Complex & Intelligent Systems (2022) 8:3395–3407
https://doi.org/10.1007/s40747-022-00671-3

ORIGINAL ARTICLE

Rethinking ResNets: improved stacking strategies with high-order schemes for image classification

Zhengbo Luo[1] · Zitang Sun[1] · Weilian Zhou[1] · Zizhang Wu[2] · Sei-ichiro Kamata[1]

Received: 27 August 2021 / Accepted: 28 January 2022 / Published online: 22 February 2022

Abstract
Various deep neural network architectures (DNNs) maintain massive vital records in computer vision. While drawing attention worldwide, the design of the overall structure lacks general guidance. Based on the relationship between DNN design and numerical differential equations, we performed a fair comparison of the residual design with higher order perspectives. We show that the widely used DNN design strategy, constantly stacking a small design (usually, 2–3 layers), could be easily improved, supported by solid theoretical knowledge and with no extra parameters needed. We reorganise the residual design in higher order ways, which is inspired by the observation that many effective networks can be interpreted as different numerical discretisations of differential equations. The design of ResNet follows a relatively simple scheme, which is Euler forward; however, the situation becomes complicated rapidly while stacking. We suppose that stacked ResNet is somehow equalled to a higher order scheme; then, the current method of forwarding propagation might be relatively weak compared with a typical high-order method such as Runge–Kutta. We propose HO-ResNet to verify the hypothesis on widely used CV benchmarks with sufficient experiments. Stable and noticeable increases in performance are observed, and convergence and robustness are also improved. Our stacking strategy improved ResNet-30 by 2.15% and ResNet-58 by 2.35% on CIFAR-10, with the same settings and parameters. The proposed strategy is fundamental and theoretical and can, therefore, be applied to any network as a general guideline.

Keywords Image classification · Deep neural networks · Neural ordinary differential equations · Image processing

Introduction

Deep neural networks (DNNs) have achieved many exemplary breakthroughs in computer vision, image processing, and signal processing with powerful learning representations from extremely deep structures and massive data. Moreover, repeated simple functions are doing a surprisingly good job while approximating complicated ones; even though this is not fully understood.

Many DNNs have been proposed for more specific tasks, which all perform well. However, despite the tremendous success, we still lack a theoretical understanding of DNNs. In this study, we examine DNNs from a continuous perspective, regard DNNs as discrete dynamical systems, and then improve the stacking strategy of DNNs with high-order numerical methods.

It is common for DNNs to have more than a hundred layers; however, one usually designs a sub-network with around 2–3 layers and then repeats it multiple times rather than designing a huge one directly. The fixed topology and relationship between blocks lead to the development of dynamic systems.

Enormous possibilities and advantages are gained from this view. For example, mathematical problems are easier to solve and provide direct links to physical sciences and solid theoretical support from differential equations. This study focuses on the depth of DNNs, specifically, the stacking strategy of blocks: constructing DNNs with a given sub-net design in high-order ways.

We show that a certain DNN (ResNet as the example in this study) can be easily improved by slight adjustments in the block stacking strategy, following numerical methods, without any changes in width and depth. Moreover, the improvements are directly proportional to the order of the

✉ Zhengbo Luo
lewisluo@fuji.waseda.jp

[1] Graduate School of IPS, Waseda University, Kitakyushu, Fukuoka, Japan

[2] Zongmu Technology, Shanghai, China

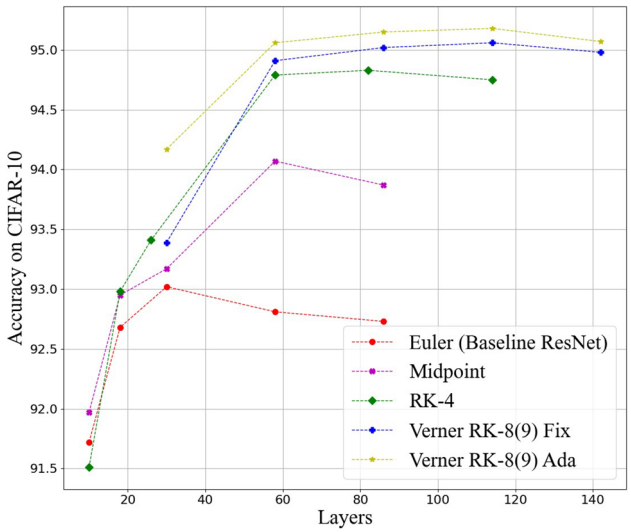

Fig. 1 HO-ResNet, stacking the same layers in higher order: given fixed layers, with the same parameters, there are more advanced stacking strategies supported by numerical methods. We implemented three of them to be HO-ResNet, which achieved noticeable improvements in several aspects. Moreover, the degradation problem occurs later with more advanced numerical schemes

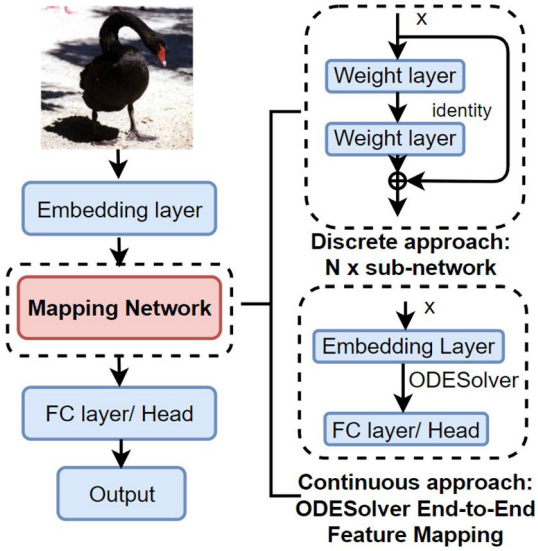

Fig. 2 Discrete and continuous approaches of feature mapping: discrete methods usually design a basic structure (2–3 layers), and repeat it N times to have various depths. The size of model, computational cost and performance changes with the depth. Meanwhile, the NODEs map features end-to-end with fully continuous flow and O(1) para size

numerical methods, not only in terms of accuracy but also in terms of convergence and robustness.

Related work

The relationship between DNNs and dynamical systems has been widely discussed in recent years. In 2016, Liao et al. [20] stated the equivalence of ResNet [12] and a specific recurrent network based on formulations from dynamical systems. A systematic proposal was published by Weinan [30] in 2017, indicating that DNNs could be thought of as a discretisation of continuous dynamical systems; the proposal and neural ordinary differential equations motivated us to rethink ResNet using advanced numerical methods.

Neural ODEs

In 2018, neural ordinary differential equations (NODEs [6]) took a step forward, replacing the discrete DNNs with solvers of ordinary differential equations (ODEs).

To highlight the equivalence, we take a residual network (ResNet) as an example. Let model ResNet be $\mathbf{RES}^L(\mathbf{x}) : \mathbb{R}^{d_{\text{in}}} \to \mathbb{R}^{d_{\text{out}}}$ as an L-layer residual neural network. We assume that the hidden state $\mathcal{N}$ belongs to $\mathbb{R}^{d_{\text{N}}}$, and the input layer is $\mathcal{N}^0(\mathbf{x}) = \mathbf{x} \in \mathbb{R}^{d_{\text{in}}}$. The residual design is described as follows:

$$\mathcal{F}\left(\mathbf{x}, \left\{\boldsymbol{W}^i\right\}\right) + \mathbf{x}, \quad \text{s.t.} \quad 1 \le i \le L-1. \tag{1}$$

i is an integer only and $\mathcal{F}\left(\mathbf{x}, \left\{\boldsymbol{W}^i\right\}\right)$ represents the residual mapping to be learned which we do not expand further.

A sequence of transformation to a hidden state $\mathcal{N}$ from depth i to $i+1$ in residual networks can be written as follows, where $i \in \{1 \dots L-1\}$ and $\boldsymbol{W}$ refers to the weight matrix:

$$\mathcal{N}^{i+1} = \mathcal{F}\left(\mathcal{N}^i, \left\{\boldsymbol{W}^i\right\}\right) + \mathcal{N}^i, \quad \text{s.t.} \quad 1 \le i \le L-1. \tag{2}$$

The situation above always takes a step $\Delta i = 1$, letting $\Delta i \to 0$, we have

$$\lim_{\Delta i \to 0} \frac{\mathcal{N}_{i+\Delta i} - \mathcal{N}_i}{\Delta i} = \frac{\mathrm{d}\mathcal{N}(i)}{\mathrm{d}i} = \mathcal{F}(\mathcal{N}(i), i), \tag{3}$$

thereby, hidden states can be parameterised using an ODE. Data point x can be mapped into a set of features by solving the initial value problem (IVP):

$$\frac{\mathrm{d}\mathcal{N}(i)}{\mathrm{d}i} = \mathcal{F}(\mathcal{N}(i), i), \quad \mathcal{N}(0) = \mathbf{x}, \tag{4}$$

at a certain position. $\mathcal{N}(i)$ corresponds to the features learned by the model. NODEs map the input to output by solving an ODE starting from $\mathcal{N}(0)$ and adjust the dynamics to fit the output of the system closer to the label.

