SYQ: Learning Symmetric Quantization For Efficient Deep Neural Networks

Julian Faraone* Nicholas Fraser# Michaela Blott# Philip H.W. Leong*
The University of Sydney*
Xilinx Research Labs#
(julian.faraone, philip.leong)@sydney.edu.au (nfraser, mblott)@xilinx.com

Abstract

Inference for state-of-the-art deep neural networks is computationally expensive, making them difficult to deploy on constrained hardware environments. An efficient way to reduce this complexity is to quantize the weight parameters and/or activations during training by approximating their distributions with a limited entry codebook. For very low-precisions, such as binary or ternary networks with 1-8-bit activations, the information loss from quantization leads to significant accuracy degradation due to large gradient mismatches between the forward and backward functions. In this paper, we introduce a quantization method to reduce this loss by learning a symmetric codebook for particular weight subgroups. These subgroups are determined based on their locality in the weight matrix, such that the hardware simplicity of the low-precision representations is preserved. Empirically, we show that symmetric quantization can substantially improve accuracy for networks with extremely low-precision weights and activations. We also demonstrate that this representation imposes minimal or no hardware implications to more coarse-grained approaches. Source code is available at https://www.github.com/julianfaraone/SYQ.

1. Introduction

Deep Neural Networks (DNNs) have produced state-of-the-art results in applications such as computer vision [17], natural language processing [4] and object detection [31]. As their size continues to grow to improve prediction capabilities, their memory and computational requirements also scales, making them increasingly difficult to deploy on embedded systems. For example, [17] achieved state-of-art-results on the ImageNet challenge using AlexNet which required 240MB of storage and 1.45 billion operations to compute inference per image. Several methods of compression [12], quantization [3] and dimensionality reduction [25] have been applied to reduce these demands, with promising results. This demonstrates the over-parametrization and redundancies in DNNs and poses an opportunity for utilizing regularization to make their representations more amenable to hardware implementations.

In particular, low-precision neural networks reduce both memory and computational requirements whilst achieving accuracies comparable to floating point [10]. For extremely low-precisions, such as binary and/or ternary weight representations and 1-8 bits for activations, most of the multiply-accumulate (MAC) operations can be replaced by simple bitwise operations. This translates to massive reductions in storage requirements and spatial complexity in hardware. Additionally, large power savings and speed gains are achieved when networks can fit in on-chip memory. The issue is that a large reduction in precision, leads to large information loss which incurs significant accuracy degradation, especially for complex datasets such as ImageNet [26]. Ideally, we can train networks which have both high prediction capabilities and minimal computational complexity.

DNN training is an iterative process which has a feed-forward path to compute the output and a backpropagation path to calculate gradients and update its parameters for learning. Low-precision networks involve having a set of full-precision weights which are quantized before computing inference. As the quantization functions are piecewise and constant, the gradients of quantized weights are calculated and applied to update their corresponding full-precision weights. Similarly, derivatives of quantized activations are calculated by using a non-constant differentiable approximation function. This type of training was first proposed as the Straight Through Estimator (STE) [1] which suggested the use of a nonzero derivative approximation to functions which are non-differentiable or have zero derivatives everywhere. The problem is that without an accurate estimator for weights and activations, there exists a significant gradient mismatch which impinges on learning. Seemingly, as discussed in [22], activations are more robust to quantization than weights for image classification problems due to weight reuse in Convolutional (CONV) layers affecting multiple operations. To overcome this, methods such as increasing the weight codebook by applying a scaling co-

efficient to all weights in a layer, provides better approximations for weight distributions and greater model capacity [19]. This is computationally inexpensive and can be represented as multiplying each weight layer's matrix by a diagonal scalar matrix which only requires storage of one value. Applying fine-grained scaling coefficients has also been shown to improve accuracy by increasing model capacity [21], [24]. The problem with all of these fine-grained approaches is either large storage requirements for the scaling coefficients or high computational complexity due to irregular codebook indices. In this paper we present Learning Symmetric Quantization (SYQ), a method to design binary/ternary networks with fine-grained scaling coefficients which preserve these complexities. We do this by learning a symmetric weight codebook via gradient-based optimizations which enables a minimally-sized square diagonal scalar matrix representation. To reduce the large information loss from CONV layer quantization, we use a more fine-grained pixel/row-wise scaling approach, rather than layer-wise scaling in Fully-Connected (FC) layers. In the process, we significantly close the accuracy gap for low-precision networks to their floating point counterpart, whilst preserving their efficient computational structures. Our work makes the following contributions:

- Our approach significantly improves the ability of convolutional weights to learn low-precision representations. This is useful as most layers in modern network architectures consist of convolutions which are typically the least redundant layers.
- The proposed method reduces the computational complexity of traditional fine-grained low-precision scaling and imposes minimal hardware costs to layer-wise scaling.
- On state-of-the-art networks such as AlexNet, ResNet and VGG, our method is empirically shown to improve accuracy for 1-2 bit weights and 2-8 bit activations.

