This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCDS.2022.3147789, IEEE Transactions on Cognitive and Developmental Systems



# Quantized Reservoir Computing for Spectrum Sensing with Knowledge Distillation

Shiya Liu, Lingjia Liu, and Yang Yi

***Abstract*—Quantization has been widely used to compress machine learning models for deployments on field-programmable gate array (FPGA). However, quantization often degrades the accuracy of a model. In this work, we introduce a quantization approach to reduce the computation/storage resource consumption of a model without losing much accuracy. Spectrum sensing is a technique to identify the idle/busy bandwidths in cognitive radio. The spectrum occupancy of each bandwidth maintains a temporal correlation with previous and future time slots. A recurrent neural network (RNN) is very suitable for spectrum sensing. Reservoir computing (RC) is a computation framework derived from the theory of RNNs. It is a better choice than RNN for spectrum sensing on FPGA because it is easier to train and requires fewer computation resources. We apply our quantization approach to the RC to reduce the resource consumption on FPGA. A knowledge distillation called teacher-student mutual learning is proposed for the quantized RC to minimize quantization errors. The teacher-student mutual learning resolves the mismatched capacity issue of conventional knowledge distillation and enables knowledge distillation on small datasets. On the spectrum sensing dataset, the quantized RC trained with the teacher-student mutual learning achieves comparable accuracy and reduces the resource utilization of digital signal processing (DSP) blocks, flip-flop (FF), and Lookup table (LUT) by 53%, 40%, and 35%, respectively compared to the RNN. The inference speed of the quantized RC is 2.4 times faster. The teacher-student mutual learning improves the accuracy of the quantized RC by 2.39%, which is better than the conventional knowledge distillation.**

***Index Terms*—Quantization, Reservoir Computing, Spectrum Sensing, Model Compression, Knowledge Distillation, Cognitive Radio.**

## I. Introduction

The usage of FPGA as a hardware acceleration platform for machine learning models is dramatically increasing recently. Compared to other platforms such as CPU and GPU, the advantages of FPGA platform is easier to reconfigure and faster time to market. Meanwhile, FPGA has lower power consumption [1, 2].

In the era of mobile computing, deploying machine learning models on embedded systems has attracted immense attention. Many quantization approaches have been proposed [3, 4, 5, 6] to compress machine learning models for increasing inference speed and reducing model size. The XNOR-net in [5] achieves 58X speed-up on convolution operation. Some quantization approaches [3] not only apply quantization during the inference phase but also on the training phase. By applying quantization on weights, activation, and gradients, both inference and training speed are improved. Meanwhile, the resource required for training is significantly reduced, which makes training on embedded systems applicable.

In these proposed quantization approaches [3, 4, 5, 6], a considerable effort is spent on the optimization of multiplication and addition operations by replacing the floating-point multiplication and addition with integer-only multiplication and addition. This is because multiplication and addition operations are the most commonly used operations during both the training and inference phases of machine learning models. Besides, multiplication is considered as one of the most complicated operations, which requires complex hardware implementation and consumes significant power. Even though the above-mentioned approaches could reduce the resource utilization and energy consumption of multiplication and addition compared to floating-point implementations, these two operations are still expensive to perform. In this work, a quantization approach is presented to enhance the efficiency of integer multiplications by bit-shift operations. Also, the quantization approach removes expensive re-scaling operations in quantized integer additions. These optimizations reduce the resource utilization of a machine learning model on FPGA significantly.

Spectrum sensing is a technique to identify the idle or busy bandwidths in cognitive radio. The spectrum occupancy of each bandwidth maintains a temporal correlation with previous and future time slots. RNN [7, 8] is a type of neural network which is designed to capture the temporal correlation in sequential data. Therefore, it is very suitable for spectrum sensing. However, RNN suffers issues of extensive computation, slow inference speed, and high power consumption on FPGA. Also, it is easy to have an overfitting issue when a dataset is small. Because of these disadvantages, an RNN is not a good choice for spectrum sensing on FPGA.

Reservoir computing (RC) [9, 10, 11] is a variant of RNN. Compared to a conventional RNN, an RC only has a reservoir layer and a readout layer. The input is mapped to a high-dimensional space by the reservoir layer. Then, the reservoir layer's output is sent to the readout layer to generate the output. In an RC, most of the weights are generated randomly and fixed during the training phase. The only weights needed to be trained are the weights in the readout layer. Compared to a conventional RNN, an RC is easier to train and more resource-efficient during inference. Moreover, it has much lower generalization errors, especially when dealing with small datasets such as spectrum sensing. In this paper, we apply

Shiya Liu, Lingjia Liu, and Yang Yi are with the Bradley Department of Electrical and Computing Engineering, Virginia Tech, Blacksburg, Virginia, 24061, USA

This work was supported in part by the U.S. National Science Foundation (NSF) under Grant CCF-1750450, Grant ECCS-1811497, and Grant CCF-1937487.

the RC to spectrum sensing on FPGA and demonstrate that it achieves better performance, compared with the traditional method using square law combining and support vector machine.

The knowledge distillation [12, 13, 14] is a model compression technique that transfers the "dark knowledge" embedded in the teacher model to the student model. Conventional knowledge distillation suffers three major issues when transferring the knowledge from a pre-trained floating-point model to a quantized model. Firstly, it is challenging to pick teacher models for student models. Several works [15, 16] found that larger teacher models do not often yield better student models due to the mismatched capacity between teacher and student models. Knowledge distillation via teacher assistant [16] is an approach to address the mismatched capacity issue by introducing teacher assistant models between teacher and student models. This multi-level knowledge distillation improves the accuracy of student models but significantly lowers the training efficiency. The second issue of conventional knowledge distillation is that a pre-trained teacher model's parameters might not be suitable for the student model and quantization. Therefore, the teacher model cannot provide useful guidance for the quantized student model. Another issue of conventional knowledge distillation is that some datasets such as spectrum sensing do not have sufficient data to train a large teacher model. If the teacher model is only slightly better than the student model or even worse due to overfitting issues, the conventional knowledge distillation will not work. To address the aforementioned three issues of knowledge distillation, a knowledge distillation called teacher-student mutual learning is proposed for quantized models. In teacher-student mutual learning, the pre-trained floating-point counterpart of the quantized model is adopted as the teacher model to ease the effect of mismatched capacity between the teacher and student model. To better transfer the knowledge, we distill knowledge from both the logits and intermediate layer of the teacher model to the student model [12, 17]. To make the teacher model more adaptable to the student model and quantization effects, we train teacher and student models in parallel and optimize the teacher model with the knowledge from the logits of the student model [18]. Through learning the knowledge transferred from the teacher model, the quantization errors of the student model are significantly reduced. Moreover, the knowledge from the student model adapts the teacher model for the student model and quantization effects. Then, the teacher model can provide better guidance for the quantized student model. Meanwhile, the knowledge generated by the student model alleviates overfitting issues of the teacher model, especially on a small dataset such as spectrum sensing. Our contributions are summarized below.

- We propose an efficient quantization approach for the RC. The teacher-student mutual learning is introduced to reduce quantization errors of the quantized RC.
- The teacher-student mutual learning not only diminishes the mismatched capacity between the teacher and student model but also enables knowledge distillation on small datasets. This approach improves the accuracy of the quantized RC by 2.39% on the spectrum sensing dataset, which is better than the conventional knowledge distillation.
- On the spectrum sensing dataset, the quantized RC reduces the digital signal processing (DSP) block, Flip-Flop (FF), and lookup table (LUT) utilization by 37%, 20%, and 15%, respectively with 2.3 times faster in speed compared to the floating-point RC. Compare to the quantized RC using the quantization method in [4], our quantized RC reduces the resource utilization of block random-access memory (BRAM), FF, and LUT by 18%, 9%, and 15% respectively and achieves the same accuracy.
- We demonstrate that the RC has better accuracy compared to traditional approaches, such as square law combining and support vector machine on spectral sensing.

## II. Background

### A. Quantization

A quantization approach for convolutional neural networks has been proposed in [4]. To replace the floating-point number with integer number, a quantization scheme [4] is proposed as,

$$r = S(q - Z), \tag{1}$$

where $r$ and $q$ are a floating-point number and the quantized integer of the floating-point number $r$ respectively. $S$ is the floating-point scaling constant and $Z$ is an integer that represents the number 0.

In Eq. (1), scaling constant $S$ is a floating-point number and thus we need to quantize it into an integer number. Eq. (2) shows a typical way to convert $S$ to an integer number [4].

$$q_S = \frac{S \times 2^{n-1}}{2^{n-1}}, \tag{2}$$

where $n$ is the number of bits of an integer representing scaling constant $S$. In Eq. (2), by multiplying $S$ with an integer number $2^{n-1}$, the scaling constant $S$ becomes an integer number. The dividend $2^{n-1}$ can be completed using bit-shift operations. Quantization of $S$ is calculated off-line since $S$ is fixed after training.

