BCNN: Binary Complex Neural Network

Yanfei Li†, Tong Geng‡, Ang Li‡, and Huimin Yu†
†Zhejiang University, Hangzhou, Zhejiang, China
‡ Pacific Northwest National Laboratory, Richland, WA, USA
aoxue18@zju.edu.cn, tong.geng@pnnl.gov, ang.li@pnnl.gov, yhm2005@zju.edu.cn

***Abstract*—Binarized neural networks, or BNNs, show great promise in edge-side applications with resource limited hardware, but raise the concerns of reduced accuracy. Motivated by the complex neural networks, in this paper we introduce complex representation into the BNNs and propose Binary complex neural network – a novel network design that processes binary complex inputs and weights through complex convolution, but still can harvest the extraordinary computation efficiency of BNNs. To ensure fast convergence rate, we propose novel BCNN based batch normalization function and weight initialization function. Experimental results on Cifar10 and ImageNet using state-of-the-art network models (e.g., ResNet, ResNetE and NIN) show that BCNN can achieve better accuracy compared to the original BNN models. BCNN improves BNN by strengthening its learning capability through complex representation and extending its applicability to complex-valued input data. The source code of BCNN will be released on GitHub.**

***Index Terms*—Binarized network networks, complex neural networks, smart edges, complex number**

I. Introduction

Deep neural networks (DNNs) recently achieved tremendous success in many computer vision applications. Aiming at replicating similar success for practical edge-side utilization, researchers are tweaking the DNN models for resources limited hardware. Binary neural networks (BNNs) [1], [2], which adopt only a bit for a neuron, stand out as one of the most promising approaches. To accommodate constrained hardware budget, rather than deducting the number of neurons in a model, BNN reduces the number of bits per neuron to the extreme — each element of the input, weight and activation of a BNN layer is merely a single binary value, implying +1 or -1.

BNNs have demonstrated several appealing features for embedded utilization: (i) *Execution efficiency:* the natural mapping from a BNN neuron to a digital bit makes BNN extremely hardware-friendly. From the computation perspective, each 32 or 64 full-precision dot-product calculations can be aggregated into a Boolean exclusive-or plus a population-count operation [1], improving the execution efficiency by more than 10x [3]. From the memory perspective, rather than using a 32-bit single-precision floating-point, or a 16-bit half-precision floating-point, in BNNs each neuron uses one bit, substantially improving the utilization of the memory storage and bandwidth [4]. The combination of the two effects can bring over three orders of latency reduction for single image inference compared to full-precision DNNs on GPUs [3]. (ii) *Low-cost:* Due to simpler hardware logic (e.g., avoiding using floating-point multiplier) and diminished memory demand, the hardware cost for implementing BNNs is much lower than DNNs [5], [6]. (iii) *Energy-efficiency:* Due to low-cost hardware operations and smaller chip-area [7], BNN-based designs are very friendly to portable devices with limited cycle life of batteries. (iv) *Robustness*: Due to the discrete parameter space through the binarization of weight, BNN shows better robustness than normal DNNs [8], while certain properties can be formally verified [9], [10]. Because of these advantages, BNNs have been utilized for a variety of practical applications, such as auto-driving [11], COVID face-cover detection [12], smart agriculture [13], image enhancement [14], 3D objection detection [15], etc.

Although BNNs exhibit these attractive features, concerns have been raised over the reduced accuracy compared to DNNs, which is largely due to the loss of information in the binarization process, and the reduced model capacity. Ever since the proposal of BNNs, continuous efforts have been invested from the machine learning community on improving BNN accuracy [16]–[25], as briefly summarized in the next section.

In the meanwhile, complex-valued neural networks [26] have been proposed as an amendment to the normal DNNs. Most existing DNNs adopt real-valued representation for the inputs and weights. However, considering the richer representational capacity [27], the better generalization capability [28], and the potential to facilitate noise-robust memory retrieval [29], deep complex networks have been formulated [26], in which the inputs, the weights and the outputs are all complex values. Correspondingly, the convolution, the batch normalization, the activation, etc. are reformed in complex operations. It has been shown that complex networks can deliver comparable or even better accuracy than DNNs under the same model capacity. A particularly attractive feature of the complex network is the ability to embed phase information of the input data naturally into the network representation. The phase information is critical for deterministic signals, such as neuronal rhythms in the brain [30], Polarimetric synthetic aperture radar (PolSAR) images [31], Fourier representation of wave, speech data [32], multi-channel images like MRI [33], etc.

Considering the adoption of complex networks for terminal scenarios, in this work, we propose a novel network called *Binary Complex Neural Network* (BCNN) that integrates BNNs and complex neural networks. BCNN extends BNN with its richer representation capacity of complex numbers, but still conserving the high computation efficiency of BNNs for resource limited hardware. In BCNN, the input, weight and output of a layer are all binary complex values, i.e., one of $\{1+i, 1-i, -1+i, -1-i\}$, using **dual-bits** per neuron — one

for the real part and one for the imaginary part. We propose binary complex convolution, which follows the complex number computation rules, but can still be calculated through the assembly-level *xnor-popcnt* machine instructions available in most hardware. Tackling the expensive computation cost of the original complex network in batch normalization [26], which involves matrix inversion and square-rooting, in this work we propose a new batch normalization approach that can significantly simplify the computation logic while facilitate the convergence of the training. Furthermore, we propose a BCNN weight initialization strategy to accelerate the convergence speed and mitigate the chances of gradient explosion/vanishing. Evaluation results on the Cifar10 and ImageNet datasets show that BCNN can achieve better accuracy than BNNs using state-of-the-art models like ResNet [34], ResNetE [35], [36], and NIN [37]. Our contribution in this paper are:

1) We propose the concept of binary complex number, including its dual-bit storage format and the binary complex computation mechanism.
2) We propose the binary complex neural network (BCNN), including quadrant binarization and its gradient.
3) We propose a BCNN-oriented batch normalization function, significantly lowering the computation cost.
4) We propose a BCNN-oriented weight initialization function, facilitating better convergence for the training.
5) Evaluations on Cifar10 and ImageNet datasets show that BCNN can achieve better accuracy than original BNNs.

This paper is organized as follows. We summarize existing literature in Section-II, covering theoretical research about BNNs, the major approaches to enhance BNN accuracy, and the complex neural networks. We present the design details of BCNN in Section-III, covering the definition of binary complex numbers (Section-III-A), the quadrant binarization function (Section-III-B), the batch-normalization (Section-III-C), and the weight initialization (Section-III-D). We evaluate BCNN in Section-IV and conclude in Section-V.

II. Related Work

We briefly discuss existing literature about BNNs and complex neural networks.

