Fixed-point (FXP) inference has proven suitable for embedded devices with limited computational resources, and yet model training is continually performed in floating-point (FLP). FXP training has not been fully explored and the non-trivial conversion from FLP to FXP presents unavoidable performance drop. We propose a novel method to train and obtain FXP convolutional keyword-spotting (KWS) models. We combine our methodology with two quantization-aware-training (QAT) techniques – squashed weight distribution and absolute cosine regularization for model parameters, and propose techniques for extending QAT over transient variables, otherwise neglected by previous paradigms. Experimental results on the Google Speech Commands v2 dataset show that we can reduce model precision up to 4-bit with no loss in accuracy. Furthermore, on an in-house KWS dataset, we show that our 8-bit FXP-QAT models have a 4-6% improvement in relative false discovery rate at fixed false reject rate compared to full precision FLP models. During inference we argue that FXP-QAT eliminates q-format normalization and enables the use of low-bit accumulators while maximizing SIMD throughput to reduce user-perceived latency. We demonstrate that we can reduce execution time by 68% without compromising KWS model’s predictive performance or requiring model architectural changes. Our work provides novel findings that aid future research in this area and enable accurate and efficient models.

Index Terms— keyword-spotting, quantization-aware-training, fixed-point-arithmetic, on-device

1. INTRODUCTION

We are interested in a small memory footprint and low latency keyword spotting (KWS) system for embedded on-the-go (OTG) devices. Modern KWS algorithms use deep neural networks (DNN) and have shown to achieve human-level performance [1]. Despite their success, DNN deployment on-device is challenging due to computational and bandwidth demand. These challenges can be greatly mitigated by converting expensive floating-point (FLP) operations to fixed-point (FXP) while simultaneously improving inference speed [2]. FXP multiplications consume 18.5× less energy and half the memory compared to FLP [3]. However, DNN training is usually conducted in FLP due to operation parallelism on commercial GPUs, and wide numerical range that allows for better gradient updates. While similar parallelization for FXP can be realized on FPGAs [4], the lack of a processor comparable to GPUs makes training significantly slow. Most approaches thereby resort to converting models trained in FLP to FXP introducing bias that degrades performance [5, 6]. We propose a novel method that addresses the limitations of FXP training and uses FLP workflows to obtain FXP models.

Computational requirements can be reduced further using low-precision inference via quantization, which allows increased operations per accessed memory byte [5, 7]. Such quantization is typically achieved by means of post-training-quantization (PTQ) [8], which however causes severe information loss affecting model accuracy. To maintain overall accuracy for quantized DNNs, quantization can be incorporated in the training phase leading to quantization-aware-training (QAT). QAT introduces quantization noise during training by means of deterministic rounding [9, 10, 11], reparameterization [12, 13] or regularization [14, 15] among few techniques, allowing DNNs to adapt to inference quantization. Notable work has shown that with QAT model parameters can be learned at binary and ternary precision [16, 17].

In this paper, we discuss methods through which we enable FXP training and combine it with QAT. Our approach allows training up to ultra-low-precision (4-bit) FXP models that are accurate, and perform on-par with full-precision (32-bit) FLP models demonstrated on publicly available Google Speech Commands v2 dataset [18]. On an inhouse KWS dataset, our 8-bit FXP-QAT models outperform the 32-bit FLP baseline by a relative 4-6%. We additionally propose key inference optimizations that make KWS on constrained OTG devices possible. Our proposed approach achieves 68% faster inference while maintaining similar accuracy to full-precision FLP models. Furthermore, we argue that with QAT, we can eliminate ~10% CPU cycles otherwise spent in normalizing activation precision. Our contributions can be summarized as follows -

- we propose novel FXP model training that leverages workflows optimized for FLP;
- we extend two QAT techniques to FXP and train low-bit KWS models with minimal accuracy degradation;
- we reduce inference execution time using low-bit accumulators enabled through FXP-QAT;
- we propose a two-tier accumulator-buffer approach to reduce overflow/saturation typical of large DNN layers;

The remainder of the paper is organized as follows: in sections 2 and 3, we present related work, and introduce QAT focusing on two methods based on squashed weight distribution and absolute-cosine regularization. In section 4, we propose and describe our contributions to these methods, which allow us to achieve fixed-point training. We describe our experimental setup, and results in section 5. After discussion, we conclude the paper with our vision on the future of FXP-QAT for small footprint devices.

