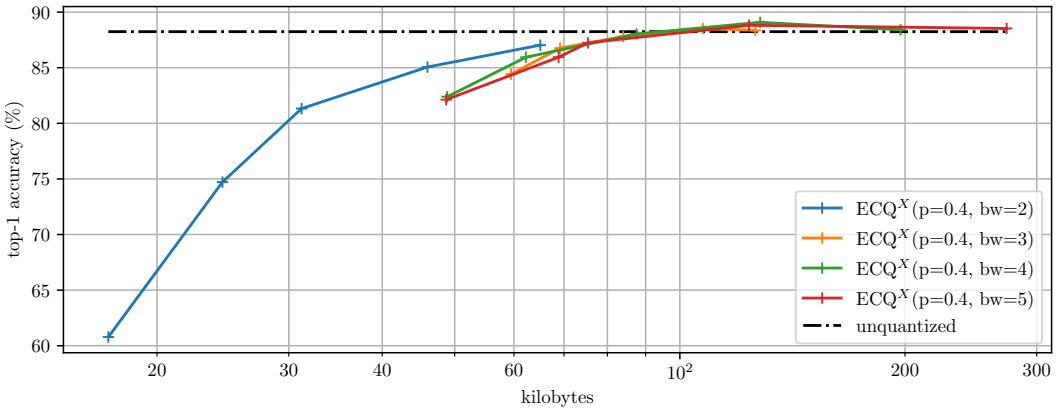

Figure 9: Resulting MLP_GSC model performances vs. memory footprint, when applying ECQ^x with 2 bit to 5 bit quantization.

It has been further shown, that network canonization can increase explanation quality [14]. With the aim of computing weight relevance scores for a BatchNorm layer's adjacent linear layer in its original (trainable) state, keeping the layers separate is more favorable than merging. Therefore, the $\alpha\beta$-rule with $\beta = 1$ is also applied to BatchNorm layers. The quantization results of the VGG architecture with BatchNorm modules are shown in Figure 8.

5.2.3 Bit Width Variation

Bit width reduction has multiple benefits over full-precision in terms of memory, latency, power consumption, and chip area efficiency. For instance, a reduction from standard 32 bit precision to 8 bit or 4 bit directly leads to a memory reduction of almost 4× and 8×. Arithmetic with lower bit width is exponentially faster if the hardware supports it. E.g., since the release of NVIDIA's Turing architecture, 4 bit integer is supported which increases the throughput of the RTX 6000 GPU to 522 TOPS (tera operations per second), when compared to 8 bit integer (261 TOPS) or 32 bit floating point (14.2 TFLOPS) [29]. Furthermore, Horowitz showed that, for a 45 nm technology, low-precision logic is significantly more efficient in terms of energy and area [21]. For example, performing 8 bit integer addition and multiplication is 30× and 19× more energy efficient compared to 32 bit floating point addition and multiplication. The respective chip area efficiency is increased by 116× and 27× as compared to 32 bit float. It is also shown that memory reads and writes have the highest energy cost, especially when reading data from external DRAM. This further motivates bit width reduction because it can reduce the number of overall RAM accesses since more data fits into the same caches/registers when having a reduced precision.

In order to investigate different bit widths in the regime of ultra low precision, we compare the compressibility and model performances of the deployed networks when quantized to 2 bit, 3 bit, 4 bit and 5 bit integer values (see Figures 9 and 10). Here, we directly encoded the integer tensors with the DeepCABAC codec of the ISO/IEC MPEG NNR standard [22]. The least sparse working points of each trial, i.e., the rightmost data points of each line, show the expected behaviour, namely that compressibility is increased by continuously reducing the bit width from 5 bit to 2 bit. However, this effect decreases or even reverses when the bit width is in the range of 3 bit to 5 bit. In other words, reducing the number of centroids from $2^5 = 32$ to $2^3 = 8$ does not necessarily lead to a further significant reduction in the resulting bitstream size if sparsity is predominant. The 2 bit quantization still minimizes the size of the bit stream, even if, especially for the VGG model, more accuracy is sacrificed for this purpose. Note that compressibility is only one reason for reducing bit width besides, for example, speeding up model