In addition to ResNet, several outstanding networks are linked to different numerical schemes (see Table 1). The NODEs are a family of DNNs that can be interpreted as a continuous equivalent of a certain discrete DNN (see Fig. 2). Inspired by several results of NODEs, we believe that

Table 1 Connections between neural networks and ODE schemes: some networks have been proposed without awareness of their connection with ODE schemes such as ResNet, RevNet, and PolyNet, while others have been proposed following the guidelines of ODE schemes such as LM-ResNet and second-Order CNNs

Networks	Corresponding ODE formula	ODE Scheme
ResNet [12]	$\mathcal{N}^{t+1} = \mathcal{N}^t + \mathcal{F}\left(\mathcal{N}^t, \boldsymbol{W}^t\right)$	Forward Euler
RevNet [10]	$\mathcal{Y}^t = \mathcal{N}^t + \mathcal{F}\left(\mathcal{N}^{t+1}, \boldsymbol{W}^t\right), \mathcal{Y}^{t+1} = \mathcal{N}^{t+1} + \mathcal{G}\left(\mathcal{Y}^t, \boldsymbol{W}^t\right)$	Forward Euler
PolyNet [32]	$\mathcal{N}^{t+1} = \mathcal{N}^t + \mathcal{F}\left(\mathcal{N}^t, \boldsymbol{W}^t\right) + \mathcal{F}\left(\mathcal{F}\left(\mathcal{N}^t, \boldsymbol{W}^t\right)\right)$	Backward Euler
LM-ResNet [22]	$\mathcal{N}^{t+1} = (1-\Delta t)\mathcal{N}^t + \Delta t\mathcal{F}\left(\mathcal{N}^{t-1}, \boldsymbol{W}^{t-1}\right) + \mathcal{F}\left(\mathcal{N}^t, \boldsymbol{W}^t\right)$	Linear-MultiStep
Second-order CNNs [23]	$\mathcal{N}^{t+1} = 2\mathcal{N}^t - \mathcal{N}^{t-1} + \Delta t^2\mathcal{F}\left(\mathcal{N}^t, \boldsymbol{W}^t\right)$	Second-order

The form of ResNet is widely discussed as the Euler forward scheme, which is the most straightforward way to solve the initial value problem. RevNet is a reversible network, which means that the dynamic can be simulated from the end time back to the initial time. PolyNet includes polynomial compositions that can be interpreted as approximations to one step of the implicit backward Euler scheme. LM ResNet adopted the known linear multistep method in numerical ODEs and achieved the same accuracy by half the parameter size. Moreover, the second-order CNN considers higher order states, which is another direction that can be further explored

continuous theoretical concepts could lead to equivalent improvements in discrete DNNs.

Common high-order numerical schemes for IVP

With respect to the design of DNNs, structures by NAS have been in a leading position in recent years. However, recent work from the Google team revisited ResNet to conclude that state-of-the-art NAS structures such as Efficient-Nets [27] are not necessarily better than ResNets [2]. Another excellent piece of work, RepVGG [9], also achieved state-of-the-art results with a simple VGG-style sub-net. The comeback of ResNet and simple sub-net motivated us to rethink the block-stacking strategy, i.e. how to stack these sub-nets in a better way. We studied ResNet because complicated NAS designs did not seem necessarily better than ResNet.

Euler's scheme is a simple and elegant one-order method, but most networks contain a few basic components. The true expression of the stacked network is nested and in a considerably high order. Will advanced numerical methods make things better? Our stacking designs are based on the corresponding high-order methods for the differential equation. Adopting Euler forward as the baseline ResNet [16], midpoint [4], Runge–Kutta 4th, and Verner's RK-8(9)th-Order Embedded Runge–Kutta method [5,24,29] are the general guidelines for our proposed stacking design.

Effective DNNs that related to ODE schemes

Table 1 shows several outstanding networks with their corresponding schemes. Besides them, many powerful networks are partially related to the ODE scheme. For example, dense connections are like shortcuts of high-order RK schemes [14], a DenseNet block is nested high-order RK; the One-Shot Aggregation (OSA) proposed by VoVNet [18] is very similar to how RK methods conclude the block output from each state inside. Moreover, in NAS, which discovered several complicated structures, its stacking links (regardless of sub-net differences) may eventually converge to a specific high-order scheme.

Contribution

Motivation. Our strategy is motivated by multiple recent works mentioned above. The comeback of the simple sub-net design made us focus more on the stacking of blocks rather than on the design of a small sub-net. The improved performance of NODEs with advanced solvers indicated to us that it would be of interest to explore stacked discrete networks, following numerical schemes with various orders.

Difference and Novelty Advanced schemes in NODEs are used to solve IVPs. NODEs use error tolerance instead of a simple depth of layers; therefore, advanced schemes require more computational cost due to complex operations. However, it is not convenient to constrain the FLOPs of NODEs with various schemes simultaneously since the complexity of calculation is related to the landscape of high-dimensional features. Meanwhile, it is rather simple in discrete DNNs. To our best knowledge, it is the first work to implement and compare DNNs in these schemes under the same conditions.

Pros and cons. Our method enables DNNs to achieve better performance, more stable loss landscape (before and after training), faster convergence, and stronger robustness, which we will show in Sect. 3. Moreover, one can hardly notice the additional cost when adopting 2–4-order methods, such as midpoint and RK-4, which we will analyse at the end of Sect. 2. However, the model requires considerable memory to store middle states if fully following a very high-order design, such as Verner's RK-8(9)th scheme.

The main contributions of this study are listed as follows:

– We propose the Higher Order ResNet (HO-Net) to explore, with a given sub-net design, how high-order

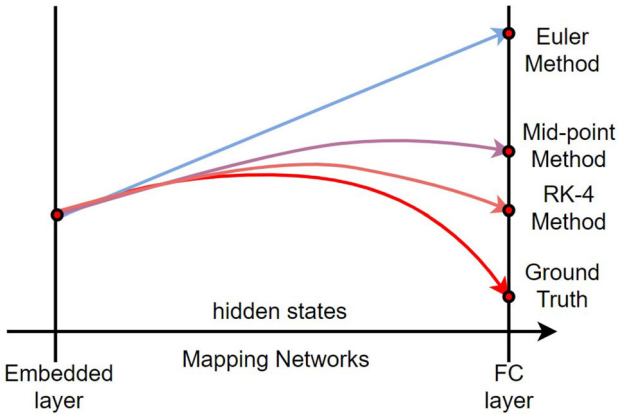

Fig. 3 Various schemes with a simple case: from end-to-end, higher order methods with more steps lead to a closer fitting of the target function but also additional steps and costs

methods help in terms of performance, convergence and robustness.
- We provide sufficient fair comparisons, stacking basic Euler scheme to the same order to enable comparison with high-order methods, therefore, no extra parameters and steps.
- Visualisations of loss landscapes are given for better understanding.
- Corresponding theoretical supports, complexity analysis and sufficient ablation studies.

High-order residual networks

NODEs compute the gradients of a scalar-valued loss with respect to all inputs of any ODE solver, without back propagating through the operations of the solver. Similar to different sub-net designs in discrete networks, the methods of the solver make a difference under the same conditions. Thus, advanced solvers make NODEs better, but incur extra computational costs. Figure 3 shows a simple case for better understanding.

Will advanced numerical schemes make ResNet better in a discrete scheme with the same cost and parameter size? In this study, baseline ResNet (Euler forward scheme), midpoint scheme, RK-4, fixed Verner-8(9), and adaptive Verner's 8(9) have a basic sub-net with [2, 4, 8, 28, 28] layers, respectively. When comparing ResNet with HO-ResNet, we maintained the same depth and width.