Multiplication and addition are the most fundamental and widely used arithmetic operations in machine learning models. Based on Eq. (1), both multiplication and addition operations could be computed using integer-only arithmetic.

*1) Quantized Multiplication:* Assume there are two floating-point numbers which are $r_1$ and $r_2$, respectively and the product result is $r_3$. Based on Eq. (1), the multiplication between these two floating-point numbers could be calculated using Eq. (3).

$$S_3(q_3 - Z_3) = S_1(q_1 - Z_1)S_2(q_2 - Z_2), \tag{3}$$

where $q_1$, $q_2$ and $q_3$ are the quantized integer of floating number $r_1$, $r_2$, and $r_3$, respectively. $S_1$, $S_2$, and $S_3$ are the positive scaling constants for floating number $r_1$, $r_2$, and $r_3$ respectively. $Z_1$, $Z_2$, and $Z_3$ are the quantized zero-point for floating number $r_1$, $r_2$, and $r_3$ respectively.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCDS.2022.3147789, IEEE Transactions on Cognitive and Developmental Systems



By rewriting the Eq. (3), the quantized representation $q_3$ for floating-point number $r_3$ could be calculated as,

$$\begin{aligned} q_3 &= Z_3 + S(q_1 - Z_1)(q_2 - Z_2), \\ S &= \frac{S_1 S_2}{S_3}. \end{aligned} \tag{4}$$

the only floating-point number $S$ in Eq. (4) can be converted to an integer using Eq. (2). Then Eq. (4) could be calculated as,

$$q_3 = Z_3 + \frac{S \times 2^{n-1}}{2^{n-1}}(q_1 - Z_1)(q_2 - Z_2). \tag{5}$$

Then, the multiplication between two floating-point numbers can be performed using integer-only arithmetic.

*2) Quantized Integer Addition:* Assume we are adding two floating-point numbers $r_1$ and $r_2$ together using integer-only arithmetic.

$$r_3 = r_1 + r_2. \tag{6}$$

Using Eq. (1), Eq. (6) could be rewrite as,

$$S_3(q_3 - Z_3) = S_1(q_1 - Z_1) + S_2(q_2 - Z_2). \tag{7}$$

To add them together, we first need to re-scale the scaling constant of $r_2$ equal to the scaling constant of $r_1$, which could be expressed as,

$$\begin{aligned} T &= \frac{S_1}{S_2} \\ S_3(q_3 - Z_3) &= S_1(q_1 - Z_1) + S_2 \times T \times \frac{q_2 - Z_2}{T} \\ S_3(q_3 - Z_3) &= S_1(q_1 - Z_1) + S_1 \times \frac{q_2 - Z_2}{T}. \end{aligned} \tag{8}$$

In Eq. (8), $S_1$, $S_2$, $S_3$, and T are floating-point numbers. We quantize them into integer numbers using Eq. (2). Assume $q_T$ is the quantized integer of the floating-point number $\frac{1}{T}$. We have,

$$\begin{aligned} S_3(q_3 - Z_3) &= S_1(q_1 - Z_1) + S_1(q_2 - Z_2)q_T \\ S_3(q_3 - Z_3) &= S_1(q_1 - Z_1 + q_2 q_T - Z_2 q_T). \end{aligned} \tag{9}$$

Finally, we need to re-scale the scaling constant of $r_1$ equal to the scaling constant of $r_3$. Assume $S = \frac{S_1}{S_3}$ and $q_S$ is the quantized integer of $S$. $q_3$ could be computed as,

$$q_3 = Z_3 + q_S(q_1 - Z_1 + q_2 q_T - Z_2 q_T). \tag{10}$$

As shown in Eq. (10), the addition operation between two floating-point numbers can be performed using integer-only arithmetic. The quantized integer addition requires two re-scale operations. The first re-scale operation re-scales the scaling constant of $r_2$ equal to the scaling constant of $r_1$. The second re-scale operation re-scales the scaling constant of $r_1$ equal to the scaling constant of $r_3$.

### B. Knowledge Distillation

Knowledge distillation is a model compression technique proposed in [12]. The idea of the knowledge distillation is to transfer the knowledge learned by a teacher model to a student model, which typically has a smaller model size. The knowledge distillation is summarized in Fig. 1. The approach adopts two targets, which are "hard target" and "soft target" respectively, to transfer the knowledge learned by the teacher model to the student model. The "hard target" is the ground truth label from the dataset. The "soft target" is the probabilities of each class predicted by the teacher model for the same input. There is an issue in the "soft target". When dealing with an easy sample, the correct class has close to 1 probability while other classes are close to 0. In such a case, the "soft target" is identical to "hard label" and does not provide much knowledge for the student model to learn. The authors in [12] claim that important information is embedded in the ratios of very small probabilities in the "soft targets". To distill the important information, the authors introduce a parameter $T$ called temperature into the Softmax function to resolve this issue. The new softmax function is defined as,

$$P(x;T) = softmax(x/T), \tag{11}$$

where $x$ is the prediction of the model and $T$ is the temperature parameter.

As shown in Fig. 1, the overall loss function of the student model includes two loss functions. The cross-entropy loss function uses "hard targets" and the distillation loss function uses "soft targets" generated by the teacher model. Mathematically, the loss function is expressed as,

$$\begin{aligned} L_s = (1-\alpha)H(y, P(x_s;T=1)) + \\ \alpha T^2 * L_{KL}(P(x_t;T=t), P(x_s;T=t)), \end{aligned} \tag{12}$$

where $H$ is the cross-entropy loss function and $L_{KL}$ is the Kullback-Leibler divergence [19] loss function. $y$ represents the ground truth label. $P$ is the softmax function with temperature parameter $T$. $\alpha$ is a hyperparameter to control the weighted average in the loss function. $x_t$ is the logits of the teacher model, and $x_s$ is the logits of the student model.

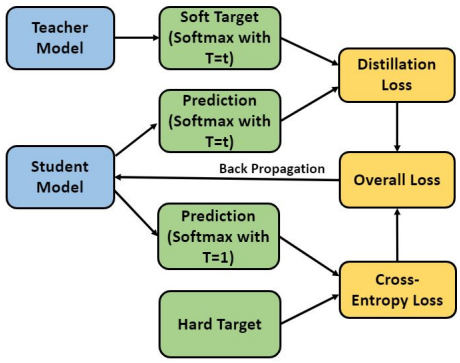


Fig. 1: Overview of Knowledge Distillation

Many research works have applied knowledge distillation to different applications [12, 20, 21]. an approach of applying knowledge distillation to object detection to improve mean average precision (mAP) of a compressed model is proposed. Through the knowledge distillation, the mAP of the compressed model is improved by 8% on the PASCAL VOC dataset [22] and 11% on the MS COCO [23]. Meanwhile, the compressed model is approximately 12 times smaller than the teacher model. Using ensembles of teacher models to train a single student model on Automatic Speech Recognition (ASR) is proposed [12]. The frame accuracy and Word Error Rate

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCDS.2022.3147789, IEEE Transactions on Cognitive and Developmental Systems



(WER) of the student model is improved by 3.2% and 1.8% respectively. The Knowledge distillation also has been applied to semi-supervised learning. In [21], the authors use ensembles of teacher model to generate labels for unlabeled data. Then, these unlabeled data are used to train the student model. The experimental results of human keypoint detection and general object detection show that the student model trained with unlabeled data outperforms the student model trained with labeled data.

## III. Reservoir Computing

### A. Advantages of Reservoir Computing

*1) Training of Reservoir Computing:* Nowadays, an RNN is trained by Backpropagation through time algorithm [24]. To calculate the gradients, an RNN is unrolled into multiple layers and then backpropagation is applied to the unrolled neural network. The training of an RNN often suffers gradient explosion or gradient vanishing problems when a sequence is very long. an RC does not suffer the gradient vanishing or gradient explosion problems since most of the weights are generated randomly and leave untrained during the training phase.

*2) Less Overfitting Issues:* When training an RNN, we have to spend time resolving overfitting issues. In contrast, an RC has less overfitting issues since most of the parameters are generated randomly and only the weights connected to readout layers are trained.

*3) Efficient Hardware Implementation:* Compared to RNN, an RC has simpler architecture, and thus the hardware implementation of the RC is very friendly. Meanwhile, the RC has faster inference speed and consumes fewer resources which are very suitable for resource-constrained devices.

*4) Memory Capacity:* Even though the architecture of RC system is simple, it still has rich memory capacity [25]. To further increase the memory capacity of an RC, the concept of deep neural network has been brought to the RC and several deep RC architectures have been proposed [25]. The authors in [25] empirically demonstrate that the deep RC architecture achieves high time-scale differentiation compared to the shallow RC architecture. Meanwhile, the deep RC has a richer memory capacity compared to the shallow RC architecture.