2. RELATED WORK

Among inference setups aimed at enabling DNN deployment for low-power platforms, FXP systems play a key role [19]. FXP of-
fered the most promising trade-off between accuracy and computational complexity among different number systems [20]. Horowitz et al. [3] found corroborative evidence with FXP multiply-accumulate (MAC) units compared to FLP on 45nm technology. Notable work has been done in FXP inference — Umuroglu et al. [7] built a framework that mapped binary weights to hardware, while other foundational works proposed to design schemes for FXP inference [21, 22]. On another front, there has been work on efficient model topologies [4], better approaches to quantize parameters to FXP with low errors [6] and methods to address limitations of FXP training [23, 24, 25]. While the majority of the work has been centered around developing suitable models [26] or efficient hardware, QAT improves model adaptation to FXP. Lin et al. [27] used SQNR based quantization to achieve better accuracy with 8-bit FXP but suffered from data overflow during inference. Anwar et al. [28] presented an exhaustive L2-error minimization based quantization but required pretraining with full-precision. In the context of speech processing, 4 to 6-bit QAT have been studied for event detection [29], speech recognition [14] and KWS [9] but limited to floating-point. All the above methods either require pretraining of model parameters or have runtime limitations and have limited discussion on inference computations.

3. QUANTIZATION AWARE TRAINING

Quantized models are typically obtained by full-precision training followed by quantization. Nevertheless, for aggressive compression, low-bit quantization or cross format conversion from FLP to FXP, such quantization degrades accuracy. With QAT, we introduce inference specific quantization-noise during training to avoid inference time errors. Our FXP training is agnostic to QAT methodology and can extend to different techniques. We choose two recent works in this domain and describe how we enable FXP-QAT for different model components.

3.1. Weight quantization

Using squashed weight distribution (SQWD), weights can be reparametrized to a finite range (e.g. [-1,1]). This can be achieved by means of a non-linear function (i.e. tanh) that maps weight \( w \) to \( \hat{w} \) in uniform distribution [12]. A regularization loss limits the distribution-spread to asymptotes of tanh for efficient mapping. Subsequently, \( \hat{w} \) is quantized to desired b-bit precision using eq. 1, where

\[
\hat{w}(flp,b) = \text{round}[2^{b-1}(\hat{w} + 1) - 0.5] + 0.5 \times 2^{b-1} - 1
\]  

(1)

Using absolute cosine regularization (ACR), QAT can be defined as a regularization problem. Most QAT techniques do not include explicit quantization loss in back-propagation [14]. Conversely with ACR, the model parameters are weighted using an absolute-cosine function that represents quantization loss. The cosine function has \( 2^b \) zero-points for b-bit quantization. We constrain our weights to the same finite range as SQWD to establish a fair comparison.

\[
\hat{w}(fxp,b) = \text{round}[2^{b-1} \times (\hat{w}(flt,p,b,32) + 1.0)]
\]  

(2)

\[
\hat{w}(flt,p,b) = \hat{w}(fxp,b,b) * 2^{b-1} - 1.0
\]  

(3)

Proposed approach with FXP

Either in SQWD or ACR, a FLP quantizer is used to quantize a weight \( w \) from full-precision FLP \( \hat{w}(flt,p,32) \) to b-bit FLP \( \hat{w}(flt,p,b) \). For a FXP inference with no quantization error, a trivial solution would be to replace the FLP quantizer in eq. 1 with a FXP quantizer. However, FXP arithmetic has limited support on GPUs which makes training large ML models challenging. To combat this, we propose a two-stage quantizer (shown in figure 1) that maps \( w \) from full-precision FLP \( \hat{w}(flt,p,32) \) to b-bit FXP \( \hat{w}(fxp,b) \) using eq. 2 and to b-bit FLP \( \hat{w}(flt,p,b) \) using eq. 3. This facilitates training in FLP arithmetic, albeit using FXP quantization. During back-propagation we use straight-through-estimators [30] for gradient calculation, to overcome the non-differentiability of the round \([\cdot]\) operation. For inference, we use eq. 2 to export parameters to FXP. With a one-to-one mapping between quantized FLP and FXP representations, we can eliminate errors from PTQ.

