Stabilizing Convolutional Neural Networks with Modified Runge–Kutta Integration

Sidhartha Sankar Pradhan[1*] and Subhendu Sekhar Sahoo[2]

[1*]Department of Artificial Intelligence and Machine Learning,
Siksha 'O' Anusandhan, Bhubaneswar, Odisha, India
[2]National Informatics Centre, Odisha, India
[1]sankar.experiment0@gmail.com
[2]subhendusahoo123@gmail.com

Abstract. Convolutional Neural Networks (CNNs) are commonly employed for image classification, but training stability and generalization are difficult due to discretization errors during feature propagation. In this paper, we provide a Modified Runge-Kutta (MRK4) integration approach as a parameter-free post-pooling stabilizer for typical CNN architectures. Experiments on the CIFAR-10 dataset compare a baseline CNN, a CNN with Batch Normalization (BN), and MRK4-augmented variants across different random seeds. Results show that the proposed CNN-BN-MRK4 model increases mean classification accuracy by up to 3.5% and reduces validation loss by approximately 18% compared to CNN-BN, while keeping comparable inference time. Furthermore, the MRK4-based technique exhibits lower performance variance and higher generalization stability. These results show that higher-order numerical integration can effectively stabilize deep CNN training without introducing new learnable parameters.

Keywords: Convolutional Neural Networks, Modified Runge–Kutta, Batch Normalization, CIFAR-10

1 Introduction

Convolutional Neural Networks (CNNs) have demonstrated state-of-the-art performance in various computer vision applications, such as segmentation, object detection, and image classification [1–3]. To achieve those objectives, CNNs employ convolution, nonlinearity, and pooling as the main tools for hierarchical feature extraction by the authors [5]. However, many CNN designs assume a fixed discrete depth structure, with learnt weights, activations, and pooling to determine the transformation from layer to layer by the authors [6]. One promising way to improve CNNs is by viewing them as a discretization of continuous dynamical systems. Conventionally, these models rely on Euler-like updates to approximate the flow of information across layers. However, recent research suggests using higher-order. Numerical integration methodologies, such as solving an ordinary differential equation (ODE), as said by the authors [7, 8], offer adaptive step sizes, richer feature propagation dynamics, enhanced stability, and improved performance.

Following this, the authors [9] augmented standard CNN architectures by embedding fourth-order Runge-Kutta (RK4)- style integration steps. It explains that RK4 can provide a more accurate approximation of the continuous feature flow, with reduced discretization error, and achieve higher accuracy with faster convergence without incurring excessive processing costs. The concept is further tested against typical CNNs and

B. Singh et al. (eds.), *Proceedings of the International Conference on Advances in Computing Technology and Artificial Intelligence (COMPUTATIA 2026)*, Atlantis Highlights in Intelligent Systems 18,
https://doi.org/10.2991/978-94-6239-713-2_46

variations (e.g., batch normalization and dropout) to quantify accuracy, computational overhead, and training stability by the authors [10].

In continuation, a network created by an ODE solver to model hidden representations as continuous flows is also investigated by the authors [11]. Adaptive solvers can also be applied when depth is treated as a continuous quantity. In this context, several works have explored normalization and robustness in the Neural ODE framework by the authors [12, 13]. To analyze the effect of classification task performance, various continuous flow models and methods, such as batch norm and layer norm, have been investigated by the authors [14]. In Sander et al., the authors examine exactly how ResNets approximate underlying ODEs, quantify error bounds, and demonstrate the conditions under which a ResNet behaves similarly to a neural ODE. This motivates the use of more accurate integrators by the authors [15]. In this regard, the most closely related work on Runge-Kutta Convolutional Neural Networks (RKCNNs) is carried out by M Zhu et al., where forward Euler has been used to reinterpret pre-activation ResNets followed by higher-order Runge- Kutta techniques to replace or enhance certain elements for the improvement in accuracy and efficiency by the authors [16]. Moreover, neural ODEs using model order reduction have also been investigated to reduce the complexity of convolutional neural ODEs while maintaining accuracy by the authors [17]. However, higher-order techniques are typically more costly, necessitating trade-off analysis. In the field of Cellular Neural Networks (a distinct but related notion), the authors have employed RK4-like embedding methods to simulate faster CNNs by the authors [18]. This demonstrates that simulation error can be decreased by using higher-order integration in spatial/temporal CNN-like scenarios.