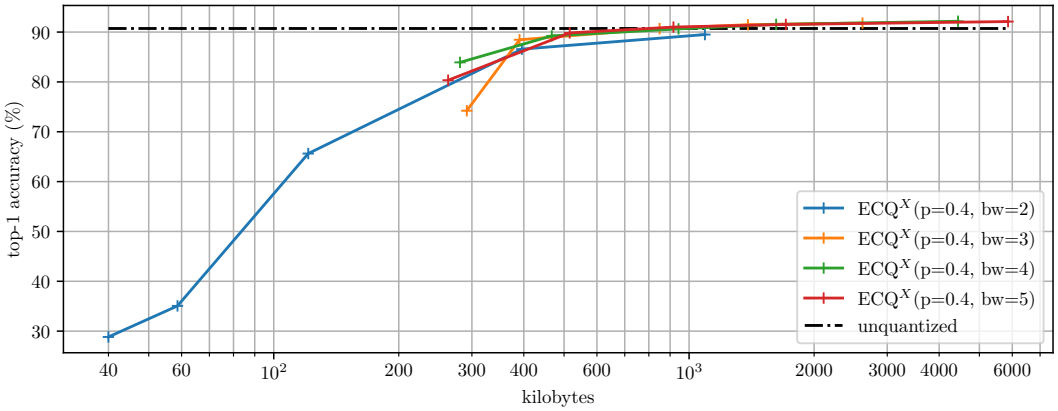

Figure 10: Resulting VGG16 model performances vs. memory footprint, when applying ECQ$^{\text{x}}$ with 2 bit to 5 bit quantization.

inference due to increased throughput.

5.2.4 ECQ$^{\text{x}}$ Results Overview

In addition to the performance graphs in the previous subsections, all quantization results are summarized in Table 1. Here, ECQ$^{\text{x}}$ and ECQ are compared specifically for a 2 and 4 bit quantization as these fit particularly well to power-of-two hardware registers. The ECQ$^{\text{x}}$ 4 bit quantization achieves a compression ratio for VGG16 of 103× with a negligible drop in accuracy of -0.1%. In comparison, ECQ achieves the same compression ratio only with a model degradation of -1.23% top-1 accuracy. For the 4 bit quantization of MLP_GSC, ECQ$^{\text{x}}$ achieves its highest accuracy ("drop", i.e., increase of $+0.71\%$ compared to the unquantized baseline model) with a compression ratio that is almost 10% larger compared to the highest achievable accuracy of ECQ ($+0.47\%$). For sparsities beyond 70% ECQ significantly reduces the model's predictive performance, e.g., at a sparsity of 80.39% ECQ shows a loss of -1.40% whereas ECQ$^{\text{x}}$ only degrades by -0.34%.

And finally, the 2 bit results in Table 1 show two major findings: 1) With only a minor model degradation all weight layers can also be quantized to only 4 discrete centroid values while still maintaining a high level of sparsity, 2) ECQ$^{\text{x}}$ renders higher compressible models in comparison to ECQ, as indicated by the higher compression ratios CR.

6 Conclusion

In this chapter we presented a new entropy-constrained neural network quantization method (ECQ$^{\text{x}}$), utilizing weight relevance information from Layer-wise Relevance Propagation (LRP). Thus, our novel method combines concepts of explainable AI (XAI) and information theory. In particular, instead of only assigning weight values based on their distances to respective quantization clusters, the assignment function additionally considers weight relevances based on LRP. In detail, each weight's contribution to inference in interaction with the transformed data, as well as cluster information content is calculated and applied. For this approach, we first utilized the observation that a weight's magnitude does not necessarily correlate with its importance or relevance for a model's inference capability. Next, we verified this observation in a relevance vs. weight (magnitude) correlation analysis and subsequently introduce our ECQ$^{\text{x}}$ method. As a result, smaller weight parameters that are usually omitted in a classical quantization process are preserved, if their relevance score indicates a stronger contribution to the overall neural network accuracy or performance.

Table 1: Quantization results for ECQ$^{\text{x}}$ for 2 bit and 4 bit quantization: highest accuracy, highest compression gain without model degradation (if possible) and highest compression gain with negligible degradation. Underlined values mark the best results in terms of performance and compressibility with negligible drop in top-1 accuracy.