### B. Three Types of Reservoir Computing

An RC has a reservoir and a readout layer. The weights connected to the reservoir layer are created randomly. The weights of the readout layer are trained. A reservoir layer is expressed as,

$$\begin{aligned} x(t) =&(1-a)x(t-1)+ \\ & a * Activation(W_{in}u(t) + W_{res}x(t-1)), \end{aligned} \tag{13}$$

where $u(t)$ and $x(t)$ represent the input and output of reservoir layer at time $t$ respectively. $x(t-1)$ is the reservoir layer's output from the previous timestamp. $a$ is a hyperparameter to control the weighted average of terms in the reservoir layer. $W_{in}$ and $W_{res}$ are the input-to-reservoir weight matrix and recurrent-to-reservoir weight matrix respectively. The readout layer is trained and could be expressed as,

$$y(t) = W_{readout}x(t) + \theta_{readout}, \tag{14}$$

where $W_{readout}$ is a reservoir-to-readout weight matrix and it is trained. $\theta_{readout}$ is the bias term for the readout layer.

There are several types of RC, such as echo state network, liquid state machine, and delay feedback reservoir.

*1) Echo State Network:* In an echo state network (ESN) [26], the weights $W_{in}$, which connects the input and the reservoir layer, and the weight $W_{res}$, which connects neurons in the reservoir layer, are created randomly. The weights $W_{readout}$ of the readout layer are optimized to learn the temporal patterns of the reservoir layer. The overall architecture of an echo state network is shown in Fig. 2.

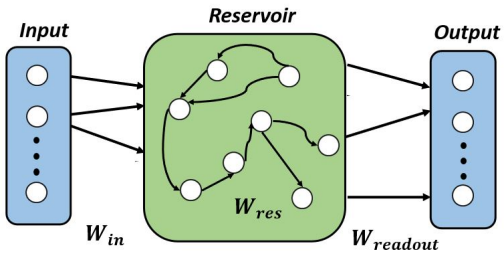


Fig. 2: Overview of Echo State Network

*2) Liquid State Machine:* Liquid State Machine (LSM) [27] is a type of spiking neural network. The overall architecture is very similar to the echo state network and is shown in Fig. 3. In LSM, the input is an array of spikes. The input layer is connected to the reservoir layers by the weight $W_{in}$ and spiking neurons inside reservoir layers are connected through the weight $W_{res}$. The readout layer uses the output of the reservoir layer as input to generate the final output. Like the echo state network, only the weight in the readout layer is trained. The difference between an echo state network and a liquid state machine is that the liquid state machine uses spiking neurons and the input is a vector of spikes.

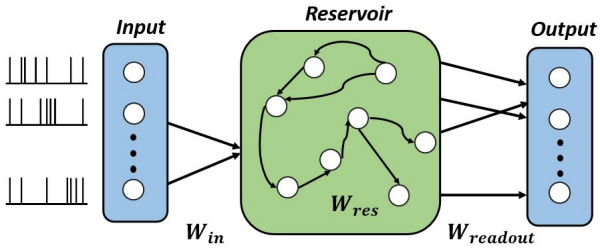


Fig. 3: Overview of Liquid State Machine

*3) Delay Feedback Reservoir:* The reservoir layer in ESN and LSM includes numerous neurons that are connected sparsely. There is only one node in the reservoir layer of a delay feedback reservoir (DFR) [28, 29, 30]. The node has several virtual nodes to form a delay feedback loop. Fig. 4 illustrates the architecture of a DFR.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCDS.2022.3147789, IEEE Transactions on Cognitive and Developmental Systems



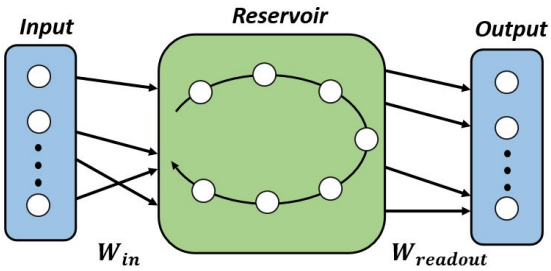


Fig. 4: Overview of DFR Computing

A DFR has fewer parameters compared to ESN and LSM. For an ESN or an LSM with $N$ hidden units and $K$ dimensional input, the total number of parameters is $K \times N + N \times N$. For a DFR, the number of parameters is $K \times N + 1$, which is much less than the ESN and LSM. Because of this merit, a DFR might be more suitable for resource-constrained devices for some applications, such as spectrum sensing.

## IV. Quantization of Delay Feedback Reservoir with Teacher-Student Mutual Learning

Multiplication operation is the most commonly used operation during both the training and inference phase of a neural network [31]. Also, multiplication is considered as one of the most complicated operations, which requires complex hardware implementation and consumes significant power. In this work, we introduce an approach to replace the expensive floating-point multiplication operation with an integer-only multiplication operation. Also, part of the integer-only multiplication operation is completed using bit-shift operations. In this way, the power consumption and the complexity of hardware implementation of a multiplication operation could be reduced significantly.

Addition operations are widely used in neural network models [32, 33]. An efficient addition operation is essential for a neural network running on resource-constrained hardware. Quantized integer addition could be expensive to perform when two quantized integers are in a different value range. The common method [4] requires two re-scale operations to perform a quantized integer addition. The details of how to perform quantized integer addition using the method in [4] is shown in II-A2. The increase of computation resources of a single quantized integer addition is negligible. However, many layers in a model such as the reservoir layer in a DFR model involve massive element-wise addition operation between two large arrays. Such massive additions would require much more computation resources. To resolve this issue, a new quantization scheme is proposed to perform quantized integer additions efficiently.

To reduce the quantization error, we present a teacher-student mutual learning approach to optimize the quantized DFR. Unlike the knowledge distillation introduced in [12], the teacher-student mutual learning approach does not suffer the issue of mismatched capacity between teacher and student model [15, 16]. Also, the method adapts the teacher model to the student model and quantization effects by transferring the knowledge from the student model to the teacher model. Finally, it enables knowledge distillation on small datasets, which do not have sufficient data to train teacher models.

### A. Quantization of Delay Feedback Reservoir

A popular quantization scheme could be expressed using Eq. (1) [4]. There are two major issues of the existing quantization scheme. The first issue is that it introduces additional multiplication because of the scaling constant $S$. Even though we could quantize scaling constant $S$ to an integer number using Eq. (2), an additional integer multiplication is needed.

The second issue is that a quantized integer addition is very expensive to perform since multiple re-scaling operations are required [4]. The details of how to perform quantized integer addition is shown in II-A2. When we are doing massive parallel quantized additions between two large arrays, the resource utilization would be increased dramatically.

To resolve the additional multiplication and expensive quantized addition issues of the existing quantization scheme, we proposed a new quantization scheme, which is shown in Eq. (15). We round scaling constant $S$ to the nearest powers of 2 using the logarithmic function. Then Eq. (1) could be expressed as,

$$\begin{aligned} n &= round(log(S)), \\ r &= 2^n(q - Z), \end{aligned} \tag{15}$$

where $log$ is a base-2 logarithmic function. A floating-point number is rounded to its nearest integer by the function $round$. $n$ is the number of bit-shift.

By using the logarithmic function, we not only quantize each $S$ into an integer number but also replace the integer multiplication between a scaling constant $S$ and a quantized integer with a bit-shift operation. Through the proposed quantization scheme, Eq. (3) could be expressed as,

$$\begin{aligned} S_3(q_3 - Z_3) &= S_1(q_1 - Z_1)S_2(q_2 - Z_2), \\ 2^{n_3}(q_3 - Z_3) &= 2^{n_1}(q_1 - Z_1)2^{n_2}(q_2 - Z_2), \\ q_3 &= \frac{2^{n_1}2^{n_2}}{2^{n_3}}(q_1 - Z_1)(q_2 - Z_2) + Z_3. \end{aligned} \tag{16}$$

As can be seen from Eq. (16), the additional integer multiplication introduced by the scaling constant $S$ is performed by a bit-shift operation. Meanwhile, the bit-shift parameter $n_1$, $n_2$, and $n_3$ are fixed during the inference phase and thus can be calculated off-line. When performing large matrix multiplication in parallel, bit-shift operations significantly reduce the resource consumption compared to integer multiplications.