While there is significant existing work, there are still open issues. Fewer studies have examined the deep embedding of RK4 into standard CNN blocks (particularly with pooling and normalization) and compared across a wider range of datasets, including small/mid-sized ones like CIFAR, Fashion-MNIST, and SVHN. Furthermore, many works in neural ODEs or RKCNNs compare higher-order methods, but frequently in the context of residual continuous flow models. There has been little research into how RK4 integration affects stability and training dynamics across different seeds and normalization settings, and whether the additional computational and memory costs are worth the performance advantages in practice. Furthermore, while studies such as ResNet demonstrate that the choice of solver is important, the interfaces between RK4-based integration and standard architectural components, such as pooling, max pooling. Skip connections, and so on, are not well investigated. Our approach differs from neural ODEs and RK-based residual networks in that it neither learns continuous dynamics nor replaces residual blocks. MRK4 is integrated as a parameter-free post-pooling feature stabilizer into typical CNN pipelines. To address the gap, the contributions of this paper are:

1. We introduce a modified Runge–Kutta (MRK4)-based integration strategy within standard CNN architectures and their regularized variants.
2. Across all experiments on CIFAR-10, we show that CNNs equipped with batch normalization benefit most from MRK4 integration, achieving improved mean accuracy and reduced variance across multiple random seeds.

3. We analyze the interaction between MRK4, batch normalization, and dropout, highlighting conditions under which higher-order integration contributes most effectively to training stability and generalization.

Although MRK4 is not always advantageous, these results demonstrate that it can effectively stabilize CNN training and enhance generalization when combined with batch normalization.

2 Methodology

2.1 Baseline Architectures

In our experiments, three CNN-based baselines for image classification on CIFAR- 10 are used. CIFAR-10 was implemented and trained entirely in a GPU environment; CNN: A typical convolutional network consisting of layers that are fully connected, pooling, ReLU, and convolution. CNN-Batch Normalization (BN): This technique enhances convergence and stabilizes training by adding batch normalization layers to the same architecture after each convolution. CNN-BN-Dropout: A further extension that uses regularization dropout layers to lessen overfitting.

2.2 MRK4-Augmented Convolutional Blocks

We provide MRK4Pooling, a parameter-free layer in our architecture that alters the typical feature flow following pooling. For two reasons, the MRK4 integration step is implemented after the pooling layers rather than the convolution layers. First, pooling operations reduce the computational cost of applying multi-stage RK4 updates by decreasing the spatial resolution of the feature maps. Second, because pooling involves down-sampling, it usually introduces sudden changes in feature activations. Prior to the subsequent convolutional block, MRK4 is applied after pooling to help stabilize feature propagation and ease these transitions. Because of its architecture, MRK4 can function as a lightweight feature stabilizer without appreciably raising computational complexity.

The feature map that is obtained following a MaxPooling operation is denoted by $h_t \in R^{m \times n \times c}$, where each entry represents an activation value in the pooled feature map. This output is sent straight to the following convolutional layer in a traditional CNN:

$$h_{t+1} = h_t.$$

Instead, we update all entries h_t using the classical fourth-order modified Runge–Kutta (MRK4) integration scheme. We interpret pooled feature activations as evolving according to a simple autonomous ordinary differential equation.