Model	**Prec.**[a]	**Method**[b]	**Acc. (%)**	**Acc. drop**	$\frac{\lvert W=0\rvert}{\lvert W\rvert}$ **(%)**[c]	**Size (kB)**	**CR**[d]
				CIFAR-10			
VGG16	W4A16	**ECQ$^{\text{x}}$**	92.27	<u>+1.55</u>	41.39	4,446.39	13.48
	W4A16	**ECQ$^{\text{x}}$**	90.86	+0.14	91.95	933.99	64.17
	W4A16	**ECQ$^{\text{x}}$**	90.62	-0.10	94.67	584.16	<u>102.59</u>
	W4A16	ECQ	92.09	+1.37	29.88	4,658.01	12.87
	W4A16	ECQ	91.03	+0.31	88.03	1,246.27	48.09
	W4A16	ECQ	89.49	-1.23	93.97	585.40	102.37
	W2A16	**ECQ$^{\text{x}}$**	90.42	<u>-0.30</u>	83.23	1.394,52	<u>42.98</u>
	W2A16	ECQ	90.19	-0.53	81.58	1,486.76	40.31
				Google Speech Commands			
MLP_GSC	W4A16	**ECQ$^{\text{x}}$**	88.95	<u>+0.71</u>	65.14	128.03	20.05
	W4A16	**ECQ$^{\text{x}}$**	88.34	+0.10	78.77	92.46	27.77
	W4A16	**ECQ$^{\text{x}}$**	87.89	-0.34	80.45	87.52	<u>29.33</u>
	W4A16	ECQ	88.71	+0.47	59.95	139.96	18.34
	W4A16	ECQ	88.32	+0.08	70.74	98.32	26.11
	W4A16	ECQ	86.84	-1.40	80.39	69.67	36.85
	W2A16	**ECQ$^{\text{x}}$**	87.46	-0.78	83.97	68.77	<u>37.33</u>
	W2A16	ECQ	87.72	<u>-0.52</u>	77.55	78.54	32.69

[a] WxAy indicates a quantization of weights and activations to x and y bit.
[b] ECQ refers to ECQ$^{\text{x}}$ w/o LRP constraint
[c] Sparsity, measured as the percentage of zero-valued parameters in the DNN.
[d] Compression ratio (full-precision size / compressed size) when applying the DeepCABAC codec of the ISO/IEC MPEG NNR standard [22].

The experimental results show that this novel ECQ$^{\text{x}}$ method generates low bit width (2-5 bit) and sparse neural networks while maintaining or even improving model performance. Therefore, in particular the 2 and 4 bit variants are highly suitable for neural network hardware adaptation tasks. Due to the reduced parameter precision and high number of zero-elements, the rendered networks are also highly compressible in terms of file size, e.g., up to 103× compared to the full-precision unquantized DNN model, without degrading the model performance. Our ECQ$^{\text{x}}$ approach was evaluated on different types of models and datasets (including Google Speech Commands and CIFAR-10). The comparative results vs. state-of-the-art entropy-constrained-only quantization (ECQ) show a performance increase in terms of higher sparsity, as well as a higher compression. Finally, also hyperparameter optimization and bit width variation results were presented, from which the optimal parameter selection for ECQ$^{\text{x}}$ was derived.

Acknowledgements

This work was supported by the German Ministry for Education and Research as BIFOLD (ref. 01IS18025A and ref. 01IS18037A), the European Union's Horizon 2020 programme (grant no. 965221 and 957059), and the Investitionsbank Berlin under contract No. 10174498 (Pro FIT programme).

References

[1] Ancona, M., Ceolini, E., Öztireli, C., Gross, M.: Gradient-based attribution methods. In: Explainable AI: Interpreting, Explaining and Visualizing Deep Learning, pp. 169–191. Springer (2019)

[2] Anders, C.J., Neumann, D., Samek, W., Müller, K.R., Lapuschkin, S.: Software for dataset-wide xai: From local explanations to global insights with Zennit, CoRelAy, and ViRelAy. CoRR **abs/2106.13200** (2021)

[3] Bach, S., Binder, A., Montavon, G., Klauschen, F., Müller, K.R., Samek, W.: On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. PLoS ONE **10**(7), e0130140 (2015)

[4] Balduzzi, D., Frean, M., Leary, L., Lewis, J., Ma, K.W.D., McWilliams, B.: The shattered gradients problem: If resnets are the answer, then what is the question? In: International Conference on Machine Learning. pp. 342–350. PMLR (2017)

[5] Bengio, Y., Léonard, N., Courville, A.C.: Estimating or propagating gradients through stochastic neurons for conditional computation. CoRR **abs/1308.3432** (2013)

[6] Bhalgat, Y., Lee, J., Nagel, M., Blankevoort, T., Kwak, N.: Lsq+: Improving low-bit quantization through learnable offsets and better initialization. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops (June 2020)

