Computers, Materials & Continua

Tech Science Press

https://doi.org/10.32604/cmc.2026.078894

ARTICLE

WCCN: An Efficient and Stable Neural Network Architecture for Complex-Valued Deep Learning

Bing-Zhou Chen[1,2], Hai-Ying Zheng[1,2], Ao-Wen Wang[1,3], Ke-Lei Xia[1,2], Li-Feng Fan[1,3], Zhong-Yi Wang[1,3] and Lan Huang[1,2,*]

[1]College of Information and Electrical Engineering, China Agricultural University, Beijing, China
[2]Key Laboratory of Agricultural Information Acquisition Technology (Beijing), Ministry of Agriculture, Beijing, China
[3]Key Laboratory of Modern Precision Agriculture System Integration Research, Ministry of Education, Beijing, China

*Corresponding Author: Lan Huang. Email: hlan@cau.edu.cn

Received: 09 January 2026; Accepted: 16 March 2026; Published: 08 May 2026

ABSTRACT: Many sensing and imaging modalities naturally yield complex-valued signals, where magnitude and phase jointly convey information. Complex-valued neural networks (CVNNs) possess unique advantages in processing phase-sensitive data (e.g., synthetic aperture radar (SAR) and magnetic resonance imaging (MRI)), yet their widespread adoption is hindered by significant computational overhead and training instability. To address these challenges, this paper presents the Wirtinger Derivative Complete Complex Network (WCCN), a unified and efficient framework for complex-valued deep learning. The proposed framework systematically addresses three key challenges in CVNNs: computational efficiency, parameter redundancy, and training stability. WCCN integrates three core components. First, an optimized complex convolution implementation (wcConv; Gauss trick + tuple-flow) is introduced to enable efficient complex-valued feature extraction, achieving a speedup of roughly 15%–20% over conventional implementations through a fused tuple-based processing strategy. Second, a Compact Complex Linear (CCL) layer based on low-rank factorization is proposed to reduce classifier parameters by up to 56.8% while preserving discriminative capacity. Third, a novel complex-valued activation function, wcPReLUJitter, is designed to enhance learning stability and effectively mitigate training collapse in deep CVNNs. In addition, a high-redundancy input mapping strategy, termed RTC6, is investigated and systematically compared with existing complex-valued input representations. RTC6 is introduced as a high-redundancy benchmark for representation analysis rather than an input-efficiency module. Experimental results demonstrate that RTC6 can effectively compensate for performance degradation caused by aggressive parameter compression. Extensive evaluations on CIFAR-10 and CIFAR-100 (Canadian Institute for Advanced Research (CIFAR) datasets), Street View House Numbers (SVHN), and SAR datasets show that WCCN achieves competitive performance relative to representative baselines under the experimental and data settings in this paper. Notably, the proposed WCCN-M model achieves 73.17% mean accuracy on CIFAR-100, using significantly fewer parameters, which highlights its effectiveness for large-scale pattern recognition tasks.

KEYWORDS: Complex-valued neural networks; parameter efficiency; Wirtinger derivatives; input mapping strategy; synthetic aperture radar (SAR) image classification

1 Introduction

Complex-Valued Neural Networks (CVNNs) have demonstrated strong capability in processing signals where magnitude and phase jointly convey information. However, their application to real-valued image

classification tasks still faces several critical challenges in input mapping, computational efficiency, and training stability, which hinder broader adoption.

The first significant challenge lies in the lack of consensus regarding the mapping of real-to-complex input methods. Existing approaches encompass various strategies, ranging from color space transformations (e.g., red–green–blue (RGB) to CIELAB (LAB)) to direct channel recombination [1,2]. However, there is a lack of systematic comparison and statistical validation across different network architectures. Consequently, researchers often lack a solid basis for selecting appropriate mapping strategies for specific tasks.

Secondly, complex-valued operators face challenges in both computational efficiency and non-linear design. On the one hand, standard complex-valued convolution is mathematically equivalent to three or four real-valued convolutions (depending on whether the Gauss trick is applied); the overhead comes less from the arithmetic itself than from intermediate tensor format conversions and memory movement in typical implementations [2]. On the other hand, although modern CVNNs routinely adopt non-holomorphic activation functions (e.g., CReLU [2]) and use Wirtinger calculus—a well-established framework—for backpropagation, activation design still faces a practical trade-off between stability and expressiveness. Simple component-wise activations (e.g., CReLU) have limited feature expression capabilities in deep networks, while advanced activation functions introducing complex gating mechanisms (e.g., GTReLU [1]) are prone to gradient instability, leading to "Training Collapse" in deep architectures. This severely restricts the extension of high-performance complex-valued networks to deeper layers.

Thirdly, the classification structure suffers from low parameter efficiency and rigid configuration. As network depth increases and classification tasks become more complex, traditional complex-valued fully connected classifiers (Complex Linear) face severe parameter efficiency issues. The number of parameters grows linearly with the feature dimension and the number of classes [2]. This rigid structural design leads to a dual dilemma in model deployment and performance optimization: in resource-constrained edge computing scenarios, the massive parameter count constitutes a deployment bottleneck; conversely, in deep networks pursuing extreme performance, a fixed-capacity linear classification head struggles to flexibly adjust its feature integration capability to match complex feature spaces without significantly increasing the overall model burden. There is an urgent need for a classifier design that decouples model capacity from parameter scale, allowing for a flexible trade-off between efficiency and performance.

To tackle the key challenges identified earlier, this study proposes the **Wirtinger Derivative Complete Complex Network (WCCN)**, a systematic framework for efficient complex-valued deep learning, as illustrated in Fig. 1. WCCN achieves collaborative optimization across three levels: input mapping strategies, full-process efficient operators, and controllable classification structures. In this design, input mapping (including RTC6) is used for representation-side analysis, while the efficiency gain is mainly attributed to operator and classifier design. The unifying design philosophy is a budget-constrained co-design objective:

$$\max \ \text{Accuracy, Robustness, Efficiency} \quad \text{s.t.} \quad P \le B_P,\ T \le B_T,\ M \le B_M \tag{1}$$

where P, T, and M denote the total parameter count, training/inference latency, and peak memory, respectively. Under this constraint, the three components are co-designed rather than assembled independently: RTC6 supplies richer phase/correlation cues at a controlled input cost; wcConv (Gauss trick + tuple-flow) reduces backbone overhead, freeing latency and memory budgets; and CCL (controllable rank) reallocates the saved budget to an appropriately sized classifier head. The framework significantly reduces the number of model parameters and improves overall computational efficiency without compromising classification accuracy.

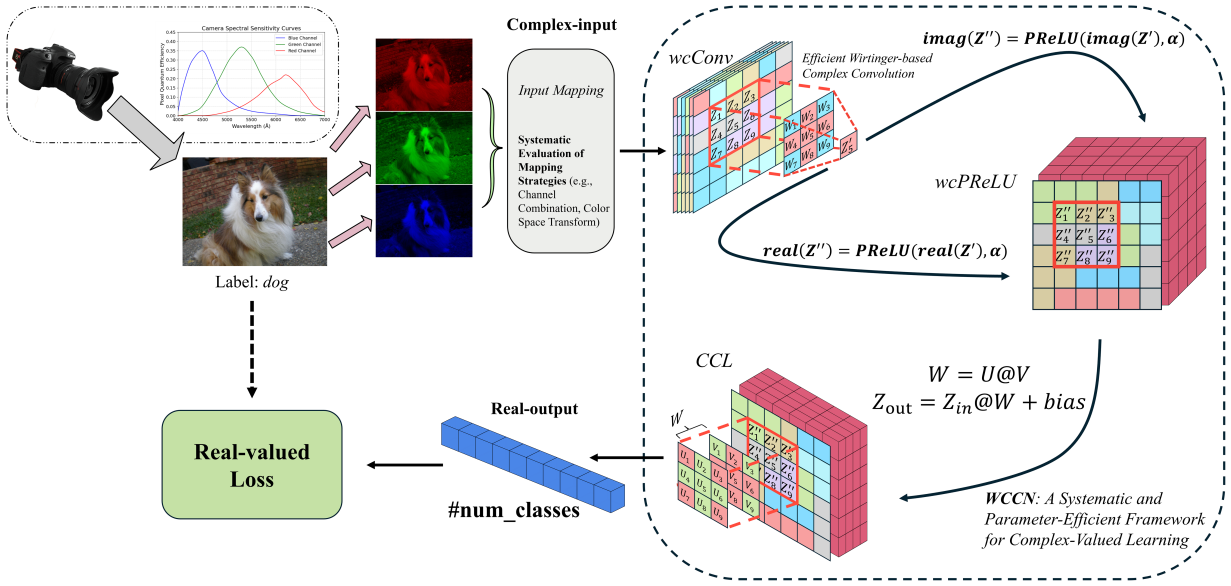


Figure 1: Overview of the efficient complex-valued network framework (WCCN).

The core contributions of this study are as follows:

(1) **Systematic Evaluation of Input Mapping Methods:** We construct a "Color Space Transformation × Input Redundancy" evaluation framework, conducting the first systematic quantitative analysis of real-to-complex mapping strategies. Empirical research indicates that shallow networks exhibit significant sensitivity to mapping strategies, while conventional deep networks generally show strong robustness. However, the proposed deep compact architecture, WCCN-M, breaks this convention. By adopting the high-redundancy RTC6 strategy, it effectively compensates for the model capacity limitations caused by substantial parameter reduction, achieving synergistic gains between input strategy and architectural design.

(2) **Efficient and Stable Complex-Valued Network Operator System:** We construct an efficient operator system comprising complex convolution (wcConv) and a novel complex-valued activation function family (wcPReLU and its variant wcPReLUJitter). wcConv is mathematically equivalent to standard complex convolution; our contribution lies in its optimized engineering implementation, which employs Gaussian decomposition (reducing four real convolutions to three) combined with a fused tuple-based processing strategy that eliminates redundant format conversions and intermediate buffer copies, yielding roughly 15%–20% end-to-end acceleration over conventional implementations. Back-propagation in all non-holomorphic components is supported by the well-established Wirtinger calculus framework. The proposed activation function family, by incorporating learnable complex parameters and a Phase-Magnitude Jittering mechanism, achieves performance superior to basic functions such as CReLU while effectively mitigating the training collapse often observed with complex gating activations like GTReLU in deep architectures, significantly enhancing convergence stability and robustness.

(3) **Compact Complex Classifier (CCL) Based on Controllable Complexity:** We propose a Compact Complex Linear (CCL) layer based on low-rank decomposition, empowering the model with the ability to trade off between parameter efficiency and performance flexibly. In shallow networks, CCL achieves up to 56.8% parameter compression through strict low-rank constraints, significantly enhancing parameter efficiency. In deep networks, benefiting from the efficiency of the WCCN-M backbone, CCL

strategically increases computational investment by raising the rank within deep structures without exceeding the parameter scale of comparison models (e.g., CDS-Large, from Co-domain Symmetry (CDS) [1]), thereby achieving stronger performance on complex tasks.

In summary, WCCN offers a comprehensive technical solution for the practical application of CVNNs in large-scale visual tasks, providing a new methodological reference for exploring more complex and efficient architectures.

2 Related Work

Research on Complex-Valued Neural Networks (CVNNs) aims to extend deep learning into the complex domain for amplitude-phase representation [3–5]. Recent work has also explored complex-valued learning for visual perception tasks [6] and remote-sensing-oriented complex-valued modeling [7]. Since the introduction of early complex-valued backpropagation algorithms [8,9], CVNNs have theoretically demonstrated superior generalization and expressive power [10,11]. However, as noted in recent surveys [5], transitioning CVNNs from theory to large-scale practical applications requires overcoming numerous challenges in operator implementation, non-linear design, lightweighting, and data interfaces, with existing technical components often scattered across disparate research efforts.

2.1 Real-to-Complex Input Mapping

Effectively mapping ubiquitous real-valued data (e.g., RGB images) into the complex domain is a prerequisite for the widespread application of CVNNs. Existing methods vary widely, including learning imaginary components through residual blocks [2], utilizing color space transformations (e.g., LAB [1]), and employing direct channel combinations (such as the Sliding method proposed in Co-domain Symmetry [1]). Another category of methods draws from signal processing, constructing complex-valued inputs via analytic signals or orthogonal transforms (e.g., Shearlet or Gabor filters), which have shown excellent performance in specific domains such as iris recognition [4,12]. Despite the existence of multiple mapping strategies, most of these methods are heuristic in nature. The field has long lacked a benchmark test under unified experimental settings to systematically compare mainstream mapping strategies (e.g., LAB, Sliding, Learned Imaginary Part, Analytic Signal/Orthogonal Coding) [5]. This lack of design basis means that front-end design choices for CVNNs dealing with real-valued data are often arbitrary and lack theoretical support.

2.2 Implementation of Complex Convolutions

Complex-valued convolution is the core operator of CVNNs. Early and influential works, such as Deep Complex Networks (DCN) [2], simulated a single complex convolution by combining four real-valued convolutions. While this method is functionally complete, combining the calculation results of each real-valued convolution into a complex value generates intermediate variables, leading to reduced efficiency. Another approach utilizes the convolution Theorem to implement convolution via element-wise multiplication in the frequency domain [13,14]. However, this approach relies on frequent and computationally expensive Fast Fourier Transforms (FFT) [15].

Most existing works have not actively pursued parameter compression through structured design at the kernel level (e.g., depthwise separable, low-rank, or polar parametrization). Consequently, low computational and parameter efficiency remain key obstacles hindering the development of CVNNs.

2.3 Lightweighting and Parameter Efficiency in CVNNs

As network depth increases, parameter redundancy becomes a critical factor limiting model performance and generalization capability [16]. In the field of CVNNs, exploration of lightweighting is relatively preliminary. Most works still employ traditional complex-valued fully connected layers as classifiers. Additionally, complex-valued networks have twice as many parameters as real-valued ones, making fully connected layers even more demanding. Some works, such as SurReal [17] and CDS [1], have proposed classifiers based on prototype distances, which reduce the number of parameters to some extent. Applying post-processing compression techniques, such as knowledge distillation, to CVNNs has also proven effective [18]. Other studies achieve parameter savings with complex pipelines, but do not analyze what drives this efficiency [6]. There remains significant room for further research and optimization regarding the parameter efficiency of CVNNs.

2.4 Non-Linearity and Network Construction in the Complex Domain

Activation functions introduce non-linearity to the network. In the complex domain, the design of activation functions faces a theoretical contradiction between analyticity and boundedness (Liouville's Theorem). This has given rise to the mainstream technical route of non-analytic activation functions, such as applying non-linearity separately to the real and imaginary parts (e.g., CReLU [2]). Recent works, such as Cardioid ReLU [19] and Cross-fused Split Activation [20], have further enriched the design space of activation functions. Simultaneously, the proposal of modules like Complex Batch Normalization (Complex BN) [2] has made the training of stable complex-valued networks possible. Although these studies have expanded the CVNN toolkit, they primarily focus on operator functionality, ignoring the impact of these designs on the overall parameter efficiency and redundancy of the network.

To conclude, while existing works have made significant progress in complex operators, activation functions, and input strategies, this research often remains fragmented. Previous efforts have either focused on constructing hybrid real-complex models [21] or addressing isolated problems, failing to resolve the core challenges CVNNs face in practical applications systematically: firstly, the lack of systematic comparison and statistical validation across different network architectures leaves design choices without a basis; secondly, the computation and design of core operators are constrained; and thirdly, structures like the classification head suffer from severe parameter redundancy, limiting model lightweighting and generalization. Developing a complex-valued network that efficiently optimizes the input, backbone, and classification head, while balancing parameters and computation, is crucial for large-scale applications in this field. The WCCN framework is proposed precisely to address these challenges.