$$dx/dt = f(x), \tag{1}$$

Our solution uses the identity dynamics f (x) = x for feature smoothing, rather than learning explicit continuous representations. For each entry of the pooled feature map, the update follows:

$$k1 = h \cdot f(x), \tag{2}$$

$$k_2 = h \cdot f\, x + h/2\, k_1, \tag{3}$$

$$k_3 = h \cdot f\, x + h/2\, k_2, \tag{4}$$

$$k4 = h \cdot f\,(x + h\, k_3), \tag{5}$$

$$x_{out} = x + (h/6)\, k_1 + 2k_2 + 2k_3 + k_4. \tag{6}$$

In our trials, the step size h is set at 0.1. This value was chosen as a compromise between numerical stability and sensitivity to feature updates. A smaller step size allows smoother RK-based feature growth and avoids sudden changes in activation values; extremely tiny values increase computational cost with little discernible performance improvement. Empirically, h = 0.1 yielded stable convergence and consistent accuracy across preliminary trials, whereas larger step sizes caused oscillatory updates in the feature maps. The modified MRK4 approach updates each element using the classical Runge-Kutta formulation by the authors [19]. The MRK4Pooling layer produces an updated feature map with the same spatial dimensions as the input, which is then passed to the next convolutional block.

2.3 Dataset and Preprocessing

The CIFAR-10 dataset, which comprises 10,000 test and 50,000 training images in 10 classes, is what we used. The images were adjusted to have a unit variance and zero mean. No extra data augmentation was used to isolate the impact of MRK4.

2.4 Training Setup

The Adam optimizer was used to train all models with a batch size of 64, 100 epochs, and a learning rate of $1 \times 10{-}3$. Five different seeds were used in the trials to guarantee repeatability. We present several performance metrics, such as test accuracy, loss, macro F1-score, number of trainable parameters, and training time per epoch, inference time per sample, overfitting gap (difference between final training and validation accuracy), and maximum overfitting gap (largest difference during training).

Experiments were repeated using many random seeds to evaluate resilience. We draw attention to the CNN-BN-MRK4 configuration's consistent gains for seed 5 (later, we checked for seed 7).

2.5 Evaluation Metrics

The following metrics were evaluated for each model for the data set CIFAR-10: Accuracy: percentage of correctly classified test samples. Loss: reduction in cross-entropy on the test set. F1-score: harmonic mean of precision and recall. Overfitting Gap: the difference between training and test accuracy. Maximum Gap: the maximum difference between training and validation accuracy that was noted during training. Training Time: Average wall-clock time per epoch. Inference Time: average time per test sample.

To account for volatility and stability, the mean and standard deviation were calculated across different seeds.

3 Results and Analysis

3.1 Theoretical Motivation

The concept of using a higher-order numerical approach to simulate feature evolution is what drives the proposed modified RK4-enhanced CNN-BN. A multi- stage update that approximates feature propagation using the fourth-order modified Runge–Kutta scheme is introduced by the modified RK4-enhanced architecture, in contrast to the traditional CNN with CNN-BN, which updates feature maps layer by layer. This improves generalization and decreases error accumulation by enabling a steadier and more precise evolution of activations by the authors 10, 14].

3.2 Quantitative Results

Classification accuracy across datasets and training and inference periods across CIFAR-10 is summarized in Table 2. The modified RK4-enhanced model consistently outperformed the baseline CNN-BN in terms of accuracy, with increases ranging from (1–4%). Furthermore, the modified RK4-based method exhibited more consistent performance, with lower standard deviations across random seeds. In a similar vein, the modified RK4-based approach reduces the validation loss in each CNN-BN-MRK4 comparison relative to CNN-BN by at least 18%. CNN- BN-MRK4 had a slight training overhead, requiring at least 6% more time per epoch than CNN-BN. However, the differences in inference times were insignificant, making the approach suitable for practical implementation.

3.3 Qualitative Results

Fig. 1(a) illustrates how CNN-BN-MRK4 continuously outperforms Fig. 1(b), CNN-BN, in terms of test accuracy, exhibiting faster convergence and fewer fluctuations. Significantly, CNN-BN-MRK4 exhibits greater robustness to overfitting, as evidenced by a smaller generalization gap between training and test accuracy. Compared with CNN-BN blocks, our results suggest that integrating MRK4 improves feature learning and stabilizes the optimization process. Similarly, Fig. 2(a) CNN-BN and Fig. 2(b) CNN-BN-MRK4 training and validation losses are compared. CNN-BN-MRK4 produces a faster loss reduction and stabilizes at a substantially lower value, even though both models show a downward trend. Notably, CNN-BN-MRK4 exhibits superior generalization, as evidenced by a smaller gap between training and validation loss. Additionally, compared to CNN-BN, the CNN-BN-MRK4 validation loss curve is smoother, indicating more stable training dynamics.