[7] Choi, Y., El-Khamy, M., Lee, J.: Towards the limit of network quantization. CoRR **abs/1612.01543** (2016)

[8] Courbariaux, M., Bengio, Y., David, J.P.: Binaryconnect: Training deep neural networks with binary weights during propagations. In: Advances in Neural Information Processing Systems. pp. 3123–3131 (2015)

[9] Dai, X., Yin, H., Jha, N.K.: Nest: A neural network synthesis tool based on a grow-and-prune paradigm. IEEE Transactions on Computers **68**(10), 1487–1497 (2019)

[10] Deng, B.L., Li, G., Han, S., Shi, L., Xie, Y.: Model compression and hardware acceleration for neural networks: A comprehensive survey. Proceedings of the IEEE **108**(4), 485–532 (2020)

[11] Denil, M., Shakibi, B., Dinh, L., Ranzato, M., de Freitas, N.: Predicting parameters in deep learning. In: Advances in Neural Information Processing Systems. pp. 2148–2156 (2013)

[12] Elsken, T., Metzen, J.H., Hutter, F.: Neural architecture search: A survey. The Journal of Machine Learning Research **20**(1), 1997–2017 (2019)

[13] Gholami, A., Kim, S., Dong, Z., Yao, Z., Mahoney, M.W., Keutzer, K.: A survey of quantization methods for efficient neural network inference. CoRR **abs/2103.13630** (2021)

[14] Guillemot, M., Heusele, C., Korichi, R., Schnebert, S., Chen, L.: Breaking batch normalization for better explainability of deep neural networks through layer-wise relevance propagation. CoRR **abs/2002.11018** (2020)

[15] Han, S., Mao, H., Dally, W.J.: Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In: 4th International Conference on Learning Representations (ICLR) (2016)

[16] Han, S., Pool, J., Tran, J., Dally, W.: Learning both weights and connections for efficient neural network. In: Advances in Neural Information Processing Systems. vol. 28. Curran Associates, Inc. (2015)

[17] 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. pp. 770–778 (2016)

[18] He, Y., Zhang, X., Sun, J.: Channel pruning for accelerating very deep neural networks. In: Proceedings of the IEEE international conference on computer vision. pp. 1389–1397 (2017)

[19] Hinton, G.E., Vinyals, O., Dean, J.: Distilling the knowledge in a neural network. ArXiv **abs/1503.02531** (2015)

[20] Hoefler, T., Alistarh, D., Ben-Nun, T., Dryden, N., Peste, A.: Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks (2021)

[21] Horowitz, M.: 1.1 computing's energy problem (and what we can do about it). In: 2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC). pp. 10–14 (2014)

[22] Kirchhoffer, H., Haase, P., Samek, W., Müller, K., Rezazadegan-Tavakoli, H., Cricri, F., Aksu, E., Hannuksela, M.M., Jiang, W., Wang, W., Liu, S., Jain, S., Hamidi-Rad, S., Racapé, F., Bailer, W.: Overview of the neural network compression and representation (nnr) standard. IEEE Transaction on Circuits and System for Video Technology pp. 1–14 (2021)

[23] Kohlbrenner, M., Bauer, A., Nakajima, S., Binder, A., Samek, W., Lapuschkin, S.: Towards best practice in explaining neural network decisions with lrp. In: 2020 International Joint Conference on Neural Networks (IJCNN). pp. 1–7. IEEE (2020)

[24] Krizhevsky, A.: Learning Multiple Layers of Features from Tiny Images (Apr 2009)

[25] LeCun, Y., Denker, J.S., Solla, S.A.: Optimal brain damage. In: Advances in neural information processing systems. pp. 598–605 (1990)

[26] Marban, A., Becking, D., Wiedemann, S., Samek, W.: Learning sparse & ternary neural networks with entropy-constrained trained ternarization (ec2t). In: The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops. pp. 3105–3113 (June 2020)

[27] Montavon, G., Binder, A., Lapuschkin, S., Samek, W., Müller, K.R.: Layer-wise relevance propagation: an overview. Explainable AI: interpreting, explaining and visualizing deep learning pp. 193–209 (2019)

[28] Montavon, G., Samek, W., Müller, K.R.: Methods for interpreting and understanding deep neural networks. Digital Signal Processing **73**, 1–15 (2018)

