Bit-Level Quantization for Efficient Layout Hotspot Detection

Zehua Pei[1,2], Wenqian Zhao[1], Zhuolun He[1,2], Bei Yu[1]
[1]Chinese University of Hong Kong [2]Shanghai Artificial Intelligence Laboratory
{zehuapei,wqzhao,zlhe,byu}@cse.cuhk.edu.hk

***Abstract*—Hotspot detection is an essential step in the physical verification flow to identify layout patterns that are sensitive to process variations. Recent advances in machine learning have enabled deep neural networks (DNNs) to achieve good performance for hotspot detection; however, these models require a high computational complexity and memory footprint. To reduce such costs, quantization provides a promising solution by compressing DNNs into low-bit inference schemes. In this paper, we propose several quantization algorithms specifically designed for a classic neural network based hotspot detector while taking into account the feature distribution of the dataset used. Our experiments show that our compressed model can achieve competitive results compared with full-precision models while significantly reducing inference runtime at the same time.**

I. Introduction

With the semiconductor industry's rapid development, the transistor's feature size shrinks rapidly and the circuit becomes more complex, which brings about significant challenges to chip manufacturing. Despite various advanced Resolution Enhancement Techniques (RETs), designers may still introduce sensitive layout patterns that lead to manufacturing defects. Therefore, efficient and accurate hotspot detection during the physical verification stage is crucial to ensure the printability of layout designs.

Among different kinds of hotspot detection methods, machine learning based methods with deep neural networks have rapid development and have achieved great success. These methods employ deep neural networks with structures of various sizes and depths. For example, Yang *et al.* [1] construct a convolutional neural network with 14 convolution layers and 3 fully connected layers. The network in Geng *et al.* [2] employs several inception blocks and attention blocks as the backbone. Despite the high performance of machine learning based methods, deep neural networks contain lots of network parameters. They take a large memory footprint and require long inference runtime, especially when thousands of layout patterns are evaluated. Given the facts above, techniques that can reduce the network size and accelerate inference are in great demand. By considering input layout clips as binary images, Jiang *et al.* [3] employ a binarized neural network (BNN) to speed up the hotspot detection process. However, they design a BNN with a special network structure, and it is hard to utilize the compression technique on other already existing models.

In this paper, we propose to employ *quantization* to compress neural networks, which can flexibly reduce the bit width of the computing elements in any existing neural network. For example, it is feasible to reduce the 32-bit precision layer weights and the intermediate outputs between layers, called activations, to 8-bit integers. In subsequent sections, we give our formulation of quantization algorithms and show their efficiency for network compression. We also propose a quantization initialization algorithm by considering the feature distribution in hotspot detection tasks, where compact and convergence-benefited quantized data will be produced. In order to demonstrate the efficiency of quantization, we adopt the quantization algorithms on a neural network proposed by a classic hotspot detection work [4]. It is reasonable to perform the quantization algorithms on this network because it can be shown that quantization can still achieve remarkable performance on hotspot detection tasks, even for a small neural network. We also compare our proposed algorithms with other classic quantization algorithms, and the results show that ours can achieve superior performance in hotspot detection.

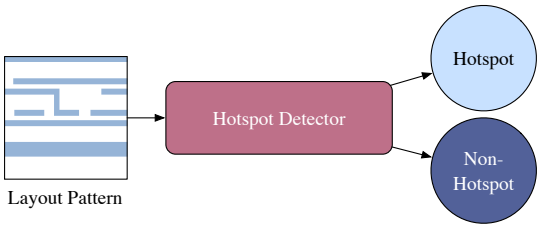

Fig. 1 Hotspot Detection Flow.

The contributions of this paper are summarized as follows:

- We come up with efficient bit-level quantization algorithms for hotspot detectors with or without retraining.
- A quantization initialization algorithm dedicated to hotspot detection is proposed by considering the feature distribution.
- Experimental results show that our algorithms can preserve inference accuracy (loss within 1%) while reducing 55.35% network runtime with an Int8 network.

The structure of this paper is organized as follows. Section II introduces related literature on hotspot detection and quantization. Section III recaps basic definitions and gives problem formulation. Section IV describes the details of our bit-level quantization formulation on the hotspot detector. Section V demonstrates the experimental results and comparisons with other algorithms, followed by the conclusion in Section VI.

II. Related Work

A. Hotspot Detection

Typically, there are three kinds of hotspot detection approaches, including lithography simulation, pattern matching and machine learning. *Lithography simulation* is a common method that can achieve high accuracy, but it is extremely time-consuming to perform over a whole chip. For *pattern matching* methods [5]–[8], they take a collection of hotspot layout patterns and use the collection to match patterns in new designs to identify the hotspots, which can speedup the hotspot detection flow and preserve the detection accuracy as much as possible. However, it is difficult for pattern matching methods to detect unknown hotspots that are not contained in the collection. *Machine learning* has achieved remarkable performance in modern chip manufacturing [1], [2], [4], [9]–[21]. For machine learning methods in hotspot detection, it can learn the hidden relationship between layout patterns and their defects characteristics, which enables strong generalization abilities to detect unknown hotspots.