3.2. Input and activation quantization

Inputs to various model components form the remainder of inference arithmetic. In the context of hidden layer activations, deep models have a cascaded bit-growth problem due to the large number of multiply and accumulate (MAC) operations. To mitigate this, we placed fake quantization nodes between consecutive layers to scale down activations to smaller \([-1,1]\) range, prior to quantization with eq. 2 and eq. 3. On the other hand, to accommodate wider numerical range of feature inputs, we traded-off bit precision on fractional part. The number of bits used to represent fractional part is termed as q-format.
A slightly modified quantization approach with $q$-format for a feature $f$ is proposed in eq. 4 and 5, where $c = 0.5$ if $f < 0$ else $-0.5$.

$$[h]f_{i,p,b,q} = \min[\max(f \times 2^q + c, -2^{b-1}), 2^{b-1} - 1]$$

$$f_{i,p,b,q} = f_{i,p,b,q} \times 2^{-q}$$

(4)

(5)

4. FIXED-POINT INERENCE

This section describes how QAT improves FXP inference by eliminating a key trade-off typical of post-training-quantized models, and leverages increased parallelization with low-bit inference on SIMD architectures. Additionally, we describe and propose a solution for archetypal saturations in low-bit accumulators. The discussion is tailored in the scope of ARM NEON chips, a popular choice for FXP inference.

4.1. $q$-format normalization

FXP inference setups - specifically designed for PTQ assign a different $q$-format ($q_i$) per axis for model parameters to reduce quantization-error, resulting in mixed representations within the model. The number of representations scale with number of layers, and incur additional memory to store $q$-format mappings. To prevent this, activations are normalized during runtime to a uniform $q$, prior to subsequent layer. These normalization operations roughly consume 10% cycles in inference. With QAT, we impose training time constraints to learn model parameters with a common $q$-format ($q$) thus eliminating additional cycles spent in normalization, speeding up inference. This difference between PTQ and QAT is illustrated in figure 2.

4.2. Increased parallelization with QAT on SIMD

ARMv7a/v8a is the most adopted architecture for embedded devices using FXP arithmetic. The advanced-SIMD extension (NEON) in these architectures accelerates processing by parallelizing load, store, multiply, and accumulate operations. The $vmlal_s8$ NEON intrinsic [31] operating on 128-bit wide NEON registers supports up to eight operations in a single cycle. During inference, MACs make up to 90% of computations, and we leverage parallelization capability of NEON to parallelize MACs and achieve faster inference. To fully utilize the $8 \times$ parallelization, we limit the accumulator size in MAC operations to low 16-bits, since $16 \times 8 = 128$, shown in figure 3. The use of low-bit accumulator is enabled through FXP-QAT by training low-bit precision models. The 16-bit accumulator is scaled-down prior to next layer to avoid bit-growth explosion.

![Fig. 3](image-url) Proposed two-tier accumulator-buffer mechanism that prevent saturations in low-bit accumulator.

4.3. Saturations in low-bit accumulators

While smaller accumulators speed up inference, they saturate faster. The 16-bit NEON accumulator can saturate in as little as 2 and 4 MACs using 8-bit and 7-bit models. Although we can employ QAT to train models up-to ultra-low-bit precision (i.e. 4-bit), model topologies influence saturations. Specifically, wide models involve greater MACs per activation and are more likely to saturate accumulators. Saturations are a significant problem for the model performance since they corrupt downstream model propagation. For the keyword-spotting problem we are interested in, we observed that 0.1% saturations caused a $\sim 30\%$ degradation in keyword detection score. As a solution, we propose a two-tier accumulator-buffer (2T-AB) mechanism that mitigates this problem while maintaining benefits from $8 \times$ SIMD parallelism. As opposed to performing all the required MACs for an activation using a single accumulator, we propose to periodically flush the 16-bit accumulator to a wider 32-bit fixed-width buffer. In table 1, we profile 6-bit KWS model and show 2T-AB reduces the number of saturations. We observe that sparsely flushing (i.e every 256 MACs) reduces saturations by 89% while flushing every 64 MACs completely eliminates saturations.

