SIAM J. IMAGING SCIENCES
Vol. 17, No. 1, pp. 540–594

PottsMGNet: A Mathematical Explanation of Encoder-Decoder Based Neural Networks*

Xue-Cheng Tai†, Hao Liu‡, and Raymond Chan§

Abstract. For problems in image processing and many other fields, a large class of effective neural networks has encoder-decoder-based architectures. Although these networks have shown impressive performance, mathematical explanations of their architectures are still underdeveloped. In this paper, we study the encoder-decoder-based network architecture from the algorithmic perspective and provide a mathematical explanation. We use the two-phase Potts model for image segmentation as an example for our explanations. We associate the segmentation problem with a control problem in the continuous setting. Then, the continuous control model is time discretized by an operator-splitting scheme, the PottsMGNet, and space discretized by the multigrid method. We show that the resulting discrete PottsMGNet is equivalent to an encoder-decoder-based network. With minor modifications, it is shown that a number of the popular encoder-decoder-based neural networks are just instances of the proposed PottsMGNet. By incorporating the soft-threshold-dynamics into the PottsMGNet as a regularizer, the PottsMGNet has shown to be robust with the network parameters such as network width and depth and has achieved remarkable performance on datasets with very large noise. In nearly all our experiments, the new network always performs better than or as well as on accuracy and dice score compared to existing networks for image segmentation.

Key words. Potts model, operator splitting, deep neural network, image segmentation

MSC codes. 68U10, 94A08

DOI. 10.1137/23M1586355

1. Introduction.

Deep neural networks have demonstrated great performance in many image processing tasks, such as image segmentation [58, 48, 13, 3, 47, 75], image denoising [73, 2], etc. Although deep neural networks have provided remarkable results, mathematical explanations for their success are still underdeveloped.

A popular explanation or intuition for network architecture is the encoder-decoder framework, which decomposes the network into two parts: the encoder part and the decoder part. When a high-dimensional variable is passed to the network, the encoder part first uses

*Received by the editors July 14, 2023; accepted for publication (in revised form) November 8, 2023; published electronically March 7, 2024.
https://doi.org/10.1137/23M1586355
Funding: The work of the first author is partially supported by HKRGC-NSFC Grant N-CityU214-19, HKRGC CRF Grant C1013-21GF and NORCE Kompetanseoppbygging program. The work of the second author is partially supported by NSFC 12201530, HKRGC ECS 22302123 and HKBU 179356. The work of the third author is partially supported by HKRGC GRF grants CityU1101120 and CityU11309922 and CRF grant C1013-21GF.
†NORCE Norwegian Research Centre, Nygårdstangen, NO-5838 Bergen, Norway (xtai@norceresearch.no, xuechengtai@gmail.com).
‡Corresponding author. Mathematics, Hong Kong Baptist University, Kowloon Tong, Hong Kong (haoliu@hkbu.edu.hk).
§Department of Mathematics, City University of Hong Kong, Hong Kong; and Hong Kong Centre for Cerebro-Cardiovascular Health Engineering (raymond.chan@cityu.edu.hk).

several computational layers and downsampling layers to extract useful low-dimensional features. Then these features are passed to the decoder to reconstruct the desired high-dimensional output. Such an architecture is widely used in networks for image segmentation, such as UNet [58], UNet++ [75], DeepLab [13], and SegNet [3]. While the encoder-decoder framework only provides a general picture of the network architecture, i.e., the decoder does a dimension reduction and the decoder does a data reconstruction, more detailed explanations, such as what is the role of each layer in the network, and what mathematical model we are solving by this network, are still unclear.

For image segmentation, a large class of models is based on theories of min cut/ max flow. The continuous max flow and min cut problems are studied in [4, 71]. Stemming from Markov random fields, many efficient methods are designed based on graph cuts [9]. A very important mathematical model for image segmentation is the Potts model [55]. Applications of the Potts model for segmentation and classification [7, 8, 72] have been extensively studied for discrete graph settings and can be efficiently solved by max flow algorithms. The new survey [63] contains a comprehensive overview of the Potts model and its fast algorithms. The Potts model was first proposed for statistical mechanics in [55] and can be taken as a generalization of the two-state Ising model to lattice [54]. It is used for binary graph cuts in [9]. In [9, 7], efficient graph cut algorithms, equipped with fast graph cut techniques, are proposed for min cut problems such as image segmentation with the Potts model. It has been shown in [4, 71] that the Potts model is equivalent to a continuous min cut and max flow problem: If the Potts model is discretized with certain approximations, it reduces to existing graph cut models (see [69] for some more detailed explanations). In [71], the alternating direction method of multipliers (ADMM) is used to solve the Potts model. However, the convergence of the algorithm is not guaranteed. Recently, in [61], based on the Eckstein–Bertsekas [21] and Fortin–Glowinski [23] splitting techniques, two novel preconditioned ADMMs for the Potts model with guaranteed convergence were proposed.