One critical problem in machine learning based methods is how to effectively extract features of layout patterns. Zhang *et al.* [9] propose

an optimized concentric circle sampling (CCS) feature with an online learning scheme. Yang *et al.* [4] apply discrete cosine transformation (DCT) to extract features that are compatible with the emerging deep learning structure, and a biased learning technique is proposed to improve hotspot detection accuracy. Geng *et al.* [2] propose to jointly perform layout feature embedding and hotspot detection in an end-to-end manner, where an attention mechanism-based deep convolutional neural network is proposed. Another problem is about data augmentation. Yang *et al.* [1] apply hotspot upsampling and random-mirror flipping before training the network to relieve the imbalance issue of the training dataset in hotspot detection. Chen *et al.* [10] propose a semi-supervised hotspot detection with a self-paced multitask learning paradigm, where both data samples with and without labels are leveraged to improve model accuracy and generalizability.

Different from classifying the layout patterns, there are also some region-based methods to detect multiple hotspots in a large region at a time. Chen *et al.* [22] utilize joint auto-encoder and inception module to encode for feature extraction, and a two-stage classification and regression flow is proposed to locate hotspot regions. Zhu *et al.* [23] propose an end-to-end single hotspot detector without refining potential regions to detect the hotspots in large scales; multiple tasks are defined to detect the center and corner points of the hotspot regions.

B. Quantization

Quantization aims to reduce the bit-precision of key network structures, such as weights and activations, where 32-bit floating-point is the dominant numerical format for deep learning structures. The target of quantization is to preserve the accuracy as much as possible and to reduce the inference runtime [24]–[26].

One option of quantization is to perform post-training quantization (PTQ), which benefits only the inference of neural networks by taking a pre-trained network and implementing it in low-bit precision. Post-traning quantizaiton does not need training and can be data-free or require a series of calibration inputs. Instead of applying the Rounding-to-nearest approach, which rounds the weight vector $\boldsymbol{w}$ to the nearest representable quantization grid value in a fixed-point grid, Adaround [27] proposes a better weight-rounding mechanism for post-training quantization by analyzing the loss degradation with the second-order Taylor series expansion. BRECQ [28] leverages the basic building blocks in neural networks and reconstructs them one by one, which has achieved a good balance between cross-layer dependency and generalization error. Qdrop [29] analyzes the influence of incorporating activation quantization into weight tuning and has achieved flatness from a general perspective.

Quantization-aware training (QAT), where weights and activations are quantized during training, is another important quantization option. In quantization-aware training, the low-precision networks can be trained to improve accuracy by inserting some quantization operations into the neural networks, where high-precision weights and activations are updated in the backward pass to adapt to the low-precision weights and activations [30]–[32].

One main issue in QAT is determining a suitable clipping range or scale factor. PACT [31] learns the activation clipping range during training for finding the optimal quantization scale. QIL [33] learns the quantization intervals jointly with the weights by directly minimizing the task loss of the network. LSQ [32] proposes a novel means to estimate and scale the task loss gradient at each weight and activation layer's quantizer step size, such that it can be learned in conjunction with other network parameters.

Another important issue in QAT is dealing with the non-differentiable rounding operator in the backpropagation of the training process, where approximation methods should be used to address this problem. A conventional approach is the so-called Straight Through Estimator (STE) [34]. In DoReFa-Net [30], it ignores the rounding operator by STE and approximates it with an identity function. Despite the coarse approximation of STE, which makes no contribution to updating the latent weights without considering the quantization conditions, other approaches have also been explored. In DSQ [35], a series of hyperbolic tangent functions are used to approach the standard quantization gradually. EWGS [36] adaptively scales up or down each gradient element and uses the scaled gradient as the one for the discretizer input to train quantized networks via backpropagation. OCTAV [37] finds optimal clipping scalars on the fly by the fast Newton-Raphson method and proposes magnitude-aware differentiation as a remedy to improve accuracy further.

III. Preliminaries

In this section, some terminology and preliminary knowledge of layout hotspot detection and quantization are introduced.

A. Hotspot Detection

The process of transferring designed patterns onto silicon wafers in chip manufacturing is called the lithographic process. However, this process may involve various variations, and some patterns are sensitive to these variations and may cause a reduction of manufacturing yield as a result of potential open or short circuit failures. Layout patterns that are sensitive to process variations are defined as hotspots. A hotspot clip is defined as a clip that includes at least one hotspot in its core region. The overall flow of hotspot detection is illustrated in Fig. 1. The following definitions and metrics are used to evaluate the performance of a hotspot detector in this paper.

Definition 1 (Accuracy)**.** *The ratio between the number of correctly predicted hotspot clips and the number of groundtruth hotspot clips.*

Definition 2 (False Alarm)**.** *The number of non-hotspot clips that are predicted as hotspots by the classifier.*

It should be noted that the accuracy is also equivalent to the true positive rate, and the false alarm corresponds to the number of false positives. In the hotspot detection process, it is important to accurately identify as many hotspots as possible and not mistakenly identify non-hotspots as hotspots.

In this paper, we mainly focus on the network runtime for layout pattern detection, which is one of the important evaluations for quantized networks.