A. Binarized Neural Networks

Binarized Neural Network (BNN) was originally evolved from *Binarized Weights Network* (BWN) [38] in which only weights are binarized. The foundation of modern BNNs were laid by the two cornerstone works [1], [2] in which the fundamental components of BNNs were proposed, including (1) the binarization function and its approximated gradient through *straight-through estimator* (STE) on latent variables; (2) batch-normalization, which is crucial for BNNs to be able to converge; (3) the necessity to keep full-precision for the first and last layers. It was later explained by Anderson and Berg [39] on why BNNs could effectively approximate DNNs: (i) the binary vector through binarization preserves the direction of DNN real-valued vectors in the high-dimensional geometry space; (ii) the bit dot-product (`popc(xnor())`), through batch-normalization, preserves the property of original DNN dot-product; (iii) the real-value convolution for the first layer can embed input images into high-dimensional binary space, which can then be effectively handled by binary operations.

BNNs are generally criticized for reduced accuracy compared to their DNN counterparts because of (1) Information loss due to input binarization and binary activation; (2) Reduced model capacity due to weight binarization (1 bit per neuron); and (3) Unsatisfied network structure or training methodologies as existing popular models and training strategies were mainly designed for real-value DNNs. Correspondingly, existing works propose to enhance BNN training accuracy via: (1) *Reducing information loss*. This can be achieved by adding gain terms (i.e., scaling factors) to better approximate DNN activation [16]. Gain terms are extracted based on the statistics of the inputs [17], [18], or gradually learned with the training [19], [20]; (2) *Enhancing BNN model capacity*. This is done by using multiple BNN components in the network (e.g., *BENN* [23] and *Group-Net* [22]), or using more bits for a neuron where each bit represents a basis. The basis can be fixed to powers-of-two (e.g.,1,2,4,8,...) [17], or adjustable as residual basis [18], [21], or even learned during training [19]; (3) *Designing BNN-specific network structure and training methods*. As most existing network models were designed for DNNs, researchers started to design BNN-oriented network structures, these include *ResnetE* and *BinaryDenseNet* [24] which adopted more shortcuts for reusing information to maintain rich information flow in the network, and *MeliusNet* [25], which conserved the mainstream information flow as full-precision in the first 256 channels, but using the two-block BNN structure (i.e., a dense block with an improvement block) for learning and attaching learned results in separated 64 channels. In this way, most information loss due to binarization could be avoided. Additionally, *Bi-Real Net* redirects the information-rich real-valued activation before binarization to the next block through a shortcut [35].

Alternative works concentrated on improving BNN training methodology. For example, focusing on the `sign` activation function and the STE gradient estimator, Alizadeh et al. showed that adapting the learning rate using second-moment methods was crucial for the successful adoption of STE in BNN training, compared with other optimizers [40]. Darabi et al. proposed a variation of the derivative of the Swish-like activation in place of the STE mechanism for obtaining more effective back-propagation functions [20]. Lahoud et al. presented to use a smooth activation function at the beginning, and then gradually sharpened it to a binary representation equivalent to `sign` during the training [41]. Hou et al. discussed loss-aware binarization, showcasing a proximal Newton algorithm with diagonal Hessian approximation that could directly minimize the loss with respect to the binary weights [42]. Observing that existing BNNs using real-valued latent weights to accumulate small update steps, Helwegen et al. viewed the latent weights as inertia, and introduced a BNN-specific optimizer called *Binary Optimizer* (Bop) [43] for the training. Focusing on other training aspects, Tang et al. used special regularization items to encourage the latent floating-point variables approaching +1 and -1 during the training [18], Umuroglu et al. placed pooling before batch normalization and activation [44]. Mishra et al. guided the training of BNNs through a well-trained,

full-precision teacher network by adjusting the loss function. Additional BNN works could be found in the two surveys [45], [46].

This work falls into the second category, aiming at enhancing BNN model capacity by enhancing the neuron's expressibility. BCNN uses dual bits for each complex binary neuron. Nevertheless, it is fundamentally different from 2-bit quantization; each 2 bits here embed a binary complex calculation logic, which, as shown later, is capable of extracting more expressive features. To ensure fairness, in BCNN, without changing the model structure, we proportionally reduce the number of channels to ensure consistent model size.

B. Complex Neural Network

Complex numbers extend one dimensional real number line (i.e., -∞ to ∞) to two dimensional complex plane by using the real axis and the imaginary axis. Although complex numbers do not exist in the real world, its unique properties and computing rules provide useful amendments to the representativeness of real numbers, especially when phase information is presented. For example, in physics, complex numbers are more suitable for representing waves, as the coefficients are complex after the Fourier transform; in neuroscience, neuronal rhythms, which are crucial for neuronal communication, are characterized by the firing rate and the phase, thus can be naturally expressed as complex numbers [30]; in geoscience, PolSAR images [31], [47] can offer much more comprehensive and robust information compared with pure SAR images. The scattering properties of PolSAR images can be naturally described by the complex-valued polarization scattering matrix, where the amplitude of each element corresponds to the back-scattering intensity of the electromagnetic wave from the target to the radar, and the phase corresponds to the distance between the sensor platform and the target. In biomedical science, being able to effectively handle phase information greatly facilitates MRI image reconstruction [33], [48].

Due to the richer representation and the need to process complex input signals with phases, there have long been efforts in constructing complex neural networks dating back to the 1990's [49]–[52]. The most recent one — *Deep Complex Networks* (DCN) was proposed by Trabelsi et al. [26], which formulated the building blocks of complex-valued deep neural networks include complex convolution, complex batch normalization, complex weight initialization strategy, etc. DCN takes into consideration the correlation between the real part and imaginary part of the complex inputs and weights, demonstrating its effectiveness on classification tasks, showing comparable or even superior performance than real-valued DNNs with only half of the real-valued network size.

This work is motivated by both DCN and BNN. We try to systematically integrate the two planes so that: (i) BCNN can show advanced accuracy compared to BNNs, with its richer representation through binary complex numbers; (ii) BCNN can drastically reduce the execution cost compared with DCNs, which is particularly attractive for embedded and edge utilization, where low cost, small size, low energy, and real-time response are usually demanding; (iii) Compared to BNNs, BCNN can naturally handle complex input signals, such as wave information directly from the sensors.

III. Binary Complex Neural Network

We present our binary complex neural network (BCNN) in this section. We first define a binary complex number and discuss its convolution process. We then present how to perform complex binarization and binary complex batch normalization. Finally, we propose a weight initialization strategy for BCNN.