Operator-splitting methods are powerful tools for solving complicated optimization problems. In general, an operator-splitting method decomposes a complicated problem into several easy-to-solve subproblems so that each subproblem either has a closed-form solution or can be solved efficiently. Based on how these subproblems are solved, operator-splitting methods can be divided into parallel splitting methods [49] and sequential splitting methods [26, 50]. As indicated by the name, parallel splitting methods solve all subproblems in parallel and then combine the results by averaging; sequential splitting methods solve subproblems sequentially. In fact, ADMM is a special type of operator-splitting method. Operator-splitting methods have been applied in numerical methods for partial differential equations [29, 42], inverse problems [28], computational fluid dynamics [6, 53], obstacle problems [45], surface reconstructions [36], and image processing [44, 18, 17, 43]. Compared to ADMM, operator-splitting methods have fewer parameters and are not sensitive to the choices of parameters. For problems from image processing, it is shown in [17, 18] that operator-splitting methods are more efficient. We refer readers to [31, 30, 32] for a comprehensive discussion on operator-splitting methods.

In this paper, we provide a mathematical explanation of encoder-decoder-based convolutional neural networks for image segmentation from the perspective of mathematical models and algorithms. We use the two-phase image segmentation as an example, but the explanations can be generalized to multiphase and other classification problems as well. We solve

the two-phase segmentation problem using the Potts model with the length represented by soft-threshold-dynamics [1, 51, 47]. We derive the Euler–Lagrange equation of the problem and associate it with an initial value problem with control variables. We propose a novel operator-splitting method, the hybrid splitting, which combines parallel splitting and sequential splitting to discretize the control problem. By incorporating the hybrid splitting method with the multigrid method and with proper choices of the control variables, we obtain a scheme we call PottsMGNet, which has the same architecture as an encoder-decoder-based neural network. Each layer in the network corresponds to a substep of the splitting scheme, and its convolutional kernels and biases correspond to the control variables. Our contributions can be summarized as follows:

- We provide a clear and concise mathematical explanation of a large class of encoder-decoder-based neural networks, which are essentially some operator-splitting schemes with a multigrid method for some optimal control problems. These explanations are important for designing and improving neural networks, providing clear guidelines for the choice of the number of layers and neurons on each layer, and making the networks more explainable, with each part having a clear mathematical meaning.
- Our proposed PottsMGNet is a multigrid-based numerical splitting scheme for solving control problems. With proper settings, our framework recovers the widely used neural networks in the literature for image segmentation problems, just with different activation functions. Each layer in the network corresponds to a substep of the splitting scheme, and its convolutional kernels and biases are the control variables.
- Numerical tests on different datasets show that PottsMGNet is robust with respect to network parameters and performs better or as well as existing popular encoder-decoder neural networks. It can handle data with large noise and is more robust than other networks with similar architectures.
- We also develop a novel hybrid splitting method for solving time-evolutional equations. Our hybrid splitting method incorporates parallel splitting and sequential splitting and we prove that it is first-order accurate. In the current work, they are used to get the neural networks. They can be used for solving other problems as well.

This work has been inspired by a wealth of pioneering research seeking mathematical explanations for neural networks. In particular, we would like to mention several key references, including [19, 20, 62, 24, 37, 74, 35, 60, 33, 34, 15, 10, 5]. It has been established that neural networks possess universal approximation properties [37, 74]. The work of E and co-authors [19, 20] proposed to view neural networks as continuous dynamical systems and as special discretizations of continuous problems. This has been a source of inspiration for our work. Sussillo and Barak [62] explored low-dimensional dynamics in high-dimensional recurrent neural networks. Ruthotto and Haber [60] introduced deep neural networks motivated by partial differential equations, and ODE concepts are used in Haber and Ruthotto [33] to obtain stable architectures for deep neural networks. The relation between deep neural networks and control problems is investigated in [59, 5]. Many researchers have observed that UNet and other encoder-decoder neural networks are related to multiscale techniques [34, 33]. Haber and Ruthotto [34] proposed multiscale methods for convolutional neural networks. In [35], MgNet is proposed as a unified framework that uses multigrid linear operators as feature extraction operators in traditional convolutional neural networks. The similarity between