[29] NVIDIA Turing GPU Architecture - Graphics Reinvented. Tech. Rep. WP-09183-001_v01, NVIDIA Corporation (2018)

[30] Park, E., Ahn, J., Yoo, S.: Weighted-entropy-based quantization for deep neural networks. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 7197–7205 (2017)

[31] Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., Lerer, A.: Automatic differentiation in pytorch (2017)

[32] Sabih, M., Hannig, F., Teich, J.: Utilizing explainable AI for quantization and pruning of deep neural networks. CoRR **abs/2008.09072** (2020)

[33] Samek, W., Montavon, G., Lapuschkin, S., Anders, C.J., Müller, K.R.: Explaining deep neural networks and beyond: A review of methods and applications. Proceedings of the IEEE **109**(3), 247–278 (2021)

[34] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L.C.: Mobilenetv2: Inverted residuals and linear bottlenecks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 4510–4520 (2018)

[35] Schütt, K.T., Arbabzadah, F., Chmiela, S., Müller, K.R., Tkatchenko, A.: Quantum-chemical insights from deep tensor neural networks. Nature communications **8**(1), 1–8 (2017)

[36] Senior, A.W., Evans, R., Jumper, J., Kirkpatrick, J., Sifre, L., Green, T., Qin, C., Zídek, A., Nelson, A.W.R., Bridgland, A., Penedones, H., Petersen, S., Simonyan, K., Crossan, S., Kohli, P., Jones, D.T., Silver, D., Kavukcuoglu, K., Hassabis, D.: Improved protein structure prediction using potentials from deep learning. Nature **577**(7792), 706–710 (2020)

[37] Shannon, C.E.: A mathematical theory of communication. The Bell System Technical Journal **27**(3), 379–423 (1948)

[38] Shrikumar, A., Greenside, P., Shcherbina, A., Kundaje, A.: Not just a black box: Learning important features through propagating activation differences. CoRR **abs/1605.01713** (2016)

[39] Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)

[40] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. The journal of machine learning research **15**(1), 1929–1958 (2014)

[41] Sze, V., Chen, Y., Yang, T., Emer, J.S.: Efficient processing of deep neural networks: A tutorial and survey. Proceedings of the IEEE **105**(12), 2295–2329 (2017)

[42] Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., Le, Q.V.: Mnasnet: Platform-aware neural architecture search for mobile. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 2820–2828 (2019)

[43] Warden, P., Situnayake, D.: TinyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-low-power Microcontrollers. O'Reilly Media (2020)

[44] Warden, P.: Speech commands: A dataset for limited-vocabulary speech recognition. CoRR **abs/1804.03209** (2018)

[45] Wiedemann, S., Kirchhoffer, H., Matlage, S., Haase, P., Marban, A., Marinc, T., Neumann, D., Nguyen, T., Schwarz, H., Wiegand, T., Marpe, D., Samek, W.: Deepcabac: A universal compression algorithm for deep neural networks. IEEE Journal of Selected Topics in Signal Processing **14**(4), 700–714 (2020)

[46] Wiedemann, S., Marban, A., Müller, K.R., Samek, W.: Entropy-constrained training of deep neural networks. In: 2019 International Joint Conference on Neural Networks (IJCNN). pp. 1–8 (2019)

[47] Wiedemann, S., Müller, K.R., Samek, W.: Compact and computationally efficient representation of deep neural networks. IEEE Transactions on Neural Networks and Learning Systems **31**(3), 772–785 (2020)

[48] Wiedemann, S., Shivapakash, S., Becking, D., Wiedemann, P., Samek, W., Gerfers, F., Wiegand, T.: FantastIC4: A Hardware-Software Co-Design Approach for Efficiently Running 4Bit-Compact Multilayer Perceptrons. IEEE Open Journal of Circuits and Systems **2**, 407–419 (2021)

[49] Yeom, S.K., Seegerer, P., Lapuschkin, S., Binder, A., Wiedemann, S., Müller, K.R., Samek, W.: Pruning by explaining: A novel criterion for deep neural network pruning. Pattern Recognition p. 107899 (2021)

[50] Zhou, S., Ni, Z., Zhou, X., Wen, H., Wu, Y., Zou, Y.: Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. CoRR **abs/1606.06160** (2016)

[51] Zhu, C., Han, S., Mao, H., Dally, W.J.: Trained ternary quantization. In: International Conference on Learning Representations (ICLR) (2017)