With the evaluation metrics defined above, we formulate the hotspot detection problem as follows:

Problem 1 (Hotspot Detection)**.** *Given a set of clips consisting of hotspot and non-hotspot patterns, the objective of hotspot detection is to train a classifier that can maximize the Accuracy and minimize the False Alarm and network runtime.*

B. Quantization

Consider some neural network F. Typically, the B-bit quantization is the process to transfer the weights $\boldsymbol{W}$ and activations $\boldsymbol{A}$ of the neural network F from full-precision to B-bit precision.

Consider some scalar data x, which is also called the latent data. The quantized data $\mathbb{Q}(x)$ is defined as:

$$\mathbb{Q}(x) = \text{clip}((r/2^{B-1}) \cdot \text{round}(x \cdot 2^{B-1}/r), -r, r), \quad (1)$$

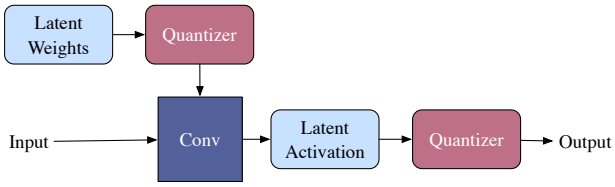

Fig. 2 Quantization flow of a convolution layer.

where the data x is first scaled by some algorithm-dependent scale factor $2^{B-1}/r$ and mapped into the integer space, with r being the clipping range in the original float space. Later, it is dequantized back to the floating point number space by rescaling. It should be noted that a clipping operation is applied to remove the quantization noise, which is defined as:

$$\text{clip}(x, l, u) = \begin{cases} l, & x < l; \\ x, & l \leq x \leq u; \\ u, & x > u. \end{cases} \tag{2}$$

In Fig. 2, a detailed flow of quantization on a convolution layer is illustrated.

The data distribution is an important condition for determining the scale factor. For unsigned data, such as the data in activations, only one side of the data needs to be scaled and clipped.

A key problem in quantization-aware training is dealing with the non-differentiable rounding operator during the backward pass. Researchers often use a straight-through estimator (STE) [34] to address this problem. By using STE, the gradient of the rounding operator is approximated as 1:

$$\frac{\partial\ \text{round}(x)}{\partial x} = 1, \tag{3}$$

which results in the fact that the gradient of quantized data is 1 w.r.t the latent data [38].

In this paper, we consider two schemes of quantization, including both quantization-aware training and post-training quantization.

IV. Algorithms

In this section, we will introduce the hotspot detection architecture by Yang *et al.* [4] and our quantization formulation. For quantization, we will discuss quantization-aware training and post-training quantization separately in order to give a clear perspective on how they are implemented.

A. DNN-based Hotspot Detector

We determine to utilize a deep-learning flow for hotspot detection task based on DAC'17 [4]. The network input is the feature tensor $\boldsymbol{X} \in \mathbb{R}^{n \times n \times k}$, which is transformed from the layout clip $\boldsymbol{I} \in \mathbb{R}^{N \times N}$ with the discrete cosine transform (DCT) and a series of transformations.

Given $\boldsymbol{X}$ as input, the neural network we will quantize is a convolutional neural network that contains two convolution stages for feature extraction and two fully connected (FC) layers for probability generation, which are demonstrated in Fig. 3. A convolution stage contains two convolution layers, and each convolution layer is followed by a ReLU activation function, and a max-pooling layer is inserted at the end of the stage. The convolution kernel size is 3×3, and the channel dimension in the two convolution stages are 16 and 32, respectively. Max-pooling layers employ a pooling window shape of 2×2. The output feature sizes of two fully connected layers are 250 and 2, respectively.

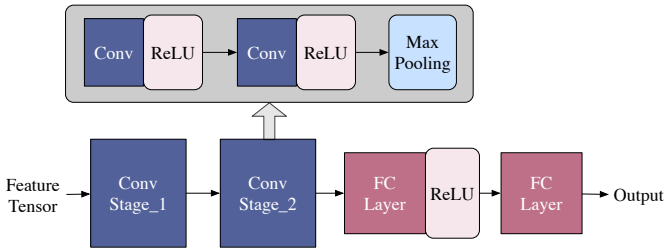

Fig. 3 The network structure of DAC'17 [4]. The shaded part denotes the convolution stage.

In this paper, all the convolution layers and fully connected layers will be quantized. We follow the classic quantization flow by inserting a quantizer after the latent data before calculation. As the example of quantized convolution layer in Fig. 2, latent weights are first inputted into the quantizer and then convoluted with the input, which can be the network input $\boldsymbol{X}$ or intermediate quantized activations.

B. Quantization-Aware Training

We propose a bit-level quantization mechanism to accelerate our hotspot detection flow by quantizing all full-precision operations into lower-precision alternatives such as FP16 and Int8.

The first approach is to conduct quantization-aware training (QAT) to retrieve the accuracy drop from quantization. In QAT, the objective is to update and optimize the latent weights during training with quantized weights and activations.