A. Binary Complex Number

Similar to a complex number $z = x + iy$ that comprises a real part "x" and an imaginary part "iy", a *binary complex number* is defined as $z^b = x^b + iy^b$ where x^b, $y^b \in \{+1, -1\}$. Therefore, z^b has four potential values: $\{-1-i, -1+i, 1-i, 1+i\}$, which can be encoded by two digital bits: the first one implies whether the real part is -1 or $+1$, while the second implies whether the imaginary part is $-i$ or i.

For dot-product, if $z^b = x^b + iy^b$ is the binary complex input, $W^b = A^b + iB^b$ is the binary complex weight, then $h = c + id$ is the full-precision complex output. The bias is also a full-precision complex number, which is omitted in the following discussion for simplicity. Therefore, its dot product follows the complex calculation rules:

$$h = c + id = (A^b * x^b - B^b * y^b) + i(B^b * x^b + A^b * y^b) \quad (1)$$

where $x^b, y^b, A^b, B^b \in \{+1, -1\}$. Compared to BNN binary dot-product, a BCNN dot-product incorporates 4 binary dot-products and two extra real-valued additions. In the matrix notation, it is expressed as:

$$\begin{bmatrix} c \\ d \end{bmatrix} = \begin{bmatrix} A^b & -B^b \\ B^b & A^b \end{bmatrix} * \begin{bmatrix} x^b \\ y^b \end{bmatrix}$$

B. Quadrant Binarization

Binarization in BNN is the process of converting a full-precision real number into a binary number: $+1$ or -1, which is generally viewed as the non-linear activation function for BNNs.

Binarization can be performed in two approaches, known as *deterministic* and *stochastic* [1], [2]. The stochastic one can potentially offer a better accuracy, but at the expense of high implementation cost, whereas the deterministic one is merely a `sign` function, as shown below:

$$\text{Forward:} \quad r^b = sign(r) = \begin{cases} +1 & r \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

Most BNN works adopt the low-cost deterministic binarization function. Since `sign` is non-differentiable at 0, and its gradient is always 0 otherwise, direct back-propagation is infeasible. Prior works proposed the *Straight-Through-Estimator* (STE) to do the back-propagation:

$$\text{Backward:} \quad \frac{\partial Loss}{\partial r} = \frac{\partial Loss}{\partial r^b} \mathbf{1}_{|r|<t_{clip}}$$

where r is the full-precision real input. $r^b \in \{+1, -1\}$ is the binary output. $Loss$ is the value of the cost function. t_{clip} is

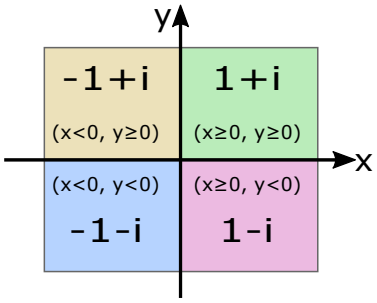

Fig. 1: Quadrant binarization into a binary complex number.

a clipping threshold, which typically sets to 1. The gradient of `sign` function is simply set as an idientity function. The threshold is used to cancel the gradient, when the inputs are geting too large, which can assist in the optimization process.

The binarization to a binary complex number is to convert a complex number into a binary complex number (i.e., one of $\{1+i, 1-i, -1+i, -1-i\}$). We propose quadrant binarization where the output is determined based on which quadrant the input complex number belongs to in the two-dimensional Cartesian system, as shown in Figure 1. Mathematically, a complex plane is a geometric representation of the complex numbers settled by the real axis x and the orthogonal imaginary axis y, where the two axes partition the plane into four quadrants, each bounded by two half-axes. Given four quadrants and four complex binary values, it is natural to link each quadrant with a complex binary value. The quadrant binarization is determined by the phase of a complex number, which is critical information in the complex signals.

This quadrant binarization essentially decouples the real part and the imaginary part so that both parts could be processed separately as an ordinary binarization. For the forward propagation, the binarization is as:

$$z^b = \text{sign}(x + iy) = \text{sign}(x) + i\text{sign}(y) = x^b + iy^b \tag{2}$$

For the backward propagation, the gradient of the binarization is through two STEs, and applied over two independent full-precision latent variables x and y:

$$\frac{\partial Loss}{\partial z} = \frac{\partial Loss}{\partial x^b}\mathbf{1}_{|x|<t_{clip}} + i\frac{\partial Loss}{\partial y^b}\mathbf{1}_{|y|<t_{clip}} \tag{3}$$

This keeps the simplicity of the binarization process for efficient hardware implementation and memory storage. Note that to improve accuracy, existing works propose various variants of the binarization function, such as scaling factor [16]–[20], approximated `sign()` function [20], [41], etc. However, according to [24], no obvious accuracy improvement had been observed by adopting these strategies. Therefore, in this work, we use the original BNN binarization function as the baseline [1], [2], which can achieve the best theoretic execution performance and model compression rate.

In BNNs, in addition to the 32x memory storage and bandwidth savings by adopting 1-bit of a neuron (compared to 32-bits per floating-point neuron), the computation efficiency gains from the approach on how bit dot-product is performed: each 32 or 64 bit dot-product in DNN can be accomplished by a single exclusive-nor (`xnor`) operation followed by a population count (`popc`) operation, leading to 10-16x speedups [3]:

$$x * w \approx x^b * w^b = \text{popc}(\text{xnor}(x^b, w^b))$$

A key requirement here is to conserve this property of being hardware friendly. Essentially in BCN,

$$\begin{aligned} z * W &= (x + iy) * (A + iB) \\ &\approx sign(x + iy) * sign(A + iB) \\ &= (x^b + iy^b) * (A^b + iB^b) \\ &= (A^b * x^b - B^b * y^b) + i(B^b * x^b + A^b * y^b) \end{aligned} \tag{4}$$

which implies that a BCNN dot-product can be computed by 4 BNN dot-products (i.e., `popc-xnor`) plus two full-precision additions. The 4 BNN dot-product can be operated in parallel at the same time, so the latency theoretically is close to one BNN dot-product.

In a BCNN convolution layer, within the input, weight and output tensors, we use the first half for the real part and the second half for the imaginary part. Specifically, if the input tensor has M complex feature maps, it is equivalent to the input has $2M$ real-valued feature maps, where the first M represent the real components x and the remaining M represent the imaginary components y. The same case applies to the output tensor. Consequently, the weight tensor is in size $(N \times M \times k \times k) \times 2$ where the former half refers to the real part of the complex weight (i.e., A in Eq. 4), and the latter half refers to the imaginary part (i.e., B in Eq. 4).