neural networks and operator-splitting methods are mentioned and utilized in [41]. Continuous UNet using a second-order ODE has been proposed in [15]. A systematic review is provided in [10]. However, none of the aforementioned references have explained neural networks as an operator-splitting discretization with a multigrid spatial approximation for the continuous Potts model considered as a control problem, as we have done in this work.

This paper is structured as follows. In section 2, we introduce the Potts model. In section 3, we formulate a control problem to solve the Potts model and our framework to learn the control variables. We introduce PottsMGNet, an operator-splitting scheme, for the control problem and its connections to neural networks in section 4. In section 5, we discuss the discretization and solution to each subproblem. In section 6, we show that PottsMGNet is in fact a neural network and discuss how to modify it to recover existing popular neural networks for image segmentation. In section 7, we present our numerical experiment results, and we conclude this paper in section 8.

Notation. Throughout this paper, we use lowercase letters to denote scalar variables and functions. Bold letters are used to denote vectors and vector-valued functions. Capital letters are used to denote tensors and operators. Calligraphic letters are used to denote sets.

2. Introduction to the Potts model.

In this section, we give a brief introduction to the Potts model [55, 63]. See Appendix A for more detailed derivations in getting this model.

We use image segmentation to present the Potts model and its numerical algorithms. Let Ω be the image domain. The continuous two-phase Potts model is of the form

$$\begin{cases} \min\limits_{\Omega_0,\Omega_1} \left\{ \sum_{k=0}^{1} \int_{\Omega_k} g_k(\mathbf{x}) d\mathbf{x} + \frac{1}{2} \sum_{k=0}^{1} |\partial\Omega_k| \right\}, \\ \Omega_0 \cup \Omega_1 = \Omega, \ \Omega_0 \cap \Omega_1 = \emptyset, \end{cases} \tag{2.1}$$

where $|\partial\Omega_k|$ is the perimeter of Ω_k and g_k's are nonnegative weight functions depending on the input image. By solving (2.1), the image domain is segmented into two regions: Ω_0 and Ω_1.

By utilizing a regularized softmax operator [47] and techniques from threshold dynamics [1, 51, 22, 47], the two-phase Potts model (2.1) can be approximated by the following problem (see Appendix A for the derivation):

$$\min_{v(\mathbf{x})\in[0,1]} \left[\int_\Omega v g d\mathbf{x} + \varepsilon \int_\Omega (v \ln v + (1-v)\ln(1-v)) d\mathbf{x} + \eta \int_\Omega v(\mathbf{x})(G_\sigma * (1-v))(\mathbf{x}) d\mathbf{x} \right] \tag{2.2}$$

for some constant $\varepsilon > 0$ and $\eta \geq 0$, where $g = g_1 - g_2$ and G_σ is the Gaussian kernel $G_\sigma(\mathbf{x}) = \frac{1}{2\pi\sigma^2} \exp(-\frac{\|\mathbf{x}\|^2}{2\sigma^2})$ for some given $\sigma > 0$. As usual $*$ denotes the convolution operator. By minimizing (2.2) and denoting the minimizer by u, the image domain is segmented into two regions corresponding to $u \leq 0.5$ and $u > 0.5$. In (2.2), the second integral is used to get the regularized softmax operator (see Appendix A for detailed explanations about this). The third integral approximates the perimeter of the two regions. One can show that as $\varepsilon \to 0$, we have $u(\mathbf{x}) \in \{0, 1\}$ for any $\mathbf{x} \in \Omega$ (see Lemma A.1 in Appendix A).