The bridge between the latent and quantized data is the scale factor, defined as s. The approach to determine the scale factor, or clipping range, is crucial for QAT to deal with the quantization noise, also called the outliers. In this work, we would like to design an appropriate determination of scale factor s. Instead of applying the maximum value or an offline scale factor, we adopt a learnable scale factor that will be updated together with the network parameters. Let's define $w \in \boldsymbol{W}$ as a real number in the network weight and B as the quantization bit-width. Inspired by Esser *et al.* [32], we define the number of positive and negative quantization levels L_P and L_N respectively, and then the quantization QAT is formulated as follows:

$$\mathbb{Q}_{QAT}(w) = s \cdot \text{round}(\text{clip}(\frac{w}{s}, L_N, L_P)). \tag{4}$$

The number of positive and negative quantization levels depends on the quantization bit-width B and the data that is quantized. For weights, $L_N = -2^{B-1}$ and $L_P = 2^{B-1} - 1$. For activations, $L_N = 0$ and $L_P = 2^B - 1$.

Since the latent weights are stored and updated, and the quantized weights and activations are used for forward and backward passes, the back-propagation from quantized weights to latent weights in the backward pass must go through the rounding operator, which is, however, non-differentiable. Here we employ the straight through estimator [34] by passing through the gradient to solve this issue:

$$\frac{\partial \mathbb{Q}_{QAT}(w)}{\partial w} = \begin{cases} 1, & L_N < w < L_P; \\ 0, & \text{otherwise.} \end{cases} \tag{5}$$

When updating the scale factor, we expect a more effective approximation. Therefore, we employ the method in Esser *et al.* [32]

as follows:

$$\frac{\partial \mathbb{Q}_{QAT}(w)}{\partial s} = \begin{cases} L_N, & \frac{w}{s} \leq L_N; \\ -\frac{w}{s} + \text{round}(\frac{w}{s}), & L_N < \frac{w}{s} < L_P; \\ L_P, & \frac{w}{s} \geq L_P, \end{cases} \quad (6)$$

where the relative proximity of w to the transition point is proposed to learn the scale factor s. In this way, the closer s is to the quantization transition point (where the quantized value changes), the larger the gradient $\frac{\partial \mathbb{Q}_{QAT}(w)}{\partial s}$ is given.

When we quantize activations $\boldsymbol{A}$ and update the scale factor of $\boldsymbol{A}$, we use the same formulation in Equation (4) and Equation (6) by replacing the real number w to a real number $a \in \boldsymbol{A}$.

C. Post-Training Quantization

We also propose the algorithm to quantize the hotspot detection network by post-training quantization to save the effort of retraining. In PTQ, we pay attention to the strategy of rounding the quantized values, i.e., either round up or down.

Inspired by Nagel *et al.* [27], we abandon the rounding-to-nearest strategy, which is sub-optimal for the entire network or the whole layer. Given a pre-trained network with the input $\boldsymbol{x}$ and the target $\boldsymbol{y}$, let Δw denote a small perturbation, and $\mathcal{L}(\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{w})$ denote the task loss; we analyze and approximate the loss degradation caused by quantization with Taylor series expansions:

$$\mathbb{E}[\mathcal{L}(\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{w} + \Delta \boldsymbol{w}) - \mathcal{L}(\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{w})] \approx \frac{1}{2} \Delta \boldsymbol{w}^\top \cdot \boldsymbol{H}^{(\boldsymbol{w})} \cdot \Delta \boldsymbol{w}, \quad (7)$$

where $\boldsymbol{w}$ is a flattened version of weight $\boldsymbol{W}$ and $\boldsymbol{H}^{(\boldsymbol{w})}$ denotes the expected Hessian of the task loss $\mathcal{L}$ w.r.t. $\boldsymbol{w}$.

Since calculating Hessian for each input data point is time-consuming, we employ an approximation inspired by Li *et al.* [28]. The Taylor second-order error can be transformed into final network outputs. Denote the neural network output as $\boldsymbol{z}_n$, and the stacked vector of weights in all n layers as $\theta = \text{vec}[\boldsymbol{w}_1^\top, ..., \boldsymbol{w}_n^\top]^\top$. It can be proved that:

$$\min_{\hat{\theta}} \Delta \theta^\top \boldsymbol{H}^{(\theta)} \Delta \theta \approx \min_{\hat{\theta}} \mathbb{E}[\Delta \boldsymbol{z}_n^\top H^{(\boldsymbol{z}_n)} \Delta \boldsymbol{z}_n], \quad (8)$$

where H is the hessian without expectation. Hence we can evaluate the error according to the change in network outputs with the output Hessian $H^{(\boldsymbol{z}_n)}$. To further improve the performance, we propose the stage-diagonal Hessian in our network. Assume that a convolution stage is formed from layer k to layer l, stage-diagonal Hessian considers the layers inside a stage $\tilde{\theta} = \text{vec}[\boldsymbol{w}_k^\top, ..., \boldsymbol{w}_l^\top]^\top$ and the intermediate stage output $\boldsymbol{z}_l$. And the two fully connected layers are still evaluated according to the network outputs. As in Li *et al.* [28], we further transform the objective by the diagonal Fisher Information Matrix (FIM):

$$\min_{\hat{\theta}} \mathbb{E}[\Delta \boldsymbol{z}_l^\top H^{(\boldsymbol{z}_l)} \Delta \boldsymbol{z}_l] = \min_{\hat{\theta}} \mathbb{E}[\Delta \boldsymbol{z}_l^\top \text{diag}((\frac{\partial \mathcal{L}}{\boldsymbol{z}_{l,1}})^2, ..., (\frac{\partial \mathcal{L}}{\boldsymbol{z}_{l,a}})^2) \Delta \boldsymbol{z}_l]. \quad (9)$$