Using our quantization scheme, the expensive quantized integer addition in Eq. (6) could be simplified as,

$$\begin{aligned} S_3(q_3 - Z_3) &= S_1(q_1 - Z_1) + S_2(q_2 - Z_2), \\ 2^{n_3}(q_3 - Z_3) &= 2^{n_1}(q_1 - Z_1) + 2^{n_2}(q_2 - Z_2), \\ q_3 &= \frac{2^{n_1}(q_1 - Z_1) + 2^{n_2}(q_2 - Z_2)}{2^{n_3}} + Z_3, \\ q_3 &= 2^{n_1 - n_3}(q_1 - Z_1) + 2^{n_2 - n_3}(q_2 - Z_2) + Z_3, \end{aligned} \tag{17}$$

where $n_1 = round(log(S_1))$, $n_2 = round(log(S_2))$, and $n_3 = round(log(S_3))$. Our method get rid of the two re-scale operations required by the method in [4] and replace all

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCDS.2022.3147789, IEEE Transactions on Cognitive and Developmental Systems



integer multiplications in a quantized integer addition with bit-shift operations. The resource consumption of the proposed quantization method and the method in [4] applied to the DFR model on the spectrum sensing task is summarized in Table VII. The proposed quantization method reduces resource consumption and achieves the same accuracy as the method in [4].

### B. Teacher-Student Mutual Learning

When transferring knowledge from a floating-point teacher model to a quantized student model, conventional knowledge distillation [12] suffers three drawbacks. Firstly, picking a suitable teacher model for a student model is challenging due to the mismatched model capacity between teacher and student models, and high accuracy teacher models do not often produce better student models [15, 16]. Sometimes, they even deteriorate the performance of student models. Secondly, a pre-trained teacher model's parameters might not be suitable for the student model and quantization so that it cannot provide useful guidance for a quantized student model. Another issue of conventional knowledge distillation is that it is difficult to train a teacher model on small datasets due to overfitting issues.

To address the aforementioned issues, we propose a knowledge distillation called teacher-student mutual learning. The overall training procedure of the teacher-student mutual learning is shown in Fig. 5. In the teacher-student mutual learning,

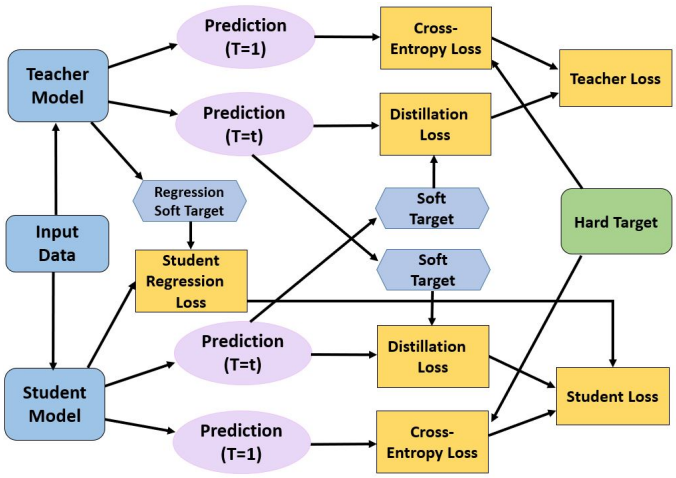


Fig. 5: The overall structure of teacher-student mutual learning

the quantized DFR and its pre-trained floating-point counterpart are used as the student and teacher model respectively to bridge the gap of model capacity between the teacher and student model. During the training phase, the student model uses three targets, which are "hard target", "classification soft target", and "regression soft target", respectively. the "hard target" is the ground truth label from the dataset and the "classification soft target" is the probability of each class predicted by the teacher model for the same input. The "regression soft target" is output feature maps of intermediate layers from the teacher model for the same input. Unlike the conventional knowledge distillation [12] that only transfers knowledge through the "classification soft target", the proposed approach transfers the teacher model's knowledge to the student model through both the "classification soft target" and "regression soft target". The additional soft target eases the training difficulty and improves the student model's performance [17]. By exploiting the transferred knowledge from the teacher model, the student model can mimic the behavior of the teacher model to minimize quantization errors. During the training, the teacher model uses two targets, which are the "classification soft target" and "hard target", respectively [18]. The "classification soft target" is the probability of each class generated by the student model, and the "hard target" is the ground truth label from the dataset. We transfer the knowledge from the student model to the teacher model through the "classification soft target" for two purposes. Firstly, the student's knowledge makes the teacher model more adaptable to the student model and quantization effects. Then, better guidance can be provided by the teacher model for the student model. Secondly, in small datasets such as the spectrum sensing dataset, the labeled data is limited. Therefore, the teacher model faces overfitting issues. The "classification soft target" introduces noise to the teacher model and helps the teacher model reducing overfitting issues.

### C. Weighted Mutual Learning Loss Function

As shown in Fig. 5, the student model is trained by three loss functions, which are the cross-entropy loss, the distillation loss, and the regression loss function. The "hard targets" is adopted in the cross-entropy loss. The "classification soft targets" and "regression soft targets" are utilized in the distillation loss and the regression loss, respectively. The teacher model is trained by two loss functions, which are the cross-entropy loss and distillation loss function respectively. The overall loss function for the student model $L_s$ can be expressed as,

$$L_s = (1-\alpha)H(y, P_s) + \alpha T^2 L_{KL}(P_t^\eta, P_s^\eta) + \beta L_{reg}(F_t, F_s), \tag{18}$$

and the overall loss function for the teacher model $L_t$ can be computed as,

$$L_t = (1-\alpha)H(y, P_t) + \alpha T^2 L_{KL}(P_s^\eta, P_t^\eta), \tag{19}$$

where $H$ is the cross-entropy loss and $y$ represents the ground truth label from the dataset. $\alpha$ and $\beta$ are two hyperparameters to control the distillation and regression loss. $T$ is the temperature parameter. $P_s$ and $P_t$ are the predicted probability of each class with temperature $T = 1$ for the student and teacher model respectively. $P_s^\eta$ and $P_t^\eta$ are the predicted probability of each class with temperature $T \neq 1$ for the student and teacher model respectively. $F_t$ is the output feature map of an intermediate layer in the teacher model, whereas $F_s$ is the corresponding output feature map of the intermediate layer in the student model. $L_{KL}$ is the distillation loss function. In classification tasks such as spectrum sensing, the class imbalance issue is very common and seriously affects the performance of the model. To address this issue, the weighted Kullback-Leibler divergence loss function is used as the distillation loss function. In the weighted Kullback-Leibler divergence loss

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCDS.2022.3147789, IEEE Transactions on Cognitive and Developmental Systems



function, each class is multiplied by a class weight assigned to the class. Then the distillation loss is expressed as,

$$L_{KL}(P_t^\eta, P_s^\eta) = \sum_{c=1}^{M} w_c P_t^\eta \log(\frac{P_t^\eta}{P_s^\eta}), \tag{20}$$

where $M$ is the number of classes of the task. $w_c$ is the class weight assigned to class $c$. $L_{reg}$ is the regression loss function and it is a L2 loss function. $L_{reg}$ is illustrated as,

$$L_{reg}(F_t, F_s) = \frac{1}{2} \|F_t - F_s\|^2 \tag{21}$$

### D. Training of Quantized Delay Feedback Reservoir

Typically, the quantized model is trained using floating-point numbers and then quantizes each floating-point number to an integer after training. This training approach is simple but leads to a large accuracy drop. We use the quantization-aware training [4, 6] to reduce quantization errors by simulating quantization effects during training. In the quantization-aware training, weight parameters are represented using floating-point numbers to better track the tiny change of each weight since gradients are usually very small. To simulate the quantization effects during the training phase, we apply the simulated quantization function $q(r)$ to each weight in the forward pass of the training. The $q(r)$ function first quantize each floating-point weight and round each weight to an integer number. Then each integer weight is converted back to a floating-point weight. Finally, these converted floating-point weights are used during the forward pass of the training phase. The simulated quantization function $q(r)$ can be expressed as,

$$\begin{aligned} s &= \frac{hi - lo}{n - 1}, \\ q(r) &= round(\frac{r - lo}{s})s + lo, \end{aligned} \tag{22}$$

where $lo$ and $hi$ are the lower and upper bound of a layer. $r$ is a floating-point number and $n$ is the number of quantization levels. A floating-point number is rounded to its nearest integer by the function $round$. $s$ is the scaling constant which is introduced in Eq. (1).

In the backward pass of the training phase, based on the straight-through estimator [34], the gradient will be passed to the floating-point weight directly for tracking the small change on the weight from the gradient.

## V. Reservoir Computing for Spectrum Sensing

Spectrum sensing is a technique to identify the idle or busy bandwidths in cognitive radio. The spectrum occupancy of each bandwidth maintains a temporal correlation with previous and future time slots. The overall structure of spectrum sensing is shown in Fig. 6.