2. Related Work

Most methods for training low-precision DNNs maintain a set of full precision weights that are deterministically or stochastically quantized during forward or backward propagation. Gradient updates computed with the quantized weights are then applied to the full precision weights [5], [15], [20]. To produce state-of-the-art results on larger models, [24] proposed scaling the quantized weights by the expectation of real-valued weights to recover the dynamic range of each layer. [19] also implemented a similar technique for ternary networks and optimised a non-zero quantization threshold as a function of the weight expectation. Other gradient-based optimization methods for the scaling coefficient have been introduced [34]. Other methods of quantization have also been implemented, i.e. re-training networks using incremental weight subgrouping to produce no accuracy loss for 5 bit weights [32]. Multiple binarizations and a scaling layer were described in [28] to improve accuracy and binarize the last layer. Logarithmic data representations were used to approximate the non-uniform distribution of the weights, activations and gradients down to 3-bits with negligible accuracy loss [22]. Activations quantization has also been investigated with frameworks created for varying activation bitwidths [33] and both weights and activations [23]. Improving the network learnability under low-precision weights and activations was analysed in [2]. More fine-grained approaches of quantization have effectively clustered weights or grouped filters together and quantize differently based on their statistical distributions [6], [21] . Increasing model capacity by applying scaling coefficients to positive and negative values separately was proposed in [34]. Furthermore, sparse representations were used as regularization to make networks more amenable to hardware [7]. Also, many low-precision DNN hardware implementations have been published [30], [11]. For example, FINN [8], [29] demonstrated the performance gains of being able to store all network weights in on-chip memory by implementing binarized neural networks on FPGAs.

3. Low-Precision Networks

In this section we discuss the motivations behind our work and fundamentals of low-precision neural networks.

3.1. Motivation

Each layer of a DNN computes dot products between weight parameters and its input values. We can represent the output of each hidden unit h, as:

$$h = g(\boldsymbol{w}^T \boldsymbol{x}) \tag{1}$$

where g is an element-wise nonlinear activation function, $\boldsymbol{x} \in \mathbb{R}^{i.w.h}$ is the input vector, and $\boldsymbol{w} \in \mathbb{R}^{i.w.h}$ provides the weight vector of a linear transformation. This computation is repeated throughout the network, therefore overall model complexity is dependant on its structure. As modern networks continue to get deeper/wider, model complexity becomes problematic for their applicability on constrained hardware environments. A solution is to efficiently quantize both weights and activations to very low-precisions (1-8 bits) with negligible or no accuracy loss. In doing so, the arithmetic operations are greatly simplified, reducing both computational and resource complexity. In the binary/ternary weight case, MACs are replaced by bit operations. For example, Figure 1 shows average resource usage on Field Programmable Gate Array (FPGA) hardware to implement a MAC operation under different precisions, which scales quadratically with the multiplier size at $\mathcal{O}(k^2)$

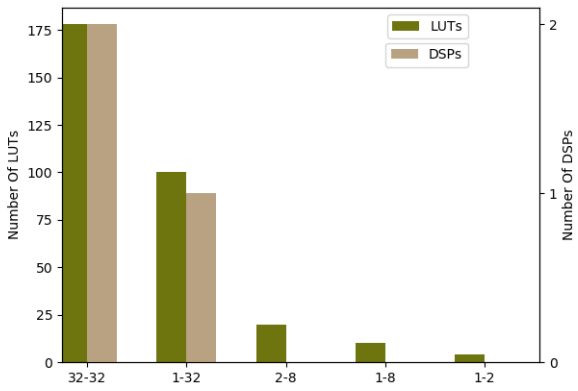

Figure 1. The average cost per MAC operation on an FPGA device for different bitwidths (weight-activation)

where k is the number of bits[1]. As shown, no high precision multipliers (known as DSPs on an FPGA) are required for precisions less than or equal to ternary weights and 8-bit activations. Furthermore, the logic element (known as LUTs on an FPGA) requirement reduces proportionally with both weight and activation precisions. Additionally, the storage requirements for both weights and activations is reduced by $8 - 32\times$. This significantly improves the network's ability to fit in on-chip memory and constrained hardware environments, and broadens the applicability of DNNs.

For a CONV layer, all weights are typically represented as a tensor $\mathbf{W}_l \in \mathbb{R}^{K\times K\times I\times N}$ where K is the filter size, I is the number of input feature maps and N, the number of output feature maps. In low-precision networks, each weight layer l can typically be represented by a diagonal scalar matrix $\boldsymbol{\alpha_l}$ multiplied by quantized weight matrix $\boldsymbol{Q}_l$ and ideally $\boldsymbol{W}_l \approx \boldsymbol{\alpha_l Q}_l$. Also, the activation function g can be approximated using a piecewise constant activation function G. In our proposed method, we observe that by ensuring quantization levels for $\boldsymbol{W}$ are symmetric around zero, we can construct efficient square diagonal matrix representations of $\boldsymbol{\alpha_l}$, which enable fine-grained quantization whilst having minimal memory requirements (of size K or K^2). This translates to a reduction in overall model complexity and high prediction capabilities. Although, we restrict ourselves by structured matrices and low-precision weights and activations, the network efficiently captures information through our gradient-based symmetric quantizer which learns the diagonal elements of $\boldsymbol{\alpha_l}$ during training.

3.2. Weight Quantization

For low-precision DNNs, the distribution of full precision weight matrices for each layer $\boldsymbol{W}_l$ are approximated by a function f, resulting in a quantized weight matrix $\boldsymbol{Q}_l$:

$$Q_{l_{i,j}} = f(W_l)_{i,j} \tag{2}$$

[1]Results are obtained from instantiating MAC modules using Vivado

for $W_{l_{i,j}} \in \mathbb{R}$ and $Q_{l_{i,j}} \in \mathbb{C}$. The codebook $\mathbb{C} = \{c_1, c_2, ..., c_r\}$ is a set of all possible values for $Q_{l_{i,j}}$ where $c_i \in \mathbb{R}$ and $i \in \mathbb{R}^+$ represent each codebook value and index respectively. For example, binary and ternary weight spaces have $\mathbb{C} = \{-1, +1\}$ and $\mathbb{C} = \{-1, 0, +1\}$ respectively. Efficient functions for binarizing and ternarizing weight parameters have been proposed as piecewise constant functions in [19], such that:

