Puppet-CNN: Continuous Parameter Dynamics for Input-Adaptive Convolutional Networks

Yucheng Xing[1] and Xin Wang[1]

Department of Electrical and Computer Engineering
Stony Brook University
Stony Brook, NY 11790, USA
{yucheng.xing, x.wang}@stonybrook.edu

Abstract. Modern convolutional neural networks (CNNs) organize computation as a discrete stack of layers whose parameters are independently stored and learned, with the number of layers fixed as an architectural hyperparameter. In this work, we explore an alternative perspective: *can network parameterization itself be modeled as a continuous dynamical system?* We introduce ***Puppet-CNN***, a framework that represents convolutional layer parameters as states evolving along a learned parameter flow governed by a neural ordinary differential equation (ODE). Under this formulation, layer parameters are generated through continuous evolution in parameter space, and the effective number of generated layers is determined by the discretization resolution used to sample the learned parameter trajectory, which can be modulated by input complexity to enable input-adaptive computation. We validate its feasibility on standard image classification benchmarks and show that continuous parameter dynamics can maintain competitive predictive performance while substantially reducing stored trainable parameters. These results suggest that viewing neural network parameterization through the lens of dynamical systems provides a structured and flexible design space for adaptive convolutional models.

1 Introduction

Modern convolutional neural networks (CNNs) organize computation as a discrete stack of layers whose parameters are independently stored and learned. In this formulation, the number of layers is fixed as an architectural hyperparameter, and layer parameters are treated as separate tensors across depth. This discrete organization has driven much of the progress in modern vision models. However, it also implicitly assumes that parameterization across depth is static rather than structured as a generative process.

In this work, we explore an alternative perspective: *can layer parameterization itself be modeled as a continuous dynamical system?* Instead of specifying each layer's parameters independently, we propose to organize convolutional kernels as states evolving along a learned trajectory in parameter space. Under this view, the executed network remains discrete, but its layers are instantiated by sampling

a shared continuous parameter trajectory rather than by independently storing a fixed tensor for each layer.

We instantiate this idea through ***Puppet-CNN***, a framework consisting of two components: a compact dynamical generator (the *puppeteer*) and a standard convolutional backbone (the *puppet*). The puppeteer is formulated as a neural ordinary differential equation (ODE) that governs the continuous evolution of convolutional parameters. By integrating this learned parameter flow, Puppet-CNN generates the kernels of successive layers through a shared dynamical mechanism, thereby introducing structured coupling across depth. While other sequential models could in principle parameterize depth-wise evolution, the continuous-time formulation provides a direct interpretation in which the effective number of generated layers corresponds to the sampling resolution of the underlying dynamics.

In many real-world scenarios, input samples exhibit heterogeneous levels of structural complexity and may benefit from different amounts of computational processing. Conventional CNNs, however, apply a fixed-depth architecture uniformly across inputs. Under the proposed dynamical formulation, computation can vary by modulating the trajectory initialization and sampling resolution according to input complexity. As a result, both network structure and parameters are generated jointly within a unified continuous framework, rather than being selected or pruned from a fixed pre-trained architecture.

We evaluate Puppet-CNN on standard image classification benchmarks to validate the feasibility of this formulation. Our results demonstrate that modeling parameterization as continuous dynamics can preserve competitive predictive performance while maintaining a compact parameterization. These findings suggest that viewing neural network parameterization through the lens of dynamical systems provides a structured and flexible design space for adaptive convolutional architectures.

The contributions of this paper are three-fold:

- We propose a continuous parameter dynamics formulation for convolutional neural networks, in which layer parameters are modeled as states evolving along a learned trajectory governed by a neural ordinary differential equation.
- We reinterpret network depth as the number of discrete parameter states sampled from the underlying continuous parameter dynamics, enabling a unified mechanism that jointly generates both network structure and layer parameters.
- We show that input-adaptive computation emerges naturally from this dynamical formulation by modulating the integration process according to input complexity, and validate the feasibility of this perspective on standard image classification benchmarks.

The organization of the remainder of the paper is as follows: In Sec. 2, we give an overview of the related works. We introduce our model in detail in Sec. 3 and evaluate it through experiments in Sec. 4. Finally, we conclude our work in Sec. 5.

2 Related Works

2.1 Depth-Variant Computation

Increasing network depth and width has been widely associated with improved representational capacity in convolutional neural networks [10,16,26]. At the same time, it has been observed that different inputs may require different amounts of computational processing [29]. This observation has motivated a line of research on depth-variant or adaptive computation mechanisms. Early-Exiting methods [1, 6, 27, 33] allow samples to exit a deep model at intermediate layers based on pre-defined criteria, such as intermediate classification confidence or uncertainty. These approaches typically operate on standard CNN backbones, sometimes augmented with multi-resolution branches or densely connected structures as in [3, 11, 38]. Other strategies modify execution paths within a fixed architecture. For example, Layer Skipping methods [28,34] use gating mechanisms to decide whether certain layers should be executed. Similarly, approaches [5, 7, 18] dynamically adjust the number of residual transformations applied within each block. These methods share a common characteristic that adaptivity is realized by selecting, skipping, or reusing components from a pre-defined network structure. The underlying layer parameters are learned and stored explicitly, while variation in computation arises from conditional activation within the existing architecture.