In spectrum sensing, the spectral efficiency is significantly improved through combining multiple-input-multiple-output (MIMO) and orthogonal-frequency-division multiplexing (OFDM) systems. As MIMO utilizes spatial multiplexing gain and frequency selective fading, inter symbol interference (ISI), and inter-channel interference (ICI) are avoided through

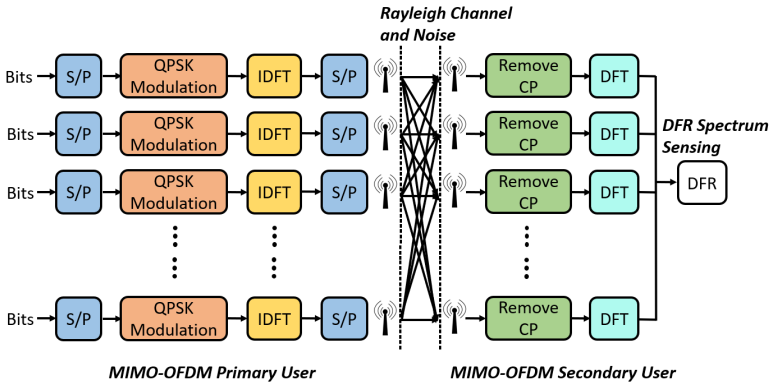


Fig. 6: The overall architecture of spectrum sensing on the MIMO-OFDM system. (DFT represents discrete Fourier transform. IDFT represents inverse discrete Fourier transform. QPSK is Quadrature Phase Shift Keying. CP represents cyclic prefix). S/P represents serial to parallel converter.

OFDM. However, the spectrum utilization is not always efficient in MIMO-OFDM systems and not all the subcarriers are utilized simultaneously [35]. To utilize the subcarriers that are not occupied by the primary users (PUs), the MIMO-OFDM-based cognitive radios propose to introduce some secondary users (SUs) that are authorized to utilize the free subcarriers in a dynamic spectrum sharing (DSS) environment. The SUs are allowed to transmit signals only on the subcarriers that are found not being used by the PU, and they should evacuate those bands as soon as the PU wants to use them. Therefore, it is fundamental for the cognitive radios to perform spectrum sensing subsequently that the available spectrum holes can be identified accurately and the interference is minimized. The spectrum sensing's performance can be significantly affected by the low signal-to-noise (SNR) ratios and fading wireless channels. In the literature, matched filtering, energy detection, and cyclo-stationary feature detection are the three classical spectrum sensing methods. These methods suffer several drawbacks such as accurate prior knowledge of the signal is needed, low detection at low SNRs, and computational complexity respectively [36, 37, 38]. To address the limitations of classical spectrum sensing methods, several machine learning-based approaches [39, 40, 41, 42] have been proposed. Compared to traditional approaches, machine learning-based approaches have several advantages. Firstly, the machine learning-based spectrum sensing approaches can learn the surrounding environment (e.g., the fading channel) of the cognitive radio effectively. Secondly, the machine learning-based approaches can find the decision boundaries more effectively [39, 40]. However, most of the machine learning-based approaches cannot capture effectively the spatial-temporal correlations existing in the received signals. Therefore, RNN is a good choice for spectrum sensing as it can capture the spatial-temporal correlation in the received signals [43]. However, RNNs are hard to train due to the vanishing gradients.

In this paper, we propose a resource-efficient quantized DFR for spectrum sensing on FPGA. We use DFR because it is both energy efficient and easy to train. Also, it can

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCDS.2022.3147789, IEEE Transactions on Cognitive and Developmental Systems



capture the spatial-temporal correlations in the received signals. The teacher-student mutual learning is adopted to reduce quantization errors of the quantized DFR. Our proposed DFR outperforms other spectrum sensing methods such as SVM [40, 44] and DSDFR [45].

## VI. Hardware Acceleration Architecture

The quantized DFR is implemented on Xilinx Zynq®-7000 FPGA board. The quantized DFR has a reservoir and a readout layer. The overall hardware architecture is shown in Fig. 7. During the inference phase, a sequence of input data is

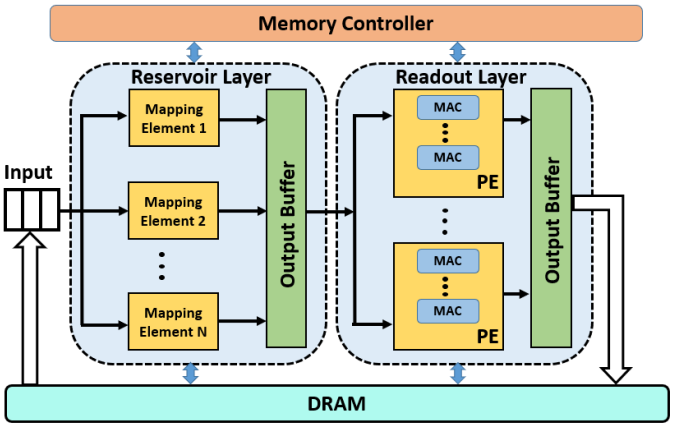


Fig. 7: Overview of Hardware Architecture

streamed into the reservoir layer from dynamic random-access memory (DRAM) on FPGA. There is an array of mapping elements inside the reservoir layer. Each mapping element has a kernel with size $1 \times G$ that maps input to high-dimensional spaces. Each mapping element has an output and weight buffer to store the mapped input and weight respectively. There is a dependency existing in the input between current time $t$ and previous time $t-1$. The mapped input at time $t$ will be added with the reservoir layer's output at time $t-1$. The added result goes through an activation function and the output of the activation function will be cached on the output buffer. A output buffer with a size of 512 is used in the reservoir layer. Once a sequence of input data is processed by the reservoir layer, the readout layer will read the data in the output buffer of the reservoir layer to perform further processing. The memory controller is used to control read and write operations of the input, output, and weight matrix of the reservoir layer.

The readout layer is performed by a matrix-vector multiplication. The input of the readout layer is a vector with a size of $N$, which is generated by the reservoir layer. The weight matrix of the readout layer has a size of $N \times M$. The generated output is a vector with a size of $M$. Loading a large matrix and performing the large matrix-vector multiplication on an FPGA is not applicable due to the hardware resource is limited. Therefore, we fold the matrix-vector multiplication onto several processing elements. The processing element is designed to perform a matrix-vector multiplication with a matrix size of $N_T \times M_T$ and a vector size of $N_T$. Assume the total number of processing elements is $P$ and then $M_T = \lceil M/P \rceil$. Each input sub-vector with a size of $N_T$ is shared by all processing elements. A processing element contains $K$ numbers of multiply-accumulate (MAC) unit. A MAC unit reads a vector of input with a size $N_T$ and a vector of weight with a size $N_T$ from a column of the weight matrix. Then, an element-wise multiplication is performed in parallel between the input and weight vector. Finally, an adder tree is used to accumulate the result of the multiplication. The structure of the MAC unit is shown in Fig. 8.

In the readout layer, the output buffer has a size of 512 elements. There is no need to create an input buffer since the input is read from the output buffer of the reservoir layer. The weight matrix is large and thus we only load part of the weight matrix each time from DRAM. We use a weight buffer with a size of $32 \times 512$ elements. The memory controller is exploited to control read and write operations of the input, output, and weight matrix of the readout layer. The number of MAC units $K$ in a processing element is 1 to reduce resource consumption. The MAC unit can process 32 elements each time.

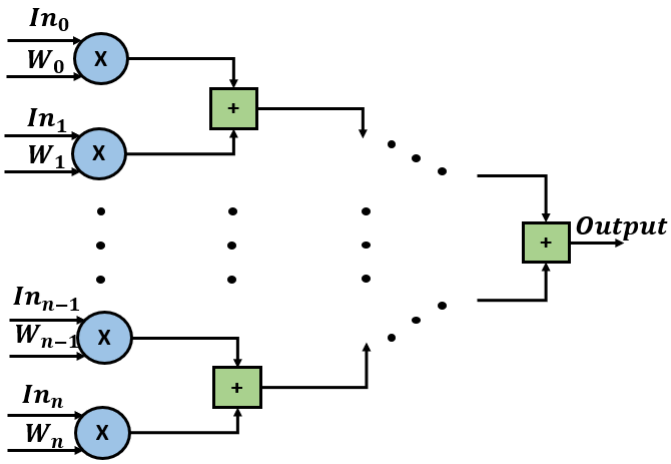


Fig. 8: Structure of MAC Unit

## VII. Application Evaluation

### A. Experimental Setup

We implement the proposed quantized DFR using Vivado HLS. The Xilinx Zynq®-7000 FPGA with Dual ARM® Cortex®-A9 MPCore™ with CoreSight™ FPGA board is used. The resource utilization and latency are reported by Vivado and Vivado HLS. The resource utilization is reported as a percentage of available resource that is utilized and the latency is reported as cycles. We report resource utilization from four resources, which are BRAM, DSP block, FF, and LUT. In our experiments, we compare ratios of resource utilization between different models. The classification accuracy is utilized as the metric to evaluate the performance of models in our experiments.