Finally, we formulate the quantization as:

$$\mathbb{Q}_{PTQ}(w) = s \cdot \text{clip}(\lfloor \frac{w}{s} \rfloor + h(v), L_N, L_P), \quad (10)$$

where the weights are first rounded by the floor operation and

$$h(v) = \text{clip}(\sigma(v)(\zeta - \gamma) + \gamma, 0, 1). \quad (11)$$

$v \in \boldsymbol{V}$ is the variable to optimize and $h(v)$ is the rectified sigmoid proposed in Louizos *et al.* [39], where $\sigma(\cdot)$ is the sigmoid function and ζ and γ are stretch parameters, fixed to 1.1 and -0.1, respectively. A regularization term is added to the objective function:

$$f_{reg}(\boldsymbol{V}) = \sum_{i,j} 1 - |2h(\boldsymbol{V}_{i,j}) - 1|^\beta, \quad (12)$$

where a progressively decreasing β makes the $h(v)$ to be either 0 or 1 and hence determinates the weights to be rounded up or down.

Furthermore, we would like to consider the noise caused by activation quantization. Inspired by Wei *et al.* [29], we model the activation noise into a multiplicative form, i.e., $\hat{a} = a \cdot (1 + u)$. Therefore, $\mathbf{1} + \mathbf{u}(\boldsymbol{X})$ is adopted to present the activation noise, which depends on specific input data point $\boldsymbol{X}$. Then the objective is reformulated as follows:

$$\min_{\hat{\boldsymbol{w}}} \mathbb{E}[\mathcal{L}(\boldsymbol{w} + \Delta \boldsymbol{w}, \boldsymbol{X}, \mathbf{1} + \mathbf{u}(\boldsymbol{X})) - \mathcal{L}(\boldsymbol{w}, \boldsymbol{X}, \mathbf{1})]. \quad (13)$$

In order to increase the flatness in as many directions as possible, we randomly disable and enable the quantization of the activation as in Wei *et al.* [29]:

$$u = \begin{cases} 0, & \text{with probability } p; \\ \frac{\hat{a}}{a} - 1, & \text{with probability } 1 - p. \end{cases} \quad (14)$$

D. Initialization of Scale Factor

Parameter initialization is an important but easily neglected issue in quantization, where the initialization of the scale factor is most representative. Fundamentally, better initialization of the learnable scale factor in QAT is helpful in finding a better local optimum. A widely used initialization strategy is by statistics, where we denote μ as the mean and σ as the standard deviation of activations (weights) in that layer. For LSQ [32], the scale factor s_{lsq} is initialized as $2 * \mu_{|x|} / \sqrt{L_P}$. In LSQ+ [40], an improved initialization for weight scale factor is set as $\max(|\mu_x - 3\sigma_x|, |\mu_x + 3\sigma_x|) / 2^{B-1}$. By observation, the data distribution of activations in the hotspot detection task is relatively compact with the binary layout patterns input, and hence a wide range initialization will be far from the converged values. Therefore, in this paper, we proposed a more effective initialization for hotspot detection task:

$$s_{ours} = (\mu_{|x|} + 2\sigma_{|x|}) / 2^{B-1}, \quad (15)$$

where we directly compute the mean and standard deviation of the absolute value of activations (weights) in that layer. In this way, the initialized range can take the negative part of the data into consideration and will produce convergence-benefited quantized data when separating most of the outliers. There exists the same problem of scale initialization in PTQ, where the scale factor almost makes use of the maximum scale factor or simply employs the same strategy as QAT. Therefore, our proposed initialization strategy can also be conducted on PTQ. In experiments, we use the algorithms in previous sections with our initialization algorithm for both QAT and PTQ.

V. Experiments

A. Experimental Setup and Implement Details

We implement the framework of Yang *et al.* [4] in Python, and we reproduce it with Pytorch. We test the results on a machine with CPU type Intel(R) Xeon(R) CPU E5-2698 v4 @ 2.20GHz and an Nvidia GTX 2080 Ti GPU. To be consistent with previous work, the experiments are conducted on the ICCAD 2012 benchmark [42], where different benchmarks and a dataset with all the $28nm$ patterns merged are evaluated to verify the scalability. The statistics of ICCAD 2012 benchmark are shown in TABLE II. Row 'Merged' denotes the merged benchmark of all the $28nm$ patterns, i.e. from benchmark2

TABLE I Experimental Results. 'FP32' and 'FP16' denote the full-precision and half-precision network, respectively. 'QAT' and 'PTQ' are Quantization-Aware Training and Post-Training Quantization, respectively, with the 8-bit precision network.