<table>
<thead>
<tr>
<th>Kernel size</th>
<th># MACs per activation</th>
<th># activations corrupted by saturations at flushing cadence</th>
</tr>
</thead>
<tbody>
<tr>
<td>(3, 4, 32)</td>
<td>384</td>
<td>10,880</td>
</tr>
<tr>
<td>(4, 4, 32)</td>
<td>512</td>
<td>2,240</td>
</tr>
<tr>
<td>(7, 4, 40)</td>
<td>1,120</td>
<td>128</td>
</tr>
<tr>
<td>(1, 1, 128)</td>
<td>128</td>
<td>160</td>
</tr>
<tr>
<td>(1, 1, 100)</td>
<td>160</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 1. Number of activations corrupted by saturations at different convolutional layers with and without 2T-AB.

5. EXPERIMENTAL STUDY

We study the proposed fixed-point QAT technique using a KWS model for resource constrained devices. We evaluate our approach using quantized models trained with our approach against an unquantized full-precision model in terms of (i) accuracy (ii) latency.

5.1. Datasets

We conducted our experimentation using an in-house KWS dataset ($D_1$) and the Google speech commands v2 dataset ($D_2$) [18]. $D_1$ consisted of 12.5K hours of de-identified labeled audio from far-field and near-field mobile phones and was split into training (10K hours) and testing (2K hours) while $D_2$ comprised of 105K one-second audio snippets sampled at 16kHz, containing 35 different keyword utterances recorded in natural environments [18]. We used the official train/validation/test splits for $D_2$.

The audio in $D_1$ and $D_2$ was down-sampled to 16kHz, converted to single channel, and segmented using a 25ms analysis window with a 10ms shift. While audio files have varying length, we used a segment length of 76 to generate fixed length inputs. Subsequently, we extracted and concatenated 64-dimensional log-mel filter bank energies (LFBE-64) to form 2-D feature inputs to our convolutional KWS model.

5.2. Model configurations

We use a 2-D fully convolutional model for all experiments. The model contains five trainable consecutive convolutional blocks preceded by a non-trainable normalization layer. Convolutional blocks vary across the network in terms of kernel size, yet each convolutional layer is followed by batch normalization and ReLu nonlinearity as activation. The output is a softmax classification layer representing the posterior probability of keyword detection. The model has a total of 199K learnable parameters and uses 125K flops.
In this paper, we presented a method for training and developing FXP models that uses floating-point arithmetic during training, but has a one-one mapping and eliminates conversion errors post-training. We applied this method in conjunction with two different QAT techniques to learn low-precision model parameters and also propose methods that apply QAT over transient variables. We test our approach in the scope of keyword-spotting that often require accurate and low-latency inference. We initially demonstrate and show that our 8-bit FXP QAT models outperform 32-bit FLP counterparts. Encouraged by the promising results, we further reduced precision, and observed that our method has a strong generalization capability allowing us to train models up to ultra-low-bit precision (4-bit) with minimal accuracy degradation. Besides accurate, our model is more efficient than traditional models, achieving 68% and 30% speed up in execution time compared to floating-point inference, or the traditional combination of post training weight quantization, followed by fixed-point inference. In conclusion, we argue that it is possible to reduce on-device computation processing time, without sacrificing accuracy, which is especially important for always-on keyword recognition systems. In the next iteration of our work we plan to explore how our approach scales to large models and different problem spaces like computer vision and complex tasks like speech recognition. Additionally, we plan to combined our work with other compression techniques like pruning, distillation, and neural architecture search to generate efficient models. We believe that our findings will broadly impact the community focused on deep learning on resource constrained devices.
7. REFERENCES

[31] “Neon programmers guide, vector multiply-accumulate,”.