If the minimizer u exists, it can be proven that $u \in (0,1)$ and it satisfies the Euler–Lagrangian equation:

$$\varepsilon \ln \frac{u}{1-u} + \eta G_\sigma * (1-2u) + g = 0 \quad \forall \mathbf{x} \in \Omega. \tag{2.3}$$

The corresponding gradient flow equation on the time interval $(0,T]$ with a proper initial state $u_0(\mathbf{x})$ is given as

$$\begin{cases} u_t = -\varepsilon \ln \frac{u}{1-u} - \eta G_\sigma * (1-2u) - g, \ (\mathbf{x},t) \in \Omega \times (0,T], \\ u(\mathbf{x},0) = u_0, \ \mathbf{x} \in \Omega. \end{cases} \tag{2.4}$$

Remark 2.1. Following the formulations in Appendix A, we can easily extend our model and algorithms proposed later to multiphase segmentation and classification problems. To make the presentations clear, we will stay with the two-phase problem in this work.

3. A control problem for the Potts model. For image segmentation, we assume f is a given image defined on the image domain Ω. We take the initial function $u_0 = H(f)$ for some appropriate operator H to be specified in section 7.1. Based on (2.3), we consider the following gradient flow equation for the Potts model with control variables $W(\mathbf{x},t)$ and $d(\mathbf{x},t)$:

$$\begin{cases} \dfrac{\partial u}{\partial t} = -\varepsilon \ln \dfrac{u}{1-u} - \eta G_\sigma * (1-2u) + W(\mathbf{x},t) * u + d(\mathbf{x},t), \ (\mathbf{x},t) \in \Omega \times (0,T], \\ u(\mathbf{x},0) = H(f), \ \mathbf{x} \in \Omega. \end{cases} \tag{3.1}$$

To clarify, in (3.1), we are considering a Potts model with two labels on a domain $\Omega \subset \mathbb{R}^2$. The function $u(\mathbf{x},t) : \Omega \times [0,T] \to \mathbb{R}$ represents the label of each point $\mathbf{x}$ in the domain at time t. The term $W(\mathbf{x},t) * u$ is the convolution of u with a weight function $W(\mathbf{x},t) : D \times [0,T] \to \mathbb{R}$ for some domain $D \subset \mathbb{R}^2$, which essentially represents a weighted average of the labels of neighboring points. The support of $W(\mathbf{x},t)$ is normally small and the domain D may differ from Ω. The parameters ε and η control the approximation of the binary functions and are used to regularize the labels. Specifically, the length regularization with the term associated with this is used to ensure that neighboring points have similar labels, thereby promoting spatial coherence of the labeling. The parameter η controls the strength of this regularization, with larger values promoting more spatial coherence.

The function $g(\mathbf{x},t)$ in (2.3) is replaced by $W(\mathbf{x},t) * u + d(\mathbf{x},t)$ in (3.1), where $W(\mathbf{x},t)$ and $d(\mathbf{x},t)$ are control functions that can be used to steer the final state $u(\mathbf{x},T)$ to some desirable state. In this way, $W(\mathbf{x},t)$ and $d(\mathbf{x},t)$ are treated as control variables that can be adjusted to achieve the desired labeling of the domain at the final time T. In (3.1), the control variable $W(\mathbf{x},t)$ is applied to u by convolution since convolution is widely used in deep learning methods for image processing. We remark that the convolution in (3.1) (and those discussed in section 4) can be replaced by any other linear operations.

In this context, we are considering an optimal control approach to obtain a segmentation operator for two-phase image segmentation. Specifically, we are given a set of I images $f_i : \Omega \to \mathbb{R}^3$ that have been segmented into foreground and background regions, denoted by

$v_i : \Omega \to \{0,1\}$ for $i = 1,2,\ldots,I$. Denote $\mathcal{N}_1$ as the mapping $f \to u(\mathbf{x},T)$, i.e., $\mathcal{N}_1(f,\theta_1) = u(x,T)$ with $u(x,T)$ being the solution of (3.2) at time T, where $\theta_1 = (W(\mathbf{x},t), d(\mathbf{x},t))$ denotes the control variables. We seek to find a set of control variables θ_1 that minimize the distance between the segmentation result obtained by applying the mapping $\mathcal{N}_1$ to each image f_i and its corresponding ground truth segmentation v_i:

$$\min_{\theta_1} \sum_{i=1}^{I} \mathcal{L}(\mathcal{N}_1(f_i,\theta_1), v_i). \tag{3.2}$$