3 Methods

The WCCN framework introduced in this paper is a complex-valued convolutional network designed to establish a theoretical foundation for selecting and developing methodologies that map real-valued images to the complex domain, while also improving the efficiency and parameter utilization of complex-valued operators. WCCN comprises three core components: an input mapping method module, an efficient complex convolution implementation (wcConv; Gauss trick + tuple-flow) under the standard complex convolution formulation (with Wirtinger calculus/Cauchy–Riemann (CR) calculus as a standard framework to support backpropagation through non-holomorphic components), and a lightweight complex-valued classification structure, complemented by additional complex-valued operation modules.

3.1 Systematic Evaluation Framework for Real-to-Complex Input Mapping

To systematically investigate the impact of real-to-complex mapping methods on the performance of Complex-Valued Neural Networks (CVNNs), we designed a dual-dimensional comparative framework. We categorize and analyze existing and proposed input strategies from two core perspectives: Color Space Transformation and Input Information Redundancy.

3.1.1 Mapping Methods Based on Color Space Transformation

These methods begin by transforming images from their original RGB format into a color space that offers enhanced perceptual qualities or is more appropriate for specific signal processing tasks. Afterwards, they create complex-valued representations within this new color space. In this study, we employ the **LAB Input** strategy as a representative approach. This method converts the image from RGB to CIELAB (CIE L*a*b*) space, utilizing the luminance channel L as an independent real-valued channel, while combining the chromaticity channels a and b into a complex-valued channel, as shown below:

$$\{R, G, B\} \rightarrow \{L^* + 0i, a^* + ib^*\} \tag{2}$$

3.1.2 Mapping Methods Based on Original RGB Channel Combination

These methods bypass color space transformation, directly leveraging the intensity values of RGB channels to construct complex-valued inputs. The primary distinctions lie in the channel combination techniques and the level of information redundancy introduced.

The most fundamental approach is the **RGB Baseline**, which serves as a direct benchmark. It assigns the three original channels (R, G, B) to the real parts of the complex input while setting the imaginary parts to zero, as defined in Eq. (3):

$$I_{re} = (R, G, B), \quad I_{im} = (0, 0, 0) \tag{3}$$

This method exhibits the lowest redundancy, as the imaginary part contributes no additional information.

Alternatively, the **Sliding Input** strategy exploits the correlation between adjacent channels. It constructs complex-valued inputs by pairing channels in a sliding window fashion:

$$\{R, G, B\} \rightarrow \{R + iG, G + iB\} \tag{4}$$

By incorporating intensity information from adjacent channels into the imaginary part, this method introduces a medium level of redundancy.

3.1.3 High Redundancy RGB Channel Combination: RTC6

To evaluate network performance under highly redundant input conditions, we propose the **Real-value To Complex-value Six-channel (RTC6)** strategy.

Based on the physical principle that spectral response curves of RGB channels in standard cameras exhibit substantial overlap, RTC6 is designed to generate highly redundant input data. It is important to note that we do not claim RTC6 to be the theoretically optimal coding strategy (which would likely require learning-based approaches); rather, it serves as a systematic "high-redundancy benchmark". This design aims to test whether robust complex-valued networks can refine useful features from information saturation or

if performance degrades. Specifically, RTC6 pairs all possible unique cross-channel combinations, assigning one as the real part and the other as the imaginary part:

$$\{R, G, B\} \rightarrow \{R + iG, R + iB, G + iB, G + iR, B + iR, B + iG\} \tag{5}$$

Although this linear combination process is simple, the resulting magnitude $|z|$ and phase θ of the complex feature $z = X + iY$:

$$|z| = \sqrt{X^2 + Y^2}, \quad \theta = \arctan(Y/X) \tag{6}$$

are non-linear functions of the original channel intensities. This effectively achieves non-linear coupling of cross-channel information at the feature level, providing a richer representation for subsequent operations.

To quantitatively validate the redundancy properties of each mapping strategy, we computed four redundancy metrics over 4000 randomly sampled images from the CIFAR-10 and CIFAR-100 training sets. Metrics are evaluated on the informative real-valued components (excluding constant zero imaginary parts): *Reuse Factor* = Components/Unique Sources (Components: the number of real-valued components after mapping; Unique Sources: the number of distinct source channels these components trace back to), Mean absolute Pearson correlation ($|corr|$), Mean Normalized Mutual Information (NMI), and Rank/Comp (numerical rank per component). Results are presented in Tables 1 and 2.

Table 1: Quantitative redundancy metrics of input mapping strategies (CIFAR-10, 4000 images).

Strategy	Comp.	Reuse Factor ↑	Mean $\|corr\|$ ↑	Mean NMI ↑	Rank/Comp ↓
LAB	3	1.000	0.060	0.050	0.471
RGB	3	1.000	0.868	0.235	0.469
Sliding	4	1.333	0.904	0.375	0.338
RTC6	12	4.000	0.904	0.443	0.117

Table 2: Quantitative redundancy metrics of input mapping strategies (CIFAR-100, 4000 images).

Strategy	Comp.	Reuse Factor ↑	Mean $\|corr\|$ ↑	Mean NMI ↑	Rank/Comp ↓
LAB	3	1.000	0.070	0.060	0.501
RGB	3	1.000	0.826	0.219	0.506
Sliding	4	1.333	0.872	0.359	0.364
RTC6	12	4.000	0.872	0.431	0.127

The arrows indicate the direction of redundancy (larger/smaller values correspond to higher redundancy) and are used for interpretability rather than implying that a metric is universally "better". RTC6 exhibits systematically elevated redundancy across all metrics: the highest reuse intensity, the strongest statistical dependency, and the most compact rank structure, quantitatively validating its positioning as a high-redundancy benchmark.

3.1.4 Comparative Experiment Design

To systematically evaluate these strategies, we classify the mapping methods into three levels of redundancy: low (RGB Baseline), medium (Sliding Input), and high (RTC6). These are cross-compared with

the color space transformation method (LAB) within a "Color Space Transformation × Input Redundancy" dual-dimensional experimental design. This framework allows for a comprehensive analysis of the impact of redundant information on model robustness and assesses the efficacy of different mapping strategies across various network architectures. Detailed performance evaluations, covering multiple datasets and architectures, are presented in Section 4, employing Friedman tests and Nemenyi post-hoc tests to verify statistical significance.

3.2 High-Efficiency Complex-Valued Operators and Network Components

To address the bottlenecks in computational efficiency and parameter overhead of CVNNs, we designed an efficient complex-valued network architecture centered on *wcConv*, complemented by our proposed Compact Complex Linear (CCL) classifier head and several supporting modules. This design adheres to the principle of "balancing performance and efficiency," improving computational speed and significantly reducing parameter count while maintaining feature expression capability.

3.2.1 wcConv: Efficient Complex Convolution Implementation

Standard complex-valued convolution $(x + iy) * (u + iv)$ is mathematically equivalent to a combination of real-valued convolutions. The PyTorch backend already applies the Gauss trick internally (see `aten/src/ATen/native/Convolution.cpp`), reducing the naïve four real convolutions to three. Nevertheless, the default implementation wraps inputs and outputs in complex tensors, incurring repeated packing/unpacking, intermediate buffer copies, and Python-side dispatch overhead.

The contribution of *wcConv* is not a new convolution algorithm but an **optimized engineering implementation** within the existing mathematical framework. To minimize these overheads, *wcConv* adopts two complementary strategies:

(1) **Gaussian decomposition (Gauss trick):** The complex convolution is implemented via three—rather than four—real-valued convolutions ($T_1 = \text{Conv}(X_r, W_r)$; $T_2 = \text{Conv}(X_i, W_i)$; $T_3 = \text{Conv}(X_r+X_i, W_r+W_i)$), directly reducing arithmetic cost.

(2) **Fused tuple-based processing (tuple-flow):** Complex features are maintained throughout the network as real–imaginary tuples (X_r, X_i) rather than being packed into high-level complex tensor objects. This eliminates the format-conversion and intermediate-copy overhead that dominates training latency in alternative implementations.

Backpropagation for all non-holomorphic components (including *wcConv* and the proposed *wcPReLU*) is supported by the well-established Wirtinger calculus (*CR*-calculus) [22]. We emphasize that Wirtinger calculus is not our methodological novelty; rather, it provides a standard gradient framework under which non-holomorphic modules can be consistently optimized. The speed improvement is entirely attributable to the implementation-level optimizations described above.

Table 3 summarizes the key differences among representative complex convolution implementations.

Table 3: Comparison of complex convolution implementation paths.

Implementation	#Real Convs ↓	Core Algorithm	Tensor Repr.	Primary Extra Overhead
wcConv (Ours)	3	Gauss + tuple-flow	(X_r, X_i) tuple	Low format-conversion cost

(Continued)

Table 3 (continued)

Implementation	#Real Convs ↓	Core Algorithm	Tensor Repr.	Primary Extra Overhead
Native complex conv	3	Gauss (backend)	Complex tensor	Temp copies/dispatch
CDS Complex-ConvFast	3	Gauss (explicit)	Stack $[B, 2, C, H, W]$	Stack/copy & graph build
DCN ComplexConv2d	4	Naïve 4-real	Complex + dual real conv	More conv calls & type conversions

In the weight initialization phase, we create separate tensors for real and imaginary parts using a Kaiming-uniform distribution and multiply by a scaling factor of $1/\sqrt{2}$ after synthesizing the complex weights to maintain signal variance stability during early training. Algorithm 1 describes *wcConv2d*; *wcConv1d* follows the same process, adjusting only the kernel and padding for 1D data.

Algorithm 1: wcConv2d (Forward Pass with Gauss Trick)

Input: $X_r, X_i \in \mathbb{R}^{B\times C_{in}\times H\times W}$
Output: $O_r, O_i \in \mathbb{R}^{B\times C_{out}\times H'\times W'}$
Precompute Weights: $W_{sum} \leftarrow W_r + W_i$
Forward (Gauss Trick):
    $T_1 \leftarrow \text{Conv2d}(X_r, W_r)$
    $T_2 \leftarrow \text{Conv2d}(X_i, W_i)$
    $T_3 \leftarrow \text{Conv2d}(X_r + X_i, W_{sum})$
Combine:
    $O_r \leftarrow T_1 - T_2$
    $O_i \leftarrow T_3 - T_1 - T_2$
Add Bias (optional): $O_r \leftarrow O_r + b_r, \quad O_i \leftarrow O_i + b_i$
Return: (O_r, O_i)

3.2.2 Compact Complex Linear Layer (CCL)

In Section 3.2.1, we introduced the efficient complex convolution operator *wcConv* as the network backbone. However, the design of the classifier head is equally crucial for an efficient network. Traditional complex classifiers suffer from severe parameter bottlenecks, where the number of learnable parameters P has a quasi-bilinear relationship with input feature dimension d and output class count C: $P \in O(d \cdot C)$.

Taking the common Complex Linear layer (CLinear) [2] and Prototype Distance layer [1,17] as examples, with input dimension $d = 128$ and output classes $C = 100$: CLinear requires two independent real linear layers, totaling $2 \times (d \times C + C) = 25800$ real parameters; Prototype Distance requires learning $d \times C$ complex prototypes plus a temperature parameter, totaling $2dC + 1 = 25601$ real parameters. Such parameter overhead contradicts the design philosophy of WCCN, which pursues extreme lightweighting.

To address this bottleneck, we evaluated several lightweight design options under a unified benchmark and found low-rank decomposition to provide the most favorable efficiency-accuracy trade-off in our setting.

Based on this, we propose the **Compact Complex Linear Layer (CCL)**. The core idea is that the complex weight matrix $W \in \mathbb{C}^{d\times C}$ is typically over-parameterized, and its intrinsic effective rank is far smaller than its dimensions. CCL does not directly learn W but decomposes it into the product of two low-rank matrices: $W \approx U \cdot V$, where $U \in \mathbb{C}^{d\times r}$, $V \in \mathbb{C}^{r\times C}$, and rank $r \ll \min(d, C)$. This reduces parameter complexity from $O(d \cdot C)$ to $O(r \cdot (d + C))$. At $r = 32$, the compression rate reaches 42.7% compared to the CLinear example.

Beyond parameter compression, the low-rank structure of CCL provides flexibility in controlling model capacity. This frees network design from the constraints of fixed-scale fully connected layers: in resource-constrained scenarios, a small value for r can be selected for lightweighting, while in performance-oriented deep networks, r can be moderately increased to reinvest the saved parameter budget into the classifier head. This allows the model to break through performance bottlenecks by enhancing feature integration capability without significantly increasing total parameters.

We assess CCL by comparing it with various classifier strategies: baseline methods (CLinear, Prototype Distance), factorization methods (DistF, DistF_Improved), adaptive approaches (AdaptiveRankClassifier), structural sparsity methods (GroupedCLinear), and hybrid architectures (HybridComplexClassifier). Detailed benchmark results in Section 4 will empirically demonstrate CCL's superior trade-off between parameters and performance.

3.2.3 Complex-Valued Activation Functions

(1) wcPReLU: Inspired by real-valued PReLU, wcPReLU introduces learnable negative slope parameters in the complex domain, effectively mitigating the "dying ReLU" problem. Specifically, it applies PReLU to the real and imaginary parts of the complex input separately but forces them to share the same set of learnable slope parameters. This avoids learning conflicting representations between real and imaginary parts, thereby enhancing training stability. Its definition is:

$$f(z, \alpha_c) = \mathrm{PReLU}(x, \alpha_c) + i \cdot \mathrm{PReLU}(y, \alpha_c) \tag{7}$$

where α_c is the channel-wise learnable slope. While non-analytic, this design is compatible with the Wirtinger/CR-calculus framework used for backpropagation in WCCN.

(2) wcPReLUJitter: To address signal drift and noise in complex scenarios, we propose the *wcPReLUJitter* module. It simulates real-world signal variations to encourage robust feature learning through two mechanisms:

Phase-Magnitude Jittering: With probability p_{jitter}, we perturb z in polar coordinates. The perturbed z' is calculated as:

$$z' = z \cdot \mathrm{polar}(1+ \epsilon_{mag}, \epsilon_{phase}) \tag{8}$$

where $\epsilon_{mag} \sim \mathcal{N}(0, \sigma_{mag})$ and $\epsilon_{phase} \sim \mathcal{N}(0, \sigma_{phase})$ are sampled from Gaussian distributions.

Conjugate Mixing: With probability p_{mix}, the perturbed activation z' is linearly mixed with its conjugate $\overline{z'}$:

$$z'' = \lambda z' + (1 - \lambda)\overline{z'} \tag{9}$$

where $\lambda \sim U(0,1)$. The final output is then passed through the standard wcPReLU: $z_{out} = \mathrm{wcPReLU}(z'')$.

3.2.4 Supporting Complex-Valued Modules

To ensure stability and efficiency, we designed supporting modules adapted for the complex domain. **wcBatchNorm** employs a split batch normalization strategy, applying standard BN to real and imaginary parts separately. This avoids complex covariance calculations, reducing overhead while maintaining stability. Similarly, **wcPooling** implements split pooling, performing average or max pooling independently on real and imaginary parts (wcAvgPool, wcMaxPool) [2]. These components ensure the core operators function at maximum efficacy within the WCCN framework.