For example, a ResNet-Euler-18 has eight (8×2 layers) Euler-style blocks, one embedding layer, and one FC layer, where ResNet-Midpoint-18 and ResNet-RK4-18 have four (4×4 layers) Midpoint-style blocks and two (2×8 layers) RK4-style blocks. The higher order design evidently impacts

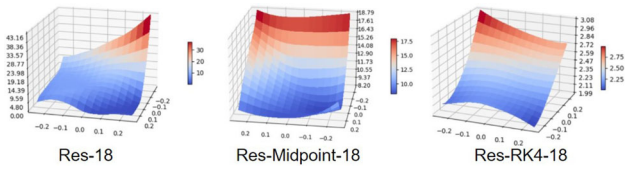

Fig. 4 Loss landscape without training: when processing the same images, ResNet-18 with Euler, midpoint, and RK-4 schemes have ranges of loss ([0, 43.16], [8.20, 18.79], [1.99, 3.08]), respectively. A high-order scheme significantly improves the stability of models for unseen samples; the loss will not shack too much, which leads to the robustness of the learning rate and other properties, which will be discussed in a later section

the network, even without training (see Fig. 4, in relation to visualisations of initial conditions.

A few definitions are necessary before introducing HO-ResNet. The situations discussed below are without the embedding layer and FC layers, which means that the input and output are all features. We use $\mathcal{F}$ to describe any sub-net/component, which could be any component such as a VGG-style two-layer sub-net, three-layer bottle net, ViT block, self-attention layer, or simple feed-forward network.

However, we used a VGG-style block in the experiments because the question 'what form shall $\mathcal{F}$ be?' was not included in our discussion. Our focus is on propagating features in various schemes with a given $\mathcal{F}$. We refer to the input/output features as $\mathcal{N}^{\text{in}}$ and $\mathcal{N}^{\text{out}}$, denote the middle states of the input as $\mathcal{N}^{\text{mid}}$ if there are two layers, $\mathbf{k}_i$ is used to indicate the hidden states of the input after a particular ith $\mathcal{F}$.

Euler forward scheme

Let a network be ResNet-6 which means that two ResBlocks are stacked, and we can write the transforms between the embedding layer and FC layer as follows:

$$\mathcal{N}^{\text{Mid}} = \mathcal{F}\left(\mathcal{N}^{\text{in}}, \left\{\boldsymbol{W}^i\right\}\right), \tag{5}$$

$$\mathcal{N}^{\text{Out}} = \mathcal{F}\left(\mathcal{N}^{\text{Mid}} + \mathcal{N}^{\text{in}}, \left\{\boldsymbol{W}^i\right\}\right) + \mathcal{N}^{\text{Mid}}. \tag{6}$$

$\mathcal{N}^{\text{Mid}}$ denotes the output of the first $\mathcal{F}$; one may find that the behaviour of two stacked ResBlocks is very similar to a midpoint ResBlock, as shown in Fig. 5.

Midpoint scheme

The Euler method updates it with the easiest one-step method, which is $\Delta i \mathcal{F}\left(i, \mathcal{N}^i\right)$, while the midpoint method updates it in a higher order manner:

$$\mathcal{N}^{i+\Delta i} = \mathcal{N}^i + \Delta i \mathcal{F}\left(i + \frac{\Delta i}{2}, \mathcal{N}^i + \frac{\Delta i}{2}\mathcal{N}\left(i, \mathcal{N}^i\right)\right). \tag{7}$$

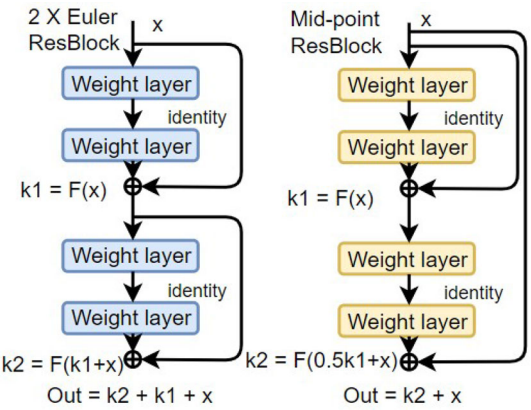

Fig. 5 Midpoint ResBlock: comparing two stacked ResBlock-Euler with a single design that follows the midpoint method

If we compare ResBlock-Euler and ResBlock-Midpoint, in terms of network design, we will have different stacking, as shown in Fig. 5. Rather than a design with two layers, we now have a block design consisting of four layers. Nevertheless, we stack the Euler scheme to the same layers/orders for a fair comparison.

There were two noteworthy differences: (1) when adding the shortcut to obtain the mid-state, the midpoint design compressed the output of $\mathcal{F}$ by half. (2) The second shortcut is from an earlier location, which is directly from the input. If we write the midpoint ResBlock design as

$$\mathcal{N}^{\text{Mid}} = \mathcal{F}\left(\mathcal{N}^{\text{in}}, \left\{\boldsymbol{W}^i\right\}\right), \tag{8}$$

$$\mathcal{N}^{\text{Out}} = \mathcal{F}\left(\frac{1}{2}\mathcal{N}^{\text{Mid}} + \mathcal{N}^{\text{in}}, \left\{\boldsymbol{W}^i\right\}\right) + \mathcal{N}^{\text{in}}. \tag{9}$$

One finds that it is very similar to the stacked ResBlock-Euler. Therefore, we assume that stacked ResNet will be easily improved with higher order methods because numerical problems have solid theoretical backups, which could be more accurate than the lower order ones.

Fourth-order Runge–Kutta Scheme

Exploring this further, with a fourth-order design, Zhu et al. [33] attempted an RK style design. In this study, we use it as a guideline for stacking rather than a certain RK-Net. The Runge–Kutta method (RK4) updates it in a 4-step manner, and our implemented ResBlock-RK4 can be written as follows:

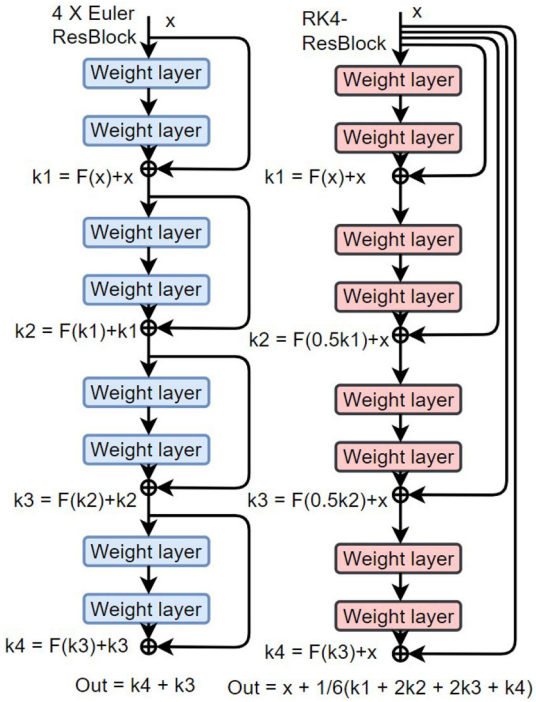

Fig. 6 RK4-ResBlock: comparing four stacked ResBlock-Euler with a single design which follows the 4th-order Runge–Kutta method

$$\begin{aligned}
\mathbf{k}_1 &= \mathcal{F}\left(\mathcal{N}^{\text{in}}\right), \\
\mathbf{k}_2 &= \mathcal{F}\left(\frac{1}{2}\mathbf{k}_1 + \mathcal{N}^{\text{in}}\right), \\
\mathbf{k}_3 &= \mathcal{F}\left(\frac{1}{2}\mathbf{k}_2 + \mathcal{N}^{\text{in}}\right), \\
\mathbf{k}_4 &= \mathcal{F}\left(\mathbf{k}_3 + \mathcal{N}^{\text{in}}\right),
\end{aligned} \tag{10}$$

In addition, the network output is

$$\mathcal{N}^{\text{Out}} = \mathcal{N}^{\text{in}} + \frac{1}{6}\left(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4\right). \tag{11}$$

Implementing a single ResBlock-RK4 requires eight layers, and thus we compare four stacked ResBlock-Euler with it. The stacked ResBlock-Euler can be written in the same way for easy comparison:

$$\begin{aligned}
\mathbf{k}_1 &= \mathcal{F}\left(\mathcal{N}^{\text{in}}\right), \\
\mathbf{k}_2 &= \mathcal{F}\left(\mathbf{k}_1 + \mathcal{N}^{\text{in}}\right), \\
\mathbf{k}_3 &= \mathcal{F}\left(\mathbf{k}_2 + \mathbf{k}_1\right), \\
\mathbf{k}_4 &= \mathcal{F}\left(\mathbf{k}_3 + \mathbf{k}_2\right),
\end{aligned} \tag{12}$$