The distance measure used is the cross entropy, which is a common choice for comparing probability distributions:

$$\mathcal{L}(u,v) = -u \ln v - (1-u)\ln(1-v).$$

Formally, the optimization problem we solve is given by (3.2), where $\mathcal{L}(\cdot,\cdot)$ measures the distance between two segmentation results and $\mathcal{N}_1(f_i,\theta_1)$ represents the segmentation result obtained by applying the mapping $\mathcal{N}_1$ to the image f_i using the control variables θ_1. Since the control variables θ_1 appear in the mapping $\mathcal{N}_1$, we can view (3.2) as an optimal control problem with multiple targeting states v_i and control variables θ_1. The control equation (3.1) contains nonlinear and bilinear terms, which makes the problem more challenging to solve. However, the controllability of a similar control equation has been shown for a single targeting state in [39], which justifies the suitability of our approach.

4. Our PottsMGNet.

To solve the optimal control problem (3.2) numerically, we need two discretizations: one for the control variables $\theta_1 = (W(\mathbf{x},t), b(\mathbf{x},t))$ and another one for the corresponding state variable u. To make the presentation clear, we will only consider the discretization of θ_1 now and leave the discretization of the state variable till later. So the state variables u are continuous functions in this section and will be discretized in section 5.

4.1. Multigrid discretizations.

The image domain Ω and the convolution kernel domain D may be different. We assume that both Ω and D have been discretized in the multigrid setting, as explained in Appendix B. As convolution is used, we need to extend the values of u and θ_1 outside the domains. Due to this, we will simply take $\Omega = D = \mathbb{R}^2$ and apply zero padding for θ_1 and u. Correspondingly, $\mathcal{T}^j, \mathcal{V}^j$ are the multigrids and spaces over $\mathbb{R}^2$ with the corresponding padding techniques, where j denotes the grid level. In this paper, we use $\mathcal{T}^1$ to represent the finest grid (the image's original resolution). The grid gets coarser as j increases. We will use the V-cycle multigrid technique, which consists of a left branch and a right branch. In the left branch, computations are conducted from fine grids to coarse grids sequentially. In the right branch, computations are conducted from coarse grids to fine grids sequentially. See Figure 1 for an illustration of a V-cycle of the multigrid method. Our notation for multigrid follows more from [70, 66, 11].

4.2. A basic decomposition for θ_1.

The approach presented in [64, 65, 66] for solving the optimization problems in the discrete multigrid setting involves decomposing the search space into many subspaces or subsets. The goal is to search for the minimizer iteratively through these subspaces either in parallel or sequentially. This approach can be seen as decomposing

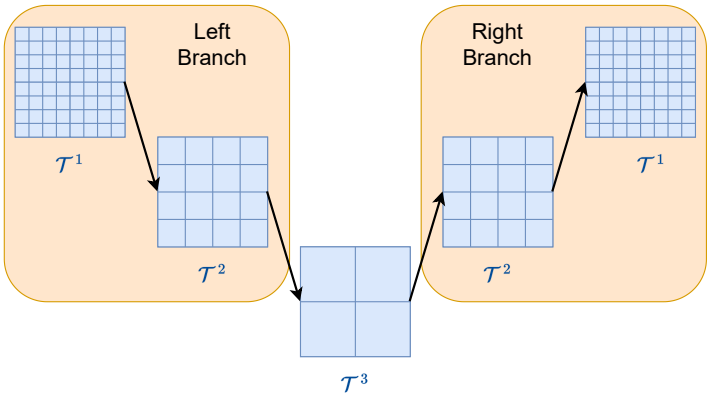

Figure 1. *An illustration of a V-cycle of the multigrid method.*

the minimization variable into a sum of variables from different subspaces. The traditional V-cycle multigrid method is a sequential iterative procedure that operates over the multigrid subspaces defined in Appendix B in a specific order, as shown in Figure 1.

Traditionally, space decomposition and subspace correction are used to interpret multigrid and domain decomposition methods [70, 11]. We will apply the space decomposition ideas [64, 65, 66] for solving the optimization problems (3.2). This idea has been used to decompose a large space into the sum of smaller spaces and then solve the problems iteratively over the smaller spaces. But here, it is used differently. The control variable θ_1 is decomposed into a sum of variables from subspaces. One purpose for this decomposition is to increase the number of unknowns for the control variables. This is different from earlier usages for the space decomposition ideas. Then we will use a hybrid splitting scheme to split the components of θ_1 into several subproblems, which are solved in parallel or sequentially. The general idea is that all components of θ_1 are gone through when we apply one iteration of the scheme. Details of the hybrid splitting scheme are discussed in Appendix D. To make the presentation clear, we will present the ideas step by step in the following on how to decompose the control variable θ_1. Our idea is to decompose θ_1 so that the control variables and operators have a similar form as those in (D.5). When there is no ambiguity, we omit the control variables' dependency on $\mathbf{x}$ or t for simplicity of presentation.