4 Experiments and Analysis

This section aims to comprehensively evaluate the effectiveness of the proposed WCCN framework through a series of experiments. We first establish a unified experimental benchmark, then demonstrate the performance comparison between WCCN and current mainstream methods across multiple datasets. Finally, we use ablation studies and mechanistic analyses to examine the sources of WCCN's gains in input strategy, network components, and parameter efficiency.

4.1 Experimental Setup

4.1.1 Datasets

Our experiments encompass three widely used real-valued image classification datasets and three open-source native complex-valued datasets:

(1) **CIFAR-10/CIFAR-100 [23]:** These datasets contain 32×32 color images across 10 and 100 classes, respectively, comprising 50,000 training images and 10,000 test images. They serve as benchmark data for evaluating model generalization capabilities in multi-class classification tasks.

(2) **SVHN [24]:** The Street View House Numbers dataset contains over 600,000 real-world digit images, used to test model robustness under varying lighting, viewpoints, and styles.

(3) **Open-Source SAR Datasets [7]:**

1) *Flevoland-1989*: L-band fully polarimetric SAR images collected by the National Aeronautics and Space Administration (NASA)/Jet Propulsion Laboratory (JPL) Airborne Synthetic Aperture Radar (AIRSAR) platform in 1989, covering the Flevoland agricultural area in the Netherlands. It is a classic benchmark for polarimetric SAR classification.
2) *Flevoland-1991*: L-band data of the same area collected by the AIRSAR platform in 1991, used for cross-comparison and model generalization evaluation.
3) *Oberpfaffenhofen*: L-band multi-look SAR images collected by the German Aerospace Center (DLR) Experimental Synthetic Aperture Radar (E-SAR) platform, covering the Oberpfaffenhofen area in Germany, featuring diverse terrain types such as urban areas, forests, and open lands.

4.1.2 Comparison Models and Reproducibility

To conduct a comprehensive performance evaluation, we compare WCCN with the following mainstream real-valued and complex-valued network models: **CNN** (a standard real-valued convolutional neural network baseline), **DCN** [2], **SurReal** [17], **CDS** [1], and **CV-CNN** [7]. Except for the CV-CNN model, all comparison models were retrained and tested using open-source code from their original papers on a machine with an Intel Core i7-12700 central processing unit (CPU), NVIDIA RTX 4070 graphics processing unit (GPU), and 32 GB random-access memory (RAM) to ensure fairness.

4.1.3 Data Splitting

For the SAR experiments, all reported results directly use the split files provided by the original authors [7] (without redefining the ratios). The training, validation, and test subsets are mutually exclusive, and all methods are evaluated under the identical partitioning scheme. We also report the mean ± standard deviation across 5 random seeds on the fixed split to separate optimization randomness from partitioning effects. The specific splits for the three datasets are as follows:

(1) *Flevoland-1989* (15 classes, 15 × 15 patches): Development partition represents approximately 10%, containing about 9% for training and 1% for validation (10,865 train/1333 val), leaving the remaining 143,898 instances for testing.
(2) *Flevoland-1991* (14 classes): Exactly 850 for training and 150 for validation per class (Total: 11,900 train/2100 val), leaving the remaining 121,350 instances for testing.
(3) *Oberpfaffenhofen/Germany* (3 classes): Development partition is approximately 1%, containing about 0.9% for training and 0.1% for validation (11,804 train/1310 val), leaving the remaining 1,298,504 instances for testing.

4.1.4 Training Details

Shallow Models: For shallow models (including CNN, DCN, SurReal, CDS-I, and WCCN), we employ the Adam with decoupled weight decay (AdamW) optimizer [25] for training. The learning rate is set to 1×10^{-3}, the weight decay coefficient is 0.1, and the momentum parameters (β_1, β_2) are set to (0.99, 0.999). All models use a batch size of 256 and are trained for a total of 50,000 iterations (steps). During training, we evaluate model performance on the validation set every 1000 iterations and report the best results.

Deep Models: For deep models (including DCN, CDS-Large, and WCCN-M), we use the stochastic gradient descent (SGD) optimizer with momentum set to 0.9 and a weight decay coefficient of 5×10^{-4}. The batch size is set to 64, and training lasts for 200 epochs. A piecewise linear learning rate schedule is adopted: warming up to 0.2 in the first 10 epochs, then decaying to 0.01, 0.001, and 0.0001 at epochs 100, 120, and 150, respectively. Models are evaluated on the validation set after each epoch, and the best performance is saved.

To evaluate stability and variance, unless otherwise specified, all experiments are independently reproduced 5 times using different random seeds. Significance testing uses $\alpha = 0.05$. To unify reporting across sections, all quantitative results in this manuscript are presented in the form of mean ± standard deviation.

4.1.5 Evaluation Metrics

The core evaluation metrics for all experiments in this study are Classification Accuracy and Parameter Count. Parameter reporting can be misleading if the underlying tensor representation is not normalized: some implementations store learnable weights as complex tensors (e.g., `torch.cfloat`), while others store real and imaginary parts as separate real-valued tensors (or stacked real tensors). Standard statistical tools (e.g., *torchinfo*) count one complex scalar as one parameter by default, even though it corresponds to two real numbers in storage and computation. Therefore, directly comparing raw parameter counts can be unfair across implementations.

To ensure a fair comparison of parameter counts across different models, this paper adopts **Equivalent Real Parameter Count** as the unified statistical standard. Specifically, since a complex parameter is equivalent to two real-valued parameters in terms of storage and computation, the equivalent real-valued parameter count is calculated as:

$$P_{equiv} = P_{real} + 2 \times P_{complex} \tag{10}$$

All parameter scales reported in subsequent experiments are based on this statistical scheme to ensure fairness and accuracy. Additionally, model parameter counts are influenced by the number of input channels. To ensure consistency, parameter counts in Tables 4 and 5 are calculated based on dual-channel input.

Table 4: Performance comparison of shallow models on real-valued datasets (Mean ± Std).

Model	CIFAR-10		CIFAR-100		SVHN
	#Params ↓	Acc (%) ↑	#Params ↓	Acc (%) ↑	Acc (%) ↑
CNN	34,426	65.62 ± 1.24	46,036	34.52 ± 0.38	88.52 ± 0.85
DCN	66,858	65.18 ± 0.73	78,468	33.95 ± 0.55	87.62 ± 0.77
SurReal	15,092	53.20 ± 0.66	26,702	25.42 ± 0.15	83.35 ± 0.19
CDS-I	24,241	69.07 ± 0.81	47,281	38.34 ± 0.73	89.01 ± 0.10
WCCN (Ours)	20,868	70.33 ± 0.65	25,368	41.63 ± 0.63	88.07 ± 0.27

Table 5: Deep model performance on CIFAR-10/100 (Params: 2ch baseline; 5 runs, seeds 40–44).

Model	CIFAR-10		CIFAR-100	
	#Params ↓	Acc (%) ↑	#Params ↓	Acc (%) ↑
DCN	1.746M	92.8	1.792M	–
CDS-Large	1.748M	93.42 ± 0.26	1.794M	72.95 ± 0.39
WCCN-M (Ours)	1.343M	93.23 ± 0.20	1.370M	73.17 ± 0.34

4.2 Main Performance Comparison

To intuitively compare the comprehensive performance of various frameworks, this subsection presents the final performance of WCCN against multiple baseline models across different tasks. The accuracy figures in this section are reported in the unified format of mean ± standard deviation. For models evaluated under multiple input strategies in prior experiments, we report the best-performing strategy by mean accuracy together with its corresponding standard deviation. A systematic analysis of the impact of different input strategies, as well as ablation studies of WCCN components, is provided in Section 4.3.

4.2.1 Performance on Real-Valued Datasets

Table 4 systematically compares the parameter scale and classification accuracy of WCCN with mainstream shallow network models on the CIFAR-10, CIFAR-100, and SVHN datasets. On CIFAR-10, WCCN achieves the highest mean accuracy of 70.33% ± 0.65% with only 20,868 parameters (the second fewest among all models). Notably, although the SurReal model has the fewest parameters (15,092), its best-strategy mean accuracy is only 53.20% ± 0.66%, significantly lower than other models, highlighting WCCN's superior balance between performance and efficiency. Furthermore, WCCN reduces the parameter count by 13.9% compared to CDS-I (24,241), which has relatively close performance, further demonstrating the model's compactness.

On the CIFAR-100 dataset, WCCN achieves "dual optimality" in both parameter count and accuracy. Its parameter count (25,368) is the lowest among all models, while its best-strategy mean accuracy (41.63% ± 0.63%) is the highest. Compared to SurReal (26,702), which has the next lowest parameter count, WCCN improves accuracy by 16.21 percentage points while reducing parameters by 5.0%. Against CDS-I (47,281)

and CNN (46,036), WCCN achieves leading classification performance with only 53.7% and 55.1% of their respective parameter counts.

On the SVHN dataset, WCCN similarly achieves the second-highest best-strategy mean accuracy of 88.07% ± 0.27% with only 20,868 parameters (the second fewest among all models). Although CDS-I reaches the highest mean accuracy (89.01% ± 0.10%), WCCN remains competitive while reducing parameters by 13.9% relative to CDS-I (24,241), demonstrating high efficiency.

WCCN exhibits particularly outstanding scalability. As the classification task expands from 10 classes (CIFAR-10) to 100 classes (CIFAR-100), WCCN's parameter count increases by only 4500, the smallest increment among all models. In contrast, CNN, DCN, and SurReal all show an increase of 11,610 parameters, and CDS-I increases by as much as 23,040. WCCN's parameter growth is 61.2% less than the smallest increment among comparison models (CNN/DCN), fully proving its parameter efficiency and structural compactness in large-scale category expansion scenarios.

To systematically evaluate the scalability of the WCCN framework in deeper and more complex network structures, this paper constructs a larger-scale variant, WCCN-M, and compares it with the classic baseline DCN and the high-performance model CDS-Large, which share the same deep architecture. Quantitative results are shown in Table 5.

On CIFAR-10, WCCN-M achieves 93.23% ± 0.20% classification accuracy using only 1.343M parameters (the fewest among the three), reducing parameters by approximately 23.1% compared to DCN (1.746M) and CDS-Large (1.748M), with a 0.19 percentage-point gap to the best baseline (CDS-Large: 93.42% ± 0.26%). For CIFAR-10, the DCN baseline value is directly cited from the CDS paper and is not reproduced in this work, while the CDS-Large result is from our practical reproduction.

On CIFAR-100, WCCN-M demonstrates dual optimality in parameter efficiency and classification performance. Its parameter count is 1.370M (lowest), a 23.6% reduction compared to CDS-Large (1.794M), while achieving a best-strategy mean accuracy of 73.17% ± 0.34%, which is higher than CDS-Large's best-strategy mean accuracy (72.95% ± 0.39%). Note that the original DCN paper [2] did not provide the deep network structure or classification results for CIFAR-100; thus, no reproduction comparison experiment was conducted for this case.

To avoid presentation inconsistency, all dataset-level numbers reported in the main text are synchronized with the corresponding table entries (same statistic definition and rounding precision).

4.2.2 Performance on Native Complex-Valued Dataset

To ensure fairness and reproducibility, this study fully reproduced the representative method CV-CNN [7] within the PyTorch framework. The network structure, training hyperparameters, and data preprocessing pipeline strictly followed the original literature and the official MATLAB (MATrix LABoratory) implementation. Because the original 10% training set indices are unavailable, our reproduced results differ slightly from the reported values. WCCN and CV-CNN employ identical model architectures and experimental configurations, including input representation, optimizer settings, and random seed initialization, ensuring training and testing under unified conditions.

As shown in Table 6, WCCN outperforms the reproduced CV-CNN on four evaluation metrics and ties on one metric across the three polarimetric synthetic aperture radar (PolSAR) benchmark datasets. The largest gain is +2.33 percentage points on Oberpfaffenhofen OA raw, while OA post on Flevoland-1991 shows a slight decrease of 0.03 percentage points (99.12% vs. 99.15%), which remains within the reported standard deviation range.

Table 6: Performance comparison on open-source SAR datasets (OA: overall accuracy; raw/post denote before/after post-processing).

Model	Flevoland-1989		Flevoland-1991		Oberpfaffenhofen	
	OA raw (%) ↑	OA post (%) ↑	OA raw (%) ↑	OA post (%) ↑	OA raw (%) ↑	OA post (%) ↑
CV-CNN (paper)	96.20	97.70	99.00	–	93.40	–
CV-CNN (repro)	96.98 ± 0.22	97.67 ± 0.21	99.45 ± 0.06	99.15 ± 0.04	89.96 ± 0.29	90.94 ± 0.35
WCCN (Ours)	97.09 ± 0.36	97.78 ± 0.34	99.45 ± 0.10	99.12 ± 0.10	92.29 ± 0.44	93.18 ± 0.45

4.3 Systematic Analysis of Input Mapping Strategies

To evaluate the impact of real-to-complex mapping strategies on the performance of shallow Complex-Valued Neural Networks (CVNNs), this study conducts a systematic cross-analysis of four input strategies based on the dual-dimensional comparative framework proposed in Section 3.1. Table 7 presents the classification accuracy ($Mean \pm Std$) of all models on the CIFAR-10 dataset. The experimental results indicate that input strategies significantly and consistently influence model performance. Specifically, under the high-redundancy RTC6 strategy, the proposed WCCN method achieves the optimal average accuracy of 70.33% ± 0.65%, representing an improvement of 0.35 percentage points over the low-redundancy RGB baseline and 1.74 percentage points over the LAB strategy.

Table 7: Performance comparison of shallow models on CIFAR-10 under different input strategies.

Method	LAB (%) ↑	RGB (%) ↑	Sliding (%) ↑	RTC6 (%) ↑
CNN	61.62 ± 1.86	64.43 ± 1.27	65.62 ± 1.24	63.97 ± 1.14
DCN	63.48 ± 1.52	64.74 ± 0.42	65.18 ± 0.73	64.23 ± 0.93
SurReal	40.42 ± 0.46	53.02 ± 0.22	50.06 ± 0.31	53.20 ± 0.66
CDS-I	65.21 ± 0.74	69.06 ± 0.65	68.45 ± 0.63	69.07 ± 0.81
WCCN (Ours)	68.59 ± 0.47	69.98 ± 0.49	68.60 ± 0.63	70.33 ± 0.65

A cross-model comparison of input strategies (Fig. 2b) clearly reveals that strategies based on original RGB channel combinations (RTC6, RGB, Sliding) significantly outperform the LAB color space transformation strategy in terms of average performance. As shown in Fig. 2c, the cross-model average accuracy for the LAB strategy is only 59.86%, whereas RTC6, RGB, and Sliding achieve 64.16%, 64.25%, and 63.58%, respectively. Furthermore, the heatmap in Fig. 2a validates this conclusion: the LAB strategy performs the worst across all five models, particularly on the SurReal model, where accuracy drops to 40.42%. This phenomenon suggests that LAB color space transformation may disrupt the physical correlations between original RGB channels, thereby negatively affecting model performance.

In addition, Fig. 2d summarizes model-wise averages across all four input strategies, showing that WCCN has the highest cross-strategy mean (69.38%), followed by CDS-I (67.95%), while SurReal is substantially lower (49.18%). This confirms that WCCN's gain is not tied to a single mapping, but remains robust under strategy variation.