2.2 Input-Conditioned Parameterization

In conventional CNNs, parameters are optimized over the training set and then fixed for all inputs during inference. To introduce input-dependent behavior, a line of work explores input-conditioned parameterization, where convolutional kernels are adapted or generated as functions of the input [21]. One group of approaches adjusts a fixed set of base parameters through input-conditioned modulation. For example, CondConv [37] combines multiple expert kernels using sample-dependent routing weights, and Dynamic Convolution [2] further constrains the normalized combination coefficients. Related designs incorporate spatially varying attention mechanisms to modulate convolutional responses according to local content [9, 25]. In these methods, the kernel tensors are pre-defined, while input-dependent coefficients determine how they are activated for each sample. Another group of approaches directly generates parameters from the input using an auxiliary network. Early work [4] factorized parameter matrices and predicted part of the factors at test time. Dynamic Filter Networks (DFNs) [12] and HyperNetworks [8] generate layer parameters on the fly through a separate model. Subsequent extensions explore spatial-specific generation [35], decoupled dynamic kernels [39], meta-learned generators [20], and grouped transformations as in WeightNet [19]. Input-conditioned generation has also been combined with additional contextual signals such as camera perspective [13] or graph edge attributes [22]. Despite differences in implementation, these approaches share a common formulation in which parameter adaptation or generation is performed in a layer-wise manner. Parameters for each layer are produced or modulated

through mappings conditioned on the corresponding layer-wise input features, while the organization of parameters across depth remains discretely defined by the underlying architecture.

2.3 Parameter Generation Paradigms

Beyond input-conditioned parameterization, prior work reflects different paradigms for organizing generated parameters at the network level. In layer-wise generation approaches [8, 12, 19, 35, 39], parameters for successive layers are instantiated independently through separate mappings. Although such parameters may be conditioned on intermediate features, each layer's weights are generated as a discrete entity, and the depth of the network remains pre-defined. A different paradigm synthesizes parameters at the global level. Diffusion-based approaches [24, 32] treat the entire parameter set as a sample from a learned distribution and generate a complete configuration for a fixed architecture. In the reported experiments, parameter synthesis is typically demonstrated on relatively small networks or on subsets of layers, where the parameter space can be represented explicitly as a generative target. In these methods, the generative "time" corresponds to the sampling process in parameter space rather than to the depth dimension of the network, and the structural organization of layers remains unchanged. Related ideas also appear in meta-learning frameworks [20], where parameter generators or meta-learners adapt model weights across tasks. In such settings, adaptation typically occurs at the task level, producing task-specific parameter configurations while preserving a pre-defined network structure. Taken together, existing approaches reflect two dominant parameter generation paradigms: layer-wise parameter mapping and global parameter synthesis under fixed architectures. In both cases, network depth is treated as a pre-defined structural dimension. Generated parameters are either mapped independently per layer or sampled as a complete set, rather than organized as states evolving along depth under a shared dynamical mechanism.

3 Methodology

This section presents the overall architecture of *Puppet-CNN* under the continuous parameter dynamics perspective. The framework consists of two components: a *puppeteer* module, which generates convolutional parameters through a continuous evolution process, and a *puppet* module, which applies the generated parameters to process input data (see Fig. 1). Instead of assigning independent parameters to each layer, the puppeteer organizes parameterization across depth as a continuous dynamical system in parameter space, so that convolutional layers are obtained by discretizing a shared continuous parameter trajectory. In addition, the evolution process can be modulated according to input characteristics, enabling adaptive computation. The following subsections formalize this formulation and describe its practical instantiation.

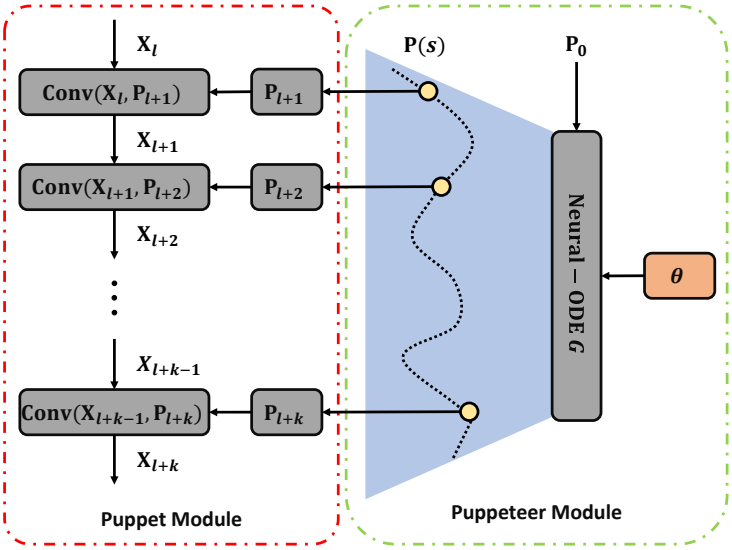


Fig. 1: Overview of Puppet-CNN. The puppeteer module governs continuous parameter evolution, and the generated parameters define the layers of the puppet network.

3.1 Continuous Parameter Evolution

We first formalize the continuous parameter evolution underlying Puppet-CNN. Let $s \in [0, 1]$ denote a normalized continuous evolution coordinate, and let $\mathbf{P}(s)$ represent the convolutional kernel parameters as a function of s. Rather than assigning independent parameter tensors to discrete layers, we model their variation along the evolution coordinate as a continuous process governed by an ordinary differential equation (ODE):

$$\frac{d\mathbf{P}(s)}{ds} = G(\mathbf{P}(s); \theta), \tag{1}$$

where $G(\cdot; \theta)$ is a learnable neural function that defines the parameter dynamics along s. This formulation induces a continuous trajectory in parameter space, where parameters are interpreted as states evolving under a shared transformation rule.