In addition, the network output is

$$\mathcal{N}^{\text{Out}} = \mathbf{k}_4 + \mathbf{k}_3. \tag{13}$$

$$
\begin{aligned}
\mathbf{k}_1 &= \mathcal{F}\left(\mathcal{N}^{in}\right), \\
\mathbf{k}_2 &= \mathcal{F}\left(\frac{h}{12}\mathbf{k}_1 + \mathcal{N}^{in}\right), \\
\mathbf{k}_3 &= \mathcal{F}\left(\frac{h}{27}\left(\mathbf{k}_1 + 2\mathbf{k}_2\right) + \mathcal{N}^{in}\right), \\
\mathbf{k}_4 &= \mathcal{F}\left(\frac{h}{24}\left(\mathbf{k}_1 + 3\mathbf{k}_3\right) + \mathcal{N}^{in}\right), \\
\mathbf{k}_5 &= \mathcal{F}\left(\frac{h}{375}\left(\left(4 + 94\sqrt{6}\right)\mathbf{k}_1 - \left(282 + 252\sqrt{6}\right)\mathbf{k}_3 + \left(328 + 206\sqrt{6}\right)\mathbf{k}_4\right) + \mathcal{N}^{in}\right), \\
\mathbf{k}_6 &= \mathcal{F}\left(h\left(\frac{9 - \sqrt{6}}{150}\mathbf{k}_1 + \frac{312 + 32\sqrt{6}}{1425}\mathbf{k}_4 + \frac{69 + 29\sqrt{6}}{570}\mathbf{k}_5\right) + \mathcal{N}^{in}\right), \\
\mathbf{k}_7 &= \mathcal{F}\left(h\left(\frac{927 - 347\sqrt{6}}{1250}\mathbf{k}_1 + \frac{7328\sqrt{6} - 16248}{9375}\mathbf{k}_4 + \frac{179\sqrt{6} - 489}{3750}\mathbf{k}_5 + \frac{14268 - 5798\sqrt{6}}{9375}\mathbf{k}_6\right) + \mathcal{N}^{in}\right), \\
\mathbf{k}_8 &= \mathcal{F}\left(\frac{h}{54}\left(4\mathbf{k}_1 + \left(16 - \sqrt{6}\right)\mathbf{k}_6 + \left(16 + \sqrt{6}\right)\mathbf{k}_7\right) + \mathcal{N}^{in}\right), \\
\mathbf{k}_9 &= \mathcal{F}\left(\frac{h}{512}\left(38\mathbf{k}_1 + \left(118 - 23\sqrt{6}\right)\mathbf{k}_6 + \left(118 + 23\sqrt{6}\right)\mathbf{k}_7 - 18\mathbf{k}_8\right) + \mathcal{N}^{in}\right), \\
\mathbf{k}_{10} &= \mathcal{F}\left(h\left(\frac{11}{144}\mathbf{k}_1 + \frac{266 - \sqrt{6}}{864}\mathbf{k}_6 + \frac{266 + \sqrt{6}}{864}\mathbf{k}_7 - \frac{1}{16}\mathbf{k}_8 - \frac{8}{27}\mathbf{k}_9\right) + \mathcal{N}^{in}\right), \\
\mathbf{k}_{11} &= \mathcal{F}\left(h\left(\frac{5034 - 271\sqrt{6}}{61440}\mathbf{k}_1 + \frac{7859 - 1626\sqrt{6}}{10240}\mathbf{k}_7 + \frac{813\sqrt{6} - 2232}{20480}\mathbf{k}_8 + \frac{271\sqrt{6} - 594}{960}\mathbf{k}_9 + \frac{657 - 813\sqrt{6}}{5120}\mathbf{k}_{10}\right) + \mathcal{N}^{in}\right), \\
\mathbf{k}_{12} &= \mathcal{F}\left(h\left(-8.14164\mathbf{k}_1 - 574.436\mathbf{k}_6 + 847.88\mathbf{k}_7 + 113.719\mathbf{k}_8 + 626.94\mathbf{k}_9 + 605.73\mathbf{k}_{10} - 328.69\mathbf{k}_{11}\right) + \mathcal{N}^{in}\right), \\
\mathbf{k}_{13} &= \mathcal{F}\left(h\left(0.0878\mathbf{k}_1 + 0.69337\mathbf{k}_6 - 1.9\mathbf{k}_7 + 0.23\mathbf{k}_8 - 0.69\mathbf{k}_9 - 0.077\mathbf{k}_{10} + 2.49\mathbf{k}_{11} + 0.0018\mathbf{k}_{12}\right) + \mathcal{N}^{in}\right), \\
\mathbf{k}_{14} &= \mathcal{F}\left(h\left(-0.1\mathbf{k}_1 + 5.575\mathbf{k}_6 + 7.486\mathbf{k}_7 - 6.23\mathbf{k}_8 + 2.27\mathbf{k}_9 - 4.89\mathbf{k}_{10} - 4.86\mathbf{k}_{11} - 0.0235\mathbf{k}_{12} + 1.78\mathbf{k}_{13}\right) + \mathcal{N}^{in}\right),
\end{aligned}
\tag{14}
$$

And the network output is:

$$
\mathcal{N}^{Out} = \mathcal{N}^{in} + h\left(0.06\mathbf{k}_1 - 0.19\mathbf{k}_8 + 0.72\mathbf{k}_9 - 0.72\mathbf{k}_{10} + 0.75\mathbf{k}_{11} + 0.0004\mathbf{k}_{12} + 0.34\mathbf{k}_{13} + 0.032\mathbf{k}_{14}\right). \tag{15}
$$

Fig. 7 Verner's RK-8(9)th-order scheme: the full method has 16 steps and extra processes for error estimation and scale factor adjustment. In our implementation, Verner's 8(9)th-order block has 28 layers instead of $(8 \times 2 \times 2)$ layers for two reasons: (1) $\mathbf{k}_{15}$ and $\mathbf{k}_{16}$ do not impact the block output $\mathcal{N}^{\text{Out}}$ and are used to calculate the adaptive factor h. (2) Moreover, one can update h via deep-learning frameworks directly by setting it as a learnable tensor value. We compared both fixed h and adaptive h, where learnable h is the only extra parameter in HO-ResNet, usually no more than 50, making a rare difference because most models have millions of parameters. h equals one in a fixed Verner block; for the adaptive version, one could use a shard h for all hidden states from $\mathbf{k}_1$ to $\mathbf{k}_{14}$ or every state could have a different h

According to the summary above, we can easily understand the difference between 4th fourth-order RK-ResNet and the four stacked ResBlocks. In Fig. 6, we find that RK4-ResBlock is somehow similar to the DenseNet design [14]. However, DenseNet will shortcut every $\mathbf{k}$ to layers after it, which the proposed RK4-ResNet does not. Moreover, HO-ResNet scales the particular output of $\mathbf{k}$ with mathematical support.

Midpoint-ResBlock and RK4-ResBlock only shortcut $\mathbf{k}$ to the block output; identity mappings inside the block are all from the input x of the block. However, some more high-order methods will start to map $\mathbf{k}$ inside the block to other states. However, the coefficient is precisely calculated, rather than simply added.

Verner's 8(9)th-order Runge–Kutta Scheme

Of course, one could keep seeking a more advanced scheme to construct ResBlock in a higher order manner, that contains more layers than just two or three. There are many other versions to guide the network design. Many of these methods are adaptive methods that use a scaling factor. This means that the compression rate of $\mathbf{k}_i$ can be adaptive depending on the error returned.

In this study, we implemented the midpoint and RK4 methods directly, as they are fixed methods. However, we did not fully implement the adaptive Verner's RK-8 [29], as shown in the caption of Fig. 7 for reasons.

The full Verner's RK-8(9) ResBlock design shall contain 16 $\mathcal{F}$ which means 32 layers. We use $\mathbf{k}_i$ to represent the

output of each block, where i is an integer in the range [1–16]. Unfortunately, no flowchart is provided because of the complicated connections between the hidden states.

The final output of the Verner-8(9) ResBlock depends only on the shortcut from the start and $\mathbf{k}_i$, where i in [1, 8, 9, 10, 11, 12, 13, 14]. The scale factor h should be adjusted depending on $\mathbf{k}_i$, where i in [1, 8, 9, 10, 11, 12, 13, 14, 15, 16].

Thereby, one can implement a fixed version with an out-of-scale factor adjustment using 14 $\mathcal{F}$. However, $\mathbf{k}_{15}$ and $\mathbf{k}_{16}$ are required to update h if one wants to fully follow Verner's method, in which we update h by directly setting it to be learnable, to compare fixed and adaptive versions with the same computational cost.