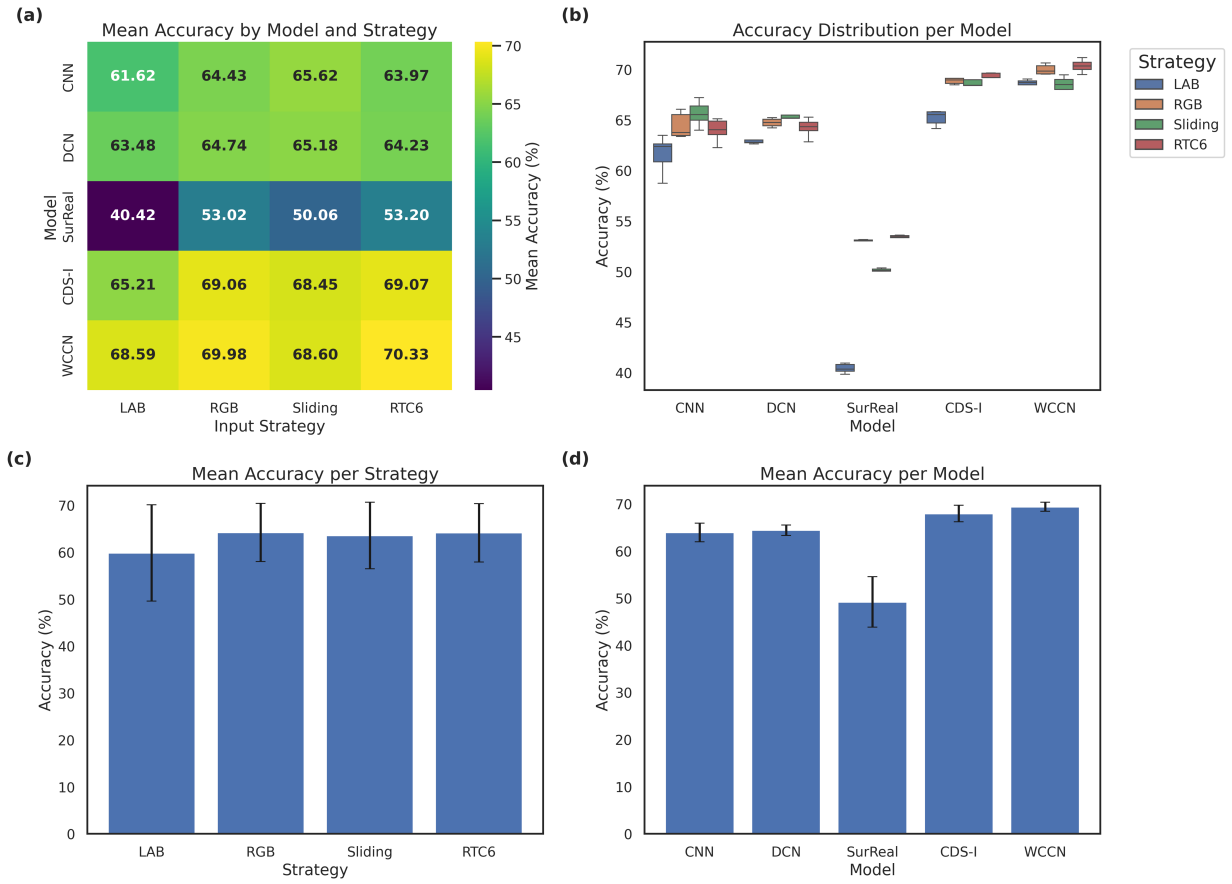


Figure 2: Comprehensive performance comparison of shallow models on the CIFAR-10 dataset under varying input strategies. (**a**) Heatmap of mean classification accuracy, plotting four input strategies (LAB, RGB, Sliding, RTC6) against five models (CNN, DCN, SurReal, CDS-I, WCCN). (**b**) Accuracy distributions of the five models across all strategies, showing test accuracy variations. (**c**) Mean accuracy of each input strategy, averaged across all models. (**d**) Mean accuracy of each model, averaged across all input strategies.

To verify the cross-comparison hypothesis designed in Section 3.1.4 from a statistical perspective, we conducted Friedman and Nemenyi tests following a randomized block design. Specifically, we use $n = 5$ blocks (random seeds 40–44) over $k = 4$ conditions (LAB, RGB, Sliding, RTC6). Ranks are computed within each seed (block) to control the variance introduced by random initialization, and the Nemenyi post-hoc test uses the same fully paired samples rather than treating the $n \times k = 20$ observations as independent.

Results show that WCCN ($p = 0.007$), SurReal ($p = 0.003$), and CDS-I ($p = 0.007$) exhibit highly significant differences at the $p < 0.01$ level; CNN ($p = 0.029$) reaches significance at the $p < 0.05$ level; while DCN ($p = 0.145$) does not reach the significance threshold.

The Nemenyi post-hoc test (Fig. 3) provides finer statistical support for these conclusions. In SurReal (Fig. 3c) and CDS-I (Fig. 3d) models, the LAB strategy is statistically significantly inferior to the RGB ($p = 0.036$; $p = 0.036$) and RTC6 ($p = 0.003$; $p = 0.008$) strategies based on original channels.

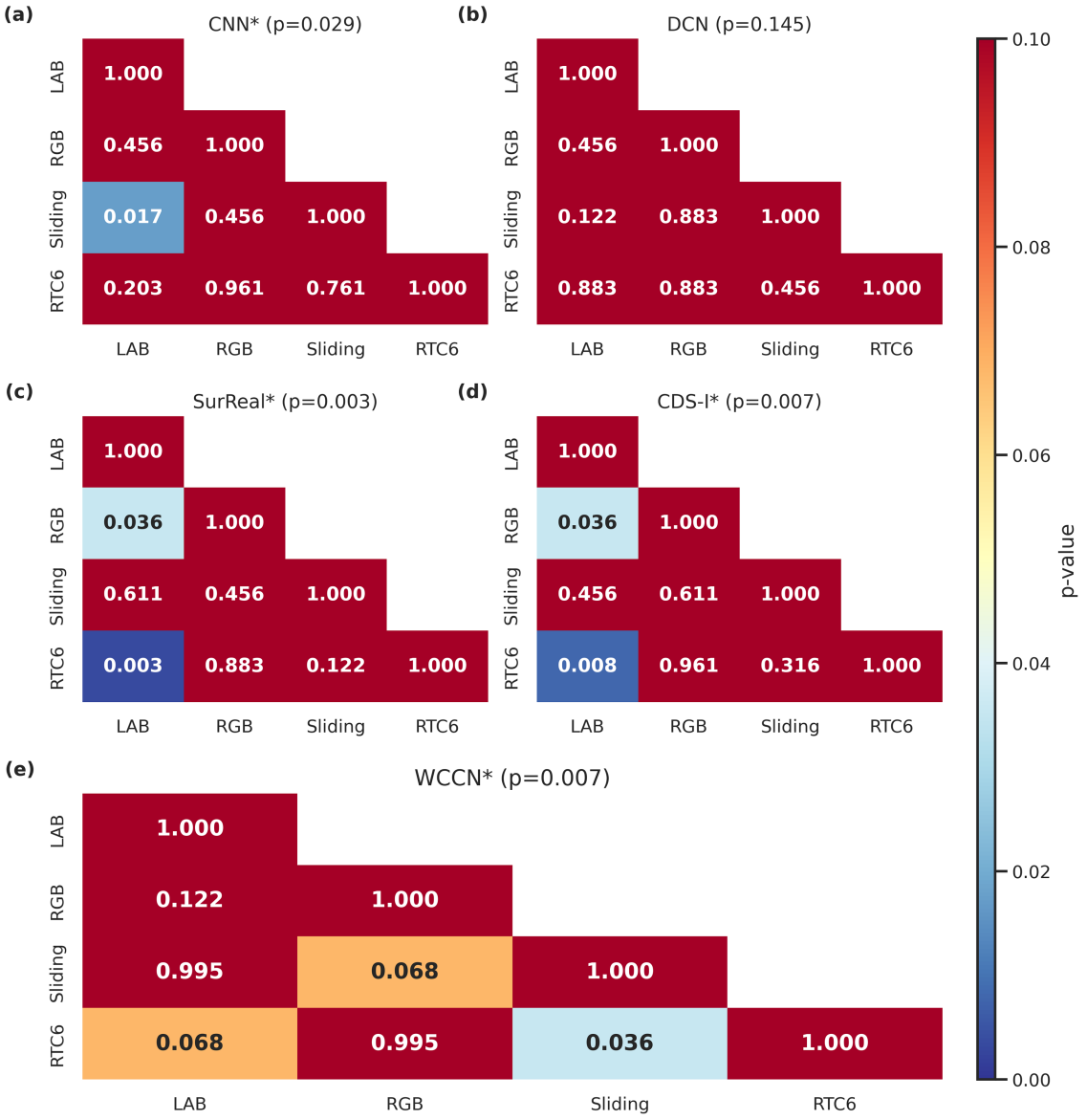


Figure 3: Nemenyi post-hoc test results for different input strategies across various shallow models on CIFAR-10. (**a**)–(**e**) Results for CNN, DCN, SurReal, CDS-I, and WCCN, respectively. Titles display Friedman test p-values, with asterisks (*) indicating statistical significance ($p < 0.05$). Heatmaps visualize pairwise p-values from Nemenyi post-hoc tests, quantifying the significance of performance differences between input strategies (LAB, RGB, Sliding, RTC6). Lower p-values denote more pronounced disparities. The analysis reveals that SurReal, CDS-I, and WCCN exhibit substantial sensitivity to input representation choices.

Regarding the second dimension proposed in Section 3.1 (redundancy levels), a comparative analysis of RTC6 (High), Sliding (Medium), and RGB (Low) strategies reveals a more complex pattern. In SurReal and CDS-I models, no statistically significant difference exists among these three strategies, indicating some robustness to medium-to-high redundancy information. However, the proposed WCCN model (Fig. 3e) exhibits distinct characteristics: the high-redundancy RTC6 strategy is significantly superior to the medium-redundancy Sliding strategy ($p = 0.036$), which is the only statistically significant strategy contrast within the WCCN model. Although RTC6 (70.33%) and RGB (69.98%) both outperform Sliding (68.60%) in

absolute accuracy, the difference between RGB and Sliding does not reach statistical significance ($p = 0.068$). Additionally, no significant differences exist between the LAB strategy and the other three strategies (RGB: $p = 0.122$; Sliding: $p = 0.995$; RTC6: $p = 0.068$).

However, a more important trend is that as model expressive power increases (e.g., from CNN to WCCN), performance differences caused by different input strategies are narrowing. To further verify the limit case of this law, this study repeated the experiment on the more challenging CIFAR-100 dataset using the more expressive CDS-Large and WCCN-M models (see Table 8).

Table 8: Performance comparison of deep models on CIFAR-100 under different input strategies.

Method	LAB (%) ↑	RGB (%) ↑	Sliding (%) ↑	RTC6 (%) ↑
CDS-Large	72.95 ± 0.39	72.81 ± 0.26	72.67 ± 0.36	72.90 ± 0.10
WCCN-M	72.80 ± 0.27	72.51 ± 0.11	72.85 ± 0.59	73.17 ± 0.34

Experimental results demonstrate that deep models exhibit strong strategic robustness. Unlike the distinct fluctuations seen in shallow models of the same series—for instance, shallow CDS-I shows an accuracy range (difference between max and min) of 3.86% with LAB significantly lagging—deep CDS-Large and WCCN-M show average accuracy ranges of only 0.28% and 0.66%, respectively. Especially for the LAB strategy, which shallow networks struggle to handle, deep models achieve parity with other strategies.

The Friedman test further confirms this trend (as shown in Fig. 4): the performance distribution of the CDS-Large model under four strategies shows no statistically significant difference (Friedman $p = 0.733$, Fig. 4a). This implies that when model capacity is sufficient, its powerful non-linear feature extraction capability is enough to compensate for information differences caused by input mapping methods, allowing the model to learn equivalent features from different complex-valued representations adaptively.

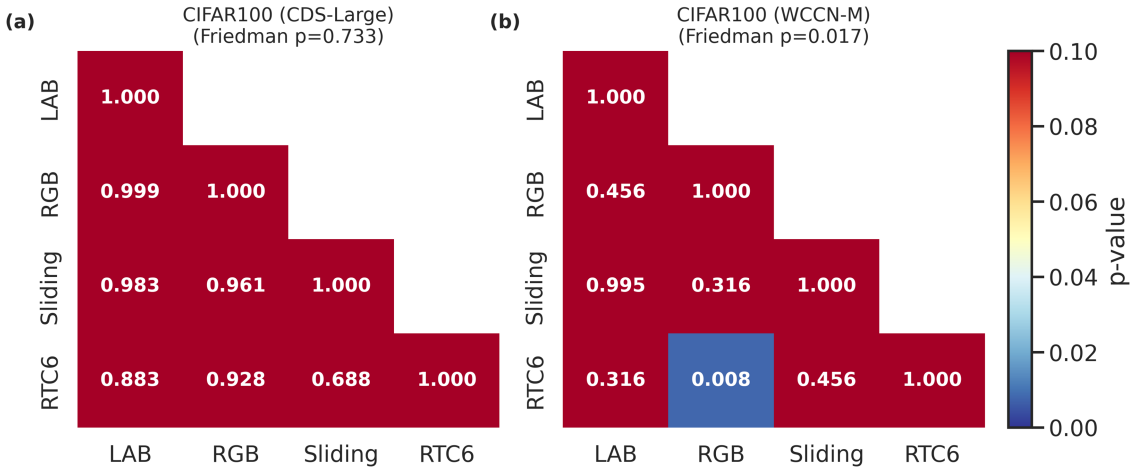


Figure 4: Nemenyi post-hoc test results for different input strategies on deep models (CIFAR-100). (**a**)–(**b**) Results for CDS-Large and WCCN-M, respectively. Titles display Friedman test p-values, assessing the statistical significance of performance variations across input strategies (LAB, RGB, Sliding, RTC6). Heatmaps visualize pairwise p-values from Nemenyi post-hoc tests, where lower values indicate more significant disparities. Notably, WCCN-M exhibits significant sensitivity to input strategy choice ($p = 0.017$), whereas CDS-Large shows no statistically significant differences ($p = 0.733$).

Despite the overall trend towards convergence, the WCCN-M model still demonstrates the uniqueness of its architecture. As shown in Fig. 4b, the Friedman test result for WCCN-M is $p = 0.017$ ($p < 0.05$), indicating statistical differences still exist between strategies. The Nemenyi post-hoc test reveals that this difference primarily stems from the contrast between the high-redundancy RTC6 strategy and the low-redundancy RGB strategy ($p = 0.008$). This illustrates that even in deep networks, the unique design of the WCCN architecture can effectively utilize the extra phase redundancy provided by the RTC6 strategy to boost performance (RTC6 achieved the highest average accuracy of 73.17%).

Synthesizing the above experimental data, we draw the following conclusions: First, shallow networks show significant sensitivity to input mapping strategies, while conventional deep networks with sufficient capacity (like CDS-Large) exhibit strong robustness (Friedman $p = 0.733$); Second, the compact deep architecture WCCN-M breaks this convention. Experiments show that the high-redundancy RTC6 strategy effectively compensates for the capacity limitations brought by significant parameter reduction (−23.6%) in WCCN-M, thereby significantly improving classification accuracy. These results suggest a compensation effect and a positive interaction between high-redundancy input and the compact WCCN architecture.

4.4 Ablation Studies and Mechanistic Analysis

In Section 4.3, we analyzed the impact of different input mapping strategies on model performance, elucidating the role of input design in both shallow and deep networks. This section focuses on the core structure of the WCCN framework. Through ablation experiments and mechanistic analysis, we explore the impact of key modules (such as efficient complex convolution, compact complex classifier head, and complex activation functions) on overall performance, helping readers understand the underlying reasons for performance improvements.