3.4 Additional Results for Seed Size-7

We also tested CNN-BN and CNN-BN-MRK4 with a random seed size of 7 to further validate the consistency of the enhancements. CNN-BN-MRK4 maintained its performance, with a maximum accuracy of 4% and a minimum loss of 10% compared with CNN-BN for each seed, according to the results shown in Table 1. This shows that the MRK4 augmentation is robust to initialization modifications and is consistent with the patterns observed for seed size 5.

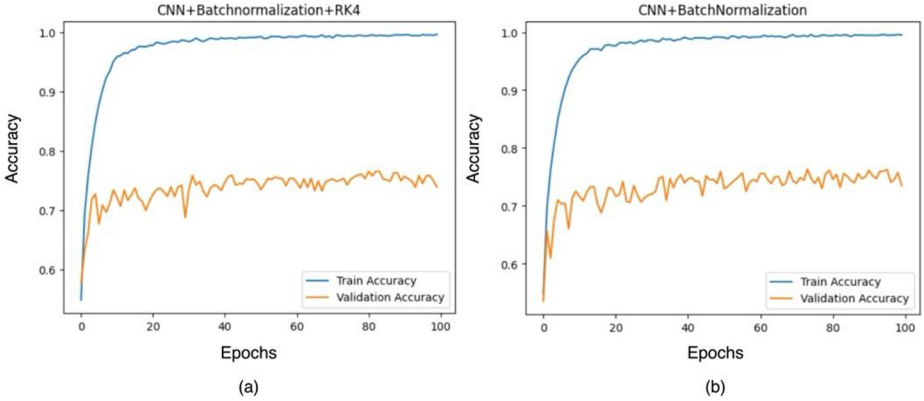

Fig. 1. Training and validation accuracy curves showing faster convergence and reduced variance for CNN-BN-MRK4

4 Discussion

The results demonstrate that adding a fourth-order modified Runge–Kutta (MRK4) integration step to convolutional neural networks with Batch Normalization consistently improves accuracy, stability, and generalization when compared to baseline CNN-BN models. Our experimental comparison focuses on popular CNN baselines, including CNN, CNN-BN, and CNN-BN-Dropout, to isolate the effect of the MRK4 integration phase. We can easily assess MRK4's effect on training stability and generalization using these architectures, which serve as common building blocks in real-world CNN pipelines.

Table 1. Table caption CIFAR-10 Experimental Results for Seed Size 7.

Seed	Accuracy	Loss	F1 Score	Overfit Gap	MaxGap	Train Time(s/epoch)
CNN+ BN (Seed Size:7)						
0	0.7419	2.9906	0.7418	0.2497	0.3495	130.49
42	0.6992	3.9605	0.6953	0.278	0.3476	124.62
123	0.7282	3.5365	0.7248	0.255	0.3169	126.01
2025	0.7416	2.898	0.7412	0.2393	0.2806	124.51
999	0.7296	2.9541	0.7301	0.2596	0.2806	133.09

2024	0.7374	3.1284	0.7359	0.2444	0.3821	131.11
777	0.7269	3.4122	0.7263	0.2579	0.3312	130.47
Mean	0.7293	3.2686	0.7279	0.2548	0.3269	128.61
Std	0.0136	0.3594	0.0147	0.0116	0.0346	3.23
CNN+BN+MRK4(Seed Size:7)						
0	0.7475	2.8124	0.7472	0.2384	0.319	132.1
42	0.7388	2.8785	0.7381	0.2408	0.3238	134.01
123	0.7227	3.2385	0.7264	0.2554	0.3001	134.72
2025	0.7276	3.2281	0.728	0.2535	0.3238	132.12
999	0.7231	3.4433	0.7225	0.2574	0.2982	134.65
2024	0.7395	3.2244	0.7383	0.2485	0.4314	135.12
777	0.7428	3.1141	0.7426	0.2265	0.3588	133.32
Mean	0.7346	3.1342	0.7347	0.2458	0.3164	133.72
Std	0.0092	0.2045	0.0085	0.0103	0.043	1.15