To instantiate a finite network from the continuous formulation, we consider the solution of Eq. (1) over a small interval $[s, s + \Delta s]$, which can be expressed in integral form as

$$\mathbf{P}(s + \Delta s) = \mathbf{P}(s) + \int_{s}^{s+\Delta s} G(\mathbf{P}(\tau); \theta)\, d\tau. \tag{2}$$

Since this integral generally does not admit a closed-form solution, we approximate it numerically. Using an explicit Euler scheme with step size Δs, the evolution can be discretized as

$$\mathbf{P}_l = \mathbf{P}_{l-1} + G(\mathbf{P}_{l-1}; \theta)\, \Delta s, \tag{3}$$

where $\mathbf{P}_l \approx \mathbf{P}(s_l)$ and $s_l = l\,\Delta s$. Each sampled state yields one instantiated convolutional transformation, whose kernel is derived from the corresponding parameter state. Under the normalized evolution interval $[0, 1]$, the effective depth of the network is determined by the sampling resolution, i.e.,

$$D = \lfloor \frac{1}{\Delta s} \rfloor, \tag{4}$$

so that network depth arises from discretizing a single underlying evolution trajectory rather than from pre-specified discrete layers.

3.2 Input-Adaptive Parameter Trajectory Sampling

The continuous parameter evolution framework naturally enables input-adaptive computation. In practical scenarios, input samples exhibit heterogeneous structural complexity and may benefit from different amounts of processing. Within our formulation, adaptivity arises by modulating the sampling of a shared continuous parameter trajectory for each input. Specifically, both the initial condition of the trajectory and the discretization resolution along it can depend on input characteristics. As a result, Puppet-CNN supports adaptive behavior at two complementary levels: *parameter-level adaptation* through trajectory initialization and *depth-level adaptation* through trajectory sampling density.

Parameter-Level Adaptation. As indicated by Eq. (2), the underlying continuous trajectory $\mathbf{P}(s)$ over the normalized interval $[0, 1]$ is uniquely determined by the shared evolution function $G(\cdot; \theta)$ and the initial state $\mathbf{P}_0 = \mathbf{P}(0)$, while the discretized samples used for network instantiation depend on the chosen step size. By allowing this initial condition to depend on a scalar complexity signal $c(\mathbf{X}_0)$ extracted from the input sample $\mathbf{X}_0$,

$$\mathbf{P}_0 = \psi(c(\mathbf{X}_0)), \tag{5}$$

where $\psi(\cdot)$ denotes a lightweight mapping from the scalar complexity measure to the initial parameter state, different inputs induce distinct parameter trajectories under the same evolution rule.

Depth-Level Adaptation. As discussed in Sec. 3.1, the effective depth of the instantiated network is determined by the discretization step size Δs. To enable input-dependent depth variation, we allow the step size to depend on the same complexity signal $c(\mathbf{X}_0)$,

$$\Delta s = \phi(c(\mathbf{X}_0)), \tag{6}$$

where $\phi(\cdot)$ is a scalar mapping that preserves a monotonic relation between the input complexity and the sampling resolution. Through this modulation, inputs with higher complexity lead to finer sampling of the shared parameter trajectory, and thus deeper instantiated networks, while simpler inputs result in coarser sampling and shallower architectures.

These two mechanisms operate on different aspects of the same continuous parameter evolution. The evolution function G remains shared across all inputs,

while input-dependent variation enters only through the initial state and sampling resolution of a single underlying trajectory. Adaptive computation is therefore not introduced as an additional control mechanism, but emerges intrinsically from the continuous organization of parameters across depth. Adaptation becomes a structural consequence of parameter evolution rather than an externally imposed architectural modification.

In practice, we instantiate the scalar complexity signal $c(\mathbf{X}_0)$ using an entropy-based measure [17, 30, 36] that combines spatial- and frequency-domain statistics of the input. Specifically, we define

$$E(\mathbf{X}_0) = -\sum_{x_i} p(x_i) \log p(x_i), \tag{7}$$

where $p(x_i)$ denotes the empirical distribution of pixel intensities in the input. To further account for structural variation in the frequency domain, we compute the entropy of the Fourier-transformed representation $\mathcal{F}(\mathbf{X}_0)$. The final complexity signal is defined as

$$c(\mathbf{X}_0) = \frac{1}{2}E(\mathbf{X}_0) + \frac{1}{2}E(\mathcal{F}(\mathbf{X}_0)). \tag{8}$$

This provides a lightweight proxy for structural complexity. We use this measure as a feasibility instantiation of input-dependent trajectory modulation, rather than as an optimized complexity estimator. The mappings $\psi(\cdot)$ and $\phi(\cdot)$ are then implemented as simple scalar transformations to generate the initial parameter state and sampling resolution, respectively. Importantly, the overall formulation is independent of the specific complexity metric, and alternative scalar measures can be incorporated within the same continuous parameter evolution framework without altering the shared evolution dynamics.

3.3 Puppet-Puppeteer Architecture

We now detail how the continuous parameter evolution framework described in Sec. 3.1 - Sec. 3.2 is instantiated within the Puppet-Puppeteer architecture, as illustrated in Fig. 2. In this realization, the *puppeteer* module governs parameter dynamics along a continuous evolution coordinate, while the *puppet* module is obtained by sampling this trajectory into discretized kernel states, which are injected into successive convolutional layers. Under this view, architectural depth is determined by how densely the trajectory is sampled, while input-dependent modulation can alter both the sampled parameters and the resulting depth.