4.4.1 CCL Performance Benchmark and Rank Selection Strategy

To address the excessive parameters in complex-valued classifiers, we investigated classifier head structures and proposed the **Compact Complex Linear Layer (CCL)** based on low-rank decomposition. Using WCCN as the backbone and wcPReLU as the activation function, we tested different classifier heads to analyze the impact of factorization on parameter count and performance.

For reproducibility, the key head configurations are fixed as follows: CCL ($rank = 24$), DistF ($codebook\_size = 32$), DistF_Improved ($codebook\_size = 32$), HybridComplexClassifier ($num\_prototypes = 16$), GroupedCLinear ($num\_groups = 4$), AdaptiveRankClassifier ($initial\_rank = 32$), and wcDistQuantized ($num\_bits = 8$).

Table 9 benchmarks nine complex-valued classifier heads with two parameter views: trainable parameters and non-trainable buffer states. Results show that while the traditional CLinear achieves the highest mean accuracy (42.57% ± 0.74%), its trainable parameter cost is substantial at 25,800. The Dist method has a comparable trainable count (25,601) but drops in performance to 40.15% ± 0.38%. Our CCL ($rank = 24$) delivers 41.63% ± 0.63% accuracy and cuts trainable parameters by 56.8% to 11,144, showing a strong efficiency-performance trade-off. Notably, CCL outperforms several other compact alternatives such as AdaptiveRankClassifier (37.44% ± 0.57%) and GroupedCLinear (35.45% ± 0.38%).

Here, "#Trainable Head Params" counts only learnable parameters in the classifier head, while "#Non-trainable Buffers" reports stored state tensors that participate in forward inference but are not updated by backpropagation. For example, DistQuantized has three learnable scalars (two scales and one temperature), while its quantized prototype tensors are stored as buffers (25,600 entries), so "3" is valid for trainable-parameter accounting but not for total stored state.

Table 9: Benchmark comparison of different complex-valued classifier heads.

Classifier Head	#Trainable Head Params ↓	#Non-Trainable Buffers	CIFAR-100 Accuracy (%) ↑
CLinear	25,800	0	42.57 ± 0.74
CCL (rank=24)	11,144	0	41.63 ± 0.63
DistF	14,593	0	40.70 ± 0.59
HybridComplexClassifier	61,401	0	40.74 ± 0.45
Dist	25,601	0	40.15 ± 0.38
DistF_Improved	14,593	3200	39.16 ± 0.54
AdaptiveRankClassifier	4770	0	37.44 ± 0.57
GroupedCLinear	6632	0	35.45 ± 0.38
DistQuantized	3	25,600	31.56 ± 0.58

To further verify the computational efficiency of CCL, we designed an inference speed benchmark. This test directly compares the inference time of three representative complex classifier heads (CLinear, Dist, CCL), systematically evaluating performance across different combinations of input feature dimensions (2^7 to 2^{11}, i.e., 128 to 2048) and output class counts (10^1 to 10^3, i.e., 10 to 1000). All tests were conducted on an NVIDIA RTX 4070 GPU (Configuration: Rank = 24, Trials = 20, Device = Compute Unified Device Architecture (CUDA) 12.9). Each configuration ran independently 20 times after warm-up and memory clearing, recording the average value to ensure precision and stability.

As shown in Fig. 5, CCL exhibits significant advantages in inference speed. Under the standard configuration (Input Features = 512, Output Features = 100), CCL's average inference time is only 0.200 ms, corresponding to speedups of 1.8× and 14.7× over CLinear (0.360 ms) and Dist (2.936 ms). Aggregated across all tested configurations, Fig. 5d summarizes the average inference time, parameter count, and throughput as 0.074 ms/56,592/13.5 for CCL, 0.158 ms/588,004/6.3 for CLinear, and 0.941 ms/587,265/1.1 for Dist; the corresponding parameter bars in Fig. 5c show that CCL uses only about 9.6% of the parameters of CLinear and Dist.

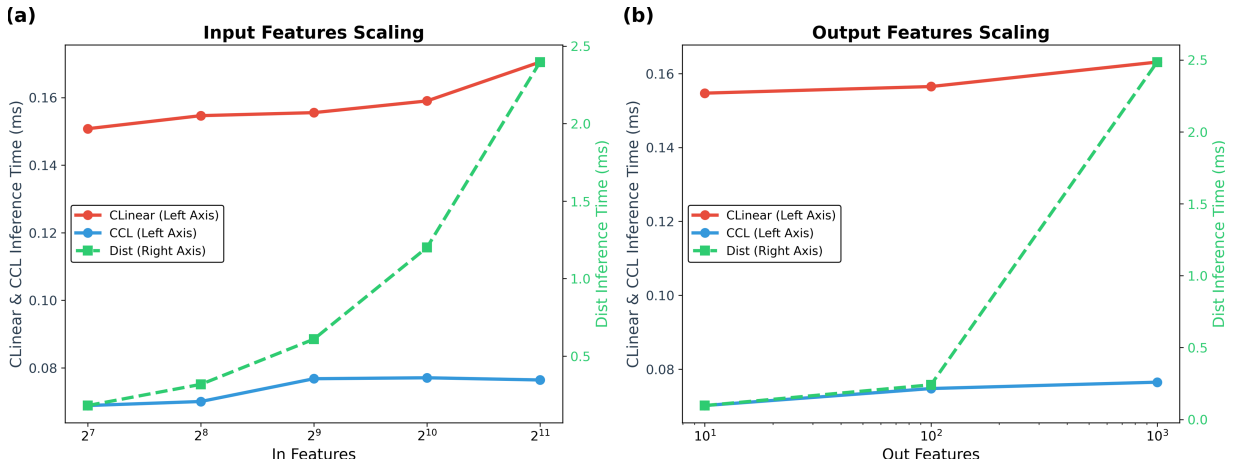
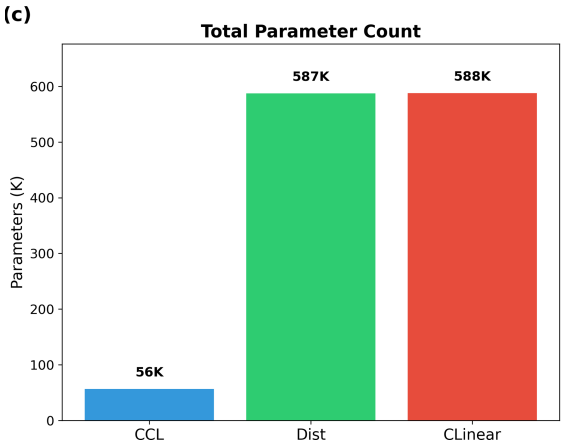


Figure 5: (Continued)



(d)

Performance Summary

Model	Avg Time(ms)	Parameters	Speed (1/ms)
CCL	0.074	56,592	13.5
CLinear	0.158	588,004	6.3
Dist	0.941	587,265	1.1

Figure 5: Computational efficiency comparison of different complex classifier heads. (**a**) Inference time vs. input feature dimensions for CCL, CLinear, and Dist (Dist plotted on secondary right axis for clarity). (**b**) Inference time vs. output feature dimensions (Dist on secondary axis). (**c**) Total parameter count comparison, highlighting CCL's substantial reduction relative to baselines. (**d**) Quantitative performance summary detailing average inference time (Avg Time), parameter count, and throughput speed (1/Avg Time). Avg Time denotes the arithmetic mean of repeated measurements under identical conditions, while Speed serves as a simplified throughput metric.

Scalability analysis further reveals CCL's superiority. In the input feature dimension scaling test (Fig. 5a), as dimensions increase from 128 to 2048, CCL's inference time maintains linear growth with the slowest rate, whereas Dist shows a rapidly deteriorating trend, approaching 2.4 ms at the highest input dimension on its secondary right axis. In the output dimension scaling test (Fig. 5b), CCL similarly demonstrates the best scalability; even as output classes increase from 10 to 1000, its inference time growth remains flat, almost unaffected by output dimension, while CLinear and Dist show significant performance degradation.

This inference speed test fully confirms the effectiveness of the low-rank decomposition design: by decomposing complex fully connected matrix operations into two consecutive low-rank matrix multiplications ($x \otimes U \otimes V$, where $U \in \mathbb{C}^{n\times r}$, $V \in \mathbb{C}^{r\times m}$, rank $r \ll \min(n, m)$), CCL not only drastically reduces parameter storage requirements (to 43.2% of the original) but, more importantly, significantly lowers actual computational complexity, optimizing time complexity from $O(n \times m)$ to $O(n \times r + r \times m)$. In practical applications, this design allows CCL to achieve substantial inference speed improvements while maintaining high accuracy (less than 1% mean accuracy gap to CLinear), providing strong performance guarantees for edge devices and real-time application scenarios.

Controllable Complexity and Rank Analysis: The rank parameter r serves as a key hyperparameter in the WCCN architecture for regulating model capacity and parameter efficiency. To determine the optimal configuration, we conducted a wide-range sensitivity scan for r from 4 to 128 (as shown in Table 10). The results exhibit clear two-stage characteristics:

(1) **Rapid Gain Phase ($r \in [4, 24]$):** As r increases, model accuracy shows a clear monotonic rise from 26.80% ± 0.49% (r = 4) to 41.63% ± 0.63% (r = 24), demonstrating that increasing rank significantly improves representational power.

(2) **Diminishing Returns and Saturation Phase ($r > 24$):** When r exceeds 24, further gains are small. For example, although r = 64 increases parameter count by approximately 2.6 times (from 11,144 to 29,384), accuracy improves only by about 0.50 percentage points compared with r = 24.

This behavior is consistent with the optimization dynamics of low-rank bilinear heads ($W \approx UV$). For small r, increasing rank mainly reduces approximation bias, so performance improves rapidly.

After a moderate rank, however, the effective class-discriminative subspace is already largely covered; additional rank mostly introduces redundant degrees of freedom and weakens implicit regularization. Meanwhile, the factorized parameterization has scale non-identifiability $(U\alpha, \alpha^{-1}V)$, which can make optimization increasingly ill-conditioned at large r under a fixed training budget. Therefore, larger rank does not guarantee proportionally better generalization and leads to clear saturation.

Table 10: Impact of rank parameter r in CCL on parameter-performance tradeoff.

Rank	#Params ↓	CIFAR-100 Accuracy (%) ↑	Marginal Eff. (pp/kP) ↓
4	2024	26.80 ± 0.49	—
8	3848	35.61 ± 0.54	4.83
16	7496	40.51 ± 0.42	1.34
24	11,144	41.63 ± 0.63	0.31
32	14,792	41.70 ± 0.55	0.02
64	29,384	42.13 ± 0.52	0.03
128	58,568	42.20 ± 0.39	0.002

The rightmost column of Table 10 quantifies this trend: the marginal efficiency drops by two orders of magnitude from the rapid-gain phase (4.83 pp/kP at r: $4 \rightarrow 8$) to the saturation phase (0.02 pp/kP at r: $24 \rightarrow 32$), pinpointing $r = 24$ as the inflection point of the cost–benefit curve.

This phenomenon indicates that $r = 24$ constitutes the optimal "sweet spot" for the trade-off between parameter count and performance in this task. Therefore, to maximize computational efficiency while maintaining high performance, all shallow network experiments in this study adopt $r = 24$ as the standard configuration.

To verify CCL's performance in deep complex networks, we extended the experiment to the larger capacity WCCN-M architecture and compared different rank configurations ($r \in \{24, 32, 64, 150, 155\}$) against the standard wcLinear classifier head (see Fig. 6). Results indicate that as r increases, CCL's classification accuracy gradually improves. Although CCL underperforms wcLinear when r is small (e.g., 24 or 32), accuracy surpasses the standard linear head when $r = 150$. However, as r continues to increase, model performance shows a noticeable decline. This suggests that simply increasing r is insufficient; an excessively large r can negatively impact overall performance. This result aligns with our method design: in deep networks, moderately increasing r effectively leverages the backbone's efficiency, reinvesting the saved computational budget into the classifier head to improve performance on complex tasks without significantly increasing total parameters. Even at $r = 150$, where the CCL head uses more parameters than the wcLinear head (106.8K vs. 51.4K), the total parameter count of WCCN-M (1.37M) remains significantly lower than the baseline CDS-Large (1.79M), verifying the effectiveness of the parameter reallocation strategy.

The degradation beyond $r = 150$ is also consistent with this view: once capacity is no longer the bottleneck, further rank expansion increases optimization variance and head-level overfitting risk more than useful signal capture, especially when training epochs and regularization settings are unchanged.

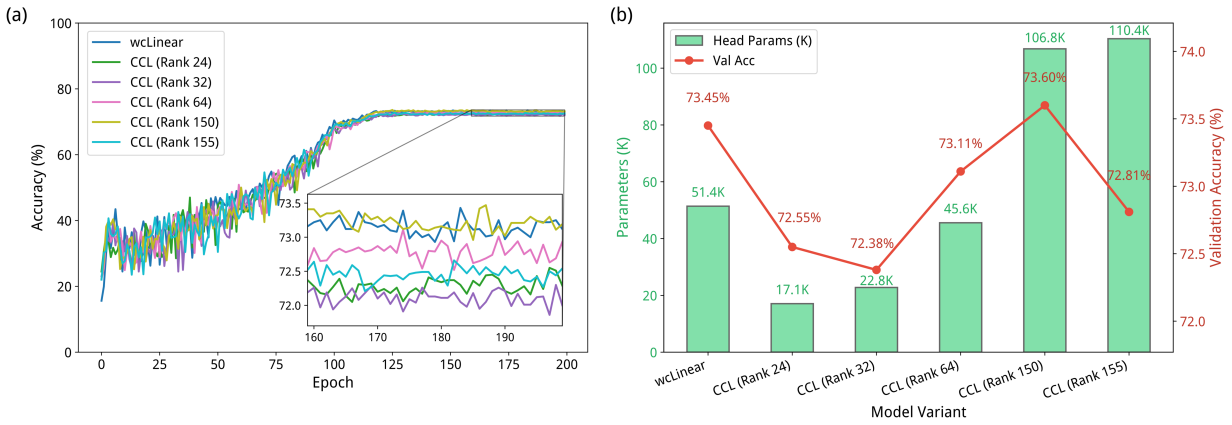


Figure 6: Comparison of CCL with different ranks and wcLinear in deep networks (CIFAR-100). (**a**) Test accuracy trajectories of wcLinear and CCL under varying rank settings, with an inset highlighting late-training fluctuations. (**b**) Parameter-accuracy trade-off analysis, plotting classifier parameter counts against validation accuracy. Results demonstrate that lower-rank CCL variants achieve substantial parameter reduction, while an optimized rank (e.g., $r = 150$) can match or surpass wcLinear accuracy.

Maintaining the WCCN-M network and training settings, we downsampled the training set to 20% and 10% and trained by replacing only the classifier head. As shown in Table 11, under the current 5-seed evaluation, wcLinear achieves higher validation accuracy than CCL at both sampling ratios: 58.87% ± 0.65% vs. 57.68% ± 0.49% (20%) and 48.66% ± 0.40% vs. 47.05% ± 0.44% (10%). This indicates that under low-data settings, CCL's main advantage remains parameter efficiency rather than peak accuracy.

Table 11: Performance comparison of CCL and CLinear under low data regimes.