C. *Complex Gaussian Batch Normalization*

Batch normalization (BN) [53] has been proposed to accelerate convergence speed and contribute to better training accuracy. In real-value DNN, BN first normalizes the input so that the mean becomes zero and the variance becomes one. It then adjusts the normalized input by scaling through a learnable gain factor, and shifting through a learnable bias, as shown below:

$$BN(r) = \frac{r - \mu}{\sqrt{\sigma^2 + \epsilon}} * \gamma + \beta \tag{5}$$

where r is the input, μ is the mean of the batch, σ is the variance of the batch, γ is the learned scale, β is the learned shift, ϵ is a tiny number for numerical stability.

BN is important for DNNs, but is vital for BNNs. In addition to the normalization of the input, the learned gain and bias essentially increase the model capacity, or learning capability of a BNN layer. Without BN, the training of BNNs is even unlikely to converge.

Different from Eq 5, standardizing complex input to normal complex distribution is much more complicated, because in addition to normalizing the mean and variance, Batch normalization in *complex neural networks* needs to ensure equal variance of the real and imaginary components. In deep complex Network [26], the complex batch normalization is treated as a 2D whitening transformation – scaling the complex input by the square root of their variance along the real and imaginary components. This is achieved by multiplying the

0-centered data by the inverse square root of the covariance matrix:

$$\tilde{z} = (V)^{-\frac{1}{2}}(z - E[z]) \qquad \text{BN}(\tilde{z}) = \gamma\tilde{z} + \beta \tag{6}$$

where z is the complex input, $E[z]$ is the mean of z, V is the 2×2 covariance matrix. The scaling parameter γ is a 2×2 positive semi-definite matrix with three degrees of freedom (γ_{ri} and γ_{ir} is the same). The shifting parameter β is also a complex parameter. γ_{ri}, γ_{ir}, β are initialized with 0; γ_{rr} and γ_{ii} are initialized with $1/\sqrt{2}$.

As shown in Eq 6, complex batch normalization involves the computation of matrix inversion and matrix square-root, which is too costly for the hardware. Besides, directly adopting this complex batch normalization approach in BCNN leads to poor training accuracy, or even non-convergence, which is shown in Section IV. Consequently, we propose a novel batch normalization method called complex Gaussian batch normalization (CGBN), which is more efficient and light-weight.

Our objective is to normalize the input complex signal to a standard complex normal distribution (CN) [54], [55]. The standard complex normal random variable, also known as standard complex Gaussian random variable, is a complex random variable z whose real and imaginary parts are independent normally distributed random variables with mean equals to zero and variance equals to 1/2. In mathematical form, $z \sim CN(0, 1)$ implies:

$$R(z) \perp I(z) \text{ and } R(z) \sim N(0, 1/2) \text{ and } I(z) \sim N(0, 1/2)$$

Consequently, we can separately normalize the real part and imaginary part of the input complex signal to a normal distribution with zero mean and $1/2$ variance:

$$\tilde{z} = \left(\frac{z_r - \mu_r}{\sqrt{2\sigma_r^2 + \epsilon}}\right) + i\left(\frac{z_i - \mu_i}{\sqrt{2\sigma_i^2 + \epsilon}}\right) \tag{7}$$

The scaling parameter and shifting parameter are learnable complex values, the complex Gaussian batch-normalization is as shown below:

$$\begin{aligned} \text{CGBN}(\tilde{z}) &= \gamma * \tilde{z} + \beta \\ &= \gamma_r\tilde{z}_r - \gamma_i\tilde{z}_i + \beta_r + i(\gamma_r\tilde{z}_i + \gamma_i\tilde{z}_r + \beta_i) \end{aligned} \tag{8}$$

where both the scaling parameter γ and shifting parameter β are learned during the training. γ is initialized as $\frac{1}{\sqrt{2}} + i\frac{1}{\sqrt{2}}$. β is initialized as $0 + i0$.

This complex Gaussian batch normalization significantly reduces the computation complexity compared to Eq 6 by avoiding the calculation of the inverse square-root of the covariance matrix. The complex Gaussian batch normalization leads to faster convergence speed and can converge in all of the models and datasets we have evaluated.

D. Binary Complex Weight Initialization

A proper weight initialization strategy can largely avoid exploding or vanishing gradient problem during backpropagation and accelerate the convergence speed during network training. Usually, the weight initialization follows two rules: (i) the input and output have the same variance in the forward propagation; (ii) the gradient of input and output have the same variance in the backward propagation.

Two weight initialization strategies are broadly used for deep neural networks: *Xavier* [56] and *He* [57]. Xavier is suitable for symmetric activation functions such as `tahn`, `softsign`, etc. The initial weight parameters follow a uniform distribution with zero mean and variance equals to $2/(fan\_in + fan\_out)$. *He* is specially designed for `ReLU` like activation functions. The variance of the initialization distribution is $2/fan\_in$ instead.

Following the *Xavier* [56] and *He* approach [57], the complex neural network [26] derives the variance of the complex weight parameters. In the complex weight Initialization, a complex weight has a polar form:

$$W = |W|e^{i\theta} = R\{W\} + iI\{W\} \tag{9}$$

The variance of W is related to its amplitude not its phase, So the amplitude $|W|$ is set to follow the Rayleigh distribution, with the probability density function being:

$$f(x, \sigma) = \frac{x}{\sigma^2}e^{-x^2/(2\sigma^2)}, x \geq 0$$

For Xavior [56], to ensure $Var(W) = 2/(fan\_in + fan\_out)$, then the parameter $\sigma = 1/\sqrt{fan\_in + fan\_out}$. For He [57], to meet $Var(W) = 2/fan\_in$, set $\sigma = 1/\sqrt{fan\_in}$. The phase θ is unrelated to the variance, so it is initialized to follows the uniform distribution between $-\pi$ and π.