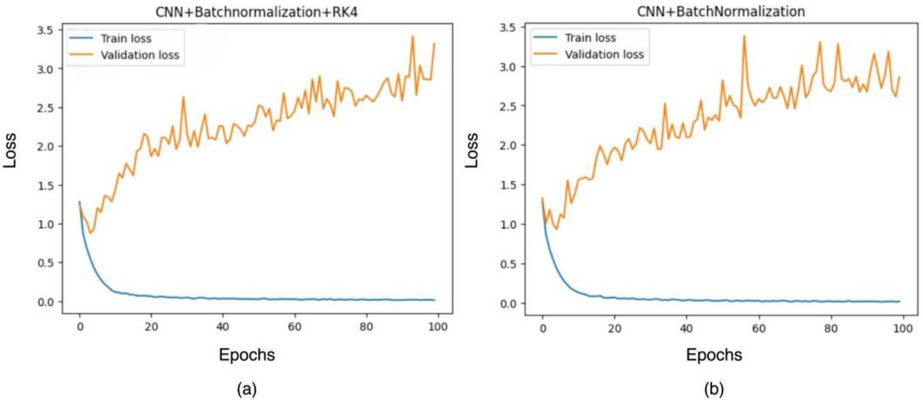

Fig. 2. Training and Test loss curves for (a) CNN-BN, (b) CNN-BN-MRK4

4.1 Why MRK4 Helps

When features in a CNN shift from one layer to the next, the MRK4 technique provides a superior approximation by the authors [14,15]. Within the same layer change, MRK4 takes multiple smaller steps rather than a single one. These extra processes stop errors from compounding across layers and gather more information about the feature maps. As a result, CNN-BN-MRK4 exhibits less variance and learns features more steadily when trained with various random seeds. In numerical analysis, MRK4 is utilized for the purpose of solving differential equations. Reducing error is its natural role, and we apply the same concept here: MRK4 helps CNNs increase accuracy by lowering propagation error between layers. In addition to updating individual feature map entries, MRK4 in CNN- BN considers the neighborhood of each entry. Feature learning is therefore more accurate, reliable, and smooth than CNN-BN or regular CNN.

Table 2. CIFAR-10 Experimental Results for Seed Size 5.