Benchmark	FP32 [4]			FP16 [41]			Our QAT			Our PTQ		
	Acc (%)	FA (%)	Runtime (ms)	Acc (%)	FA (%)	Runtime (ms)	Acc (%)	FA (%)	Runtime (ms)	Acc (%)	FA (%)	Runtime (ms)
Benchmark1	100.00	33.08	0.56	100.00	33.27	0.34	100.00	33.09	0.25	100.00	33.12	0.25
Benchmark2	98.59	2.76		98.51	2.67		98.59	2.66		98.59	2.78	
Benchmark3	98.89	10.57		98.83	10.69		98.88	10.86		98.89	10.66	
Benchmark4	90.40	4.19		88.13	3.50		89.78	3.91		89.83	4.12	
Benchmark5	97.56	3.52		97.55	3.77		97.56	2.58		97.51	3.48	
Merged	97.54	3.17		96.39	3.22		97.46	3.05		97.50	3.14	

TABLE II Benchmark Statistics of ICCAD 2012.

Benchmark	#Train HS	#Train NHS	#Test HS	#Test NHS
Benchmark1	99	340	226	319
Benchmark2	174	5285	498	4146
Benchmark3	909	4643	1808	3541
Benchmark4	95	4452	177	3386
Benchmark5	26	2716	41	2111
Merged	1204	17096	2524	13184

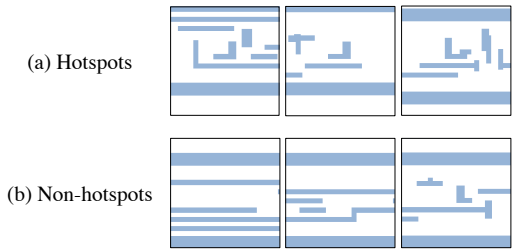

Fig. 4 (a) Hotspots and (b) Non-hotspots in ICCAD 2012 benchmark.

to benchmark5. Columns '#Train HS' and '#Train NHS' list the total number of hotspots and the total number of non-hotspots in the training set. Columns '#Test HS' and '#Test NHS' list the total number of hotspots and the total number of non-hotspots in the testing set. In Fig. 4, we demonstrate some layout examples of hotspots and non-hotspots in ICCAD 2012 benchmark. We implement the quantization algorithms and deploy the quantized neural networks with the help of a Quantization benchmark named MQBench [41]. The GPU-accelerated inference and runtime evaluation of quantized neural networks is conducted on TensorRT, which is a high-performance deep learning inference engine.

The neural network in Yang *et al.* [4] consists of four convolution layers and two fully connected layers. Different from the original configuration, we insert Batch Normalization (BN) between each convolution layer and the ReLU activation function, and we also remove the dropout on the first fully connected layer. In the training of the full-precision network, we use the Adam optimizer [43] with an initial learning rate of 0.001 and iteration of 10000. We employ the cosine learning rate decay [44] to update our learning rate. For quantization-aware training, we employ the same optimizer and learning rate scheduler as in full-precision training. The difference is that we use an initial learning rate of 0.0001 and an iteration of 2000. For post-training quantization, we use 25% of the training set for calibration.

B. Experimental Results

To evaluate the performance of quantization on hotspot detection, we apply our proposed algorithms in Section IV to quantize the neural network of Yang *et al.* [4] into an 8-bit precision network on the ICCAD 2012 benchmark. To demonstrate the high efficiency of quantization on runtime reduction, we use the inference time of a batch size of 1000 layout patterns on TensorRT.

TABLE III Quantization-Aware Training Results.

Method	Accuracy (%)	False Alarm (%)
DoReFa [30]	93.62	9.46
PACT [31]	97.06	9.01
DSQ [35]	97.42	4.34
LSQ [32]	96.24	2.86
Our QAT	97.46	3.05

TABLE IV Post-Training Quantization Results.

Method	Accuracy (%)	False Alarm (%)
AdaRound [27]	97.20	3.16
BRECQ [28]	97.33	3.13
Qdrop [29]	97.41	3.19
Our PTQ	97.50	3.14

The experiment results are listed in TABLE I. 'FP32' denotes the results with the original full-precision network of DAC'17 [4], while 'FP16' denotes the results with a half-precision network implemented by the default QAT setting of MQBench [41]. 'QAT' means quantization-aware training, while 'PTQ' means post-training quantization. Columns 'Acc (%)', 'FA (%)', 'Runtime (ms)' denote the hotspot detection accuracy rate, false alarm rate, and network runtime, respectively. The results demonstrate that 8-bit quantization can preserve the reduction of accuracy of hotspot detection within 1% compared to the full-precision model. With well-designed quantization algorithms, the accuracy of 8-bit quantization can even be better than a half-precision network. In the meantime, the false alarm rate of the 8-bit precision network is also maintained in an acceptable range. The most notable result is the runtime improvement of the 8-bit precision network, which reduces about 55.35% runtime compared with the original full-precision network. The results show that quantization can achieve remarkable runtime reduction even with the small network structure as in Fig. 3.

We also compare our proposed algorithm with other classic quantization algorithms. For quantization-aware training, we compare the performance of DoReFa-Net [30], PACT [31], DSQ [35] and LSQ [32]. For post-training quantization, the algorithms of AdaRound [27], BRECQ [28] and Qdrop [29] are compared on bit-8 precision. The results in TABLE III and TABLE IV show that our algorithms can maintain better accuracy compared with other algorithms.