For BCNN, however, the complex weight initialization strategy does not work. After the quadrant binarization, the amplitude of binary complex weight is always $sqrt(2)$, which diminishes the efficiency of the original initialization strategy, which will be presented in our evaluation (Section IV). Here, following the Xavier approach [56], we derive our BCNN's weight initialization. As discussed in Section III-A, in a BCNN layer l, the complex output $h_l = c_l + id_l$ is obtained by the convolution of the complex input $z_l = x_l + iy_l$ and the complex weight $W_l = A_l + iB_l$, the complex bias is ignored for simplification. The f is the activation function, we have:

$$\begin{aligned} c_l &= x_l * A_l - y_l * B_l \qquad & x_{l+1} &= f(c_l) \\ d_l &= x_l * B_l + y_l * A_l \qquad & y_{l+1} &= f(d_l) \end{aligned}$$

The variance of real part and imaginary part can be written as:

$$Var[c_l] = fan\_in * (Var[x_l]Var[A_l] + Var[y_l]Var[B_l])$$
$$Var[d_l] = fan\_in * (Var[x_l]Var[B_l] + Var[y_l]Var[A_l])$$

If C_{in} and C_{out} are the channel size of the complex input and output, then $fan\_in = k^2 \times C_{in}$, $fan\_out = k^2 \times C_{out}$. For the Forward propagation, we ensure the real/imaginary part of input and output have the same variance: $Var[c_l] = Var[x_l]$, $Var[d_l] = Var[y_l]$. At the same time, we assume the real part and the imaginary part of complex feature maps have the same variance: $Var[c_l] = Var[d_l]$, $Var[x_l] = Var[y_l]$, and for the complex weight as well: $Var[A_l] = Var[B_l]$, so we have:

$$2 \times fan\_in \times Var[A_l] = 1$$

$$2 \times fan\_in \times Var[B_l] = 1$$

$$Var[A_l] = Var[B_l] = \frac{1}{2 \times fan\_in} \tag{10}$$

for the backward propagation, the gradient of is computed as:

$$\frac{\partial Loss}{\partial x_l} = \tilde{A}\frac{\partial Loss}{\partial c_l} + \tilde{B}\frac{\partial Loss}{\partial d_l} = f'(c_l)\tilde{A}\frac{\partial Loss}{\partial x_{l+1}} + f'(d_l)\tilde{B}\frac{\partial Loss}{\partial y_{l+1}}$$

$$\frac{\partial Loss}{\partial y_l} = \tilde{A}\frac{\partial Loss}{\partial d_l} - \tilde{B}\frac{\partial Loss}{\partial c_l} = f'(d_l)\tilde{A}\frac{\partial Loss}{\partial y_{l+1}} - f'(c_l)\tilde{B}\frac{\partial Loss}{\partial x_{l+1}}$$

The weight A and B here is a C_{in}-by-k^2C_{out} matrix while the gradient of the weight $\tilde{A}$ and $\tilde{B}$is a C_{out}-by-k^2C_{in} matrix. For input and output, the gradient of the real/imaginary part should have the same variance. With the assumption that for complex feature maps, the variance of the gradient for real part and imaginary part are the same, we have:

$$Var[\frac{\partial Loss}{\partial x_l}] = 2 \times fan\_out \times Var[A] \times Var[\frac{\partial Loss}{\partial x_{l+1}}]$$

$$Var[\frac{\partial Loss}{\partial y_l}] = 2 \times fan\_out \times Var[B] \times Var[\frac{\partial Loss}{\partial y_{l+1}}]$$

Therefore, for the backward pass, we have:

$$Var[A_l] = Var[B_l] = \frac{1}{2 \times fan\_out} \tag{11}$$

With a compromise of Eq 11 and Eq 11, we have:

$$Var[A_l] = Var[B_l] = \frac{1}{fan\_in + fan\_out} \tag{12}$$

Therefore, in BCNN, we initialize the weight matrix by following the normal distribution with mean $\mu = 0.0$ and variance $\sigma = \sqrt{1/(fan\_in + fan\_out)}$.

IV. Evaluations

In this section, we evaluate the performance of BCNN for image classification using three deep neural network models: *NIN-Net*, *ResNet18*, and *ResNetE18* on two popular image classification datasets, *CIFAR10* and *ImageNet*.

A. Experimental Setup

We use PyTorch to train all models. First we compare the BCNN and BNN with similar architecture and the same parameter size. Then, we evaluate and compare three different batch normalization and three weight initialization strategies for BCNN.

Complex-valued Input Data Generation: As the raw data in Cifar10 and ImageNet datasets are real-valued, it is necessary to extend them to complex domain first. Our BCNN adopts a prior-art learning-based methodology proposed in [26] to generate imaginary parts. As shown in Figure 2, the imaginary parts are learned by a real-valued residual blocks, then the concatenation of raw real-valued data and the learned imaginary parts can serve as the complex-valued inputs. The two conv layers are both 1×1 conv kernel, the channel of input and output are 3, so the real-valued residual block is lightweight in terms of both computation and storage.

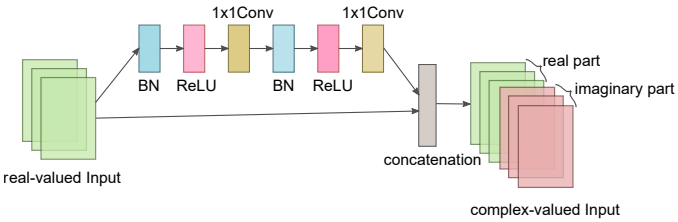

Fig. 2: Structure to generate complex input from real-valued input.

BCNN Model Configuration: Usually in BNNs, the first and the last layer are full-precision. Using full-precision for the first layer is to conserve the maximum information flow from the input images. Using the full-precision for the last layer is to conserve the maximum state-space before the final output, which is in particular meaningful for large dataset like ImageNet. For BCNN, we adopt the same strategy. Using full-precision complex convolution for the first layer. For the last layer, we use a full-precision real-valued layer by treating M complex input as $2M$ real-valued input.

For fair comparison, all networks used in our evaluation of BCNN and BNN are with very similar architecture and the same model size. As BCNN uses complex-valued parameters, the model size of BCNN is about twice the size of BNNs with the same network configurations. Therefore, we tune the numbers of channels at each layer of BCNNs to approximately $1/\sqrt{2}$ of the ones in BNNs.

Network: Three networks are used for evaluation: NIN-Net, ResNet18 and ResNetE18.

(a) *NIN-Net* consists of three stacked mlpconv layers followed by a spatial 2×2 MAX-Pooling and a global Average-Pooling layer. For Cifar10 dataset, we use the original NIN-Net proposed in [37]; for ImageNet dataset, we use the enhanced version of NIN-Net proposed in [18], which enlarges kernel sizes of the first four mlpconv layers from 1×1 to 3×3, and increase the output channels at the first two mlpconv layers from 96 to 128.

(b) *ResNet18 and ResNetE18:* The block of ResNet18 and ResNetE18 are as shown in Figure.3 and Figure.4 (the stride of first convolution is 2, when the stride is 1, the bypass will be identical). ResNetE is a modified version of ResNet, which is equipped with extra shortcuts and adopts full-precision down-sampling conv layer. With these modifications, ResNetE can preserve the information flow of the network better and process low-precision data, especially binary data, more efficiently.

B. Result on CIFAR-10

We first evaluate BCNN with NIN-net [37] and ResNet18 [34] on CIFAR-10 dataset. In training, Adam optimizer is adopted with the initial learning rate set as 5e-3. All models are trained for up to 300 epochs. We adjust the learning rates during training by multiplying them by 0.2 at the 80th, 150th, 200th, 240th, and 270th epochs respectively.