Seed	Accu-racy	Loss	F1 Score	Overfit Gap	MaxGap	Traina-ble Params	Train Time(s/epoch)	Inference Time(ms/sam-ple)
CNN								
0	0.717	3.8564	0.7166	0.2587	0.281	3568	109.569	0.18
42	0.72	3.8679	0.7207	0.2572	0.2702	356810	65.792	0.137
123	0.72	3.6335	0.7188	0.2613	0.2788	356810	67.208	0.14
2025	0.711	3.8643	0.712	0.2548	0.2842	356810	66.785	0.123
999	0.713	3.9705	0.7102	0.2629	0.2749	356810	63.007	0.123
Mean	0.7162	3.8385	0.7156	0.259	0.2778	356810	74.4722	0.1406
Std	0.0036	0.1107	0.0039	0.0028	0.00485	0	17.609	0.0209
CNN+MRK4								
0	0.7081	4.2286	0.7081	0.2622	0.2909	3568	75.9404	0.4921
42	0.7163	4.084	0.7163	0.2563	0.2736	356810	78.8196	0.5484
123	0.7171	3.9649	0.7142	0.2636	0.2861	356810	78.8822	0.5543
2025	0.6988	3.8442	0.6983	0.2725	0.289	356810	78.6051	0.5293
999	0.7192	3.6616	0.7179	0.2664	0.2825	356810	79.3731	0.5549
Mean	0.7119	3.9566	0.711	0.2652	0.2844	356810	78.3241	0.5358
Std	0.0075	0.1949	0.0071	0.0055	0.0061	0	1.218	0.0237
CNN+Batch Normalization								
0	0.742	2.9906	0.7418	0.2497	0.3495	357258	122.071	0.146
42	0.699	3.9605	0.6953	0.278	0.3476	357258	123.752	0.142
123	0.728	3.5365	0.7248	0.255	0.3169	357258	153.657	0.188
2025	0.742	2.898	0.7412	0.2393	0.2806	357258	139.644	0.161
999	0.73	2.9541	0.7301	0.2596	0.2806	357258	142.048	0.155
Mean	0.7282	3.2679	0.7266	0.2563	0.315	357258	136.2344	0.1584
Std	0.0157	0.4157	0.017	0.0128	0.0304	0	11.8776	0.0162
CNN+Batch Normalization+MRK4								
0	0.7475	2.8124	0.7472	0.2384	0.3189	357258	143.8453	0.6237
42	0.739	2.8785	0.7381	0.2408	0.3238	357258	137.1288	0.5479
123	0.7227	3.2385	0.7264	0.2554	0.3001	357258	135.7068	0.5437
2025	0.7276	3.2281	0.7281	0.2535	0.3238	357258	133.9332	0.5441
999	0.7231	3.4433	0.7225	0.2574	0.2982	357258	139.2704	0.5236
Mean	0.7319	3.1202	0.7324	0.2491	0.3129	357258	137.9769	0.5566
Std	0.0097	0.238	0.0089	0.0079	0.0114	0	3.4157	0.0346
CNN+Batch Normalization+Dropout								
0	0.7575	1.8047	0.7571	0.2096	0.2436	357258	136.1803	0.6556
42	0.7671	2.0269	0.7644	0.1933	0.2226	357258	124.2461	0.5792
123	0.7573	1.7325	0.7578	0.2203	0.2519	357258	140.4581	0.5513
2025	0.7616	1.9358	0.7605	0.2042	0.2509	357258	137.0614	0.5691
999	0.765	1.9689	0.7638	0.2058	0.2485	357258	131.7688	0.5348
Mean	0.7616	1.8938	0.767	0.2067	0.2435	357258	133.943	0.5778

Std	0.0039	0.1087	0.003	0.0087	0.0108	0	5.5845	0.0417
CNN+Batch Normalization+Dropout+MRK4								
0	0.757	1.8885	0.756	0.1965	0.2158	357258	151.1735	0.5025
42	0.7663	1.8709	0.7643	0.1944	0.2335	357258	133.4146	0.5929
123	0.7592	1.8912	0.7603	0.2045	0.2322	357258	133.5419	0.525
2025	0.7579	2.1594	0.7575	0.2007	0.2283	357258	134.1493	0.6534
999	0.7613	1.806	0.7604	0.2121	0.2345	357258	135.9854	0.5349
Mean	0.7603	1.9323	0.7597	0.2016	0.2289	357258	137.6529	0.5618
Std	0.0033	0.122	0.0028	0.0063	0.0069	0	6.8224	0.0547

4.2 Comparison with Other Variants

Even though CNN-BN-Dropout and plain CNNs were also used to test MRK4, the results showed only slight improvements. MRK4's error-reducing capacity in CNNs without normalization is limited because its intermediate steps are unstable due to a lack of scaling control. Similarly, in CNN-BN-Dropout, MRK4's smooth propagation is disrupted by Dropout's strong regularization, resulting in diminishing returns. MRK4 and normalization, on the other hand, work in conjunction with CNN-BN: MRK4 provides a higher-order approximation of feature propagation, yielding the greatest benefits, while batch normalization stabilizes intermediate states.

5 Conclusion

This study shows that using a fourth-order modified Runge-Kutta integration strategy in convolutional neural networks improves training stability and generalization, especially when combined with batch normalization. The CNN-BN- MRK4 framework improves performance on CIFAR-10 without requiring extra learnable parameters or significant inference overhead. Although the trials are limited to CIFAR-10, the constant behavior across seeds suggests universal applicability. Further research on larger datasets, such as CIFAR-100 and SVHN, is recommended.