$$\boldsymbol{Q}_l = sign(\boldsymbol{W}_l) \odot \boldsymbol{M}_l \tag{3}$$

with,

$$M_{l_{i,j}} = \begin{cases} 1 & \text{if} \quad |W_{l_{i,j}}| \geq \eta_l \\ 0 & \text{if} \quad -\eta_l < W_{l_{i,j}} < \eta_l \end{cases} \tag{4}$$

where $\boldsymbol{M}$ represents a masking matrix, η is the quantization threshold hyperparameter. $\eta = 0$ for binary networks and in our work we set $\eta = 0.05 \times max(|W_l|)$ for ternary networks as in [34]. The issue with discretization of the weights, is that it leads to the vanishing gradients problem [1]. To overcome this, an STE is defined to replace the zero derivatives from the piecewise constant function in (3), by a non-zero surrogate derivative [15]. During training $\boldsymbol{Q}_l$ is used for inference and backpropagation, and the corresponding elements in $\boldsymbol{W}_l$ are updated based on these gradients. Hence the STE is defined as:

$$\frac{\partial \hat{E}}{\partial W_{l_{i,j}}} = \frac{\partial \hat{E}}{\partial Q_{l_{i,j}}} \tag{5}$$

where $\hat{E}$ is the error function for a network without scaling coefficients. After training, the full precision weights are discarded and we require only the quantized weights for deployment. Whilst these methods greatly reduce computational complexity by eliminating floating point MACs, they increase the difficulty of learning.

3.3. Scaling

The introduction of scaling coefficients improves learning capabilities by providing greater model capacity and compensating for the large information loss due to binary/ternary quantization. Scaling discrete weight representations requires multiplying all $Q_{l_{i,j}}$ by positive scaling coefficients $\alpha \in \mathbb{R}^+$. We want to find optimal scaling coefficients for each layer, α_l, which minimize our error function:

$$\alpha_l^* = \underset{\alpha}{\operatorname{argmin}}\, E(\alpha, \boldsymbol{Q}) \quad s.t. \quad \alpha \geq 0,\ Q_{l_{i,j}} \in \mathbb{C} \tag{6}$$

with E representing the error function with scaling coefficients. Finding the optimal α_l is vital to reducing gradient mismatches in the forward and backward functions. It was proposed in [33] as the mean of absolute weight values for each layer:

$$\alpha_l = \frac{\|W_l\|_1}{Z_l} \tag{7}$$

where Z_l is the total number of layer weights. The codebook for each layer after scaling in (7) is symmetric: $\hat{\mathbb{C}}_l = \{-\alpha_l, +\alpha_l\}$ and the scalars become per-layer learning rate multipliers. Additionally, the STE in (8) reduces the gradient mismatch from (5) by including information from the full precision weights:

$$\frac{\partial E}{\partial W_{l_{i,j}}} = \frac{\partial E}{\partial Q_{l_{i,j}}} = \alpha_l \frac{\partial \hat{E}}{\partial Q_{l_{i,j}}} \tag{8}$$

Gradient-based optimizations for scaling coefficients were also introduced in [34] which applied different scaling coefficients for positive and negative $Q_{l_{i,j}}$ to improve model capacity and accuracies. These are updated during backpropagation using gradients:

$$\frac{\partial E}{\partial \alpha_l^p} = \sum_{i,j \in S_l^p} \frac{\partial E}{\partial W_{l_{i,j}}}, \; \frac{\partial E}{\partial \alpha_l^n} = \sum_{i,j \in S_l^n} \frac{\partial E}{\partial W_{l_{i,j}}} \tag{9}$$

where initially $\alpha_{l_0}^p, \alpha_{l_0}^n = 1$ and S_l is the codebook indices for each layer, i.e. $S_l^p = \{i, j | W_{l_{i,j}} \geq \eta\}$ and $S_l^n = \{i, j | W_{l_{i,j}} \leq -\eta\}$. This allows each layer's codebook values to be asymmetric around zero, such that $\hat{\mathbb{C}}_l = \{-\alpha_l^n, +\alpha_l^p\}$. The codebook indices are then highly irregular and unordered which increases computational complexity as the matrices cannot be easily decomposed. Rather we have to check the sign of every element before computation, leading to extra branching instructions for conventional computing platforms such as CPUs/GPUs and additional logic for custom hardware. The difficulty of designing low-precision networks which have both high learning capabilities and computational efficiency can be solved by learning a symmetric codebook during training and exploiting structured matrix representations.

4. SYQ Structural Representations

We now propose matrix representations of SYQ by partitioning the quantization into weight subgroups. Diagonal matrix representations consist of mainly zeros and have non-zero entries along the main diagonal. For a matrix $\boldsymbol{D}$ to be diagonal, $\boldsymbol{D} = 0$ if $D_{i,j} = 0 \; \forall \; i \neq j$, and square if $\boldsymbol{D} \in \mathbb{R}^{m \times m}$. A square diagonal matrix consisting of all equal main diagonal entries is a scalar matrix. A diagonal matrix $\boldsymbol{\alpha_l}$ is defined by the vector $\boldsymbol{\alpha_l} = [\alpha_l^1, ..., \alpha_l^m]$:

$$\boldsymbol{\alpha} = diag(\boldsymbol{\alpha}) := \begin{bmatrix} \alpha^1 & 0 & .. & 0 & 0 \\ 0 & \alpha^2 & .. & : & 0 \\ : & : & .. & \alpha^{m-1} & : \\ 0 & 0 & .. & 0 & \alpha^m \end{bmatrix}$$

Diagonal matrix multiplication is very computationally efficient as it can be easily decomposed and only the scalar vector requires storage.