Complexity analysis

We compare the situation in which the same size, number, and design of the layers are used. The space complexity can be described by the

$$\text{Space} \sim O\left(\sum_{i=1}^{D} K_i^2 \cdot C_{l-1} \cdot C_i + \sum_{i=1}^{D} M^2 \cdot C_i\right), \qquad (16)$$

where K is the kernel size, C refers to the number of channels, and D denotes the depth of the network. Thus, the complexity of various ResNet designs is the same, with no extra parameters, theoretically. However, in practice, extra space is required to maintain the output inside the high-order block because the block output depends on more states at particular layers rather than two fixed positions, compared with the baseline ResNet.

Specifically, baseline ResNet only maintains a shortcut from the block input, as does the midpoint block. For the Runge–Kutta 4 block, we need three more shortcuts for the block output. However, this could be optimised to one more shortcut.

For Verner's RK-8 block, this situation becomes rather complicated. From the state $\mathbf{k}_5$, every state starts to depend on other previous states, besides the input from the last state; we shall maintain 16 more shortcuts for the full version or 14 more shortcuts for our version. Furthermore, the space requirement is extremely high while forwarding, which is a crucial shortage; sufficient space should be provided when using such a design. Therefore, it is not recommended to use Verner's block; applying the adaptive step from Verner's block to a fourth-order scheme would be a better solution.

Our design does not affect the first term, $\sum_{i=1}^{D} K_i^2 \cdot C_{l-1} \cdot C_i$, but only the second term, $\sum_{i=1}^{D} M^2 \cdot C_i$. Baseline ResNet maintains only one feature map for shortcut as $\sum_{i=1}^{D} M^2 \cdot (C_i + 1)$, similar to the midpoint block. RK4-Block and Fixed-RK8-Block will take $\sum_{i=1}^{D} M^2 \cdot (C_i + 2)$ and $\sum_{i=1}^{D} M^2 \cdot (C_i + 15)$, respectively. We could say that

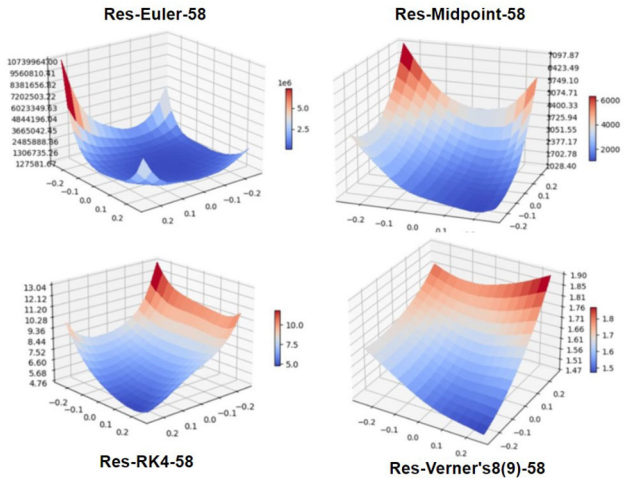

Fig. 8 Visualisation of untrained ResNet-58 with various schemes: the range of loss exploded with a low-order scheme. Euler, Midpoint, RK-4, and Verner's 8(9) have loss ranges ([1e5, 1e7], [1028, 7097], [4.76, 13.04], [1.47, 1.90]). The stability is significantly improved with higher order, which means that gradient problems, such as vanishing or exploding, are less likely to occur

the extra space requirement could be ignored generally for second–fourth-order schemes but is slightly too large for full Verner's RK-8(9).

We may describe the time complexity by

$$\text{Time} \sim O\left(\sum_{i=1}^{D} M_i^2 \cdot K_i^2 \cdot C_{i-1} \cdot C_i\right). \qquad (17)$$

According to the explanation above, we know that HO-ResNet will frequently scale the shortcuts, unavoidably creating extra multi/add processes. Compared with the baseline ResNet, the midpoint block has one additional multiplication operation for every four layers. Therefore, we shall add $\frac{1}{4}\sum_{i=1}^{D} M^2 \cdot C_i$.

The RK-4 block requires six more multiplication operations and three additional operations every eight layers. Therefore, we shall add $\frac{9}{8}\sum_{i=1}^{D} M^2 \cdot C_i$. It is difficult to observe the complexity changes among Euler, Midpoint, and RK4. However, Verner's RK-8 requires a considerable number of storage spaces because of the highly correlated hidden states and extensive additional multi/add scaling. The time consumption is approximately 1.15x–1.2x while the previous three remain the same.

Table 2 shows the multiple relationships of memory and running time in practice, taking the ResNet as the baseline. There is no room to optimise the huge cost of Verner's 8(9), but one could remove the blending of RK-4 and output the $\mathcal{N}^{in} + \mathbf{k}_4$ only instead of $\mathcal{N}^{\text{in}} + \frac{1}{6}(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4)$. RK-4 will have the same memory requirement as baseline Euler by doing so with no evident performance drop.

Table 2 Multiple relationships of memory requirements and running time: the second- to fourth-order scheme has rare extra operations, and thus does not reflect on the running time

Schemes	Euler	Midpoint	RK-4	Verner-8(9)
Memory	1x	1x	1.5x (1x)	7x
Time	1x	1x	1x	1.15x–1.2x

However, more memory is required because RK-4 blends all hidden states for the output. Verner's 8(9) requires huge amounts of memory because it blends not only hidden states for output but also previous states for hidden states in a later stage

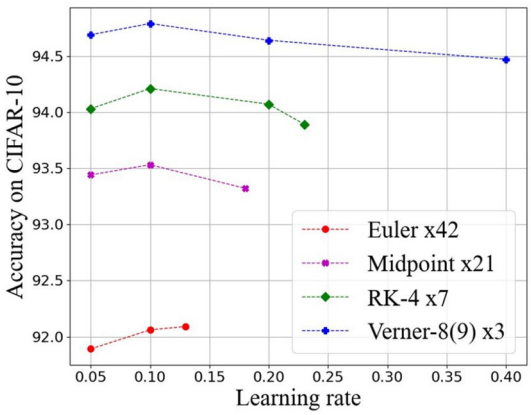

Fig. 9 ResNet-86 with various schemes against different initial learning rates: under the same condition, ResNet with higher order could be trained with a noticeable larger learning rate. Baseline Euler scheme diverged with learning rate 0.13 and all lr larger than it, while midpoint could converge with lr close to 0.2, RK-4 could converge at lr 0.23 and Verner's-8(9) can converge at lr 0.4 (or even above)

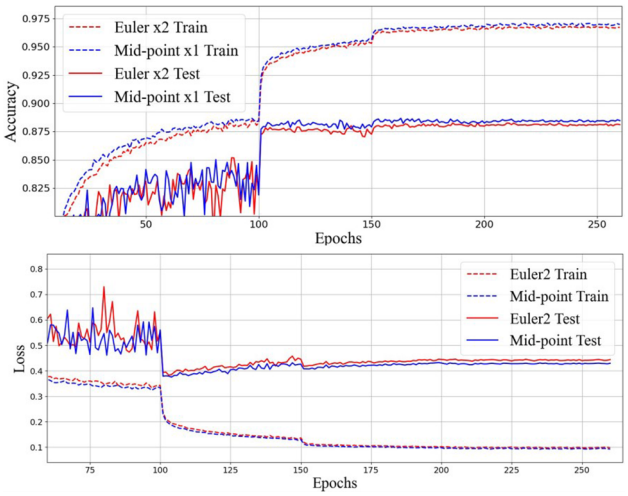

Fig. 10 One midpoint block vs. two Euler blocks: the second-order scheme shows noticeable improvements in both accuracy and loss during training and testing

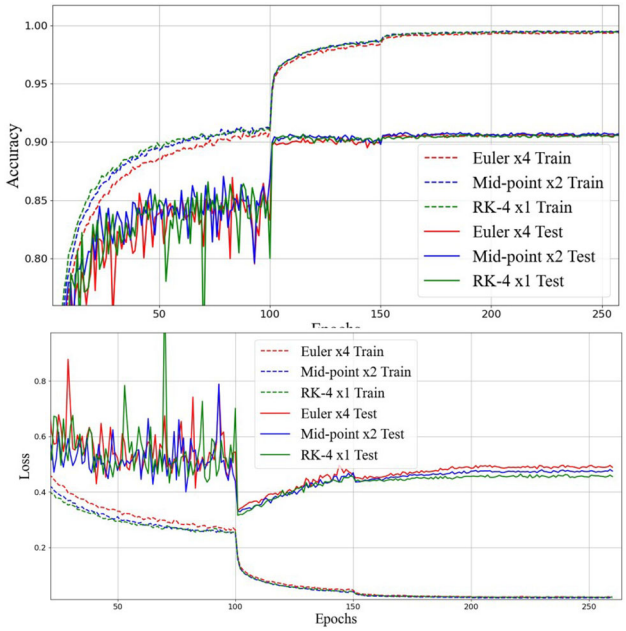

Fig. 11 One Rk4 block vs. two Midpoint blocks and four Euler Blocks: the fourth-order RK-4 scheme outperforms the midpoint and baseline Euler as a basic sub-net. Note that the advantages will become increasingly evident for deeper situations

Experiments

Datasets and implementation

In this study, we conducted most studies on the CIFAR–10 datasets [17], which all consist of coloured natural images with 32 × 32 pixels with 50,000 images for training and 10,000 images for testing, which has 10 target classes in total for CIFAR-10 with 6000 images per class.