Data Rate	Method	CIFAR-100 Accuracy (%) ↑
20%	CLinear	58.87 ± 0.65
20%	CCL (r = 150)	57.68 ± 0.49
10%	CLinear	48.66 ± 0.40
10%	CCL (r = 150)	47.05 ± 0.44

4.4.2 Efficiency Analysis of the Core Operator wcConv

To evaluate the computational efficiency of convolutional layers in different complex-valued models, we conducted a comprehensive benchmark on four Cartesian complex convolution implementations: (1) PyTorch native complex convolution (`F.conv2d` with `torch.cfloat`), used as the baseline; (2) *ComplexConvFast* from CDS-I, which applies the Gauss trick with a channel-stacking data format; (3) *ComplexConv2d* from DCN, which uses the standard 4-real-convolution decomposition; and (4) our proposed *wcConv2d*, which combines the Gauss trick with a fused tuple-based data flow. We note that *ComplexConv2Deffangle* from SurReal is excluded because it does not implement standard Cartesian complex convolution—it operates in polar coordinates using `nn.Unfold` and hand-written weighted sums rather than `nn.Conv2d`, making it architecturally incomparable. To ensure fairness, each implementation receives input in its native tensor format (CDS uses its stacked-real format; the others use `torch.cfloat`), and all models use `bias=False`. The benchmark measured the total execution time, including forward, backward, and an SGD update step, to reflect realistic training scenarios. We employed CUDA Events for precise GPU

timing and ensured proper synchronization. Each configuration underwent 50 warmup iterations followed by 20 benchmark iterations with 5 repetitions.

As illustrated in Fig. 7, we systematically varied six key parameters: (a) input spatial size (32–112), (b) input channel count (16–256), (c) kernel size (1/3/5/7), (d) output spatial size (by fixing stride = 2 and sweeping input size 32–192), (e) output channel count (32–512), and (f) batch size (1–32). Using native PyTorch complex convolution as the baseline, we report the speedup ratios of the other three implementations in each subplot.

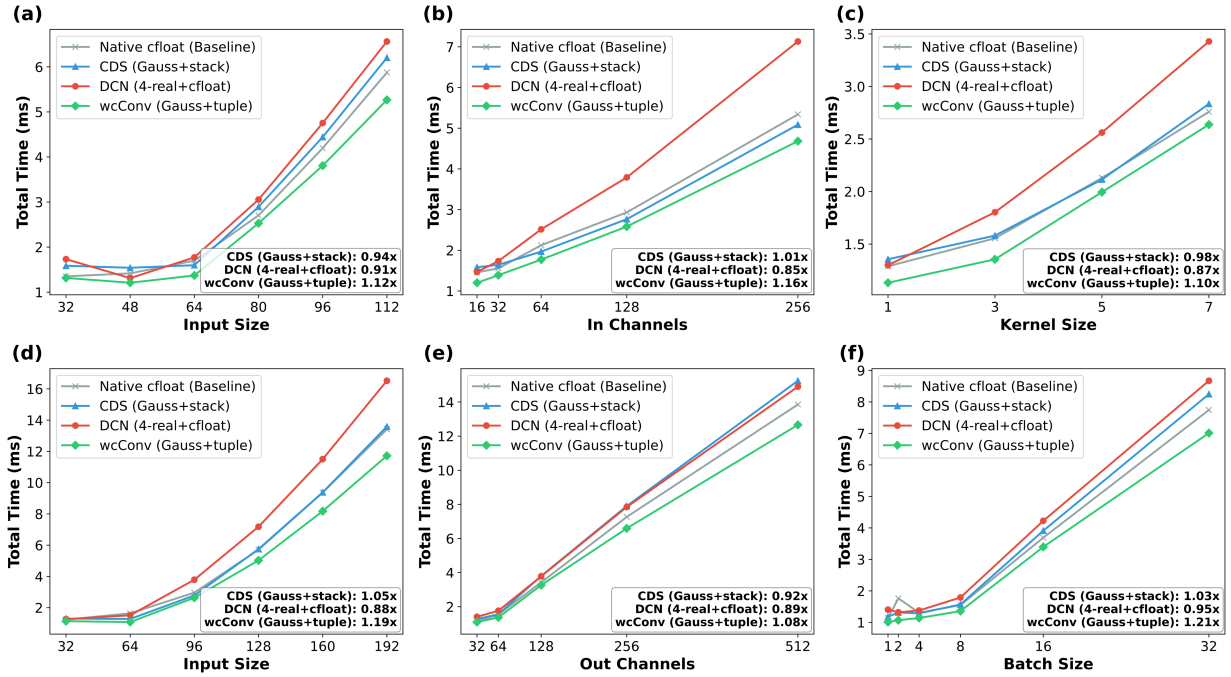


Figure 7: Computational efficiency comparison of Cartesian complex convolution implementations. Using native PyTorch complex convolution as the baseline, subplots (**a**)–(**f**) evaluate training-step latency under varying configurations, including input spatial size, channel dimensions, kernel size, output resolution (via stride), and batch size. Results consistently show that wcConv2d (Gauss + tuple-flow) achieves the lowest latency across all tested conditions.

Under all experimental conditions, our *wcConv2d* (Gauss + tuple-flow) consistently achieved the lowest latency across all tested configurations, with a mean speedup of 1.15× over the native PyTorch complex convolution baseline (range: 1.03×–1.65×). The efficiency gains come from two complementary optimizations: the Gauss trick reduces 4 real-valued convolutions to 3, and the tuple-flow data format eliminates the Python-side overhead associated with repeated tensor splitting, stacking, and complex object creation. By maintaining a clean tuple data flow (X_r, X_i) and inlining the Gauss trick, we maximize the utilization of the PyTorch C++ backend while minimizing interpreter intervention. CDS-I, which also employs the Gauss trick but uses a channel-stacking format, achieves performance close to the native baseline (mean 0.99×), indicating that its stack-based format conversion partially offsets the algorithmic savings of the Gauss trick. DCN's standard 4-real decomposition with `cfloat` is generally ~10% slower than the native baseline (mean 0.90×), reflecting the overhead of its explicit `.real/.imag` extraction and `+1j*` recombination pattern.

Unified Latency Benchmark and Attribution Analysis. To precisely attribute the sources of speedup, we conducted end-to-end latency benchmarks under a unified setting (RTX 4070, PyTorch 2.1.2,

batch = 8, in= 32, hidden = 64, 4 layers, input size = 64 × 64, k = 3, warmup = 24, bench = 50, repeat = 6). Each implementation uses its native I/O format; timing deliberately omits cross-format adaptations. Results are reported in Table 12 (mean ± 95% confidence interval (CI)).

Table 12: Breakdown of training step latencies across complex convolution implementations (ms/step).

Method	Forward Latency ↓	Backward Latency ↓	Total Step Latency ↓
Native cfloat (PyTorch complex conv)	3.176 ± 0.062	6.861 ± 0.047	10.036 ± 0.093
Explicit 4-real-conv + cfloat	3.247 ± 0.116	7.360 ± 0.062	10.607 ± 0.081
Explicit Gauss (3-real-conv) + cfloat	2.916 ± 0.142	6.249 ± 0.102	9.165 ± 0.087
Explicit Gauss (3-real-conv) + tuple-flow	2.296 ± 0.030	4.560 ± 0.024	6.856 ± 0.045
CDS Gauss (3-real-conv) + stack	2.837 ± 0.061	5.835 ± 0.037	8.672 ± 0.054

The profiler attribution analysis (Table 13) reveals the root cause: tuple-flow reduces format-conversion overhead to 0.8% of total self CUDA time, compared with 10.8%–17.2% for other implementations. This confirms that the efficiency gains come from implementation-level data-flow optimization, not from the convolution algorithm itself.

Table 13: Profiler attribution summary (Self CUDA, 12-step window).

Method	Total Self CUDA (ms)	Conv Share	Format Conv Share	Others
Native cfloat	203.07	32.0%	16.2%	51.8%
Explicit 4-real + cfloat	197.36	44.1%	17.2%	38.7%
Explicit Gauss + cfloat	163.88	38.4%	16.6%	45.0%
Explicit Gauss + tuple-flow	103.30	64.3%	0.8%	34.9%
CDS Gauss + stack	158.71	38.0%	10.8%	51.2%

We further verified forward and backward numerical equivalence using shared inputs and weights (Table 14). Gauss-based implementations are bitwise identical to native complex convolution; the small discrepancy of the 4-real decomposition is due to floating-point reordering and does not affect the optimization objective.

Table 14: Maximum absolute error of each implementation relative to native.

Impl.	Fwd Δy↓	Δw_r↓	Δw_i↓	Δx_r↓	Δx_i↓
Gauss-cfloat	0.000e+00	0.000e+00	0.000e+00	0.000e+00	0.000e+00
Gauss-tuple	0.000e+00	0.000e+00	0.000e+00	0.000e+00	0.000e+00
CDS-stack	0.000e+00	0.000e+00	0.000e+00	0.000e+00	0.000e+00
Standard-4-Real	2.956e−05	3.755e−05	4.330e−05	1.397e−08	1.490e−08

Finally, a multi-layer depth sweep (2–12 layers, Table 15) confirms that the tuple-flow advantage is consistent across depth, maintaining approximately 1.5× speedup from 2 layers to 12 layers relative to the native implementation. Each configuration is repeated 7 times (warmup = 16, bench = 36), and results are reported as mean ± standard deviation.

Table 15: Total training step latency across varied model depths (ms, mean ± std).

Layers	Native	4-real	Gauss+cfloat	CDS	Gauss+tuple
2	4.690 ± 0.066	4.778 ± 0.107	4.163 ± 0.056	4.108 ± 0.044	3.154 ± 0.076
4	10.266 ± 0.072	10.760 ± 0.103	9.193 ± 0.026	8.715 ± 0.064	6.875 ± 0.074
6	15.596 ± 0.082	16.155 ± 0.091	13.914 ± 0.065	13.556 ± 0.060	10.706 ± 0.056
8	21.375 ± 0.037	22.313 ± 0.082	18.943 ± 0.105	17.784 ± 0.061	14.149 ± 0.057
10	26.919 ± 0.227	28.295 ± 0.085	23.987 ± 0.104	22.313 ± 0.061	17.869 ± 0.075
12	32.416 ± 0.285	34.122 ± 0.084	28.935 ± 0.118	27.280 ± 0.252	21.445 ± 0.055

System-Level Engineering Metrics. Beyond per-operator benchmarks, we report comprehensive system-level engineering metrics for the full model pipeline in Table 16. Settings: CIFAR-100, batch = 64; training timing uses warmup = 20, bench = 40, repeat = 5; inference timing uses warmup = 40, bench = 120, repeat = 5; floating-point operations (FLOPs) are counted for the forward pass only. Memory is reported in megabytes (MB) as the peak allocated memory during inference. It is important to note that the "efficient" claim in WCCN refers to the implementation path (wcConv + tuple-flow + CCL), not to RTC6, which serves as a high-redundancy benchmark mapping for analyzing the capacity-compensation mechanism.

Table 16: Engineering metrics under unified settings (CIFAR-100, batch = 64; Params: actual 3/6ch).

Model	Params	FLOPs/Batch (GFLOPs)	Training Step (ms) ↓	Inference (ms/batch) ↓	Throughput (img/s) ↑	Peak Infer Mem (MB) ↓
WCCN-M(tuple), RGB(3ch)	1,371,312	62.314	56.205	10.958	5840.35	250.30
WCCN-M(tuple), RTC6(6ch)	1,374,768	62.994	55.386	11.151	5739.36	250.31
DCN-M, RTC6(6ch)	1,798,400	72.294	154.429	43.339	1476.74	379.65
CDS-Large, RTC6(6ch)	1,798,208	54.079	81.896	22.330	2866.10	318.65

Under the same RTC6(6ch) input, WCCN-M achieves 3.89× faster inference than DCN-M and 2.00× faster than CDS-Large, while the total training-step latency is 2.79× and 1.48× faster, respectively. Peak inference memory is also lower (250.31 vs. 379.65/318.65 MB). The incremental cost of expanding 3 to 6 ch within WCCN-M is minimal: FLOPs +1.09%, inference latency +1.76%, and peak inference memory < 0.01%, confirming that the RTC6 overhead is small and controllable at the system level. Note that in WCCN-M, expanding RGB(3ch) to RTC6(6ch) only changes the input dimension of the first convolutional layer; the net parameter increase is 3456 (+0.25%).

Notably, WCCN-M has slightly higher forward FLOPs (62.994 GFLOPs) than CDS-Large (54.079 GFLOPs), yet achieves a lower total training-step latency (55.386 vs. 81.896 ms). This is because FLOPs count forward arithmetic operations only and do not capture backward propagation, optimizer updates, memory access, or format conversion overhead. A training-time profiler attribution (Table 17) reveals the root cause: CDS-Large incurs 297.696 ms in format-conversion operations (copy/cat) compared to only 22.068 ms for WCCN-M, which dominates the training latency difference.

Table 17: Training-time profiler attribution (20 Steps, Self CUDA Time).

Model	Total Self CUDA (ms)	Conv (ms)	BN (ms)	Pool (ms)	Format Conv (copy/cat) (ms)	Elem/Others (ms)
WCCN-M	1546.486	398.354	105.509	27.240	22.068	651.909
CDS-Large	2929.678	520.813	0.000	3.887	297.696	1104.274

We further investigate why inference speedups are larger than training speedups through a jitter ablation study (Table 18). Disabling the wcPReLUJitter perturbation branch accelerates training by approximately 1.45× (56.390 → 38.947 ms), while inference latency is nearly identical (10.91 vs. 11.01 ms) because the training-perturbation branch is not activated during inference. This confirms that the main reason training and inference speedups differ is not the convolution backbone itself, but the additional training-only operators (especially the jitter branch) and their backward overhead.

Table 18: Jitter ablation: impact of wcPReLUJitter on training and inference latency.

Config	Phase	Forward (ms)	Backward (ms)	Total (ms)
Default	Train	16.675	39.715	56.390
No-jitter	Train	12.059	26.888	38.947
Default	Infer	10.91	–	10.91
No-jitter	Infer	11.01	–	11.01

4.4.3 Ablation Study and Stability Analysis of Activation Functions

To validate the effectiveness and generalization capability of the proposed wcPReLU and its variant (wcPReLUJitter), this section designs a set of ablation experiments comparing them against the baseline complex-valued activation function CReLU [2] and the advanced gated activation function GTReLU [1]. To comprehensively evaluate performance across different network depths, we conducted comparative tests on both shallow (WCCN) and deep (WCCN-M) architectures. All experiments were conducted on the CIFAR-100 dataset, strictly maintaining training settings consistent with previous experiments (including optimizer, learning rate strategy, and preprocessing pipelines) to ensure fairness. Each experimental group was repeated independently 20 times. Comprehensive performance in terms of accuracy, stability, and computational efficiency was evaluated by calculating the mean, standard deviation, and coefficient of variation (CV) (results are shown in Table 19).

Table 19: Performance comparison of complex-valued activation functions across network depths (CIFAR-100; 20 runs, seeds 40–59).