4.1. Layers

CONV and FC layers have differing computational requirements and sensitivities to network redundancies. CONV weights are reused many times across the input feature map whereas FC weights are used only once per image. Hence, the quantization error of each weight in a CONV layer impacts the dot products across the entire input feature map volume rather than just once for FC weights. Thus, a fine-grained approach to CONV layers is effective at compensating for this error. Quantized CONV weights are represented as a tensor $\mathbf{Q}_l \in \mathbb{R}^Z$ with $Z = K \times K \times I \times N$. As typically $I, N \gg K$, it is optimal to have a diagonal scalar of size $K \times K$ or even $K^2 \times K^2$ as only small scalar vectors are required for storage. By reshaping the tensor $\mathbf{Q}_l$, we form a matrix $\boldsymbol{Q}_l \in \mathbb{R}^{\hat{Z}}$ where $\hat{Z} = K^2 \times (IN)$ or $\hat{Z} = K \times (INK)$ and represent our scalar matrix multiplication as $diag(\boldsymbol{\alpha_l})\boldsymbol{Q}_l^T$ with the square diagonal matrix, $diag(\boldsymbol{\alpha_l}) \in \mathbb{R}^{K^2 \times K^2}$ or $diag(\boldsymbol{\alpha_l}) \in \mathbb{R}^{K \times K}$ respectively. FC layers are represented as a matrix $\boldsymbol{Q}_l \in \mathbb{R}^{L \times H}$ where H is the number of hidden nodes and L the activation neurons. As FC layers are more robust to quantization, one learnable scaling coefficient (layer-wise) for the FC layer can sufficiently approximate the distribution and also can be represented with scalar matrix computation. All elements in $\boldsymbol{\alpha_l}$ are then equal and we only require storage of one value.

4.2. Subgroups

More fine-grained quantization can improve approximations of the statistical distributions of weights. We implement pixel-wise scaling for CONV layers which involves grouping all spatially equivalent pixels along the $I \times N$ dimension. This results in different values for all the main diagonal elements in $diag(\boldsymbol{\alpha}) \in \mathbb{R}^{K^2 \times K^2}$. With this representation, we can still decompose the matrix computation along each pixel dimension and exploit the parallel nature of convolutions as shown in Figure 2. We do this by creating subgroups $1 \leq i \leq K^2$ with codebook indices $S_l^i = \{j | W_{l_{i,j}}\}$. Other granularities such as row-wise scaling involve grouping all pixels along a row or column ($I \times N \times K$), resulting in $S_l^j = S_l^i \cup S_l^{i+1} ... \cup S_l^K$ where $1 \leq j \leq K$ (as illustrated in Figure 2) and also layer-wise scaling: $S_l = S_l^i \cup S_l^{i+1} ... \cup S_l^{K^2}$. Different granularities affect both accuracy and computation as further explored in Sections 6 & 7.

5. SYQ Training

In this section, we now describe the methodology to efficiently train SYQ networks.

5.1. Symmetric Quantizer

When training low-precision inference networks, the aim is to have the smallest possible codebook. Typically, as

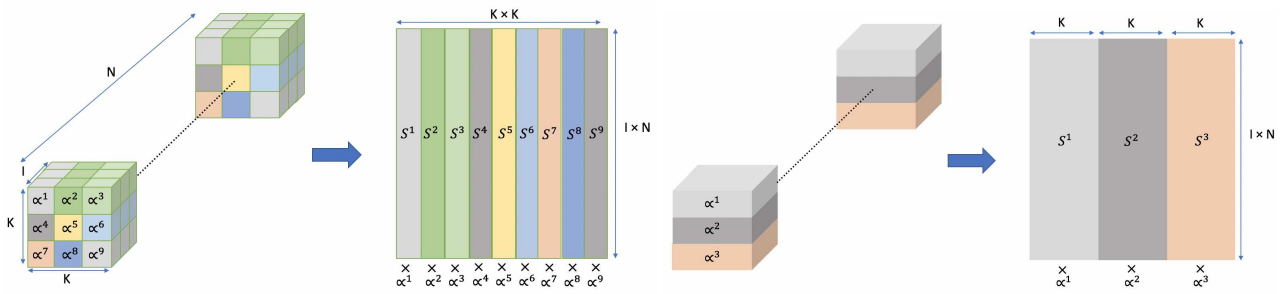

Figure 2. Computational structure of pixel-wise (Left) and row-wise (Right) subgrouping of a CONV layer (K, I = 3). The tensors represent the weight layer structure during training and the matrices represent the matrix decomposition for deployment.

the codebook size increases, a network will approach full-precision performance but increase hardware cost. However, there are certain codebook representations which are significantly more hardware friendly than others and won't necessarily impose any hardware costs. Given a codebook C, and the nonzero codebooks $C^p = \{c_i | c_i > 0\}$ and $C^n = \{c_j | c_j < 0\}$, a quantizer is denoted as symmetric if:

$$\forall c_i \in C^p, \quad \exists\, |c_j| \in C^n \quad \text{where} \quad c_i = |c_j| \tag{10}$$

Learning this type of codebook requires updating one scaling coefficient during training for two bi-polar codebook values. The gradient of each scaling coefficient for each subgroup becomes:

$$\frac{\partial E}{\partial \alpha_l^i} = \sum_{j \in S_l^i} \frac{\partial E}{\partial W_{l_{i,j}}} \tag{11}$$