(i) First, we decompose the control variables $W(\mathbf{x},t)$ and $d(\mathbf{x},t)$ as in the following:

$$W(\mathbf{x},t) = A(\mathbf{x},t) + \widetilde{A}(\mathbf{x},t),\ d(\mathbf{x},t) = b(\mathbf{x},t) + \widetilde{b}(\mathbf{x},t). \tag{4.1}$$

These variables will be further split next. Above, A, b contain the control variables in the left branch of the multigrid V-cycle, and $\widetilde{A}, \widetilde{b}$ contain the control variables in the right branch of the multigrid V-cycle (see later steps for details). We also decompose the operator as follows:

$$-\varepsilon \ln \frac{u}{1-u} - \eta G_\sigma * (1-2u) = S(u) + \widetilde{S}(u). \tag{4.2}$$

(ii) Second, the control variables are further decomposed as

$$A=\sum_{j=1}^{J}A^j,\quad b=\sum_{j=1}^{J-1}b^j,\quad S=\sum_{j=1}^{J}S^j, \tag{4.3}$$

$$\widetilde{A}=\sum_{j=1}^{J-1}\widetilde{A}^j+A^*,\quad \widetilde{b}=\sum_{j=1}^{J-1}\widetilde{b}^j+b^*,\quad \widetilde{S}=\sum_{j=1}^{J-1}\widetilde{S}^j+S^*. \tag{4.4}$$

Above, $A^j, b^j, S^j, \widetilde{A}^j, \widetilde{b}^j, \widetilde{S}^j$ contain control variables at grid level j, and A^*, b^*, and S^* contain control variables that are applied to the output of the V-cycle at the finest mesh, i.e., $A^j, \widetilde{A}^j \in \mathcal{V}^j, A^* \in \mathcal{V}^1, b^j, \widetilde{b}^j, b^* \in \mathbb{R}$.

(iii) At grid level j in the left branch, we further decompose

$$A^j=\sum_{k=1}^{c_j}A_k^j,\quad b^j=\sum_{k=1}^{c_j}b_k^j,\quad S^j=\sum_{k=1}^{c_j}S_k^j. \tag{4.5}$$

Above, c_j's are positive integers which are often called the channel number at grid level j and they are fixed parameters for the network. The reason to do these decompositions is to increase the number of unknowns in the control variables, which enables us to handle large datasets. We compute c_j intermediate outputs with the A_k^j's. Variables $A_k^j \in \mathcal{V}^j, b_k^j \in \mathbb{R}$ contain control variables producing the kth intermediate output. This is accomplished via a hybrid splitting scheme with c_j parallel splittings (see Appendix D for details). We do the same decomposition for the right branch, i.e.,

$$\widetilde{A}^j=\sum_{k=1}^{c_j}\widetilde{A}_k^j,\quad \widetilde{b}^j=\sum_{k=1}^{c_j}\widetilde{b}_k^j,\quad \widetilde{S}^j=\sum_{k=1}^{c_j}\widetilde{S}_k^j. \tag{4.6}$$

(iv) At grid level j for the kth intermediate output, we again further decompose

$$A_k^j=\sum_{s=1}^{c_{j-1}}A_{k,s}^j. \tag{4.7}$$

The purpose is also to increase the number of unknowns for the control variables. The kth intermediate output at grid level j is computed using all intermediate outputs from the previous grid level. Variable $A_{k,s}^j$ is used to convolve with the sth intermediate output from grid level $j-1$. We do the same decomposition for the right branch.

(v) The A^* variable is also further decomposed as

$$A^*=\sum_{s=1}^{c_1}A_s^*, \tag{4.8}$$

where A_s^* is used to convolve with the sth output from level 1 of the right branch.