Network Architecture	Activation Function	Mean Accuracy (%) ↑	Accuracy Range (%) ↑	CV ↓	Forward Time (ms) ↓
Shallow Net	CReLU	38.92 ± 0.56	37.73–39.61	0.0145	1.432
	GTReLU	41.18 ± 0.50	40.21–42.00	0.0122	2.427
	wcPReLU	40.26 ± 0.58	39.20 – 41.49	0.0145	1.621
	wcPReLUJitter	41.59 ± 0.48	40.82–42.39	0.0115	1.619
Deep Net	CReLU	71.04 ± 0.31	70.62–71.84	0.0043	4.292
	GTReLU	Training Collapse	–	–	–
	wcPReLU	72.59 ± 0.36	71.94–73.32	0.0050	4.775
	wcPReLUJitter	72.79 ± 0.35	72.26–73.60	0.0048	4.988

Note that WCCN-M uses wcPReLUJitter by default; thus, the WCCN-M CIFAR-100 result in Table 5 corresponds to the same setting as the "Deep Net + wcPReLUJitter" row in Table 19. The two tables use different numbers of random seeds (Table 5: 5 runs, seeds 40–44; Table 19: 20 runs, seeds 40–59). The overlap subset (40–44) matches exactly; the mean discrepancy (73.17 vs. 72.79) is due to the additional seeds (45–59) included in Table 19.

Experimental results indicate that in the shallow WCCN architecture, although GTReLU achieves better accuracy than the baseline CReLU (41.18% vs. 38.92%) through its complex gating mechanism, this comes at a significant computational cost—its forward propagation time reaches 2.427 ms, which is 1.7 times that of CReLU. In contrast, the proposed wcPReLUJitter not only achieves the highest average accuracy (41.59%), significantly outperforming CReLU and GTReLU, but also maintains extremely high inference efficiency (1.619 ms, only about 13% slower than CReLU). Furthermore, wcPReLUJitter possesses the lowest coefficient of variation ($CV = 0.0115$), demonstrating excellent training stability in shallow networks.

When network depth extends to WCCN-M, the numerical stability of activation functions becomes a critical bottleneck constraining performance. As shown in Fig. 8, wcPReLU maintains stable convergence and gradient flow throughout training. In contrast, Figs. 9 and 10 show that GTReLU, with its introduced complex gating mechanism, suffers from severe "Training Collapse" in the deep architecture (WCCN-M), causing the model to fail to converge. This indicates that the complex gating mechanism of GTReLU introduces numerical instability during backpropagation in deep networks, thereby affecting training stability. Conversely, the proposed wcPReLU series resolves this issue via a simpler activation design that preserves gradient flow; gradients for these non-holomorphic activations are computed under the standard Wirtinger/CR-calculus backpropagation framework. Specifically, although wcPReLUJitter has a slightly increased single inference time (4.988 ms) compared to CReLU (4.292 ms), it boosts accuracy by 1.75% (from 71.04% to 72.79%) with minimal accuracy fluctuation ($CV = 0.0048$), showing significant advantages in high-performance scenarios. For use cases or devices with stringent latency requirements, wcPReLU (excluding the jitter mechanism) presents a streamlined alternative, delivering an accuracy of 72.59%, which surpasses that of CReLU (71.04%), while incurring only an 11% increase in computational overhead. This makes wcPReLU well-suited for applications where low latency is critical. Therefore, the wcPReLU series demonstrates superior stability and adaptability in deep networks.

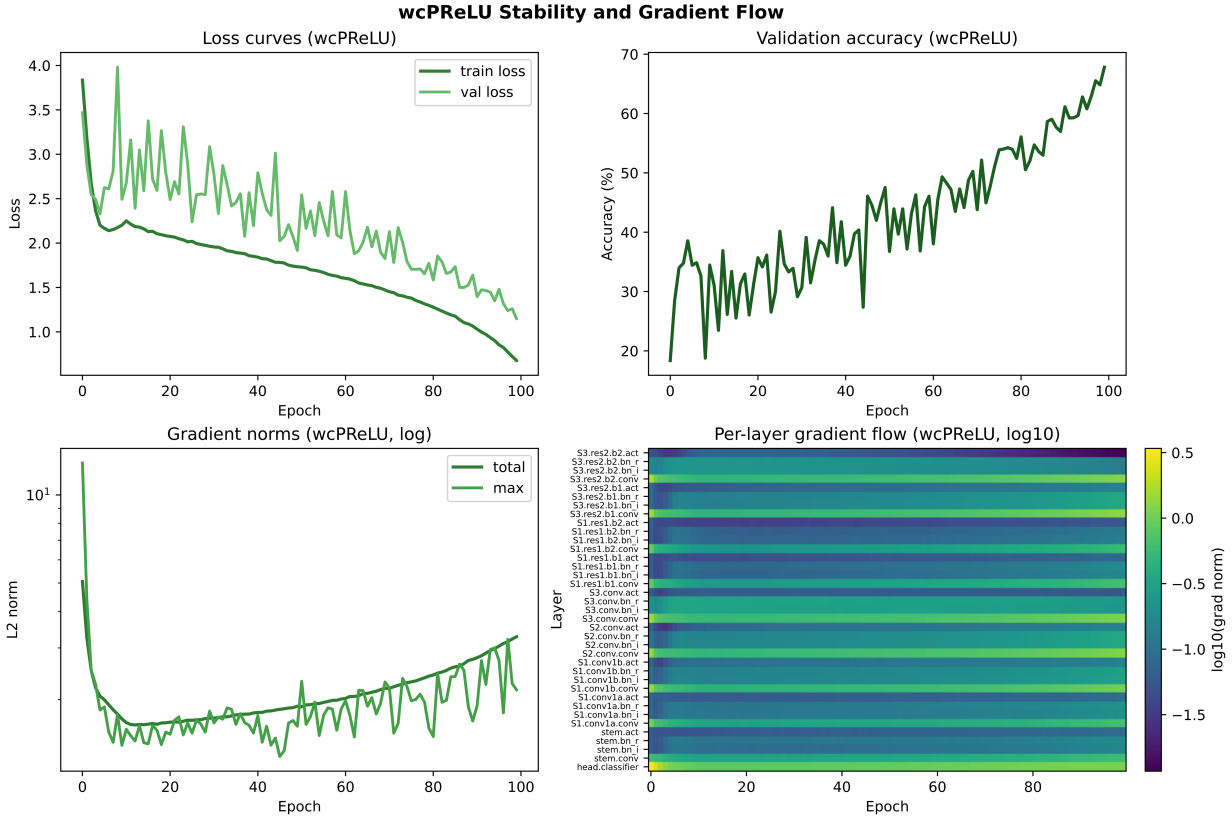


Figure 8: Training stability and gradient flow evidence of wcPReLU on WCCN-M (CIFAR-100, 100-epoch diagnostic setting). The panels show epoch-wise training loss, validation accuracy, gradient norms, and layer-wise gradient flow, confirming stable convergence with no gradient hollowing.

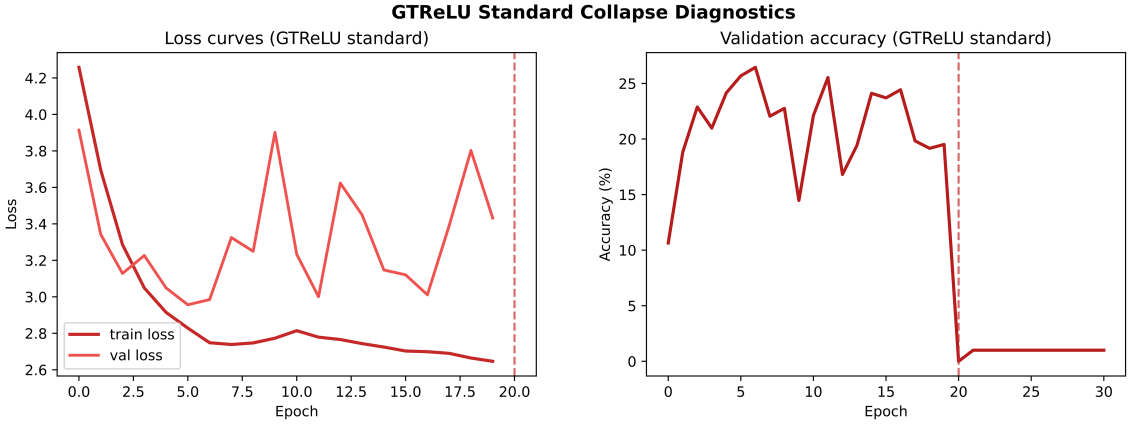


Figure 9: (Continued)

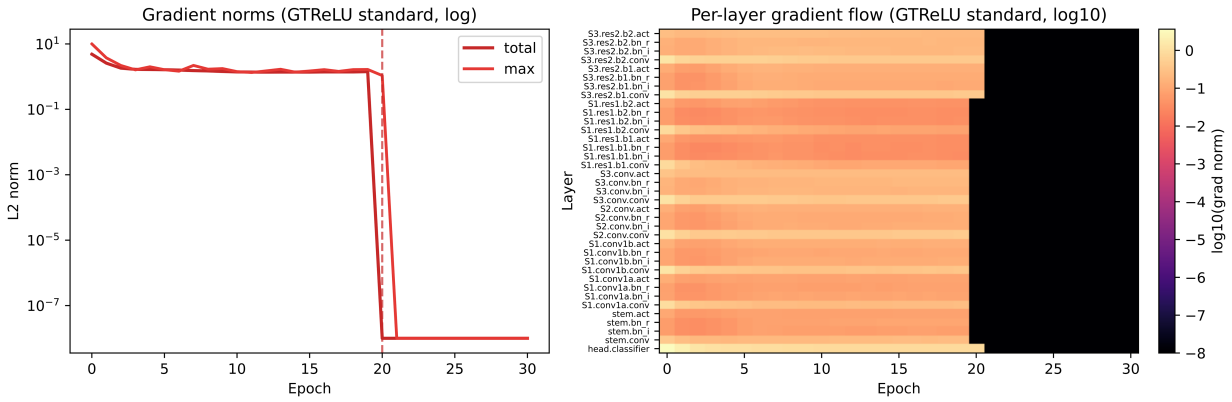


Figure 9: Training collapse trajectory of GTReLU under standard settings on WCCN-M (CIFAR-100). Loss diverges to Inf around epoch 20, and validation accuracy stagnates at approximately 26%.

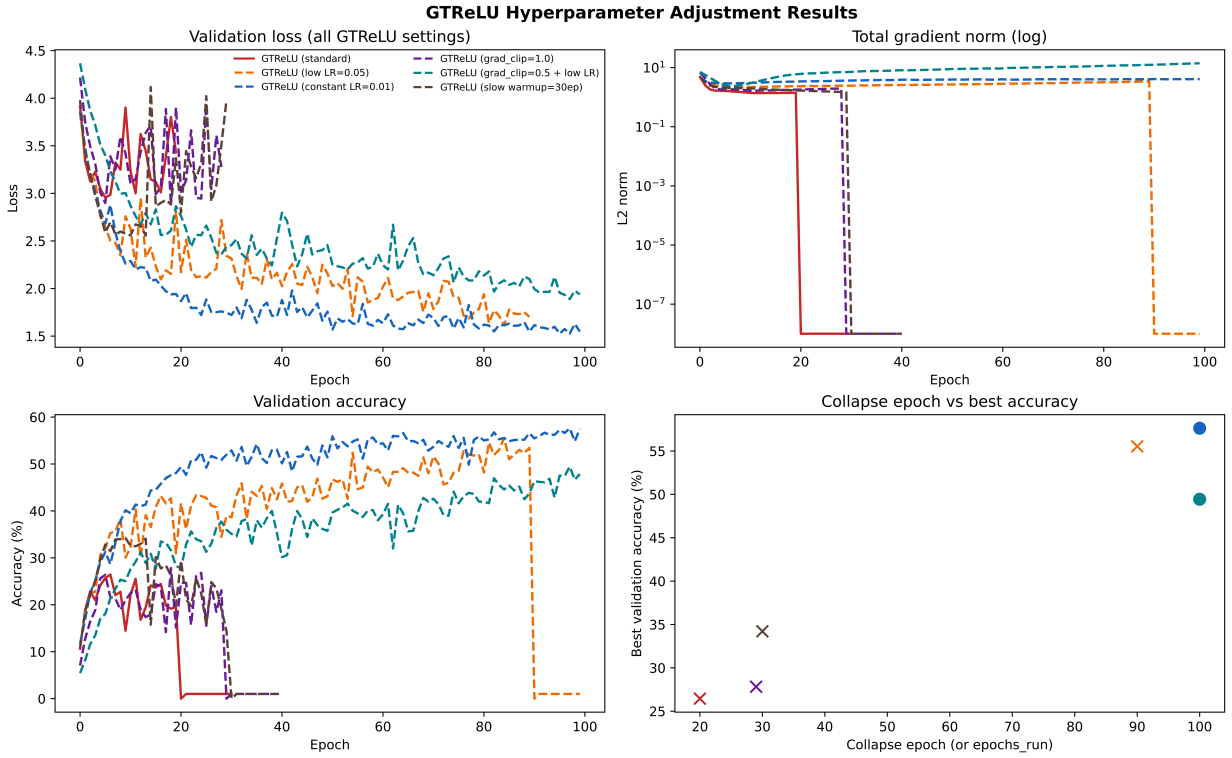


Figure 10: GTReLU hyperparameter sensitivity analysis: common tuning strategies (gradient clipping, warmup, reduced learning rate) only delay the collapse onset but cannot reliably prevent it. Even conservative non-collapsing settings yield significantly lower accuracy than wcPReLU.

Mechanistic Analysis of wcPReLU Stability. To understand why wcPReLU avoids the collapse observed with GTReLU, we provide comprehensive training diagnostics in Fig. 8. wcPReLU employs piecewise linear mappings for the real and imaginary parts separately: the slope on the positive semi-axis is fixed at 1, while the slope on the negative semi-axis is a learnable parameter $\alpha_c > 0$. Critically, the local derivative always remains strictly non-zero (either 1 or α_c), which continuously maintains gradient flow and significantly mitigates the risks of gradient blockage and cascading feature decay during training in deep

networks. As shown in the diagnostic panels, wcPReLU maintains stable convergence over 100 epochs on WCCN-M (CIFAR-100), achieving a best validation accuracy of 67.81% under the unified diagnostic setting, with layer-wise gradients consistently showing no signs of layer-wide gradient hollowing throughout the training process.

Detailed GTReLU Collapse Diagnostics and Hyperparameter Sensitivity. To provide concrete evidence for the GTReLU collapse claim, we conducted comprehensive diagnostics under multiple hyperparameter configurations (Figs. 9 and 10). Under standard settings, GTReLU enters a collapse regime around epoch 20, with training and validation loss reaching Inf starting from epoch 21 and a best validation accuracy of merely 26.45%. Common hyperparameter tuning strategies only delay, but cannot prevent, the collapse: applying gradient clipping (`grad_clip` = 1.0) delays collapse to epoch 29–30 (best accuracy 27.82%); adding warmup (`warmup` = 30) delays to epoch 30–31 (best accuracy 34.21%); and reducing the peak learning rate to 0.05 pushes collapse back to around epoch 90–91, but it still eventually occurs (best accuracy 55.54%).

Even under conservative non-collapsing settings, GTReLU performance significantly lags behind wcPReLU: a constant learning rate of 0.01 yields no Inf/NaN but achieves only 57.63% accuracy (vs. wcPReLU's 67.81% under the same 100-epoch diagnostic setting), while aggressive clipping (`clip` = 0.5 + low learning rate) produces only 49.44% accuracy with the global mean $|a|$ (where a denotes the activation output of each network layer; the global mean $|a|$ averages $|a|$ across all layers) dropping from ~0.88 to 0.014, indicating severe representational degradation. Therefore, under the same training budgets and conventionally tunable hyperparameter ranges, the stability and performance robustness of GTReLU are notably weaker than those of wcPReLU.