We directly stacked blocks 64 dim after a conv(3, 64). Stochastic gradient descent [3] under a weight decay of 1e-4 and momentum of 0.9, was used with weight initialisation in [11] and batch norm [15]. The models were trained with a batch size of 128, starting with an initial learning rate of 0.1, step scheduler with [100, 150, 200, 230], factor 0.1, and 260 epochs trained.

Augmentations and tricks

Our focus was on the differences in behaviour for each of the stacking strategies to verify whether the high-order stacking could provide general guidance for neural network design, but not on pushing a certain line of results or seeking a more considerable number. Therefore, we intentionally com-

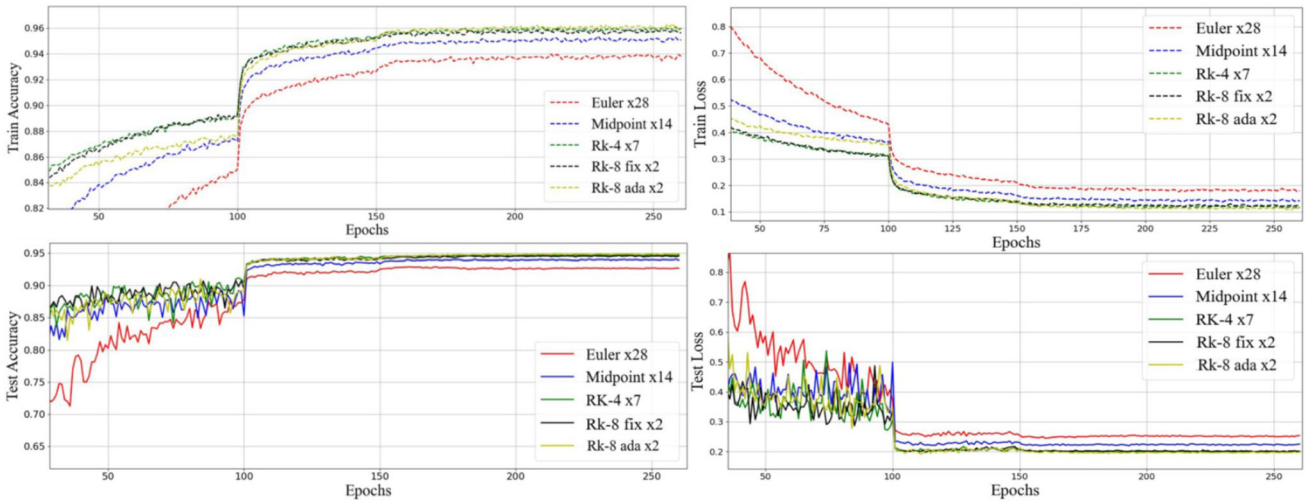

Fig. 12 ResNet-58 with various schemes: training accuracy and loss on top and testing accuracy and loss at the bottom. Note that the difference between schemes is becoming increasingly noticeable compared with observations on shallow versions

pared those stacking strategies using methods that enabled fair comparisons to be drawn.

When comparing a single block design with the corresponding deep baseline ResNet, we used raw data and observed overfitting even under 10 layers. Therefore, we used the autoaugments' CIFAR10 policy [7] to address this question.

Robustness against learning rate

Figure 8 shows a deeper situation with different schemes. Comparing with the case of ResNet-18 shown by Fig. 4, the loss exploded due to the chain rule. Meanwhile, higher order schemes show significant advantages, such as more flatten loss surfaces being provided. Such a phenomenon greatly enhances the robustness of models when different learning rates are given.

The learning rate is one of the primary hyperparameters of deep-learning systems. Slight adjustments in the learning rate could cause vast impacts on the whole system, large learning rates will easily lead to divergence, and small learning rates usually fail to converge well.

Taking ResNet-86 as an example, 84 layers could consist of [42, 21, 7, 3] of [Euler, Midpoint, RK-4, Verner's 8(9)] blocks. Figure 9 shows how higher order schemes are dealing with a larger range of learning rates. ResNet-86 in the 4th/8th-order scheme could be well trained with 2x/4x learning rate at the point where the baseline Euler diverged

Fair comparison of single block

If one compares each block design with equivalent deep baseline ResBlocks, it is found that the improvements are stable but not apparent in the shallow situation. However, the advantages were rapidly becoming evident in more profound cases.

Euler-x2 and Midpoint-x1 Given ResNet-6, we have four layers beside the input and output layers. We performed experiments on CIFAR-10 raw data with rare tricks; as shown in Fig. 10, loss and accuracy are stably improved.

Euler-x4, Midpoint-x2 and RK4-x1 If ResNet-10 was given, one could stack four baseline ResBlocks, which is the red line in Fig. 11, while Midpoint-2 and single RK-4 block benefit from more accurate updating methods.

Deeper situation with augmentation

We noticed overfitting when using ResNet shallower than 10. Therefore, we added auto-augmentation [7] in this comparison. Without the input and output layers, 56 layers could be divided into Baseline ResBlock X28, Midpoint x14, RK4-Block x7, and fixed-RK8 X2.

As shown in Fig. 12, baseline ResNet, represented by the red line, is clearly improved by high-order stacking methods, using the same setting and parameters. We noticed that the fixed-RK8 method has the potential to improve. In the early stages, it was above RK-4. In Verner's RK-8(9) [29] design, one should also maintain a scale factor h and $error$ to adjust the step side. Adaptive-step Verner's RK-8(9) converges at a slightly slower rate but outperforms all other schemes overall.

Table 3 shows more details against different depths; all schemes perform similarly in shallow cases but show more evident gaps while getting deeper. Moreover, there are numerous benefits to be gained from high order, besides the performance, such as loss landscape and robustness against hyperparameters, Fig. 13 and Table 4 show the improvement in the speed of converging.

Table 3 Classification accuracy on CIFAR-10: various schemes with different depth are compared

Models	Layers	Params	Cifar-10
Euler X4	10	0.3M	91.72
Midpoint X2	10	0.3M	91.97
RK4 X1	10	0.3M	91.51
Euler X8	18	0.59M	92.68
Midpoint X4	18	0.59M	92.95
RK4 X2	18	0.59M	92.98
Euler X14	30	1.03M	93.02
Midpoint X7	30	1.03M	93.17
RK8 X1 Fix	30	1.03M	93.39
RK4 X3	26	0.89M	93.41
RK8 X1 Adaptive	30	1.03M+1	94.17
Euler X28	58	2.07M	92.81
Midpoint X14	58	2.07M	94.08
RK4 X7	58	2.07M	94.79
RK8 X2 Fix	58	2.07M	94.91
RK8 X2 Adaptive	58	2.07M+2	95.06
Euler X42	86	3.01M	92.73
Midpoint X21	86	3.01M	93.87
RK4 X10	82	2.87M	94.83
RK8 X3 Fix	86	3.01M	95.02
RK8 X3 Adaptive	86	3.01M+3	95.15

All schemes perform similar when depth smaller than 18. After that, higher order schemes are showing more and more apparent advantages in terms of classification accuracy

According to Fig. 1 and Table 3, most schemes have difficulties achieving better performance as the depth increases. Therefore, we added warm-up and adaptive steps in training, and more epochs were trained to study situations over 100 layers and a rather aggressive case with 1000 layers. Table 5 shows comparisons under deeper situations. Especially the case with 1000 layers shows a noticeable drop compared with depth 194 within limited epochs. However, ResNets in advanced schemes performed better and much stabler with fewer drops, which support previous observations.