Starting from an input-dependent initialization $\mathbf{P}_0$ with Eq. (5), the puppeteer propagates parameters along the normalized evolution coordinate $s \in [0, 1]$ according to the shared dynamical function $G(\cdot; \theta)$. The continuous state $\mathbf{P}(s)$ is defined over a maximal tensor shape $(C^{\text{out}}_{\max}, C^{\text{in}}_{\max}, K_{\max}, K_{\max})$, where each dimension corresponds to the largest channel or kernel configuration within the puppet module. This ensures that parameter evolution is performed within a fixed-dimensional state space. For processing within the dynamical function $G(\cdot; \theta)$, the spatial kernel dimensions are reorganized into a convolution-compatible representation, as illustrated in Fig. 3, while preserving a unified parameter state. The

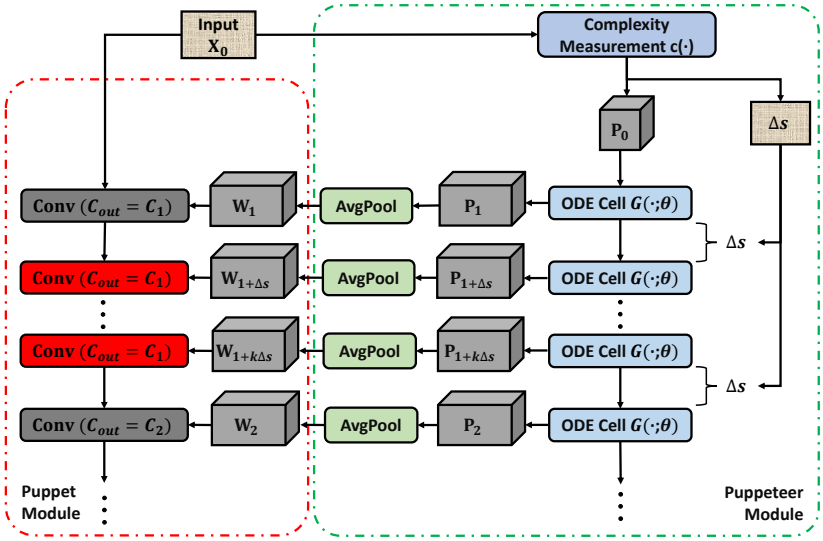


Fig. 2: Architectural instantiation of the Puppet-Puppeteer framework. The puppeteer module evolves parameter states through an ODE trajectory and generates discretized kernel states, which are resized and injected into successive convolutional layers of the puppet network. The trajectory initialization and discretization step are determined by the input complexity measurement.

discretized states $\mathbf{P}_l$ are obtained by sampling the evolution trajectory as defined in Eq. (3). For each convolutional layer l within the puppet module, the sampled state is projected to the required kernel dimension (C_l^{out}, C_l^{in}, K_l, K_l) through a deterministic resizing operation. This design accommodates heterogeneous layer configurations while preserving a single continuous evolution trajectory.

As formalized in Sec. 3.1 (see Eq. (4)), discretizing the evolution interval with step size Δs yields a finite set of sampled states $\{\mathbf{P}_l\}$ whose cardinality determines the effective depth D of the network. Within the Puppet-Puppeteer architecture, each propagation step along the discretized trajectory gives rise to one convolutional transformation in the puppet module, with kernels derived from the corresponding sampled states. Depth therefore emerges from the sampling density of a single continuous trajectory, rather than from pre-defined discrete layer allocation. When Δs is modulated as described in Sec. 3.2, this mechanism naturally enables input-adaptive depth variation.

Bringing the above components together, the parameter generation and feature transformation at layer index l follow a unified forward process. Starting from the input-dependent initialization defined in Eq. (5),

$$\mathbf{P}_0 = \psi(c(\mathbf{X}_0)), \tag{9}$$

the puppeteer module propagates parameters along the discretized evolution trajectory as defined in Eq. (3),

$$\mathbf{P}_l = \mathbf{P}_{l-1} + G(\mathbf{P}_{l-1}; \theta)\, \Delta s, \tag{10}$$

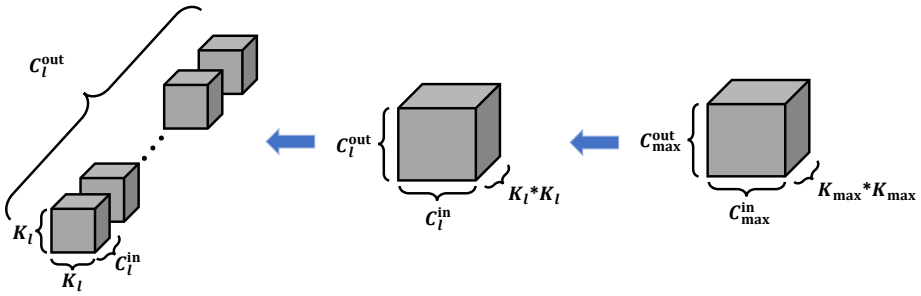


Fig. 3: Organization of the parameter state for continuous evolution and layer-wise instantiation. The continuous evolution is performed in a fixed-dimensional maximal tensor space, where spatial kernel dimensions are reshaped for processing within the dynamical function $G(\cdot;\theta)$. Sampled states are subsequently projected to match the layer-specific kernel configurations.

where the step size Δs may depend on the same complexity signal as described in Eq. (6),

$$\Delta s = \phi(c(\mathbf{X}_0)). \tag{11}$$

For each layer l in the puppet module, the sampled state $\mathbf{P}_l$ is resized to match the required kernel dimension (C_l^{out}, C_l^{in}, K_l, K_l), producing the convolutional kernel $\mathbf{W}_l$. The puppet module then applies this kernel as the convolutional transformation at layer l to update the feature representation,

$$\mathbf{X}_{l+1} = \text{Conv}(\mathbf{X}_l, \mathbf{W}_l). \tag{12}$$

Through this unified process, continuous parameter evolution, trajectory sampling, and discrete convolutional computation are coupled in a single dynamical mechanism, from which both parameter values and architectural depth emerge. Furthermore, since all convolutional kernels are generated from the shared dynamical function $G(\cdot;\theta)$, training the puppet network implicitly reduces to optimizing the parameters θ of the puppeteer through backpropagation along the discretized trajectory.