When computing binary/ternary weight representations followed by a scale, it is ideal to have a codebook which is symmetric around zero, as the codebook storage requirements are almost halved. This is because only the absolute value of the two symmetric values needs to be stored. Additionally, codebook indices become highly regular and ordered for the scalar multiply which greatly reduces computational complexity. The nature of symmetric quantization enables the opportunity to implement fine-grained quantization (pixel/row-wise) whilst maintaining the scalar matrix multiplication structure used in layer-wise scaling. This is also advantageous as the scaling coefficients become fine-grained adaptive learning rate multipliers for each pixel/row in a CONV layer, i.e. the STE becomes:

$$\frac{\partial E}{\partial W_{l_{i,j}}} = \frac{\partial E}{\partial Q_{l_{i,j}}} = \alpha_l^i \frac{\partial \hat{E}}{\partial Q_{l_{i,j}}} \tag{12}$$

As the use of scaling coefficients can more accurately approximate subgroups and are gradient-based, the gradient mismatch is significantly reduced for weight quantization which enhances network learning.

5.2. Initialization

The solution to non-convex optimizations such as gradient descent depend heavily on parameter initialization to avoid vanishing or exploding activations/gradients and ensure network convergence [9]. For low-precision networks, excessive gradient mismatches between the forward and backward functions must be minimized, otherwise the gradients will not propagate well. To deal with this concern, the scaling coefficients coefficients are initialized as the mean of full precision weights in it's corresponding subgroup. For example, the scaling coefficient in pixel-wise scaling is:

$$\alpha_{l_0}^i = \frac{\sum_{j \in S_l^i} |W_{l_{i,j}}|}{I \times N} \tag{13}$$

Layer-wise scaling in FC layers has α_{l_0} as the mean of all layer weights. By incorporating information from the full precision weights, we aim to reduce the mismatch initially and the scaling coefficients are then optimized during back-propagation.

5.3. Activations Quantization

Our forward path approximation to g in (1) uniformly quantizes a real number $x \in [0, M]$ to a k-bit number:

$$G(x) = \frac{1}{2^f} floor((2^f)x + \frac{1}{2}) \tag{14}$$

where $floor$ represents the round down operation and M is the upper bound. M itself is bounded by its arbitrary unsigned two's complement fixed point representation where f is the number of fractional bits and $M = 2^{k-f} - 2^{-f}$. Uniform quantization translates to a reduction in hardware implementation complexity. To achieve this, we use the following STE for the activations:

$$\frac{\partial E}{\partial x} = \frac{\partial E}{\partial G} \tag{15}$$

Differences in the forward and backward activation functions create a gradient mismatch which can result in unstable and inefficient learning. To minimize this issue, we adjust M as a hyperparameter. The overall SYQ training process is summarized in Algorithm 1.

Algorithm 1 SYQ Training Summary For DNNs.

Initialize: Set subgrouping granularity for S_l^i and set $\alpha_{l_0}^i$.
Inputs: Minibatch of inputs & targets (I, Y), Error function $E(Y, \hat{Y})$, current weights $\boldsymbol{W_t}$ and learning rate, γ_t
Outputs: Updated $\boldsymbol{W_{t+1}}$, $\boldsymbol{\alpha_{t+1}}$ and γ_{t+1}

SYQ Forward:
for l=1 to L **do**
  $\boldsymbol{Q}_l = sign(\boldsymbol{W}_l) \odot \boldsymbol{M}_l$ **with** η, using (3) & (4)
  for ith subgroup in lth layer **do**
    Apply α_l^i to S_l^i
  end for
end for
$\hat{Y}$ = **SYQForward** $(I, Y, \boldsymbol{Q}_l, \boldsymbol{\alpha}_l)$ using (14)

SYQ Backward:
$\frac{\partial \hat{E}}{\partial \boldsymbol{Q}_l}$ = **WeightBackward**$(\boldsymbol{Q}_l, \boldsymbol{\alpha}_l, \frac{\partial \hat{E}}{\partial \hat{Y}})$ using (12) & (15)
$\frac{\partial \hat{E}}{\partial \boldsymbol{\alpha}_l}$ = **ScalarBackward**$(\frac{\partial \hat{E}}{\partial \boldsymbol{Q}_l}, \boldsymbol{\alpha}_l, \frac{\partial \hat{E}}{\partial \hat{Y}})$ using (11)
$\boldsymbol{W_{t+1}}$ = **UpdateWeights**$(\boldsymbol{W_t}, \frac{\partial \hat{E}}{\partial \boldsymbol{Q}_l}, \gamma)$
$\boldsymbol{\alpha_{t+1}}$ = **UpdateScalars**$(\boldsymbol{\alpha_t}, \frac{\partial \hat{E}}{\partial \boldsymbol{\alpha}_l}, \gamma)$
γ_{t+1} = **UpdateLearningRate**(γ_t, t)

6. Experiments

To demonstrate the versatility of SYQ, we applied it to several state-of-the-art benchmark models, all with different network topologies. We use binary/ternary weights and varying activation bitwidths for classification of the large-scale ImageNet dataset. The ILSVRC-2012 ImageNet is a natural high resolution visual classification dataset consisting of 1000 classes, 1.28 million training images and 50K validation images. Inputs are resized to 256×256 before being randomly cropped to 224×224. We report our single-crop evaluation results using Top-1 and Top-5 accuracy.