Furthermore, sometimes we find that some layers in the neural networks are sensitive to quantization, and high accuracy degradation will be caused when these layers are quantized. Therefore, we may consider keeping the precision of these layers, which is called mixed-precision quantization. It is an interesting direction for the quantization of hotspot detection in the future.

VI. Conclusion

Recently, hotspot detection tasks have seen remarkable development by utilizing deep learning networks. However, deeper and deeper networks come with high computational complexity and expensive memory cost. In this paper, we employ quantization methods to address this problem by compressing a classic hotspot detection model. Generally, we propose quantization-aware training and post-training quantization algorithms and propose a quantization initialization algorithm dedicated to hotspot detection. The experimental results show that the final quantized model can achieve competitive performance and can significantly reduce the inference runtime.

Acknowledgments

This work is supported The Research Grants Council of Hong Kong SAR (Project No. CUHK14208021).

References

[1] H. Yang, L. Luo, J. Su, C. Lin, and B. Yu, "Imbalance aware lithography hotspot detection: a deep learning approach," *JM3*, vol. 16, no. 3, p. 033504, 2017.

[2] H. Geng, H. Yang, L. Zhang, J. Miao, F. Yang, X. Zeng, and B. Yu, "Hotspot detection via attention-based deep layout metric learning," in *Proc. ICCAD*, 2020.

[3] Y. Jiang, F. Yang, H. Zhu, B. Yu, D. Zhou, and X. Zeng, "Efficient Layout Hotspot Detection via Binarized Residual Neural Network," in *Proc. DAC*, 2019, pp. 147:1–147:6.

[4] H. Yang, J. Su, Y. Zou, B. Yu, and E. F. Y. Young, "Layout hotspot detection with feature tensor generation and deep biased learning," in *Proc. DAC*, 2017, pp. 62:1–62:6.

[5] S.-Y. Lin, J.-Y. Chen, J.-C. Li, W.-Y. Wen, and S.-C. Chang, "A novel fuzzy matching model for lithography hotspot detection," in *Proc. DAC*, 2013, pp. 68:1–68:6.

[6] W.-Y. Wen, J.-C. Li, S.-Y. Lin, J.-Y. Chen, and S.-C. Chang, "A fuzzy-matching model with grid reduction for lithography hotspot detection," *IEEE TCAD*, vol. 33, no. 11, pp. 1671–1680, 2014.

[7] Y.-T. Yu, Y.-C. Chan, S. Sinha, I. H.-R. Jiang, and C. Chiang, "Accurate process-hotspot detection using critical design rule extraction," in *Proc. DAC*, 2012, pp. 1167–1172.

[8] S. S.-E. Tseng, W.-C. Chang, I. H.-R. Jiang, J. Zhu, and J. P. Shiely, "Efficient search of layout hotspot patterns for matching SEM images using multilevel pixelation," in *Proc. SPIE*, vol. 10961, 2019.

[9] H. Zhang, B. Yu, and E. F. Y. Young, "Enabling online learning in lithography hotspot detection with information-theoretic feature optimization," in *Proc. ICCAD*, 2016, pp. 47:1–47:8.

[10] Y. Chen, Y. Lin, T. Gai, Y. Su, Y. Wei, and D. Z. Pan, "Semi-supervised hotspot detection with self-paced multi-task learning," in *Proc. ASPDAC*, 2019, pp. 420–425.

[11] D. Ding, J. A. Torres, and D. Z. Pan, "High performance lithography hotspot detection with successively refined pattern identifications and machine learning," *IEEE TCAD*, vol. 30, no. 11, pp. 1621–1634, 2011.

[12] H. Geng, H. Yang, B. Yu, X. Li, and X. Zeng, "Sparse VLSI layout feature extraction: A dictionary learning approach," in *Proc. ISVLSI*, 2018, pp. 488–493.

[13] H. Yang, S. Zhang, K. Liu, S. Liu, B. Tan, R. Karri, S. Garg, B. Yu, and E. F. Young, "Attacking a CNN-based Layout Hotspot Detector Using Group Gradient Method," in *Proc. ASPDAC*, 2021, pp. 885–891.

[14] H. Yang, Z. Li, K. Sastry, S. Mukhopadhyay, M. Kilgard, A. Anandkumar, B. Khailany, V. Singh, and H. Ren, "Generic Lithography Modeling with Dual-band Optics-Inspired Neural Networks," in *Proc. DAC*, 2022, pp. 973–978.

[15] H. Yang, P. Pathak, F. Gennari, Y.-C. Lai, and B. Yu, "Detecting Multi-Layer Layout Hotspots with Adaptive Squish Patterns," in *Proc. ASP-DAC*, 2019, pp. 299–304.

[16] H. Yang, S. Li, Y. Ma, B. Yu, and E. F. Young, "GAN-OPC: Mask optimization with lithography-guided generative adversarial nets," in *Proc. DAC*, 2018, pp. 1–6.

[17] W. Zhao, X. Yao, Z. Yu, G. Chen, Y. Ma, B. Yu, and M. D. Wong, "AdaOPC: A Self-Adaptive Mask Optimization Framework For Real Design Patterns," in *Proc. ICCAD*, 2022, pp. 1–9.

[18] G. Chen, W. Chen, Y. Ma, H. Yang, and B. Yu, "DAMO: Deep agile mask optimization for full chip scale," in *Proc. ICCAD*, 2020, pp. 1–9.