Table I compares the accuracy of BCNNs, DNNs and BNNs. For NIN-Net and ResNet18, BCNN achieves 1.85% and 0.52% improvement on accuracy respectively over BNNs with the same model sizes. Figure.5 and Figure.6 show the variation of training loss and testing loss from epoch 0 to epoch 300.

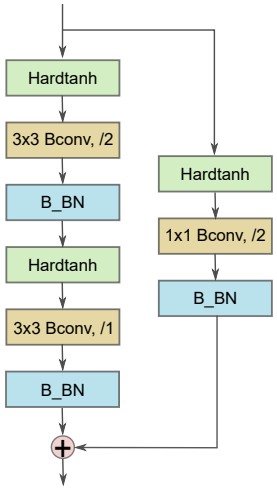

Fig. 3: ResNet18

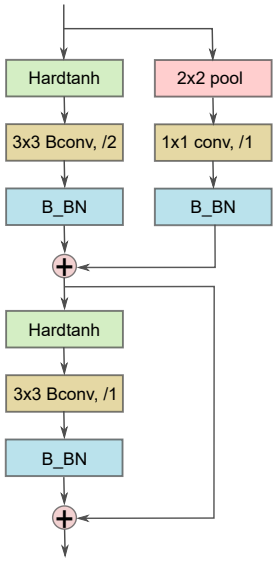

Fig. 4: ResNetE18

TABLE I: Test accuracy on Cifar10.

Network	Type	Params	Top-1(%)
NIN-Net	DNN	3.69M	89.64
	BNN	0.191M	85.77
	BCNN	0.187M	**87.62**
ResNet18	DNN	42.63M	93.02
	BNN	1.39M	90.67
	BCNN	1.39M	**91.19**

TABLE II: Test accuracy with respect to batch normalization and weight initialization strategies on Cifar10. CGBN refers to Gaussian batch normalization proposed in this work. BN refers to normal batch normalization method [53]. CBN refers to the baseline batch normalization approach proposed in deep complex network [26]. BCW refers to the binary complex weight initialization proposed in this work. Xavier refers to the Xavier weight initialization strategy [56]. Ray refers to the complex weight initialization approach proposed in deep complex network [26]. NA means non-convergence.

Network	BatchNorm	Weight_init	Top-1(%)
NIN-Net	CGBN	BCW	**87.62**
	CGBN	Xavier	86.76
	CGBN	Ray	86.07
	CBN	BCW	**87.86**
	BN	BCW	87.38
	CBN	Ray	NA
ResNet18	CGBN	BCW	**91.19**
	CGBN	Xavier	90.56
	CGBN	Ray	89.92
	CBN	BCW	**91.31**
	BN	BCW	90.25
	CBN	Ray	NA

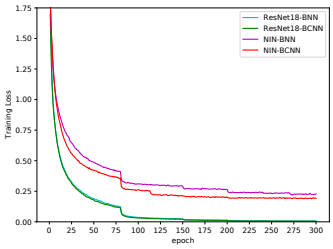

Fig. 5: Training loss on Cifar10.

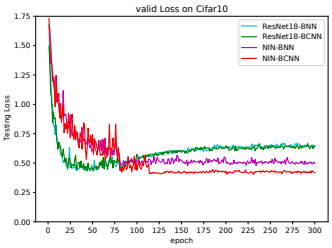

Fig. 6: Testing loss on Cifar10.

In Table II , we show the impact of different batch normalization methodologies and weight initialization strategy on BCNN. We evaluate three different batch normalization. BN (batch normalization) is the most popular batch normalization that being used in real-valued neural networks. CBN (complex batch normalization) was proposed in deep complex network. CGBN (complex gaussian batch normalization) is proposed in this work for our BCNN. Compared with the BN, the proposed CGBN improves the accuracy on both NIN-Net and ResNet18. For Cifar-10 dataset, our experimental results show that the CBN Batch Normalization can get higher accuracy compared to our CGBN. However, CBN is more computationally intensive and leads to non-convergence with large dataset, e.g. ImageNet. The detailed results on ImageNet will be given in the next section (NA in the table means non-convergence).

Different parameter initialization schemes also affect performance of BCNNs. As shown in Table II . BCW (binary complex weight initialization) is proposed for BCNN in this paper. Xavier is used for real-valued network, Ray was proposed for deep complex network. Results show the proposed initialization technique results in 1.55% and 1.27% accuracy improvement for NIN-Net and ResNet18 respectively.

C. BCNNs on ImageNet

In this section, we show the performance of BCNNs on ImageNet dataset. We use the standard pre-processing: all images are resized to 256×256 and randomly cropped to 224×224 for training. The validation dataset is with a single center crop. We use ADAM optimizer in training and set the initial learning rate as 5e-3. This learning rate is gradually adjusted during training by being divided by 5 at epochs of 25, 35, 40 and 45. Each model is trained for 50 epochs. We use 3 models in our evaluation: NIN-E (expanded version of NIN-net as introduced in Section. IV(A)), ResNet18, and ResnetE18.

Table III compares the top-1 and top-5 accuracy of BCNNs, DNNs and BNNs. BCNN always has higher accuracy than BNN. For NIN-E, ResNet18 and ResNetE18, BCNN provides 0.8%, 1.32% and 0.17% higher accuracy of top-1 than the ones of BNNs respectively. Figure.7 and Figure.8 show the changes of training and validation loss from epoch 0 to epoch 50.

TABLE III: Test accuracy on ImageNet.

Network	Type	Params	Top-1(%)	Top-5(%)
NIN-E	DNN	28.96M	57.09	79.34
	BNN	5.01M	50.018	73.936
	BCNN	5.0M	**51.08**	**74.738**
ResNet18	DNN	44.59M	70.142	89.274
	BNN	3.36M	54.308	77.388
	BCNN	3.36M	**55.598**	**78.708**
ResNetE18	BNN	4.0M	57.332	79.85
	BCNN	4.0M	**57.652**	**80.016**

The accuracy comparison of BCNNs with different Batch Norm techniques and weight initialization strategies are shown in Table IV.

BCNN with the proposed CGBN always provides higher accuracy than BN for all models. For NIN-E which is a relatively shallow network structure, CBN plus BCW have the highest accuracy. However, for the relativity deeper structures, e.g, ResNet18 and ResNetE18, CBN lead to non-coverage

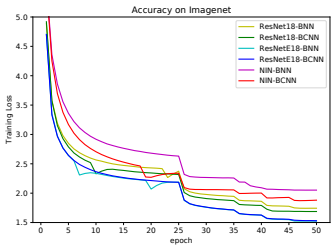

Fig. 7: Training loss on ImageNet.