4.5 Analysis of Complex-Valued Modeling Advantages

In the preceding sections, we have systematically demonstrated the significant advantages of the WCCN framework in terms of parameter efficiency and structural innovation. To further illustrate the unique modeling capabilities of complex-valued neural networks in native complex-valued tasks, this section focuses on analyzing the contribution of magnitude and phase information fusion. This aims to reveal WCCN's ability to utilize and fuse key information in complex scenarios.

This experiment utilizes three typical SAR datasets and evaluates three distinct input modes to isolate the contribution of different information components: (1) **Magnitude+Phase (Full Complex)**, where the input consists of unmodified complex-valued data containing both magnitude and phase information; (2) **Magnitude-Only**, where the imaginary part of the input is set to zero, retaining only magnitude information; and (3) **Phase-Only**, where the magnitude of the input is normalized to unity, retaining only phase information.

By comparing the classification performance under these input modes, we systematically evaluated the discriminative contribution of magnitude and phase information across different scenes. The experimental results are presented in Table 20.

A detailed analysis of Table 20 yields three critical insights regarding the mechanism of complex-valued learning. Firstly, complex-valued information fusion brings consistent improvements in most settings: the Magnitude+Phase mode achieves the best performance on five out of six metrics and remains close to the best on the remaining metric (Oberpfaffenhofen OA post). Secondly, both magnitude and phase serve as important discriminative characteristics, with their respective influences varying according to the specific scene. For instance, in the Flevoland-1991 dataset, the Phase-Only model achieves high OA values, indicating that phase features are highly informative in this region, whereas in Flevoland-1989, magnitude-phase fusion

provides a clearer advantage. Finally, the results demonstrate that WCCN possesses robust information fusion capabilities. The model can adaptively extract and utilize complementary features from both magnitude and phase channels to maximize classification accuracy. This adaptive fusion capability represents a core advantage distinguishing complex-valued networks from traditional real-valued counterparts and serves as an important reason for WCCN's strong performance in complex scenarios.

Table 20: Contribution analysis of magnitude and phase information on different SAR datasets.

Input Mode	Flevoland-1989		Flevoland-1991		Oberpfaffenhofen	
	OA raw (%) ↑	OA post (%) ↑	OA raw (%) ↑	OA post (%) ↑	OA raw (%) ↑	OA post (%) ↑
Magnitude+Phase	97.09 ± 0.36	97.78 ± 0.34	99.45 ± 0.10	99.12 ± 0.10	92.29 ± 0.44	93.18 ± 0.45
Magnitude-Only	92.41 ± 0.90	93.42 ± 0.93	94.18 ± 1.35	94.23 ± 1.33	90.82 ± 1.45	91.92 ± 1.43
Phase-Only	89.84 ± 0.56	91.16 ± 0.61	97.42 ± 0.24	97.35 ± 0.20	92.19 ± 0.36	93.71 ± 0.30

4.6 Discussion and Limitation Analysis

Although the experiments confirm the significant advantages of the WCCN framework in terms of parameter efficiency, inference speed, and multi-scenario classification performance, a thorough reflection on the experimental results suggests that the current framework still has limitations in several aspects that warrant focused attention in future research.

Firstly, the absence of support for mixed-precision training restricts large-scale training efficiency. Although Section 4.4.2 verified a consistent inference-phase acceleration of the wcConv operator (mean 1.15× over the native PyTorch complex convolution baseline, range 1.03×–1.65×), the current implementation still relies on 32-bit floating-point (FP32) complex arithmetic during the training phase. Due to the imperfect support for automatic mixed precision (AMP) for complex types in existing deep learning frameworks (such as PyTorch), WCCN struggles to utilize the Tensor Cores of modern GPUs for half-precision acceleration. This results in potential bottlenecks regarding video memory occupation and training throughput when processing ultra-large-scale datasets (e.g., ImageNet).

Secondly, the low-rank decomposition strategy of CCL lacks dynamic adaptability. In the parameter sensitivity analysis in Section 4.4.1 (Table 10 and Fig. 6), we discovered a distinct "Sweet Spot" for the selection of the rank parameter r, and networks of different scales (WCCN vs. WCCN-M) require manual adjustment of the optimal rank ($r = 24$ vs. $r = 150$). While this fixed-rank strategy effectively balances model compression and feature expression, it cannot dynamically adjust the subspace dimension based on the complexity of input samples. Particularly when facing long-tail distribution data with huge inter-class differences, the fixed low-rank constraint may limit the model's effective capture of features for minority classes (tail samples), thereby affecting the balance of overall performance.

Finally, the interaction mechanism between magnitude and phase lacks explicit modeling. Although Section 4.5 proved that WCCN can effectively utilize phase information to enhance performance, and Section 4.3 showed that high-redundancy input (RTC6) can bring compensation effects, the current framework's fusion of magnitude-phase information mainly relies on the natural coupling of convolution operations and the non-linear transformation of wcPReLU. Because there is no clear Attention Mechanism to assess the importance of magnitude and phase actively, the model's maximum performance could be affected in situations with high phase noise or very limited phase information.

In addition, it is important to note that the experiments in this work are primarily designed for controlled comparisons and mechanism validation under unified experimental settings, rather than exhaustive leaderboard-style benchmarks. Due to limited public implementations and differences in training protocols

across methods, the current results mainly support the mechanism validity and engineering feasibility of the proposed approach. Therefore, we limit the scope of our conclusions to "competitive under the experimental and data settings in this paper" and do not extrapolate to claim overall state-of-the-art superiority. Adding comparisons with more recent methods under unified settings remains a key direction for future work.

5 Conclusion and Future Work

This paper proposes the WCCN framework, which systematically addresses the core challenges of complex-valued deep learning in visual tasks. Through a comprehensive evaluation, we deeply analyzed the impact of input mapping in different deep networks and revealed the key compensation mechanism of high-redundancy input for the performance of compact architectures. WCCN constructs an efficient operator system including wcConv, wcPReLU, and CCL. Here, RTC6 is treated as a high-redundancy benchmark mapping rather than an efficiency component, while the efficiency advantage is primarily delivered by the operator/classifier implementation path. While improving inference efficiency, WCCN uses wcPReLU to reduce the training instability associated with complex gating mechanisms in deep networks. Furthermore, the proposed Compact Complex Linear (CCL) layer enables flexible reallocation of parameter budgets across devices, supporting extreme compression and configuration from edge ends to high-performance computing platforms. Experiments demonstrate that WCCN achieves competitive performance relative to existing methods under the experimental and data settings in this paper, while maintaining parameter and computational efficiency. We note that the current evaluation is limited to classification scenarios; conclusions should not be extrapolated beyond this scope without further validation. Although WCCN exhibits strong potential, combined with the limitation analysis in this paper, future research will focus on the following five directions:

(1) **Training Efficiency and Engineering Optimization:** Addressing the efficiency bottleneck caused by the current framework's lack of support for mixed-precision training, future work will be dedicated to integrating automatic mixed precision (AMP) into complex-valued operators. By optimizing underlying computational kernels, we aim to fully utilize the computing power of modern hardware (such as Tensor Cores) while guaranteeing numerical precision, thereby resolving training throughput and memory occupation issues with large-scale datasets.

(2) **Dynamic Adaptation of Low-Rank Decomposition:** To address the limitations of the fixed-rank strategy in CCL when dealing with long-tail distributions or complex samples, we will explore Adaptive Rank Selection mechanisms. The goal is to empower the network with the ability to dynamically adjust subspace dimensions according to the complexity and class features of input samples, thereby achieving a better balance between extreme compression and the representation capability of tail samples.

(3) **Explicit Modeling of Magnitude-Phase Interaction and Cross-Task Extension:** Given the implicit nature of the current magnitude-phase fusion mode, we will introduce Complex-Valued Attention mechanisms. By explicitly modeling the complementary relationship between magnitude and phase, we aim to release further the model's performance potential in phase-sensitive tasks such as polarimetric SAR interpretation. Simultaneously, we will extend WCCN's efficient operator system beyond classification to a broader range of complex-valued tasks, including object detection, semantic segmentation, complex-valued regression, and generative modeling, to systematically evaluate its task generalization capability.

(4) **Extension toward Broader Algebraic Formulations:** In future work, we will examine whether the operator-design principles of WCCN can generalize to other algebraic representations under unified experimental settings.

(5) **Broader Comparisons under Unified Settings:** To strengthen the empirical evidence for WCCN's competitiveness, we will incorporate comparisons with more recent CVNN and hybrid real/complex-valued architectures under unified training protocols and datasets.

Acknowledgement: None.

Funding Statement: This work was supported by the National Natural Science Foundation of China [Grant Number 62271488].

Author Contributions: The authors confirm contribution to the paper as follows: Conceptualization, Lan Huang; methodology, Bing-Zhou Chen, Hai-Ying Zheng, Zhong-Yi Wang and Lan Huang; software, Bing-Zhou Chen; validation, Ao-Wen Wang and Li-Feng Fan; formal analysis, Ao-Wen Wang; investigation, Bing-Zhou Chen, Hai-Ying Zheng and Ke-Lei Xia; data curation, Ke-Lei Xia; writing—original draft preparation, Bing-Zhou Chen; writing—review and editing, Bing-Zhou Chen, Li-Feng Fan, Zhong-Yi Wang and Lan Huang; funding acquisition, Lan Huang. All authors reviewed and approved the final version of the manuscript.

Availability of Data and Materials: To ensure full reproducibility and transparency, we have made all artifacts associated with this study publicly available at Mendeley Data (https://doi.org/10.17632/gc3w7d4xtc.1). This comprehensive repository contains the complete source code, datasets, and detailed training logs. In particular, to facilitate result verification, we have provided the specific scripts and underlying data corresponding one-to-one with every figure and table presented in this manuscript.

Ethics Approval: Not applicable.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Singhal U, Xing YF, Yu SW. Co-domain symmetry for complex-valued deep learning. In: Proceedings of the 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR); 2022 Jun 18–24; New Orleans, LA, USA. p. 671–80.
2. Trabelsi C, Bilaniuk O, Zhang Y, Serdyuk D, Subramanian S, Santos JF, et al. Deep complex networks. In: Proceedings of the 6th International Conference on Learning Representations; 2018 Apr 30–May 3; Vancouver, BC, Canada. p. 1–19.
3. Gao J, Deng B, Qin Y, Wang H, Li X. Enhanced radar imaging using a complex-valued convolutional neural network. IEEE Geosci Remote Sens Lett. 2019 Jan;16(1):35–9. doi:10.1109/lgrs.2018.2866567.
4. Nguyen K, Fookes C, Sridharan S, Ross A. Complex-valued iris recognition network. IEEE Trans Pattern Anal Mach Intell. 2023 Jan;45(1):182–96. doi:10.1109/tpami.2022.3152857.
5. Lee C, Hasegawa H, Gao S. Complex-valued neural networks: a comprehensive survey. IEEE/CAA J Autom Sinica. 2022 Aug;9(8):1406–26. doi:10.1109/jas.2022.105743.
6. Sikdar A, Udupa S, Sundaram S. Fully complex-valued deep learning model for visual perception. In: Proceedings of the 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP); 2023 Jun 4–10; Rhodes Island, Greece. p. 1–5.
7. Zhang Z, Wang H, Xu F, Jin YQ. Complex-valued convolutional neural network and its application in polarimetric SAR image classification. IEEE Trans Geosci Remote Sens. 2017 Dec;55(12):7177–88. doi:10.1109/tgrs.2017.2743222.
8. Leung H, Haykin S. The complex backpropagation algorithm. IEEE Trans Signal Process. 1991 Sep;39(9):2101–4. doi:10.1109/78.134446.
9. Benvenuto N, Piazza F. On the complex backpropagation algorithm. IEEE Trans Signal Process. 1992 Apr; 40(4):967–9. doi:10.1109/78.127967.
10. Hirose A. Complex-valued neural networks: distinctive features. In: Hirose A, editor. Complex-valued neural networks. Berlin/Heidelberg, Germany: Springer; 2012. p. 17–56. doi: 10.1007/978-3-642-27632-3_3.

11. Reichert DP, Serre T. Neuronal synchrony in complex-valued deep networks. In: Proceedings of the 2014 International Conference on Learning Representations; 2014 Apr 14–16; Banff, AB, Canada. p. 1–9.
12. Ko M, Panchal UK, Andrade-Loarca H, Mendez-Vazquez A. CoShNet: a hybrid complex valued neural network using shearlets. arXiv:2208.06882. 2022 Aug.
13. Mathieu M, Henaff M, LeCun Y. Fast training of convolutional networks through FFTs. In: Proceedings of the 2014 International Conference on Learning Representations (ICLR); 2014 Apr 14–16; Banff, AB, Canada. p. 1–9.
14. Rippel O, Snoek J, Adams RP. Spectral representations for convolutional neural networks. In: Proceedings of the 29th International Conference on Neural Information Processing Systems; 2015 Dec 7–12; Montreal, QC, Canada. p. 2449–57.
15. Chakraborty M, Aryapoor M, Daneshtalab M. Frequency domain complex-valued convolutional neural network. Expert Syst Appl. 2026 Jan;295(4):128893. doi:10.1016/j.eswa.2025.128893.
16. Wang X, Yu SX. Tied block convolution: leaner and better CNNs with shared thinner filters. Proc AAAI Conf Artif Intell. 2021 May;35(11):10227–35.
17. Chakraborty R, Xing Y, Yu SX. SurReal: complex-valued learning as principled transformations on a scaling and rotation manifold. IEEE Trans Neural Netw Learn Syst. 2022 Mar;33(3):940–51.
18. Wu J, Ren H, Kong Y, Yang C, Senhadji L, Shu H. Compressing complex convolutional neural network based on an improved deep compression algorithm. arXiv:1903.02358. 2019 Mar.
19. Virtue P, Yu SX, Lustig M. Better than real: complex-valued neural nets for MRI fingerprinting. In: Proceedings of the 2017 IEEE International Conference on Image Processing (ICIP); 2017 Sep 17; Beijing, China. p. 3953–7.
20. Wang H, Ji Z, Hua Q, Xiong B, Zhang Y, Kang N, et al. CRIA: an enhancement method For CV-CNN based on cross-fusion of complex information of real and imaginary activations. In: 2024 IEEE International Geoscience and Remote Sensing Symposium; 2024 Jul 7–12; Athens, Greece. p. 10285–8.
21. Popa CA. Deep hybrid real-complex-valued convolutional neural networks for image classification. In: Proceedings of the 2018 International Joint Conference on Neural Networks (IJCNN); 2018 Jul 8–13; Rio de Janeiro, Brazil. p. 1–6.
22. Kreutz-Delgado K. The complex gradient operator and the CR-calculus. arXiv:0906.4835. 2009 Jun.
23. Krizhevsky A. Learning multiple layers of features from tiny images. Toronto, ON, Canada: University of Toronto; 2009.
24. Netzer Y, Wang T, Coates A, Bissacco A, Wu B, Ng AY. Reading digits in natural images with unsupervised feature learning. In: NIPS Workshop on Deep Learning and Unsupervised Feature Learning; 2011; Granada, Spain [cited 2026 Mar 15]. Available from: https://ai.stanford.edu/~twangcat/papers/nips2011_housenumbers.pdf.
25. Loshchilov I, Hutter F. Decoupled weight decay regularization. In: 7th International Conference on Learning Representations (ICLR 2019); 2019 May 6–9; New Orleans, LA, USA: OpenReview.net. [cited 2026 Mar 15]. Available from: https://openreview.net/forum?id=Bkg6RiCqY7.