Scaling factors

The most effective DNNs have a relatively light stacking design between blocks and rarely feature scaling factors on block outputs. Following the Midpoint and RK4 scheme, the output should be compressed with a scaling factor of 0.5, and RK4 will have 1/6 and 1/3 factors when averaging the output inside the block.

In RK-8 [29], a constraint [0.125, 4] was given while scaling the output from a particular layer. Although we did not further explore the adaptive stacking design, we tried a simple test on how various scaling factors impact the same DNNs.

Taking ResNet-58 as an example, which has two fixed Verner's RK-8 blocks and two extra parameters, see Table 4. Although the full Verner's RK-8 constrains the factor inside [0.125, 4], we did not follow the constraint when applying the adaptive step for practical reasons. However, the results still show a noticeable improvement, as expected.

Ablation study on various sub-nets

Most previous works focussed on the specific design of the sub-network; however, we consider the certain sub-network abstractly as a function. In the previous section, we present stackings under various depths with the VGG-style block only. Several famous sub-nets are studied in this sub-section under common depths around 50 and 100 to demonstrate our strategy's effectiveness and generalisation ability.

Various sub-nets show more or less performance due to different designs; however, our strategy works well on them as shown in Table 6. Not only on VGG-style block without change in dimension, but also on BottleNeck, SE block and CoT.

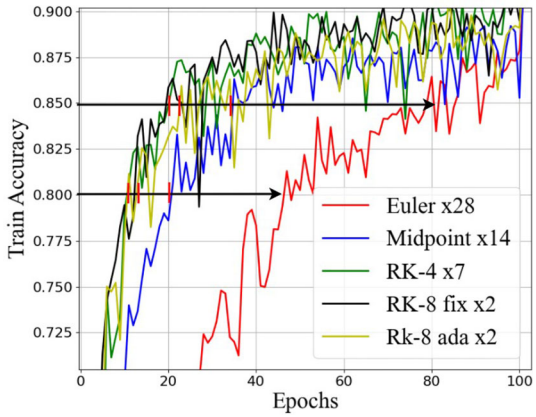

Fig. 13 Running time comparison approaching 0.8, and 0.85 train accuracy (ResNet-58): high-order schemes have evident advantages in terms of the speed of convergence. The second-order midpoint takes half of the time to reach 0.8, and 0.85, compared with Euler, while the 4th- and 8(9)th- order methods take 20–30 percent of the time. See details in Table 4

Table 4 Ratio of time cost (ResNet-58): note that Verner's RK-8 costs a 1.15 increase in time compared to the others; this situation leads RK-4 to be deemed to achieve the best performance in terms of time

Schemes	Euler	Midpoint	RK-4	Fix RK-8	Ada RK-8
To 0.8	1x	2.27x	4.43x	3.31x	3.84x
To 0.85	1x	2.28x	3.92x	3.53x	3.03x
Accuracy	92.71	94.08	94.79	94.72	95.06
Extra para	–	–	–	–	2

Adaptive Verner's RK-8 appears to struggle in the mid-stage but does converge to achieve the best overall performance in the end, with only two extra parameters needed

Table 5 Depth over 100 layers and a aggressive case with 1000 layers: with some training tricks and more epochs, deep networks could be trained better

Models	Layers	Params	Cifar-10
Euler X72	146	5.40M	94.76
Midpoint X36	146	5.40M	95.17
RK4 X18	146	5.40M	95.42
RK8 X5 Fix	142	5.26M	95.57
RK8 X5 Adaptive	142	5.26M+5	95.71
Euler X96	194	7.18M	94.93
Midpoint X48	194	7.18M	95.28
RK4 X24	194	7.18M	95.79
RK8 X7 Fix	198	7.32M	95.97
RK8 X7 Adaptive	198	7.32M+7	96.06
Euler X500	1002	36.89M	94.53
Midpoint X250	1002	36.89M	94.87
RK4 X125	1002	36.89M	95.59
RK8 X28 Fix	982	36.16M	95.82
RK8 X28 Adaptive	982	36.16M+28	95.91

The phenomena we observed before still exist in these situations

Table 6 Stacking various sub-networks with different schemes under common depths (around 56 and 110): including VGG-style block [25], BottleNeck block [12], Squeeze-and-Excitation [13] version VGG and BottleNeck, Mobile-Conv block searched by NAS [27] and Contextual Transformer Block [19]

Depth 56 ± 2	Euler	Midpoint	RK-4	Fix RK-8	Ada RK-8
VGG	92.81	94.08	94.79	94.91	95.06
BottleNeck	93.32	92.78	94.65	94.83	94.92
SE-VGG	93.51	94.36	94.92	95.14	95.31
SE-BottleNeck	93.65	93.97	94.85	95.03	95.16
MBConv	94.54	95.02	95.21	95.87	95.93
CoT	95.02	95.34	95.65	95.91	96.12
Depth 110 ± 4	Euler	Midpoint	RK-4	Fix RK-8	Ada RK-8
VGG	92.71	93.65	94.81	95.04	95.15
BottleNeck	93.21	92.64	94.51	94.93	95.21
SE-VGG	93.92	94.21	94.97	95.19	95.42
SE-BottleNeck	93.45	93.87	94.74	95.09	95.31
MBConv	94.72	95.24	95.47	95.94	96.07
CoT	95.23	95.52	95.79	96.13	96.32

The overall situations with various sub-networks are as expected

Table 7 ResNet around depth 56 and 110 with various schemes on Cifar-100 and ImageNet: the phenomena we observed before still exist on more challenging datasets

Cifar-100	Euler	Midpoint	RK-4	Fix RK-8	Ada RK-8
Depth 56 ± 2	70.14	70.41	70.83	71.25	71.42
Depth 110 ± 4	71.08	71.43	71.81	72.05	72.24
ImageNet	92.71	94.08	94.79	94.72	95.06
Depth 56 ± 2	76.23	76.31	76.47	76.52	76.63
Depth 110 ± 4	76.74	76.82	76.95	77.03	77.12

More challenging datasets

Besides those advantages on several aspects, one may wonder about the generalisation on more challenging datasets. Therefore, we also verify our strategy on Cifar-100 and IMAGENET with common depths around 56 and 110. Cifar-100 contains more specific labels than Cifar-10, which is evidently harder to train. Moreover, the ImageNet contains 1.28 million training images that consist of 1000 classes. We evaluated models on the validation set with 50 k images and obtained final results on the test set containing 100 k images. Table 7 shows that results are as expected.

Discussing

Stacking strategy is vital In the past few years, some network designs have been regarded as ODE schemes such as ResNet and PolyNet [12,21]; some networks are designed under the guidance of ODE schemes such as LM-ResNet and RK-Net

[22,33]. However, most of them focus more on the sub-net design, while the stacking strategy is also critical. Block-to-block is similar to layer-to-layer, especially when some networks will even stack 84 blocks in a single stage.

Three principles from high-order schemes (1) One shall apply an adaptive time step; it can easily be plugged in to any network. In fact, several excellent studies have already proposed methods in this manner from other perspectives. Highway networks [26], Fixup [31], SkipInit [8], ReZero [1], and LayerScale [28] are the approaches that are most closely related. (2) One should consider connections over long distances, but not nested ones such as DenseNet [14] to avoid huge memory requirements. (3) Blending from different hidden states works, but it is not as important as the previous two points. Moreover, it requires at least 1x extra memory.

Unavoidable degradation ResNet [12] significantly alleviated the degradation problem that deeper networks shall perform better in theoretical but worse in practice. The shortcut does not provide extra capacity but makes models easier for training, so does HO-ResNet. We implemented ResNet with advanced numerical methods based on the equivalence between ResNet and Euler Forward scheme; the results in Fig. 1 show that the degradation was better handled and further alleviated with more advanced numerical schemes; however, the degradation will still come eventually.

Conclusion

Inspired by several existing studies [2,6,12,22,30], we assume that DNNs are extremely complicated numerical solvers for differential equations. Most designs are relatively low order, while the networks are frequently deeper than 100 or 200. We propose a general higher order stacking strategy, following several numerical methods, which proved that high-order numerical methods could be adopted as general guidance for block stacking and network design. Sufficient experiments show that high-order improvements are stable, robust, and fully explainable in terms of math. We show that with very few changes in stacking, ResNet has shown remarkable durability.