We apply the proposed quantized DFR to the application of spectrum sensing of MIMO communication systems combined with OFDM [46]. In the quantized DFR, an 8-bit integer is used to represent each weight and intermediate result from an activation layer. The spectrum sensing dataset is from the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCDS.2022.3147789, IEEE Transactions on Cognitive and Developmental Systems



RWTH Aachen University Static Spectrum Occupancy Measurement Campaign database [47]. The dataset incorporates the static spectrum occupancy measurement of the PU activity in different frequency bands and time slots. The occupancy of each subcarrier is modeled by the frequency occupancy model extracted from the database. The target of the spectrum sensing task is to determine if a subcarrier is busy or idle. Assume the $p_{th}$ subcarrier is busy and a signal $Y_p(k)$ presents. The received signal at the cognitive radio that is transmitted via the $p_{th}$ subcarrier is defined as $R_p(k) = Y_p(k) + N_p(k)$, where $N_p(k)$ represents the discrete Fourier transform of complex additive white Gaussian noise and $k = 1, ..., K$ where $K$ is the number of OFDM received symbols. If $p_{th}$ subcarrier is idle and the signal absents on the $p_{th}$ subcarrier, the received signal at the cognitive radio is expressed as $R_p(k) = N_p(k)$. We calculate the average received signal's energy of $K$ symbols as $E_p = \frac{1}{K}\sum_{k=1}^{K} |R_p(k)|^2$. In our experiments, a sequence of energy of the received signals from a sequence of time slots by the SUs is the input to the DFR model. The output of the DFR model is to determine whether a subcarrier is busy or idle.

In the experiments, we compare our floating-point DFR and quantized DFR trained using the teacher-student mutual learning (TSML) with the state-of-the-art methods such as SVM [40, 44] and DSDFR [11]. The performance of the traditional method such as square law combining and the deep learning method such as floating-point convolutional neural network (CNN) and the floating-point RNN are also compared. To further demonstrate the effectiveness of the TSML, we compare the accuracy improvement of our TSML on the quantized DFR system with the conventional knowledge distillation [12] and the weighted cross-entropy loss [20] on the same quantized DFR system.

To illustrate the efficiency of the proposed quantization method, we compare the resource utilization and inference speed between the quantized DFR using our method, the quantized DFR using the method in [4], and the floating-point DFR. We use the following abbreviations to represent different models in our experiments. The input sequence length for all models is 8 and each element in the sequence is the energy of the received signals by the SUs.

- **"SVM":** SVM represents the SVM model with radial basis function (RBF) kernel introduced in [40, 44].
- **"SLC":** SLC represents the square law combining (SLC) method [48, 49].
- **"DSDFR":** DSDFR is the DSDFR model proposed in [11].
- **"FPCNN":** FPCNN represents the floating-point CNN model with three layers. The first layer is a 1D convolution layer with kernel size 3. The input and output channels are 1 and 16 respectively. The second layer is a 1D convolution layer with kernel size 3. The input and output channels are 16 and 32 respectively. The last layer is a fully-connected layer with an input and output size of 256 and 2 respectively.
- **"FPRNN":** FPRNN represents the floating-point RNN model. It is a Many-to-one RNN and has one recurrent layer. The hidden state in the recurrent layer has a size of 32. There are two fully-connected layers after the recurrent layer. The first fully-connected layer has an input and output size of 32 and 16 respectively. The second fully-connected layer has an input and output size of 16 and 2 respectively.
- **"FPDFR":** FPDFR represents the floating-point DFR model trained without TSML. There is only one reservoir layer in the FPDFR. The hidden state in the reservoir layer has a size of 32. There are two fully-connected layers after the reservoir layer. The first fully-connected layer has a input and output size of 32 and 16 respectively. The second fully-connected layer has an input and output size of 16 and 2 respectively.
- **"FPDFR+TSML":** FPDFR+TSML is the floating-point DFR model trained with TSML. It has the same architecture as the FPDFR.
- **"QDFR":** QDFR is the quantized DFR model trained without TSML. The proposed quantization approach is adopted to quantize the model. It has the same architecture as the FPDFR but uses an 8-bit integer for each weight and activation.
- **"QDFR+TSML":** QDFR+TSML represents the quantized DFR model trained with TSML. It has the same architecture as the QDFR.
- **"Reference QDFR":** Reference QDFR is the quantized DFR model trained without TSML. The quantization approach introduced in [4] is adopted to quantize the model. It has the same architecture as the QDFR.
- **"QDFR+KD":** QDFR+KD is the quantized DFR model trained with conventional knowledge distillation (KD) [12]. It has the same architecture as the QDFR.
- **"QDFR+WCE":** QDFR+WCE represents the quantized DFR model trained with the weighted cross-entropy (WCE) loss [20]. It has the same architecture as the QDFR.

### *B. Training Setup*

All models are trained with a mini-batch size of 32. We use Adam learning algorithm [50], and the optimizer parameters are learning rate=0.01, beta1=0.9, beta2=0.999, epsilon=1e-07. We add a learning rate scheduler and the learning rate is multiplied by 0.1 every 30 epochs. The training epoch is 100. Models are trained to minimize a cross-entropy loss. Hyperparameters such as $w_c$, $T$, $\alpha$, and $\beta$ in Eq. (18), (19), and (20) are summarized in Table I. We use the ideas in [51, 52] and the hyperparameter optimization toolkit provided in [53] to determine these hyperparameters in our experiments.

TABLE I: Parameters settings of the TSML loss function

| SNR | Antennas | $w_c$ | $T$ | $\alpha$ | $\beta$ |
|---|---|---|---|---|---|
| -10dB | 4 Tx & 4 Rx | class 0=1.0<br>class 1=1.0 | 10 | 0.1 | 0.3 |
| | 6 Tx & 6 Rx | class 0=1.0<br>class 1=1.0 | 10 | 0.1 | 0.3 |
| -20dB | 4 Tx & 4 Rx | class 0=1.2<br>class 1=1.0 | 10 | 0.2 | 0.1 |
| | 6 Tx & 6 Rx | class 0=1.2<br>class 1=1.0 | 10 | 0.1 | 0.1 |

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCDS.2022.3147789, IEEE Transactions on Cognitive and Developmental Systems



### C. Accuracy Comparison of Different Models

The accuracy comparison between the SLC, SVM, DSDFR, FPCNN, FPRNN, FPDFR+TSML, and QDFR+TSML on the spectrum sensing dataset with different SNR ratios and numbers of antennas are illustrated in Table II. We evaluate these models with SNR at -10 and -20 dB. -10/-20 dB is a low signal-to-noise ratio. However, it is a reasonable assumption for spectrum sensing in communication systems. For example, the IEEE 802.22 standard for wireless regional area network (WRAN) requires spectrum sensing techniques that can detect the primary signal with the sensing receiver sensitivity being –116 dBm [54].

TABLE II: The accuracy comparison between SLC, SVM, DSDFR, FPCNN, FPRNN, FPDFR+TSML, and QDFR+TSML on the spectrum sensing dataset. (The result of the proposed method is highlighted in blue.)

| SNR | Model | 4Tx 4Rx | 6Tx 6Rx |
|---|---|---|---|
| -10dB | SLC | 98.32% | 99.46% |
| | SVM | 98.40% | 99.58% |
| | DSDFR | 98.67% | 99.54% |
| | FPCNN | 98.92% | 99.63% |
| | FPRNN | 99.06% | 99.90% |
| | FPDFR+TSML | 98.81% | 99.75% |
| | QDFR+TSML | 98.53% | 99.70% |
| -20dB | SLC | 66.81% | 95.26% |
| | SVM | 86.04% | 96.38% |
| | DSDFR | 87.42% | 96.61% |
| | FPCNN | 87.45% | 96.24% |
| | FPRNN | 88.10% | 96.52% |
| | FPDFR+TSML | 88.83% | 96.75% |
| | QDFR+TSML | 88.71% | 96.59% |

As demonstrated in Table II, all models achieve better accuracy as the number of antennas increases. This is because spatial multiplexing gain is improved when more antennas available and the improved spatial multiplexing gain is beneficial for spectrum sensing. At SNR=-10dB with Tx and Rx antennas is 6, all models have almost the same performance. At SNR=-20dB and only 4 Tx and Rx antennas available, the FPDFR+TSML model outperforms other models in terms of accuracy. The QDFR+TSML model has slightly lower accuracy than the FPDFR+TSML model. However, it achieves better performance than other models. In later section, we will show that the QDFR+TSML model is faster and more resource-efficient on hardware platform such as FPGA than the FPDFR+TSML.

### D. Accuracy Improvement using TSML

To demonstrate the accuracy improvement using the TSML, the spectrum sensing dataset at SNR=-20dB with Tx and Rx antennas is 4 is adopted because spectrum sensing is more difficult when a smaller number of antennas is available and more noise is added. In the experiment of TSML, the FPDFR and the QDFR are used as the teacher and student model respectively. We follow the training procedure shown in Fig. 5 to optimize the teacher and student model in parallel. The teacher model transfers the knowledge to the student model through the "classification soft target" and "regression soft target". The student model transfers the knowledge to the teacher model through the "classification soft target". The overall structure of the knowledge transfer is detailed in Fig. 9.