After these decompositions, we see that the control variables are decomposed as

$$A(\mathbf{x},t)=\sum_{j=1}^{J}\sum_{k=1}^{c_j}\sum_{s=1}^{c_{j-1}}A_{k,s}^{j}(\mathbf{x},t),\qquad \widetilde{A}(\mathbf{x},t)=\sum_{j=1}^{J-1}\sum_{k=1}^{c_j}\sum_{s=1}^{c_{j-1}}\widetilde{A}_{k,s}^{j}(\mathbf{x},t)+\sum_{s=1}^{c_1}A_{s}^{*}(\mathbf{x},t), \tag{4.9}$$

$$b(\mathbf{x},t)=\sum_{j=1}^{J}\sum_{k=1}^{c_j}b_{k}^{j}(\mathbf{x},t),\qquad \widetilde{b}(\mathbf{x},t)=\sum_{j=1}^{J-1}\sum_{k=1}^{c_j}\widetilde{b}_{k}^{j}(\mathbf{x},t)+\widetilde{b}^{*}(\mathbf{x},t), \tag{4.10}$$

and the operators $S(u),\widetilde{S}(u)$ are decomposed as

$$S(u)=\sum_{j=1}^{J}\sum_{k=1}^{c_j}S_{k}^{j}(u),\ \widetilde{S}(u)=\sum_{j=1}^{J-1}\sum_{k=1}^{c_j}\widetilde{S}_{k}^{j}(u)+S^{*}(u). \tag{4.11}$$

The Potts gradient flow is transferred into

$$\begin{cases}\frac{\partial u}{\partial t}=A*u+\widetilde{A}*u+b+\widetilde{b}+S(u)+\widetilde{S}(u),\ (\mathbf{x},t)\in\Omega\times[0,T],\\ u(\mathbf{x},0)=H(f),\ \mathbf{x}\in\Omega.\end{cases} \tag{4.12}$$

With the decomposition discussed above, (4.12) has a similar form as (D.5) with $M_j=1$, $c_{j,1}=c_{2J-j}=c_j$ for $1\le j\le J$. We solve problem (4.12) by the hybrid splitting method proposed in Appendix D. Let us divide the time interval $[0,T]$ into N subintervals with time step size $\Delta t=T/N$. We denote our numerical solution at time $t^n=n\Delta t$ by U^n. In our scheme, we use u and v to denote intermediate variables. Their superscript j indicates that they are discretized on $\mathcal{T}^j$, i.e., $v_k^j,u_k^j,\bar{u}^j\in\mathcal{V}^j$. The resulting scheme by applying Algorithm D.2 to (4.12) for updating U^n to U^{n+1} is summarized in Algorithm 4.1, where the dependency of control variables on $\mathbf{x}$ is omitted. Note that in (4.15) and (4.16), there are factors 2^{j-1} and 2^j in the denominator on the left-hand side. These factors are introduced to compensate for the averaging effects in the relaxation step (4.17). With these factors and when all operators are linear, we can show that Algorithm 4.1 is a first-order scheme to solve a time evolution equation (see Theorem D.2 for details).

The architecture of Algorithm 4.1 is illustrated in Figure 2. The explanations of all indices for operators and variables of the left branch are summarized in Table 1. In Algorithm 4.1, for each grid level, a relaxation is used to pass information from the left branch to the right branch, as indicated by the green arrows.

Let us denote $\theta_2=\{\theta_2^n\}_{n=1}^N$ with

$$\theta_2^n=\left(\{A_{k,s}^{j}(\mathbf{x},t^n)\}_{j,k,s},\{\widetilde{A}_{k,s}^{j}(\mathbf{x},t^n)\}_{j,k,s},\{A_{s}^{*}(\mathbf{x},t^n)\}_{s},\{b_{k}^{j}(\mathbf{x},t^n)\}_{j,k},\{\widetilde{b}_{k}^{j}(\mathbf{x},t^n)\}_{j,k},\widetilde{b}^{*}(\mathbf{x},t^n)\right). \tag{4.13}$$

We also denote $\mathcal{N}_2$ as the mapping

$$\mathcal{N}_2: f\to H(f)\to U^1\to\cdots\to U^N,$$

which maps f to U^N by applying Algorithm 4.1 N times. Parameters θ_2 are learned by solving

$$\min_{\theta_2}\sum_{i=1}^{I}\mathcal{L}(\mathcal{N}_2(f_i,\theta_2),v_i). \tag{4.14}$$

Algorithm 4.1. A basic V-cycle multigrid control algorithm.