These findings suggest that numerical integration techniques can improve deep learning architectures and encourage additional research into larger datasets and complicated network topologies.

References

1. Kayalibay, B., Jensen, G., van der Smagt, P.: CNN-based segmentation of medical imaging data. arXiv:1701.03056 (2017)
2. Wang, Z., Liu, J.: A review of object detection based on convolutional neural network. In: Proc. 36th Chinese Control Conference (CCC), pp. 11104–11109 (2017)
3. Elngar, A.A., Arafa, M., Fathy, A., Moustafa, B., Mahmoud, O., Shaban, M., Fawzy, N.: Image classification based on CNN: a survey. Journal of Cybersecurity and Information Management 6(1), 18–50 (2021)

4. Pradhan, S.S., Sahoo, S.S.: Detecting Fake WhatsApp Messages with Knowledge Graphs Using Machine Learning and Deep Learning. Arabian Journal for Science and Engineering, 1–26 (2026).
5. Sadeghzadeh, H., Koohi, S., Paranj, A.F.: Free-space optical neural network based on optical nonlinearity and pooling operations. IEEE Access 9, 146533–146549 (2021)
6. Mishkin, D., Sergievskiy, N., Matas, J.: Systematic evaluation of convolution neural network advances on the ImageNet. Computer Vision and Image Understanding 161, 11–19 (2017)
7. Alkaabi, K., Sarfraz, U., Al Darmaki, S.: A deep learning framework for flash-flood- runoff prediction: Integrating CNN-RNN with neural ordinary differential equations (ODEs). Water 17(9), 1283 (2025)
8. Li, H., Sun, C.: Nonlinear time-varying system response modeling via a real-time updated Runge–Kutta physics-informed neural network. Engineering Applications of Artificial Intelligence 144, 110067 (2025)
9. Goyal, P., Benner, P.: Learning dynamics from noisy measurements using deep learn- ing with a Runge–Kutta constraint. arXiv:2109.11446 (2021)
10. Garbin, C., Zhu, X., Marques, O.: Dropout vs. batch normalization: an empirical study of their impact to deep learning. Multimedia Tools and Applications 79(19), 12777–12815 (2020)
11. Chen, R.T.Q., Rubanova, Y., Bettencourt, J., Duvenaud, D.K.: Neural ordinary differential equations. Advances in Neural Information Processing Systems 31 (2018)
12. Yan, H., Du, J., Tan, V.Y.F., Feng, J.: On the robustness of neural ordinary differential equations. arXiv:1910.05513 (2019)
13. Carrara, F., Caldelli, R., Falchi, F., Amato, G.: On the robustness to adversarial examples of neural ODE image classifiers. In: Proc. IEEE International Workshop on Information Forensics and Security (WIFS), pp. 1–6 (2019)
14. Gusak, J., Markeeva, L., Daulbaev, T., Katrutsa, A., Cichocki, A., Oseledets, I.: Towards understanding normalization in neural ODEs. arXiv:2004.09222 (2020)
15. Sander, M., Ablin, P., Peyr´e, G.: Do residual neural networks discretize neural ordinary differential equations? Advances in Neural Information Processing Systems 35, 36520–36532 (2022)
16. Zhu, M., Chang, B., Fu, C.: Convolutional neural networks combined with Runge– Kutta methods. Neural Computing and Applications 35(2), 1629–1643 (2023)
17. Lehtima¨ki, M., Paunonen, L., Linne, M.-L.: Accelerating neural ODEs using model order reduction. IEEE Transactions on Neural Networks and Learning Systems 35(1), 519–531 (2022)
18. Ponagalagusamy, R., Senthilkumar, S.: A new method of embedded fourth order with four stages to study raster CNN simulation. International Journal of Automa- tion and Computing 6(3), 285–294 (2009)
19. Sahoo, S.S., Pati, A., Pradhan, S.S.: Enhancing temporal scoping in knowledge bases: Neural network approaches for time interval incorporation using Wikidata. In: International Conference on Applied Intelligence and Informatics, pp. 415–429 (2024)