6.1. Networks

We compare our results to the full precision baseline and benchmark reference model accuracies in Table 1[2], showing that SYQ training achieves similar accuracy to floating point. This suggests the noise induced from replacing floating point weight layers with SYQ versions, provides effective regularization during training. An AlexNet [18] variant is implemented which eliminates dropout and includes batch normalization [16]. A mini batch size of 64 is used, L2 weight decay of 5e-6, and our learning rate is initially 1e-4 with step decays of scale factor 0.2. For ResNet [13], we test on the 18, 34 and 50 layer variations. Our batch size is 128, learning rate is initially 1e-3 with step decay of factor 0.2. We also test on a variant of VGG-16 [27], using model-A in [14] with the spp layer replaced by a max pool and only 3 CONV layers rather than 5 for input size blocks of 56, 28 and 14, as in [2]. Batch sizes are set to 32 and our learning rate is initially 1e-4 with a step decay of factor 0.2. The VGG and ResNet models were initialized from floating point baseline weights. Full-precision weights are used for the first and last layer. All other CONV layers are quantized with SYQ pixel-wise scaling, FC layers with layer-wise scaling and the activations of all layers using (14).

[2]Our ResNet and AlexNet reference results are obtained from https://github.com/facebook/fb.resnet.torch and https://github.com/BVLC/caffe, respectively

Table 1. Summary of Results for 8-bit activations and binary (1-8) and ternary (2-8) weights

Model		1-8	2-8	Baseline	Reference
AlexNet	Top-1	**56.6**	**58.1**	56.6	57.1
	Top-5	**79.4**	**80.8**	80.2	80.2
VGG	Top-1	**66.2**	**68.7**	69.4	-
	Top-5	**87.0**	**88.5**	89.1	-
ResNet-18	Top-1	**62.9**	**67.7**	69.1	69.6
	Top-5	**84.6**	**87.8**	89.0	89.2
ResNet-34	Top-1	**67.0**	**70.8**	71.3	73.3
	Top-5	**87.6**	**89.8**	89.1	91.3
ResNet-50	Top-1	**70.6**	**72.3**	76.0	76.0
	Top-5	**89.6**	**90.9**	93.0	93.0

Table 2. AlexNet accuracy differences between using row/layer-wise and pixel-wise symmetric quantization

		Row-wise		Layer-wise	
Weights	**Act.**	**Top-1**	**Top-5**	**Top-1**	**Top-5**
1	2	-0.7	-0.5	-1.4	-2.2
1	8	-0.1	-0.3	-0.4	-2.2
2	2	+0.1	-0.0	-1.3	-1.5
2	8	-0.1	-0.1	-1.9	-1.7

6.2. Changing Granularity Via Weight Subgroups

Weight subgroups can be arbitrarily designed for a given hardware application. Table 2 shows accuracy differences between using row/layer-wise vs pixel-wise scaling on AlexNet and suggests pixel-wise and row-wise are marginally different, especially for higher precisions, but both are considerably more accurate than layer-wise. This demonstrates the effectiveness of fine-grained quantization of CONV layers over layer-wise and promotes the exploration for efficient representations of scalar computation. It also shows the effectiveness of row-wise quantization as it typically incurs a smaller memory requirement with a small accuracy drop, for a significant gain in the potential parallelism of the network.

6.3. Comparisons To Previous Work

We compare SYQ explicitly using AlexNet, ResNet-18 and ResNet-50 in Tables 3, 4 & 5 as they've been extensively studied in the literature. Our ternary results with 8 bit activations (2w-8act) improves on the state-of-the-art for all

Table 3. Comparison to previously published AlexNet results

Model	Weights	Act.	Top-1	Top-5
DoReFa-Net [33]	1	2	49.8	-
QNN [15]	1	2	51.0	73.7
HWGQ [2]	1	2	52.7	76.3
SYQ	**1**	**2**	**55.4**	**78.6**
DoReFa-Net [33]	1	4	53.0	-
SYQ	**1**	**4**	**56.2**	**79.4**
BWN [24]	1	32	56.8	79.4
SYQ	**1**	**8**	**56.6**	**79.4**
SYQ	**2**	**2**	**55.8**	**79.2**
FGQ [21]	2	8	49.04	-
TTQ [34]	2	32	57.5	79.7
SYQ	**2**	**8**	**58.1**	**80.8**

Table 4. Comparison to previously published ResNet-18 results

Model	Weights	Act.	Top-1	Top-5
BWN [24]	1	32	60.8	83.0
SYQ	**1**	**8**	**62.9**	**84.6**
TWN [19]	2	32	65.3	86.2
INQ [32]	2	32	66.0	87.1
TTQ [34]	2	32	66.6	87.2
SYQ	**2**	**8**	**67.7**	**87.8**

Table 5. Comparison to previously published ResNet-50 results

Model	Weights	Act.	Top-1	Top-5
HWGQ [2]	1	2	64.6	85.9
SYQ	**1**	**4**	**68.8**	**88.7**
SYQ	**1**	**8**	**70.6**	**89.6**
FGQ [21]	2	4	68.4	-
SYQ	**2**	**4**	**70.9**	**90.2**
FGQ [21]	2	8	70.8	-
SYQ	**2**	**8**	**72.3**	**90.9**

three networks. Our 2w-4act for ResNet-50 also improves on the state-of-the-art FGQ. This is also the case for binary weights, such as 1w-8act ResNet-18 and AlexNet with 1w-2/4act. For extremely low 1w-2act representations, SYQ also has a 2.7% increase in Top-1 accuracy over the state-of-the-art HWGQ. This demonstrates SYQ's superiority for producing high accuracy. Additionally, it shows that multiple learnable scaling coefficients effectively reduce the gradient mismatch in the forward and backward paths, translating to efficient learning under low-precision constraints.