[19] R. Chen, S. Hu, Z. Chen, S. Zhu, B. Yu, P. Li, C. Chen, Y. Huang, and J. Hao, "A unified framework for layout pattern analysis with deep causal estimation," in *Proc. ICCAD*, 2021, pp. 1–9.

[20] G. Chen, Z. Yu, H. Liu, Y. Ma, and B. Yu, "Develset: Deep neural level set for instant mask optimization," in *Proc. ICCAD*, 2021, pp. 1–9.

[21] Z. Yu, G. Chen, Y. Ma, and B. Yu, "A GPU-enabled level set method for mask optimization," in *Proc. DATE*, 2021.

[22] R. Chen, W. Zhong, H. Yang, H. Geng, X. Zeng, and B. Yu, "Faster Region-based Hotspot Detection," in *Proc. DAC*, 2019, pp. 146:1–146:6.

[23] B. Zhu, R. Chen, X. Zhang, F. Yang, X. Zeng, B. Yu, and M. D. Wong, "Hotspot detection via multi-task learning and transformer encoder," in *Proc. ICCAD*, 2021, pp. 1–8.

[24] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," *Proc. NIPS*, vol. 28, 2015.

[25] F. Li, B. Zhang, and B. Liu, "Ternary weight networks," *arXiv preprint*, 2016.

[26] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, "Incremental Network Quantization: Towards Lossless CNNs with Low-precision Weights," in *Proc. ICLR*, 2017.

[27] M. Nagel, R. A. Amjad, M. Van Baalen, C. Louizos, and T. Blankevoort, "Up or Down? Adaptive Rounding for Post-Training Quantization," in *Proc. ICML*, 2020, pp. 7197–7206.

[28] Y. Li, R. Gong, X. Tan, Y. Yang, P. Hu, Q. Zhang, F. Yu, W. Wang, and S. Gu, "BRECQ: Pushing the Limit of Post-Training Quantization by Block Reconstruction," in *Proc. ICLR*, 2021.

[29] X. Wei, R. Gong, Y. Li, X. Liu, and F. Yu, "QDrop: Randomly Dropping Quantization for Extremely Low-bit Post-Training Quantization," in *Proc. ICLR*, 2022.

[30] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, "DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients," *arXiv preprint*, pp. arXiv–1606, 2016.

[31] J. Choi, Z. Wang, S. Venkataramani, P. I.-J. Chuang, V. Srinivasan, and K. Gopalakrishnan, "PACT: Parameterized Clipping Activation for Quantized Neural Networks," *arXiv preprint*, 2018.

[32] S. K. Esser, J. L. McKinstry, D. Bablani, R. Appuswamy, and D. S. Modha, "Learned Step Size Quantization," in *Proc. ICLR*, 2020.

[33] S. Jung, C. Son, S. Lee, J. Son, J.-J. Han, Y. Kwak, S. J. Hwang, and C. Choi, "Learning to Quantize Deep Networks by Optimizing Quantization Intervals with Task Loss," in *Proc. CVPR*, 2019, pp. 4350–4359.

[34] Y. Bengio, N. Léonard, and A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," *arXiv preprint*, 2013.

[35] R. Gong, X. Liu, S. Jiang, T. Li, P. Hu, J. Lin, F. Yu, and J. Yan, "Differentiable Soft Quantization: Bridging Full-Precision and Low-Bit Neural Networks," in *Proc. ICCV*, 2019, pp. 4852–4861.

[36] J. Lee, D. Kim, and B. Ham, "Network Quantization with Element-wise Gradient Scaling," in *Proc. CVPR*, 2021, pp. 6448–6457.

[37] C. Sakr, S. Dai, R. Venkatesan, B. Zimmer, W. Dally, and B. Khailany, "Optimal Clipping and Magnitude-aware Differentiation for Improved Quantization-aware Training," in *Proc. ICML*, 2022, pp. 19 123–19 138.

[38] H. Wu, P. Judd, X. Zhang, M. Isaev, and P. Micikevicius, "Integer Quantization for Deep Learning Inference: Principles and Empirical Evaluation," *arXiv preprint*, 2020.

[39] C. Louizos, M. Welling, and D. P. Kingma, "Learning Sparse Neural Networks through L_0 Regularization," in *Proc. ICLR*, 2018.

[40] Y. Bhalgat, J. Lee, M. Nagel, T. Blankevoort, and N. Kwak, "LSQ+: Improving low-bit quantization through learnable offsets and better initialization," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2020, pp. 2978–2985.

[41] Y. Li, M. Shen, J. Ma, Y. Ren, M. Zhao, Q. Zhang, R. Gong, F. Yu, and J. Yan, "MQBench: Towards Reproducible and Deployable Model Quantization Benchmark," in *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021.

[42] A. J. Torres, "ICCAD-2012 CAD contest in fuzzy pattern matching for physical verification and benchmark suite," in *Proc. ICCAD*, 2012, pp. 349–350.

[43] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *Proc. ICLR*, 2015.

[44] I. Loshchilov and F. Hutter, "SGDR: Stochastic Gradient Descent with Warm Restarts," in *Proc. ICLR*, 2017.