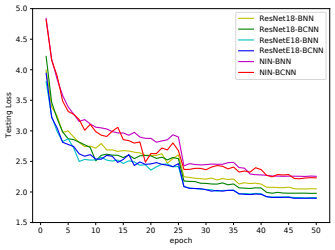

Fig. 8: Testing loss on ImageNet.

TABLE IV: Test accuracy with respect to batch normalization and weight initialization strategies on ImageNet.

Network	BatchNorm	Weight_init	Top-1(%)	Top-5(%)
NIN-E	CGBN	BCW	**51.08**	**74.738**
	CGBN	Xavier	50.992	74.554
	CGBN	Ray	50.55	74.26
	CBN	BCW	**51.79**	**75.162**
	BN	BCW	50.622	74.276
	CBN	Ray	NA	
ResNet18	CGBN	BCW	**55.598**	**78.708**
	CGBN	Xavier	54.98	78.238
	CGBN	Ray	NA	
	CBN	BCW	NA	
	BN	BCW	54.91	78.148
	CBN	Ray	NA	
ResNetE18	CGBN	BCW	**57.652**	**80.016**
	CGBN	Xavier	57.250	79.976
	CGBN	Ray	NA	
	CBN	BCW	NA	
	BN	BCW	57.642	80.188
	CBN	Ray	NA	

during training. In contract, our proposed CGBN can still work efficiently.

We further evaluate the effects of different weight initialization strategies on ImageNet dataset. As listed in Table IV, the proposed weight initialization BCW shows increased accuracy over Ray for NIN-Net. For ResNet18 and ResNetE18, the proposed BCW leads to faster convergence with higher accuracy than BNNs. As a comparison, BCNN with Ray cannot converge in our testing.

Overall, we show that BCNN, together with the proposed batch normalization and weight initialization strategies, can achieve better training accuracy on some large datasets such as ImageNet. As the next step, we will seek efficient implementation of BCNN on various hardware platforms, including GPUs [3], GPU Tensorcores [4], FPGAs [7], [58] and ASICs [5], [6], for practical utilization in embedded systems and edge domains.

V. Conclusion

In this work we propose the binary complex neural network, which combines the advantages of both BNNs and complex neural networks. Compared to BNNs, it achieves enhanced training accuracy and is able to learn from complex data; compared to complex neural networks, it is much more computation efficient, which is in particular beneficial to terminal scenarios such as smart edges and smart sensors. Future work includes the demonstration of BCNN on complex datasets, the implementation of BCNN on embedded hardware devices, and its practical applications.

References

[1] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1," *arXiv preprint arXiv:1602.02830*, 2016.

[2] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *Proceedings of the 30th international conference on neural information processing systems*. Citeseer, 2016, pp. 4114–4122.

[3] A. Li, T. Geng, T. Wang, M. Herbordt, S. L. Song, and K. Barker, "Bstc: A novel binarized-soft-tensor-core design for accelerating bit-based approximated neural nets," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2019, pp. 1–30.

[4] A. Li and S. M. Su, "Accelerating binarized neural networks via bit-tensor-cores in turing gpus," *IEEE Transactions on Parallel and Distributed Systems*, 2020.

[5] T. Geng, T. Wang, C. Wu, C. Yang, W. Wu, A. Li, and M. C. Herbordt, "O3bnn: An out-of-order architecture for high-performance binarized neural network inference with fine-grained pruning," in *Proceedings of the ACM International Conference on Supercomputing*, 2019, pp. 461–472.

[6] T. Geng, A. Li, T. Wang, C. Wu, Y. Li, R. Shi, W. Wu, and M. Herbordt, "O3bnn-r: An out-of-order architecture for high-performance and regularized bnn inference," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 1, pp. 199–213, 2020.

[7] T. Geng, T. Wang, C. Wu, C. Yang, S. L. Song, A. Li, and M. Herbordt, "Lp-bnn: Ultra-low-latency bnn inference with layer parallelism," in *2019 IEEE 30th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, vol. 2160. IEEE, 2019, pp. 9–16.

[8] A. Galloway, G. W. Taylor, and M. Moussa, "Attacking binarized neural networks," *arXiv preprint arXiv:1711.00449*, 2017.

[9] N. Narodytska, "Formal analysis of deep binarized neural networks." in *IJCAI*, 2018, pp. 5692–5696.

[10] N. Narodytska, S. Kasiviswanathan, L. Ryzhyk, M. Sagiv, and T. Walsh, "Verifying properties of binarized deep neural networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.

[11] G. Chen, H. Meng, Y. Liang, and K. Huang, "Gpu-accelerated real-time stereo estimation with binary neural network," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 12, pp. 2896–2907, 2020.

[12] N. Fasfous, M.-R. Vemparala, A. Frickenstein, L. Frickenstein, and W. Stechele, "Binarycop: Binary neural network-based covid-19 face-mask wear and positioning predictor on edge devices," *arXiv preprint arXiv:2102.03456*, 2021.

[13] C.-H. Huang, "An fpga-based hardware/software design using binarized neural networks for agricultural applications: A case study," *IEEE Access*, vol. 9, pp. 26 523–26 531, 2021.

[14] Y. Ma, H. Xiong, Z. Hu, and L. Ma, "Efficient super resolution using binarized neural network," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2019, pp. 0–0.

[15] C. Ma, Y. Guo, Y. Lei, and W. An, "Binary volumetric convolutional neural networks for 3-d object recognition," *IEEE Transactions on Instrumentation and Measurement*, vol. 68, no. 1, pp. 38–48, 2018.

[16] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *European conference on computer vision*. Springer, 2016, pp. 525–542.

[17] 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 arXiv:1606.06160*, 2016.

[18] W. Tang, G. Hua, and L. Wang, "How to train a compact binary neural network with high accuracy?" in *Thirty-First AAAI conference on artificial intelligence*, 2017.

[19] X. Lin, C. Zhao, and W. Pan, "Towards accurate binary convolutional neural network," in *Advances in Neural Information Processing Systems*, 2017, pp. 345–353.

[20] S. Darabi, M. Belbahri, M. Courbariaux, and V. P. Nia, "Bnn+: Improved binary network training," *arXiv preprint arXiv:1812.11800*, 2018.

[21] M. Ghasemzadeh, M. Samragh, and F. Koushanfar, "Rebnet: Residual binarized neural network," in *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2018, pp. 57–64.