4 Experiments

We evaluate the proposed Puppet-CNN framework on image classification to assess its effectiveness in practical convolutional architectures. Although the formulation is task-agnostic and applies to general convolutional models, classification provides a controlled setting to examine parameter efficiency, adaptive depth behavior, and overall predictive performance. Accordingly, the experiments focus on feasibility validation of the proposed parameterization mechanism under controlled training settings. Through these experiments, we analyze whether organizing convolutional parameters via continuous evolution can reduce parameter redundancy while maintaining competitive predictive performance under heterogeneous input complexity.

4.1 Datasets

CIFAR-10 [14]: CIFAR-10 consists of 60,000 RGB images of size 32×32 across 10 categories, with 50,000 training and 10,000 testing samples. In our experiments, we randomly select 10,000 images from the official training set for model training to evaluate performance under practical limited-data settings, where compact parameterization is especially relevant. From this subset, 20% of the samples are further used for validation, and the remaining samples are used for training.
CIFAR-100 [15]: CIFAR-100 contains 60,000 RGB images of size 32×32 across 100 categories, with 50,000 training and 10,000 testing samples. We follow the standard split and use the full training set, where 20% of the training data is held out as validation for hyperparameter selection.
mini-ImageNet [31]: mini-ImageNet is a subset of ImageNet containing 60,000 images across 100 categories. We split the dataset into training, validation, and testing sets with a ratio of 8:1:1 using a fixed random seed. All images are resized and randomly cropped to 64×64 during preprocessing.

4.2 Implementation Details

All models are implemented in PyTorch and trained from scratch on a single NVIDIA GeForce RTX 4090 GPU. To ensure fair comparison, all baseline methods are re-implemented and trained under identical data splits and optimization settings. For CIFAR-10, 10,000 training samples are used as described in Sec. 4.1. All models are trained for 800 epochs using the Adam optimizer with a batch size of 64. The learning rate is initialized at 1×10^{-3} and decayed to 1×10^{-5} by the end of training. To ensure comparable model capacity across all methods, we restrict the maximum number of output channels to 512 for both Puppet-CNN and all baseline models. No additional data augmentation is applied unless explicitly specified. This controlled protocol is intended to focus the comparison on the parameterization mechanism rather than dataset-specific training recipes.

For Puppet-CNN, we adopt a convolutional backbone with channel configurations $\{64, 128, 256, 512\}$. Each convolutional layer is followed by batch normalization and ReLU activation. When the channel dimension changes, a 2D max-pooling layer is applied. All convolutional kernels are of size 3×3 and are generated from discretized parameter states produced by the ODE propagation defined in Eq. (3). The dynamical function G is implemented as a one-layer depthwise-separable convolution followed by batch normalization and a tanh activation. Sampled parameter states are resized via 3D average pooling to match the required kernel dimensions before instantiation. The adaptive discretization step size is implemented as

$$\Delta s = \phi(c(\mathbf{X}_0)) = \tanh\big(c(\mathbf{X}_0)^{-1}\big)\,, \tag{13}$$

where $c(\mathbf{X}_0)$ denotes the entropy-based complexity defined in Eq. (8). The effective depth is therefore determined by $D = \lfloor 1/\Delta s \rfloor$. The initial parameter state is initialized as

$$\mathbf{P}_0 = \psi(c(\mathbf{X}_0)) = \exp\big(1 - c(\mathbf{X}_0)^{-2}\big)\,, \tag{14}$$

which is expanded to match the maximal kernel tensor dimension. These mappings are deterministic and introduce no additional learnable parameters.

4.3 Overall Performance

We evaluate Puppet-CNN on CIFAR-10 and compare it with representative adaptive-parameter methods, including DFN [12], WeightNet [19], and DDFN [39] (modified to sample-wise, denoted as DDFN-SW), as well as adaptive-depth methods such as BranchyNet [27], SkipNet [34], and DRNN [7]. All models are trained under the same settings described in Sec. 4.2. Performance is evaluated using Top-1 and Top-5 accuracy (↑), while efficiency is measured by trainable parameter size (Params (MB), ↓) and inference speed (Speed (s/img), ↓).

Table 1: Overall comparison among different models.

		Top-1 Acc (↑)	Top-5 Acc (↑)	Params (↓)	Speed (↓)
Adaptive Params	DFN	68.59%	93.13%	75.89	0.0012
	WeightNet	62.77%	93.52%	45.87	0.0040
	DDFN-SW	55.55%	93.34%	300.83	0.0059
Adaptive Depth	BranchyNet	70.00%	93.94%	27.69	0.0015
	SkipNet	55.82%	87.95%	68.88	0.0115
	DRNN	41.71%	63.89%	45.78	0.0025
Ours	**Puppet-CNN**	**72.51**%	**96.85**%	**1.08**	0.0039

As shown in Tab. 1, Puppet-CNN attains strong classification accuracy in this setting, while using only 1.08 MB of trainable parameters, substantially fewer than the compared adaptive architectures. Overall, the results suggest that organizing convolutional parameters through continuous evolution can yield a compact model that remains competitive in predictive performance, while supporting sample-wise parameter and depth adaptation within a unified framework.

4.4 Ablation Study

To better understand the contribution of different components in our model, we conduct ablation studies from three aspects: the *Puppet-Puppeteer parameterization scheme*, *parameter-level adaptation*, and *depth-level adaptation*. The goal of these experiments is not to demonstrate uniformly higher accuracy, but to examine whether continuous parameter dynamics can serve as a practical and compact parameterization mechanism for convolutional neural networks.