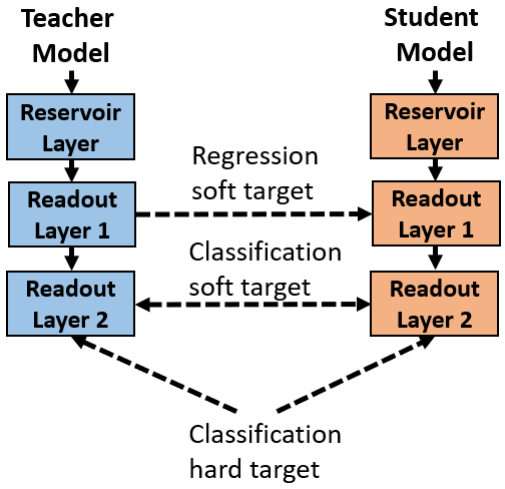


Fig. 9: Detailed structure of the knowledge transfer between the teacher and student model

The accuracy improvement of the FPDFR and QDFR model trained with TSML is shown in Table III. The QDFR+TSML model improves the accuracy by $2.39\%$ compared to the QDFR model. The FPDFR+TSML model is $2.18\%$ better than the FPDFR.

TABLE III: Accuracy improvement of the FPDFR+TSML and QDFR+TSML model on the spectrum sensing dataset at SNR(dB)=-20dB with Tx and Rx antennas is 4

| Model | Accuracy w/o TSML | Accuracy w/ TSML | Improvement |
|---|---|---|---|
| FPDFR | 86.93% | 88.83% | 2.18% |
| QDFR | 86.64% | 88.71% | 2.39% |

We also compare the QDFR+TSML model with the QDFR+KD, and the QDFR+WCE model. Results are summarized in Table IV. In the experiment of the QDFR+KD model, we tried both the FPDFR and the FPRNN as the teacher model. The best result we have is $86.80\%$, which is only slightly better than the baseline result of the QDFR. As demonstrated in Table IV, the accuracy of the QDFR+TSML is approximately $2.20\%$ and $1.46\%$ better than the QDFR+KD and the QDFR+WCE respectively.

TABLE IV: Accuracy comparison between the QDFR+TSML, QDFR+KD, and QDFR+WCE on the spectrum sensing dataset at SNR(dB)=-20dB with Tx and Rx antennas is 4.

| Model | Accuracy |
|---|---|
| QDFR+TSML | 88.71% |
| QDFR+KD | 86.80% |
| QDFR+WCE | 87.43% |

The accuracy comparison between the QDFR+TSML and QDFR model on the spectrum sensing dataset with different numbers of antennas at SNR=-10dB is illustrated in Table V.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCDS.2022.3147789, IEEE Transactions on Cognitive and Developmental Systems



TABLE V: Accuracy comparison between the QDFR+TSML and QDFR model on spectrum sensing dataset at SNR(dB)=-10dB with different number of antennas.

| Model | 4Tx 4Rx | 6Tx 6Rx |
|---|---|---|
| QDFR | 98.28% | 99.68% |
| QDFR+TSML | 98.53% | 99.70% |

Table VI shows the accuracy comparison between the QDFR+TSML and QDFR model on the spectrum sensing dataset with different numbers of antennas at SNR=-20dB.

TABLE VI: Accuracy comparison between the QDFR+TSML and QDFR model on spectrum sensing dataset at SNR(dB)=-20dB with different number of antennas.

| Model | 4Tx 4Rx | 6Tx 6Rx |
|---|---|---|
| QDFR | 86.64% | 96.20% |
| QDFR+TSML | 88.71% | 96.59% |

### *E. Resource Consumption*

To demonstrate the improvement of the model efficiency using the proposed quantization method, we compare the resource utilization and inference speed of the QDFR model with the Reference QDFR that uses the quantization approach introduced in [4] and the FPDFR model. The resource utilization of these three models is shown in Table VII. The accuracy in Table VII is measured on the spectrum sensing dataset at SNR(dB)=-20dB with Tx and Rx antennas is 4. The inference speed of these three models is illustrated in Table VIII.

TABLE VII: Comparison of resource utilization and accuracy between the QDFR, Reference QDFR, and FPDFR model on the spectrum sensing dataset

| Model | BRAM | DSP | FF | LUT | Accuracy |
|---|---|---|---|---|---|
| FPDFR | 1.00X | 1.00X | 1.00X | 1.00X | 86.93% |
| Reference QDFR | 0.95X | 0.63X | 0.88X | 1.00X | 86.69% |
| QDFR | 0.78X | 0.63X | 0.80X | 0.85X | 86.64% |

TABLE VIII: Comparison of the inference speed between the QDFR, Reference QDFR, and FPDFR model on the spectrum sensing dataset.

| Model | Inference Speed |
|---|---|
| FPDFR | 1.00X |
| Reference QDFR | 2.31X |
| QDFR | 2.31X |

As demonstrated in Table VII and Table VIII, compared to the FPDFR, the QDFR reduces the resource utilization of DSP, FF, and LUT by 37%, 20%, and 15% respectively and improve the inference speed by 2.3 times. Compare to the Reference QDFR, the QDFR reduces the resource utilization of BRAM, FF, and LUT by 18%, 9%, and 15% respectively, and achieves almost the same accuracy.

The energy consumption per sample of the QDFR on GPU and FPGA is shown in Table IX. The energy consumption per sample of the quantization method in [4] on FPGA is also included. The GPU we used is a single NVIDIA GeForce RTX 2080. The FPGA we used is the Xilinx Zynq®-7000 FPGA board. We measure the GPU power consumption using the NVIDIA GPU management and monitoring tool. The FPGA power consumption is measured using the Xilinx Vivado tool.

TABLE IX: The QDFR model energy consumption comparison using GPU and FPGA between different quantization methods on the spectrum sensing dataset

| Method & Hardware Platform | Energy/Sample(mJ) |
|---|---|
| Our quantization method (GPU) | 1.632 |
| Our quantization method (FPGA) | 0.018 |
| Quantization Method in [4](FPGA) | 0.020 |

To illustrate the resource efficiency of the DFR over RNN, the resource utilization and inference speed of the FPDFR and FPRNN model are shown in Table X and XI, respectively. As demonstrated in Table X and XI, the resource utilization of the FPDFR is reduced significantly compared to the FPRNN.

TABLE X: Comparison of resource utilization between FPDFR, QDFR, and FPRNN

| Model | BRAM | DSP | FF | LUT |
|---|---|---|---|---|
| FPRNN | 1.00X | 1.00X | 1.00X | 1.00X |
| FPDFR | 1.00X | 0.75X | 0.75X | 0.77X |
| QDFR | 0.78X | 0.47X | 0.60X | 0.65X |

TABLE XI: Comparison of the inference speed between FPDFR, QDFR, and FPRNN

| Model | Inference Speed |
|---|---|
| FPRNN | 1.00X |
| FPDFR | 1.03X |
| QDFR | 2.38X |

## VIII. Conclusion

In this paper, a quantization approach is introduced to accelerate the inference speed and reduce the resource utilization of the DFR on FPGA. The FPGA implementation of the quantized DFR reduces the DSP, FF, and LUT utilization by 37%, 20%, and 15% respectively compared to the floating-point DFR on spectrum sensing. Besides, the quantized DFR improves the inference speed by approximately 2.3 times. Compare to the quantized DFR using the common quantization method, the proposed quantized DFR reduces the resource utilization of BRAM, FF, and LUT by 18%, 9%, and 15% respectively and achieves almost the same accuracy. We propose a new knowledge distillation called TSML to reduce quantization errors of a quantized model. The TSML addresses the issue of mismatched capacity between the teacher and student model. Also, it adapts the teacher model to the student model and quantization effects by transferring the knowledge from the student model to the teacher model. Finally, it enables knowledge distillation on small datasets with limited labeled data. With the help of TSML, the accuracy of the quantized DFR is improved by 2.39% on the spectrum sensing dataset at SNR=-20dB with Tx and Rx antennas is 4 compared to

the model without TSML. Meanwhile, the floating-point DFR trained with TSML achieves better accuracy than the RNN and reduces the resource consumption on FPGA.

## REFERENCES


[1] Hardik Sharma et al. "Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network". In: *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE. 2018, pp. 764–775.

[2] Eriko Nurvitadhi et al. "Accelerating binarized neural networks: Comparison of FPGA, CPU, GPU, and ASIC". In: *2016 International Conference on Field-Programmable Technology (FPT)*. IEEE. 2016, pp. 77–84.