Acknowledgements This work was supported by JST SPRING, Grant number JPMJSP2128.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Bachlechner T, Majumder BP, Mao HH, Cottrell G, McAuley J (2021) Rezero is all you need: fast convergence at large depth. In: de Campos CP, Maathuis MH, Quaeghebeur E (eds.) Proceedings of the thirty-seventh conference on uncertainty in artificial intelligence, UAI 2021, Virtual Event, 27–30 July 2021, Proceedings of Machine Learning Research, vol. 161, pp. 1352–1361. AUAI Press. https://proceedings.mlr.press/v161/bachlechner21a.html
2. Bello I, Fedus W, Du X, Cubuk ED, Srinivas A, Lin TY, Shlens J, Zoph B (2021) Revisiting resnets: improved training and scaling strategies. arXiv preprint arXiv:2103.07579
3. Bottou L (2010) Large-scale machine learning with stochastic gradient descent. In: Lechevallier Y, Saporta G (eds.) 19th international conference on computational statistics, COMPSTAT 2010, Paris, France, August 22–27, 2010—Keynote, Invited and Contributed Papers. Physica-Verlag, pp 177–186. https://doi.org/10.1007/978-3-7908-2604-3_16
4. Burrage K, Lenane I, Lythe G (2007) Numerical methods for second-order stochastic differential equations. SIAM J Sci Comput 29(1):245–264. https://doi.org/10.1137/050646032
5. Butcher JC (1976) On the implementation of implicit Runge-Kutta methods. BIT Numer Math 16(3):237–240
6. Chen RT, Rubanova Y, Bettencourt J, Duvenaud DK (2018) Neural ordinary differential equations. In: Advances in neural information processing systems, pp 6571–6583
7. Cubuk ED, Zoph B, Mané D, Vasudevan V, Le QV (2019) Autoaugment: learning augmentation strategies from data. In: IEEE conference on computer vision and pattern recognition, CVPR 2019, Long Beach, CA, USA, June 16–20, 2019, pp 113–123. Computer Vision Foundation/IEEE (2019). https://doi.org/10.1109/CVPR.2019.00020
8. De S, Smith SL (2020) Batch normalization biases residual blocks towards the identity function in deep networks. In: Larochelle H, Ranzato M, Hadsell R, Balcan M, Lin H (eds.) Advances in neural information processing systems 33: annual conference on neural information processing systems 2020, NeurIPS 2020, December 6–12, 2020, virtual
9. Ding X, Zhang X, Ma N, Han J, Ding G, Sun J (2021) Repvgg: making vgg-style convnets great again. In: IEEE conference on computer vision and pattern recognition, CVPR 2021, virtual, June 19–25, 2021, pp 13733–13742. Computer Vision Foundation/IEEE
10. Gomez AN, Ren M, Urtasun R, Grosse RB (2017) The reversible residual network: backpropagation without storing activations. In: Guyon I, von Luxburg U, Bengio S, Wallach HM, Fergus R, Vishwanathan SVN, Garnett R (eds) Advances in neural information processing systems 30: annual conference on neural information processing systems 2017, December 4–9, 2017, Long Beach, CA, USA, pp 2214–2224
11. He K, Zhang X, Ren S, Sun J (2015) Delving deep into rectifiers: surpassing human-level performance on imagenet classification. In: 2015 IEEE international conference on computer vision, ICCV

2015, Santiago, Chile, December 7–13, 2015, pp 1026–1034. IEEE Computer Society. https://doi.org/10.1109/ICCV.2015.123
12. He K, Zhang X, Ren S, Sun J (2016) Deep residual learning for image recognition. In: 2016 IEEE conference on computer vision and pattern recognition, CVPR 2016, Las Vegas, NV, USA, June 27–30, 2016, pp 770–778. IEEE Computer Society. https://doi.org/10.1109/CVPR.2016.90
13. Hu J, Shen L, Sun G (2018) Squeeze-and-excitation networks. In: 2018 IEEE conference on computer vision and pattern recognition, CVPR 2018, Salt Lake City, UT, USA, June 18–22, 2018, pp 7132–7141. Computer Vision Foundation/IEEE Computer Society.https://doi.org/10.1109/CVPR.2018.00745. http://openaccess.thecvf.com/content_cvpr_2018/html/Hu_Squeeze-and-Excitation_Networks_CVPR_2018_paper.html
14. Huang G, Liu Z, van der Maaten L, Weinberger KQ (2017) Densely connected convolutional networks. In: 2017 IEEE conference on computer vision and pattern recognition, CVPR 2017, Honolulu, HI, USA, July 21–26, 2017, pp 2261–2269. IEEE Computer Society. https://doi.org/10.1109/CVPR.2017.243
15. Ioffe S, Szegedy C (2015) Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: F.R. Bach, D.M. Blei (eds.) Proceedings of the 32nd international conference on machine learning, ICML 2015, Lille, France, 6–11 July 2015, JMLR workshop and conference proceedings, vol. 37, pp 448–456. JMLR.org. http://proceedings.mlr.press/v37/ioffe15.html
16. Jacod J, Protter P et al (1998) Asymptotic error distributions for the Euler method for stochastic differential equations. Ann Probab 26(1):267–307
17. Krizhevsky A, Hinton G, et al (2009) Learning multiple layers of features from tiny images
18. Lee Y, Hwang J, Lee S, Bae Y, Park J (2019) An energy and gpu-computation efficient backbone network for real-time object detection. In: IEEE conference on computer vision and pattern recognition workshops, CVPR Workshops 2019, Long Beach, CA, USA, June 16–20, 2019, pp. 752–760. Computer Vision Foundation/IEEE. https://doi.org/10.1109/CVPRW.2019.00103
19. Li Y, Yao T, Pan Y, Mei T (2021) Contextual transformer networks for visual recognition. CoRR arxiv:2107.12292
20. Liao Q, Poggio TA (2016) Bridging the gaps between residual learning, recurrent neural networks and visual cortex. CoRR arXiv:1604.03640
21. Liu W, Rabinovich A, Berg AC (2015) Parsenet: looking wider to see better. CoRR arXiv:1506.04579
22. Lu Y, Zhong A, Li Q, Dong B (2018) Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations. In: Dy JG, Krause A (eds) Proceedings of the 35th international conference on machine learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10–15, 2018, Proceedings of machine learning research, vol. 80, pp 3282–3291. PMLR. http://proceedings.mlr.press/v80/lu18d.html
23. Ruthotto L, Haber E (2020) Deep neural networks motivated by partial differential equations. J Math Imaging Vis 62(3):352–364. https://doi.org/10.1007/s10851-019-00903-1
24. Shampine LF (1986) Some practical Runge-Kutta formulas. Math Comput 46(173):135–150
25. Simonyan K, Zisserman A (2015) Very deep convolutional networks for large-scale image recognition. In: Bengio Y, LeCunY (eds.) 3rd international conference on learning representations, ICLR 2015, San Diego, CA, USA, May 7–9, 2015, conference track proceedings. arXiv:1409.1556
26. Srivastava RK, Greff K, Schmidhuber J (2015) Highway networks. CoRR arXiv:1505.00387
27. Tan M, Le QV (2019) Efficientnet: Rethinking model scaling for convolutional neural networks. In: Chaudhuri K, Salakhutdinov R (eds) Proceedings of the 36th international conference on machine learning, ICML 2019, 9–15 June 2019, Long Beach, California, USA, Proceedings of machine learning research, vol. 97, pp 6105–6114. PMLR. http://proceedings.mlr.press/v97/tan19a.html
28. Touvron H, Cord M, Sablayrolles A, Synnaeve G, Jégou H (2021) Going deeper with image transformers. arXiv preprint arXiv:2103.17239
29. Verner JH (1978) Explicit Runge-Kutta methods with estimates of the local truncation error. SIAM J Numer Anal 15(4):772–790
30. Weinan E (2017) A proposal on machine learning via dynamical systems. Commun Math Stat 5(1):1–11
31. Zhang H, Dauphin YN, Ma T (2019) Fixup initialization: residual learning without normalization. In: 7th international conference on learning representations, ICLR 2019, New Orleans, LA, USA, May 6–9, 2019. OpenReview.net. https://openreview.net/forum?id=H1gsz30cKX
32. Zhang X, Li Z, Loy CC, Lin D (2017) Polynet: a pursuit of structural diversity in very deep networks. In: 2017 IEEE conference on computer vision and pattern recognition, CVPR 2017, Honolulu, HI, USA, July 21–26, 2017, pp 3900–3908. IEEE Computer Society. https://doi.org/10.1109/CVPR.2017.415
33. Zhu M, Fu C (2018) Convolutional neural networks combined with runge-kutta methods. CoRR arXiv:1802.08831