Puppet-Puppeteer Scheme. We first examine whether organizing convolutional parameters through a continuous evolution process can serve as a practical parameterization mechanism for CNN models. To this end, we apply the proposed Puppet-Puppeteer framework to several representative CNN backbones, including

AlexNet [16], VGG [23], and ResNet [10]. These models represent several fundamental design patterns widely used in convolutional networks, including shallow convolutional architectures, deep sequential convolutional structures, and residual connection-based networks. Tab. 2 compares the original backbones with their

Table 2: Progressive ablation study on representative CNN backbones with the Puppet-Puppeteer scheme and parameter adaptation.

	Fixed-CNN			+ Puppet-Puppeteer			+ Parameter Adaptation		
	Top-1	Top-5	Params	Top-1	Top-5	Params	Top-1	Top-5	Params
AlexNet	65.86%	92.05%	11.65	69.33%	94.05%	1.08	70.05%	94.13%	1.08
VGG	67.84%	91.98%	39.01	67.78%	93.02%	1.08	68.42%	90.13%	1.08
ResNet	68.94%	95.17%	44.84	66.69%	95.46%	1.08	68.25%	94.31%	1.08

Puppet variants. Across these architectures, replacing independently learned layer parameters with parameters generated from a shared evolution process maintains competitive predictive performance while dramatically reducing the number of trainable parameters. This suggests that continuous parameter evolution can serve as a viable alternative to conventional layer-wise parameterization across diverse CNN structures.

Table 3: Comparison between Puppet-CNN and lightweight CNN architectures.

	Top-1 Acc	Top-5 Acc	Params	Speed
AlexNet	65.86%	92.05%	11.65	0.0006
MobileNet-v1	58.58%	91.17%	7.52	0.0018
MobileNet-v2	66.73%	93.33%	8.90	0.0032
SqueezeNet	63.31%	94.34%	3.96	0.0022
Puppet-CNN	**72.51**%	**96.85**%	**1.08**	0.0039

To further contextualize the efficiency and performance characteristics of the proposed model, Tab. 3 compares Puppet-CNN with several lightweight CNN architectures designed for parameter-efficient learning. Under the same experimental setting, Puppet-CNN achieves competitive accuracy while using substantially fewer parameters, highlighting the parameter efficiency naturally induced by the proposed parameter evolution mechanism.

For reference, we also report the corresponding accuracy comparison under both partial and full training sets in Tab. 4. Across different backbone templates, the Puppet variants remain comparable to their fixed counterparts in both regimes, indicating that the effectiveness of the proposed parameterization is not limited to a particular training data scale.

Parameter-Level Adaptation. Since all convolutional parameters in the puppet module are generated through the shared parameter evolution process, the

Table 4: Top-1 / Top-5 accuracy comparison between fixed CNNs and their Puppet-Puppeteer variants under partial (10k samples) and full (complete training set) CIFAR-10 training data.

	Partial Set		Full Set	
	Fixed-CNN	Puppet-Puppeteer	Fixed-CNN	Puppet-Puppeteer
AlexNet	65.86%/92.05%	69.33%/94.05%	80.89%/93.91%	83.75%/96.90%
VGG	67.84%/91.98%	67.78%/93.02%	81.05%/95.37%	80.57%/95.13%
ResNet	68.94%/95.17%	66.69%/95.46%	82.24%/96.11%	80.47%/96.70%

framework naturally enables input-adaptive parameter generation conditioned on input complexity. As shown in Tab. 2, introducing parameter adaptation improves Top-1 accuracy over the Puppet-Puppeteer-only variant in some backbones, and helps close the gap to the corresponding fixed-parameter baselines.

Depth-Level Adaptation. Generating convolutional parameters through the puppeteer ODE module introduces additional computational cost when the network structure is kept fixed. To control this overhead, our framework further introduces an input-dependent depth adaptation mechanism that adjusts the number of instantiated convolutional layers according to the input complexity. Tab. 5 compares three variants: the original ResNet with fixed parameters, a Puppet-ResNet where parameters are generated but the network depth remains fixed, and the proposed Puppet-CNN with adaptive depth. While Puppet-ResNet incurs substantially higher computational cost due to parameter generation, Puppet-CNN maintains a number of operations close to that of the original ResNet. This observation suggests that the proposed depth adaptation mechanism helps regulate the overall inference cost by adjusting the number of instantiated convolutional layers. Together with the results in Tab. 2, these results also support the effectiveness of the input complexity measurement introduced in Sec. 3.2.

Table 5: Computational cost comparison between fixed-depth and adaptive-depth variants.

	Params	Mult-Adds(G)	Speed
ResNet	44.84	11.40	0.0019
Puppet-ResNet	1.08	23.44	0.0061
Puppet-CNN	1.08	12.34	0.0039

4.5 Robustness Study

To further examine the generalization of the proposed parameter evolution mechanism, we evaluate Puppet-CNN on more challenging classification datasets, including CIFAR-100 and mini-ImageNet. Compared with CIFAR-10, these