6.4. Varying Activation Bitwidth

The most important result is that SYQ efficiently quantizes networks with low-precisions for both weights and activations. From Figure 3, we can see that lowering the precision of the activations does not severely alter the training curve, suggesting that the gradient information from pixel-wise scaling coefficients in SYQ compensates well

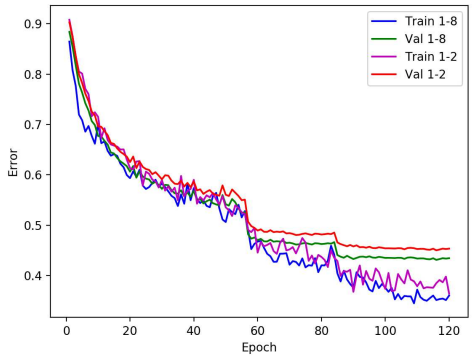

Figure 3. Top-1 training and validation error for binary AlexNet with varying activation precisions

Table 6. Number of scaling coefficients and operations per layer, for different techniques

Method	Scalars	Ops
Layer (DoReFa)	1	P
Row (SYQ)	K	P
Pixel (SYQ)	K^2	P
Asymmetric (TTQ)	2	$P+Z$
Grouping (FGQ)	$K^2N/4$	P
Channel (HWGQ/BWN)	N	P

for the loss of information. However, when quantizing down to 2-bits, the training error curve does become more volatile, demonstrating instabilities in network learning. We also report the classification accuracies for varying activations and bitwidths on AlexNet and ResNet-50 in Tables 3 & 5, which shows that there is minimal discrepancy from the full-precision networks with as low as 4-bit activations. These results are extremely promising and have strong implications for specialized hardware implementations of low-power DNNs.

7. Hardware Implications

In this section we discuss the computational implications of different scaling operations and present a design for specialized hardware implementations.

7.1. Computational and Memory Complexity

Considering a CONV layer with Ops, $P = K \times K \times I \times N \times F \times F$, where F is the IFM dimension. The layer-wise scaling, as in DoReFa-Net, requires one scaling coefficient per P operations. For channel-wise scaling in HWGQ and BWN, it requires N scaling coefficients as there is one per output feature map, where typically $N \gg 1$. TTQ implements asymmetric layer-wise quantization which requires two scaling coefficients per layer and $P + Z$ operations as we add a branching operation for each weight due to irregular codebook indices, as described in Section 3.3. FGQ uses pixel-wise scaling for every 4 filters, whereas SYQ uses

pixel-wise scaling per N filters, hence it requires $K^2N/4$ scaling coefficients and P operations. For pixel-wise SYQ scaling, K^2 scaling coefficients and P operations are required, where $K = 3$ for most CONV layers in modern networks. For row-wise SYQ scaling it requires K scaling coefficients and P operations. These results are displayed in Table 6, demonstrating the benefits of maintaining a diagonal representation for the scalar matrix multiplication of each layer as we either improve computational or memory complexity against all other fine-grained methods. Another key benefit of SYQ is its amenability to highly parallel processors.

7.2. Architectural Design

For the CONV layer, the operations are a sum of dot products between the input and kernel filter. In order to reduce compute complexity, we increase the number of operations in each dot product, while significantly decreasing the complexity of each operation. For example, the size of the input vector, in the calculation of each dot product is: $L_v = K^2I$. The number of operations is $Op^L_{mul} = L_v$ for multiplies and $Op^L_{add} = L_v - 1$ for additions. Given that we have a limited codebook for our weights, we can break it into sub-dot products where we apply the scaling factor, α^i, after we have computed the sub-dot product for that set of symmetrically constrained weights. For pixel-wise quantization, the total multiplies becomes $Op^P_{mul} = L_v + K^2$ and the total adds become $Op^P_{add} = K^2(L_v/K^2 - 1) + (K^2 - 1) = L_v - 1$. However, the first term in each of these calculations can be done at significantly lower precision. For multiplies this means a binary or ternary multiple - which can often be implemented as a bit-flip. To compute this in specialized hardware, for layer-wise scaling, we have a parallel MAC tree which consists of a multiply of an input and binary/ternary number (represented as a dot) followed by an adder tree to sum up the outputs. Outputs of these are fed into a multiplier to compute the scale, followed by an accumulator to store the outputs before being fed into the activation function. This architecture is shown in Figure 4. For every hardware block of this type, our per-pixel/row scaling only requires one additional ring counter which stores scaling coefficients and shifts the input to the scaling multiplier through an index counter as each row/pixel is finished computing which is computationally inexpensive. As in the equivalent layer-wise scaling architecture, we can still maintain one multiplier in hardware and only increase memory slightly to store the scaling coefficients. Table 7 shows the resource and performance estimates provided by Vivado HLS of the described hardware architecture for a target Xilinx ZU3 FPGA device at an estimated clock frequency of over 300 MHz. The main design is based on the MVTU described in FINN [29], with an extension to 2-bit activations and pixel-wise and row-wise

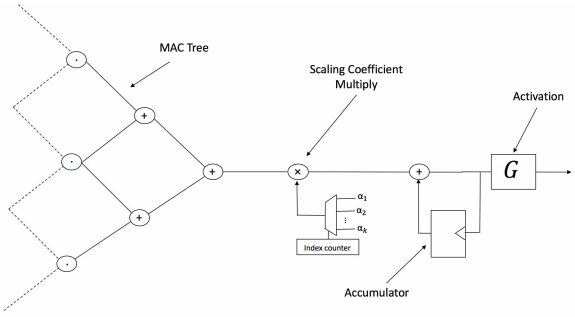

Figure 4. Hardware description of MAC for SYQ layers

Table 7. Resource Usage of a Matrix-Vector Processing Unit with Layer-wise and Pixel-wise Quantization for target Xilinx ZU3