[3] Shuchang Zhou et al. "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients". In: *arXiv preprint arXiv:1606.06160* (2016).

[4] Benoit Jacob et al. "Quantization and training of neural networks for efficient integer-arithmetic-only inference". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 2704–2713.

[5] Mohammad Rastegari et al. "Xnor-net: Imagenet classification using binary convolutional neural networks". In: *European Conference on Computer Vision*. Springer. 2016, pp. 525–542.

[6] Itay Hubara et al. "Binarized neural networks". In: *Advances in neural information processing systems*. 2016, pp. 4107–4115.

[7] Tomáš Mikolov et al. "Recurrent neural network based language model". In: *Eleventh annual conference of the international speech communication association*. 2010.

[8] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. "Learning to forget: Continual prediction with LSTM". In: (1999).

[9] Claudio Gallicchio et al. "Randomized Machine Learning Approaches: Recent Developments and Challenges." In: *ESANN*. 2017.

[10] Daniel Brunner, Miguel C Soriano, and Guy Van der Sande. *Photonic Reservoir Computing: Optical Recurrent Neural Networks*. Walter de Gruyter GmbH & Co KG, 2019.

[11] Kian Hamedani et al. "Reservoir computing meets smart grids: Attack detection using delayed feedback networks". In: *IEEE Transactions on Industrial Informatics* 14.2 (2017), pp. 734–743.

[12] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. "Distilling the knowledge in a neural network". In: *arXiv preprint arXiv:1503.02531* (2015).

[13] Mary Phuong and Christoph Lampert. "Towards understanding knowledge distillation". In: *International Conference on Machine Learning*. 2019, pp. 5142–5151.

[14] Takashi Fukuda et al. "Efficient Knowledge Distillation from an Ensemble of Teachers." In: *Interspeech*. 2017, pp. 3697–3701.

[15] Jang Hyun Cho and Bharath Hariharan. "On the efficacy of knowledge distillation". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2019, pp. 4794–4802.

[16] Seyed-Iman Mirzadeh et al. "Improved Knowledge Distillation via Teacher Assistant". In: *arXiv preprint arXiv:1902.03393* (2019).

[17] Adriana Romero et al. "Fitnets: Hints for thin deep nets". In: *arXiv preprint arXiv:1412.6550* (2014).

[18] Ying Zhang et al. "Deep mutual learning". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 4320–4328.

[19] Solomon Kullback and Richard A Leibler. "On information and sufficiency". In: *The annals of mathematical statistics* 22.1 (1951), pp. 79–86.

[20] Guobin Chen et al. "Learning efficient object detection models with knowledge distillation". In: *Advances in Neural Information Processing Systems*. 2017, pp. 742–751.

[21] Ilija Radosavovic et al. "Data distillation: Towards omni-supervised learning". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 4119–4128.

[22] Mark Everingham et al. "The pascal visual object classes (voc) challenge". In: *International journal of computer vision* 88.2 (2010), pp. 303–338.

[23] Tsung-Yi Lin et al. "Microsoft coco: Common objects in context". In: *European conference on computer vision*. Springer. 2014, pp. 740–755.

[24] Paul J Werbos. "Backpropagation through time: what it does and how to do it". In: *Proceedings of the IEEE* 78.10 (1990), pp. 1550–1560.

[25] Claudio Gallicchio, Alessio Micheli, and Luca Pedrelli. "Deep reservoir computing: A critical experimental analysis". In: *Neurocomputing* 268 (2017), pp. 87–99.

[26] Herbert Jaeger. "Echo state network". In: *scholarpedia* 2.9 (2007), p. 2330.

[27] Thomas Natschläger, Wolfgang Maass, and Henry Markram. "The" liquid computer": A novel strategy for real-time computing on time series". In: *Special issue on Foundations of Information Processing of TELEMATIK* 8.ARTICLE (2002), pp. 39–43.

[28] Nicholas D Haynes et al. "Reservoir computing with a single time-delay autonomous Boolean node". In: *Physical Review E* 91.2 (2015), p. 020801.

[29] Kian Hamedani et al. "Detecting dynamic attacks in smart grids using reservoir computing: A spiking delayed feedback reservoir based approach". In: *IEEE Transactions on Emerging Topics in Computational Intelligence* 4.3 (2019), pp. 253–264.

[30] Kangjun Bai and Yang Yi. "DFR: An energy-efficient analog delay feedback reservoir computing system for brain-inspired computing". In: *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 14.4 (2018), pp. 1–22.

[31] Zhouhan Lin et al. "Neural networks with few multiplications". In: *arXiv preprint arXiv:1510.03009* (2015).

[32] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[33] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation". In: *International Conference on Medical image computing and computer-assisted intervention*. Springer. 2015, pp. 234–241.

[34] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. "Estimating or propagating gradients through stochastic neurons for conditional computation". In: *arXiv preprint arXiv:1308.3432* (2013).

[35] Jingwei Xu and Gwan Choi. "Compressive sensing and reception for MIMO-OFDM based cognitive radio". In: *2015 International Conference on Computing, Networking and Communications (ICNC)*. IEEE. 2015, pp. 884–888.

[36] Arun Kumar and P NandhaKumar. "OFDM system with cyclostationary feature detection spectrum sensing". In: *ICT Express* 5.1 (2019), pp. 21–25.

[37] Tianyi Xiong et al. "Multiband spectrum sensing in cognitive radio networks with secondary user hardware limitation: Random and adaptive spectrum sensing strategies". In: *IEEE Transactions on Wireless Communications* 17.5 (2018), pp. 3018–3029.

[38] Ping-Rong Lin et al. "Cooperative spectrum sensing and optimization on multi-antenna energy detection in Rayleigh fading channel". In: *2018 27th Wireless and Optical Communication Conference (WOCC)*. IEEE. 2018, pp. 1–5.

[39] Chunxiao Jiang et al. "Machine learning paradigms for next-generation wireless networks". In: *IEEE Wireless Communications* 24.2 (2016), pp. 98–105.

[40] Karaputugala Madushan Thilina et al. "Machine learning techniques for cooperative spectrum sensing in cognitive radio networks". In: *IEEE Journal on selected areas in communications* 31.11 (2013), pp. 2209–2221.

[41] Charles Clancy et al. "Applications of machine learning to cognitive radio networks". In: *IEEE Wireless Communications* 14.4 (2007), pp. 47–52.

[42] Woongsup Lee, Minhoe Kim, and Dong-Ho Cho. "Deep cooperative sensing: Cooperative spectrum sensing based on convolutional neural networks". In: *IEEE Transactions on Vehicular Technology* 68.3 (2019), pp. 3005–3009.

[43] Timothy J O'Shea, Seth Hitefield, and Johnathan Corgan. "End-to-end radio traffic sequence recognition with recurrent neural networks". In: *2016 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*. IEEE. 2016, pp. 277–281.

[44] Kemal Davaslioglu and Yalin E Sagduyu. "Generative adversarial learning for spectrum sensing". In: *2018 IEEE International Conference on Communications (ICC)*. IEEE. 2018, pp. 1–6.

[45] Kian Hamedani et al. "Deep spiking delayed feedback reservoirs and its application in spectrum sensing of mimo-ofdm dynamic spectrum sharing". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 02. 2020, pp. 1292–1299.

[46] Hao Song, Xuming Fang, and Yuguang Fang. "Millimeter-wave network architectures for future high-speed railway communications: Challenges and solutions". In: *IEEE Wireless Communications* 23.6 (2016), pp. 114–122.

[47] Matthias Wellens, Alexandre de Baynast, and Petri Mahonen. "Exploiting historical spectrum occupancy information for adaptive spectrum sensing". In: *2008 IEEE Wireless Communications and Networking Conference*. IEEE. 2008, pp. 717–722.

[48] Vijaykumar Kuppusamy and Rajarshi Mahapatra. "Primary user detection in OFDM based MIMO cognitive radio". In: *2008 3rd International Conference on Cognitive Radio Oriented Wireless Networks and Communications (CrownCom 2008)*. IEEE. 2008, pp. 1–5.

[49] Hao Chen et al. "Optimal resource allocation for sensing-based spectrum sharing D2D networks". In: *Computers & Electrical Engineering* 44 (2015), pp. 107–121.

[50] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[51] Seyed Iman Mirzadeh et al. "Improved knowledge distillation via teacher assistant". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 04. 2020, pp. 5191–5198.

[52] Fabian Ruffy and Karanbir Chahal. "The state of knowledge distillation for classification". In: *arXiv preprint arXiv:1912.10850* (2019).

[53] Microsoft Research. *Neural Network Intelligence*. 2020. URL: https://github.com/microsoft/nni.

[54] Carl R Stevenson et al. "IEEE 802.22: The first cognitive radio wireless regional area network standard". In: *IEEE communications magazine* 47.1 (2009), pp. 130–138.