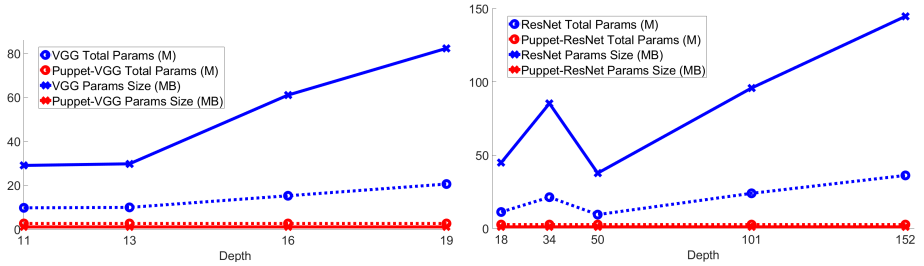
datasets contain more categories and fewer training samples per category, resulting in a more challenging classification setting. In our CIFAR-10 experiments, only a subset of the training data is used, resulting in approximately 800 training samples per category. In CIFAR-100, each class contains only 500 images in total, with 400 used for training. A similar situation also appears in mini-ImageNet, where the number of training samples per category is also limited. Tab. 6 reports the performance comparison between Puppet-CNN and several representative CNN backbones on these datasets. We focus on relative behavior under a unified training protocol rather than saturating benchmark accuracy. Across both datasets, Puppet-CNN maintains competitive Top-1 and Top-5 accuracy while using substantially fewer parameters. These results suggest that organizing convolutional parameters through continuous evolution remains effective in more challenging classification scenarios and can provide a compact parameterization of convolutional networks. Similar trends across different datasets indicate that the proposed parameter evolution mechanism generalizes beyond a specific dataset or training configuration.

Table 6: Performance comparison on CIFAR-100 and mini-ImageNet.

	CIFAR-100			mini-ImageNet		
	Top-1 Acc	Top-5 Acc	Params	Top-1 Acc	Top-5 Acc	Params
AlexNet	41.46%	67.67%	11.84	36.92%	64.35%	11.84
VGG	45.06%	67.66%	39.20	32.95%	59.94%	39.20
ResNet	44.07%	70.96%	45.02	38.41%	68.58%	45.02
Puppet-CNN	**53.26**%	**79.63**%	**1.08**	**46.46**%	**74.38**%	**1.08**

4.6 Quantitative Study

We further analyze the parameterization behavior of the proposed Puppet-CNN with respect to network depth and channel capacity. Fig. 4 illustrates how model size changes as network depth increases for VGG and ResNet architectures. In this experiment, the depth-adaptation component is disabled so that the depth of the puppet module can be manually controlled. The puppet module follows the architectural templates of VGG and ResNet while the number of layers is increased. For conventional CNNs, the number of parameters grows with network depth because each layer contains independently learned convolutional kernels. In contrast, the Puppet variants maintain nearly constant parameter sizes across different depths since convolutional kernels are generated from the shared parameter evolution process. This behavior indicates that the proposed parameterization effectively decouples model size from network depth. We also examine how the parameter size changes with respect to the maximum number of output channels in the puppet module. As described in Sec. 3.3, the number of variables in the puppeteer ODE module depends on the maximum output channel capacity. Tab. 7 reports the resulting variable scale and trainable parameter size under different channel configurations. As expected, increasing the maximum



(a) VGG vs. Puppet-VGG **(b)** ResNet vs. Puppet-ResNet

Fig. 4: Parameter size comparison between conventional CNNs and their Puppet counterparts under increasing network depth for VGG and ResNet architectures.

Table 7: Variable scale and trainable parameter size under different maximum output channel settings in the puppet module.

$C_{\max}^{\text{out}}$	64	128	256	512	1024	2048	4096
Total Variables (M)	0.04	0.17	0.66	2.63	10.50	41.97	167.83
Trainable Param Size (MB)	0.02	0.07	0.28	1.08	4.26	16.90	67.35

output channel leads to a quadratic growth in the variable scale due to the dimensionality of the generated convolutional kernels. Nevertheless, the overall parameter size remains relatively small compared to conventional CNN models with independently learned parameters.

5 Conclusion

We propose Puppet-CNN, a convolutional framework in which convolutional kernels and network depth are generated through a shared parameter evolution process. In the proposed architecture, the parameters of the convolutional layers are produced by a puppeteer ODE module conditioned on the input complexity, enabling sample-dependent parameter and depth adaptation. The proposed parameterization significantly reduces the number of trainable parameters while allowing deep convolutional architectures to be instantiated through a compact evolution mechanism. In addition, the framework is compatible with existing CNN backbones and can be integrated into different network structures. Experimental results across several image classification benchmarks demonstrate that Puppet-CNN maintains competitive predictive performance while using substantially fewer parameters than conventional CNN architectures. These results suggest that organizing convolutional parameters through continuous evolution provides an effective alternative to conventional layer-wise parameterization. In future work, we plan to evaluate this parameter evolution mechanism on larger-scale benchmarks and additional vision tasks, and investigate its extension to modern backbones and neural architectures beyond convolutional models. Improving the efficiency of the parameter generation process is another important direction for further study.