Config	SIMD	PE	BRAMs	LUTs (k)	DSPs
Layer	32	32	64	29.8	4
Layer	64	32	64	56.5	4
Layer	32	64	64	58.9	4
SYQ(P)	32	32	64	29.4	36
SYQ(P)	64	32	64	56.1	36
SYQ(P)	32	64	64	57.7	68
ZU3	-	-	432	70.6	360

SYQ. The layer-wise baseline uses no multiplies, as these can absorb into quantization thresholds for activations [29]. The MVTU was configured for a convolution layer with $I = 384$, $N = 256$, $K = 3$, while scaling the size of the MAC tree (SIMD) and the number of parallel processors (PE). As shown, the BRAM (memory blocks on an FPGA (18k)) and LUT usage is almost identical, while the DSP usage increases proportionally with the number of parallel output channels which are processed. The increase in DSPs is not necessarily costly for the ZU3 as we are able to utilize more of the total available resources. Resource usage is only shown for pixel-wise SYQ, as row-wise only differed in LUT usage by less than 2%.

8. Conclusions

The problem of efficiently training large DNNs with low-precision weights and activations is considered. We propose learning symmetric quantization for DNNs in order to maximize network learning whilst minimizing hardware complexity. This was achieved by constraining the solution to low-precision representations and learning a diagonal scalar matrix using gradient-based optimizations for efficient computation. As a result, we reduce the computational requirements of fine-grained quantization and achieve state-of-the-art accuracies on modern benchmark networks.

Acknowledgements

This research was partly supported under the Australian Research Councils Linkage Projects funding scheme (project number LP130101034) and Zomojo Pty Ltd.

References

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

[2] Z. Cai, X. He, J. Sun, and N. Vasconcelos. Deep learning with low precision by half-wave gaussian quantization. *CoRR*, abs/1702.00953, 2017.

[3] W. Chen, J. T. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen. Compressing neural networks with the hashing trick. *CoRR*, abs/1504.04788, 2015.

[4] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. P. Kuksa. Natural language processing (almost) from scratch. *CoRR*, abs/1103.0398, 2011.

[5] M. Courbariaux, Y. Bengio, and J. David. Binaryconnect: Training deep neural networks with binary weights during propagations. *CoRR*, abs/1511.00363, 2015.

[6] Y. Duan, J. Lu, Z. Wang, J. Feng, and J. Zhou. Learning deep binary descriptor with multi-quantization. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[7] J. Faraone, N. J. Fraser, G. Gamberdella, M. Blott, and P. H. W. Leong. Compressing low precision deep neural networks using sparsity-induced regularization in ternary networks. *CoRR*, abs/1709.06262, 2017.

[8] N. J. Fraser, Y. Umuroglu, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers. Scaling binarized neural networks on reconfigurable logic. In *Proceedings of the 8th Workshop and 6th Workshop on Parallel Programming and Run-Time Management Techniques for Many-core Architectures and Design Tools and Architectures for Multicore Embedded Computing Platforms*, PARMA-DITAM '17, pages 25–30, New York, NY, USA, 2017. ACM.

[9] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS10). Society for Artificial Intelligence and Statistics*, 2010.

[10] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan. Deep learning with limited numerical precision. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 1737–1746. JMLR.org, 2015.

[11] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally. EIE: efficient inference engine on compressed deep neural network. *CoRR*, abs/1602.01528, 2016.

[12] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. *CoRR*, abs/1510.00149, 2015.

[13] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[14] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852, 2015.

[15] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *CoRR*, abs/1609.07061, 2016.

[16] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pages 1097–1105, USA, 2012. Curran Associates Inc.

[18] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[19] F. Li and B. Liu. Ternary weight networks. *CoRR*, abs/1605.04711, 2016.

[20] Z. Lin, M. Courbariaux, R. Memisevic, and Y. Bengio. Neural networks with few multiplications. *CoRR*, abs/1510.03009, 2015.

[21] N. Mellempudi, A. Kundu, D. Mudigere, D. Das, B. Kaul, and P. Dubey. Ternary neural networks with fine-grained quantization. *CoRR*, abs/1705.01462, 2017.

[22] D. Miyashita, E. H. Lee, and B. Murmann. Convolutional neural networks using logarithmic data representation. *CoRR*, abs/1603.01025, 2016.

[23] E. Park, J. Ahn, and S. Yoo. Weighted-entropy-based quantization for deep neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[24] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. *CoRR*, abs/1603.05279, 2016.

[25] S. Ravi. Projectionnet: Learning efficient on-device deep networks using neural projections. *CoRR*, abs/1708.00630, 2017.

[26] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. Imagenet large scale visual recognition challenge. *Int. J. Comput. Vision*, 115(3):211–252, Dec. 2015.

[27] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[28] W. Tang, G. Hua, and L. Wang. How to train a compact binary neural network with high accuracy? In *AAAI*, 2017.

[29] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. H. W. Leong, M. Jahre, and K. A. Vissers. FINN: A framework for fast, scalable binarized neural network inference. *CoRR*, abs/1612.07119, 2016.

[30] G. Venkatesh, E. Nurvitadhi, and D. Marr. Accelerating deep convolutional networks using low-precision and sparsity. *CoRR*, abs/1610.00324, 2016.

[31] P. Viola and M. Jones. Robust real-time object detection. In *International Journal of Computer Vision*, 2001.

[32] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen. Incremental network quantization: Towards lossless cnns with low-precision weights. *CoRR*, abs/1702.03044, 2017.

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

[34] C. Zhu, S. Han, H. Mao, and W. J. Dally. Trained ternary quantization. *CoRR*, abs/1612.01064, 2016.