[22] B. Zhuang, C. Shen, M. Tan, L. Liu, and I. Reid, "Structured binary neural networks for image recognition," *arXiv preprint arXiv:1909.09934*, 2019.

[23] S. Zhu, X. Dong, and H. Su, "Binary ensemble neural network: More bits per network or more networks per bit?" in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4923–4932.

[24] J. Bethge, H. Yang, M. Bornstein, and C. Meinel, "Binarydensenet: developing an architecture for binary neural networks," in *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, 2019, pp. 0–0.

[25] J. Bethge, C. Bartz, H. Yang, Y. Chen, and C. Meinel, "Meliusnet: An improved network architecture for binary neural networks," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2021, pp. 1439–1448.

[26] C. Trabelsi, O. Bilaniuk, Y. Zhang, D. Serdyuk, S. Subramanian, J. F. Santos, S. Mehri, N. Rostamzadeh, Y. Bengio, and C. J. Pal, "Deep complex networks," *arXiv preprint arXiv:1705.09792*, 2017.

[27] S. Wisdom, T. Powers, J. R. Hershey, J. L. Roux, and L. Atlas, "Full-capacity unitary recurrent neural networks," *arXiv preprint arXiv:1611.00035*, 2016.

[28] A. Hirose and S. Yoshida, "Generalization characteristics of complex-valued feedforward neural networks in relation to signal coherence," *IEEE Transactions on Neural Networks and learning systems*, vol. 23, no. 4, pp. 541–551, 2012.

[29] I. Danihelka, G. Wayne, B. Uria, N. Kalchbrenner, and A. Graves, "Associative long short-term memory," in *International Conference on Machine Learning*. PMLR, 2016, pp. 1986–1994.

[30] D. P. Reichert and T. Serre, "Neuronal synchrony in complex-valued deep networks," *arXiv preprint arXiv:1312.6115*, 2013.

[31] Y. Cao, Y. Wu, P. Zhang, W. Liang, and M. Li, "Pixel-wise polsar image classification via a novel complex-valued deep fully convolutional network," *Remote Sensing*, vol. 11, no. 22, p. 2653, 2019.

[32] H.-S. Choi, J.-H. Kim, J. Huh, A. Kim, J.-W. Ha, and K. Lee, "Phase-aware speech enhancement with deep complex u-net," in *International Conference on Learning Representations*, 2018.

[33] E. K. Cole, J. Y. Cheng, J. M. Pauly, and S. S. Vasanawala, "Analysis of deep complex-valued convolutional neural networks for mri reconstruction," *arXiv preprint arXiv:2004.01738*, 2020.

[34] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[35] Z. Liu, B. Wu, W. Luo, X. Yang, W. Liu, and K.-T. Cheng, "Bi-real net: Enhancing the performance of 1-bit cnns with improved representational capability and advanced training algorithm," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 722–737.

[36] J. Bethge, M. Bornstein, A. Loy, H. Yang, and C. Meinel, "Training competitive binary neural networks from scratch," *arXiv preprint arXiv:1812.01965*, 2018.

[37] M. Lin, Q. Chen, and S. Yan, "Network in network," *arXiv preprint arXiv:1312.4400*, 2013.

[38] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," *arXiv preprint arXiv:1511.00363*, 2015.

[39] A. G. Anderson and C. P. Berg, "The high-dimensional geometry of binary neural networks," *arXiv preprint arXiv:1705.07199*, 2017.

[40] M. Alizadeh, J. Fernández-Marqués, N. D. Lane, and Y. Gal, "An empirical study of binary neural networks' optimisation," 2018.

[41] F. Lahoud, R. Achanta, P. Márquez-Neila, and S. Süsstrunk, "Self-binarizing networks," *arXiv preprint arXiv:1902.00730*, 2019.

[42] L. Hou, Q. Yao, and J. T. Kwok, "Loss-aware binarization of deep networks," *arXiv preprint arXiv:1611.01600*, 2016.

[43] K. Helwegen, J. Widdicombe, L. Geiger, Z. Liu, K.-T. Cheng, and R. Nusselder, "Latent weights do not exist: Rethinking binarized neural network optimization," in *Advances in neural information processing systems*, 2019, pp. 7531–7542.

[44] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "Finn: A framework for fast, scalable binarized neural network inference," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2017, pp. 65–74.

[45] T. Simons and D.-J. Lee, "A review of binarized neural networks," *Electronics*, vol. 8, no. 6, p. 661, 2019.

[46] H. Qin, R. Gong, X. Liu, X. Bai, J. Song, and N. Sebe, "Binary neural networks: A survey," *Pattern Recognition*, vol. 105, p. 107281, 2020.

[47] J. Gao, B. Deng, Y. Qin, H. Wang, and X. Li, "Enhanced radar imaging using a complex-valued convolutional neural network," *IEEE Geoscience and Remote Sensing Letters*, vol. 16, no. 1, pp. 35–39, 2018.

[48] S. Wang, H. Cheng, L. Ying, T. Xiao, Z. Ke, H. Zheng, and D. Liang, "Deepcomplexmri: Exploiting deep residual network for fast parallel mr imaging with complex convolution," *Magnetic Resonance Imaging*, vol. 68, pp. 136–147, 2020.

[49] G. M. Georgiou and C. Koutsougeras, "Complex domain backpropagation," *IEEE transactions on Circuits and systems II: analog and digital signal processing*, vol. 39, no. 5, pp. 330–334, 1992.

[50] T. Kim and T. Adalı, "Approximation by fully complex multilayer perceptrons," *Neural computation*, vol. 15, no. 7, pp. 1641–1666, 2003.

[51] H. Leung and S. Haykin, "The complex backpropagation algorithm," *IEEE Transactions on signal processing*, vol. 39, no. 9, pp. 2101–2104, 1991.

[52] T. Kim and T. Adali, "Fully complex multi-layer perceptron network for nonlinear signal processing," *Journal of VLSI signal processing systems for signal, image and video technology*, vol. 32, no. 1-2, pp. 29–43, 2002.

[53] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[54] N. R. Goodman, "Statistical analysis based on a certain multivariate complex gaussian distribution (an introduction)," *The Annals of mathematical statistics*, vol. 34, no. 1, pp. 152–177, 1963.

[55] B. Picinbono, "Second-order complex random vectors and normal distributions," *IEEE Transactions on Signal Processing*, vol. 44, no. 10, pp. 2637–2640, 1996.

[56] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, 2010, pp. 249–256.

[57] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034.

[58] T. Geng, C. Wu, C. Tan, B. Fang, A. Li, and M. Herbordt, "Cqnn: a cgra-based qnn framework," in *2020 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 2020, pp. 1–7.