References

1. Bolukbasi, T., Wang, J., Dekel, O., Saligrama, V.: Adaptive neural networks for efficient inference. In: International Conference on Machine Learning. pp. 527–536. PMLR (2017)
2. Chen, Y., Dai, X., Liu, M., Chen, D., Yuan, L., Liu, Z.: Dynamic convolution: Attention over convolution kernels. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 11030–11039 (2020)
3. Dai, X., Kong, X., Guo, T.: Epnet: Learning to exit with flexible multi-branch network. In: Proceedings of the 29th ACM International Conference on Information & Knowledge Management. pp. 235–244 (2020)
4. Denil, M., Shakibi, B., Dinh, L., Ranzato, M., De Freitas, N.: Predicting parameters in deep learning. Advances in neural information processing systems **26** (2013)
5. Figurnov, M., Collins, M.D., Zhu, Y., Zhang, L., Huang, J., Vetrov, D., Salakhutdinov, R.: Spatially adaptive computation time for residual networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 1039–1048 (2017)
6. Guan, J., Liu, Y., Liu, Q., Peng, J.: Energy-efficient amortized inference with cascaded deep classifiers. arXiv preprint arXiv:1710.03368 (2017)
7. Guo, Q., Yu, Z., Wu, Y., Liang, D., Qin, H., Yan, J.: Dynamic recursive neural network. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 5147–5156 (2019)
8. Ha, D., Dai, A.M., Le, Q.V.: Hypernetworks. In: International Conference on Learning Representations (2017), `https://openreview.net/forum?id=rkpACe1lx`
9. Harley, A.W., Derpanis, K.G., Kokkinos, I.: Segmentation-aware convolutional networks using local attention masks. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 5038–5047 (2017)
10. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)
11. Huang, G., Chen, D., Li, T., Wu, F., Van Der Maaten, L., Weinberger, K.Q.: Multi-scale dense networks for resource efficient image classification. arXiv preprint arXiv:1703.09844 (2017)
12. Jia, X., De Brabandere, B., Tuytelaars, T., Gool, L.V.: Dynamic filter networks. Advances in neural information processing systems **29** (2016)
13. Kang, D., Dhar, D., Chan, A.: Incorporating side information by adaptive convolution. Advances in Neural Information Processing Systems **30** (2017)
14. Krizhevsky, A., Nair, V., Hinton, G.: Cifar-10 (canadian institute for advanced research). URL http://www. cs. toronto. edu/kriz/cifar. html **5**(4), 1 (2010)
15. Krizhevsky, A., Nair, V., Hinton, G.: Cifar-100 (canadian institute for advanced research). URL http://www. cs. toronto. edu/kriz/cifar. html **5**(4), 1 (2010)
16. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems **25** (2012)
17. Larkin, K.G.: Reflections on shannon information: In search of a natural information-entropy for images. arXiv preprint arXiv:1609.01117 (2016)
18. Leroux, S., Molchanov, P., Simoens, P., Dhoedt, B., Breuel, T., Kautz, J.: Iamnn: Iterative and adaptive mobile neural network for efficient image classification. arXiv preprint arXiv:1804.10123 (2018)

19. Ma, N., Zhang, X., Huang, J., Sun, J.: Weightnet: Revisiting the design space of weight networks. In: European Conference on Computer Vision. pp. 776–792. Springer (2020)
20. Munkhdalai, T., Yu, H.: Meta networks. In: International conference on machine learning. pp. 2554–2563. PMLR (2017)
21. Schmidhuber, J.: Learning to control fast-weight memories: An alternative to dynamic recurrent networks. Neural Computation **4**(1), 131–139 (1992)
22. Simonovsky, M., Komodakis, N.: Dynamic edge-conditioned filters in convolutional neural networks on graphs. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 3693–3702 (2017)
23. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)
24. Soro, B., Andreis, B., Lee, H., Jeong, W., Chong, S., Hutter, F., Hwang, S.J.: Diffusion-based neural network weights generation. arXiv preprint arXiv:2402.18153 (2024)
25. Su, H., Jampani, V., Sun, D., Gallo, O., Learned-Miller, E., Kautz, J.: Pixel-adaptive convolutional neural networks. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 11166–11175 (2019)
26. Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z.: Rethinking the inception architecture for computer vision. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 2818–2826 (2016)
27. Teerapittayanon, S., McDanel, B., Kung, H.T.: Branchynet: Fast inference via early exiting from deep neural networks. In: 2016 23rd international conference on pattern recognition (ICPR). pp. 2464–2469. IEEE (2016)
28. Veit, A., Belongie, S.: Convolutional networks with adaptive inference graphs. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 3–18 (2018)
29. Veit, A., Wilber, M.J., Belongie, S.: Residual networks behave like ensembles of relatively shallow networks. Advances in neural information processing systems **29** (2016)
30. Vila, M., Bardera, A., Feixas, M., Bekaert, P., Sbert, M.: Analysis of image informativeness measures. In: 2014 IEEE International Conference on Image Processing (ICIP). pp. 1086–1090. IEEE (2014)
31. Vinyals, O., Blundell, C., Lillicrap, T., Wierstra, D., et al.: Matching networks for one shot learning. Advances in neural information processing systems **29** (2016)
32. Wang, K., Xu, Z., Zhou, Y., Zang, Z., Darrell, T., Liu, Z., You, Y.: Neural network diffusion. arXiv preprint arXiv:2402.13144 (2024)
33. Wang, X., Luo, Y., Crankshaw, D., Tumanov, A., Yu, F., Gonzalez, J.E.: Idk cascades: Fast deep learning by learning not to overthink. arXiv preprint arXiv:1706.00885 (2017)
34. Wang, X., Yu, F., Dou, Z.Y., Darrell, T., Gonzalez, J.E.: Skipnet: Learning dynamic routing in convolutional networks. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 409–424 (2018)
35. Wu, J., Li, D., Yang, Y., Bajaj, C., Ji, X.: Dynamic filtering with large sampling field for convnets. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 185–200 (2018)
36. Wu, Y., Zhou, Y., Saveriades, G., Agaian, S., Noonan, J.P., Natarajan, P.: Local shannon entropy measure with statistical tests for image randomness. Information Sciences **222**, 323–342 (2013)

37. Yang, B., Bender, G., Le, Q.V., Ngiam, J.: Condconv: Conditionally parameterized convolutions for efficient inference. Advances in neural information processing systems **32** (2019)
38. Yang, L., Han, Y., Chen, X., Song, S., Dai, J., Huang, G.: Resolution adaptive networks for efficient inference. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 2369–2378 (2020)
39. Zhou, J., Jampani, V., Pi, Z., Liu, Q., Yang, M.H.: Decoupled dynamic filter networks. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 6647–6656 (2021)