Data: The solution U^n at time step t^n.
Result: The computed solution U^{n+1} at time step t^{n+1}.
Set $c_0 = 1, v^0 = U^n, v_1^0 = U^n$.
for $j = 1, \ldots, J$ **do**
  for $k = 1, 2, \ldots c_j$ **do**
    Compute v_k^j by solving

$$\frac{v_k^j - v^{j-1}}{2^{j-1} c_j \Delta t} = \sum_{s=1}^{c_{j-1}} A_{k,s}^j(t^n) * v_s^{j-1} + b_k^j(t^n) + S_k^j(v_k^j). \tag{4.15}$$

  end for
  Compute v^j as

$$v^j = \frac{1}{c_j} \sum_{k=1}^{c_j} v_k^j.$$

end for
Set $\bar{u}^J = v^J$ and $\bar{u}_k^J = v_k^J, k = 1, 2, \ldots c_J$.
for $j = J-1, \ldots, 1$ **do**
  for $k = 1, 2, \ldots c_j$ **do**
    Compute u_k^j by solving

$$\frac{u_k^j - \bar{u}^{j+1}}{2^j c_j \Delta t} = \sum_{s=1}^{c_{j+1}} \widetilde{A}_{k,s}^j(t^n) * \bar{u}_s^{j+1} + \widetilde{b}_k^j(t^n) + \widetilde{S}_k^j(u_k^j). \tag{4.16}$$

  end for
  Compute $\bar{u}_k^j$ and $\bar{u}^j$ as

$$\bar{u}_k^j = \frac{1}{2} u_k^j + \frac{1}{2} v_k^j \tag{4.17}$$

for $k = 1, \ldots, c_j$, and

$$\bar{u}^j = \frac{1}{c_j} \sum_{j=1}^{c_j} \bar{u}_k^j.$$

end for
Compute U^{n+1} by solving

$$\frac{U^{n+1} - \bar{u}^1}{\Delta t} = \sum_{s=1}^{c_1} A_s^*(t^n) * u_s^1 + b^*(t^n) + S^*(U^{n+1}).$$

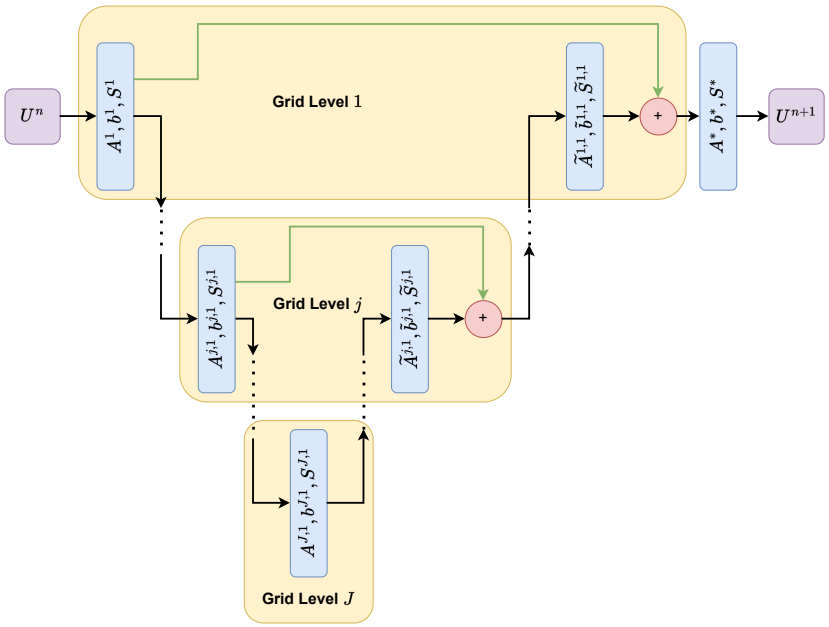

Figure 2. *Illustration of Algorithm 4.1.*

Table 1
Explanation of indices for kernels and variables in the left branch of Algorithms 4.1 and 4.2.

For $A^j_{k,s}, b^j_k, S^j_k$, $A^{j,l}_{k,s}, b^{j,l}_k, S^{j,l}_k$	j	l	k	s
Index meaning: index of	grid levels	sequential splittings	parallel splittings	output from the previous substep
For u^j_k, v^j_k, $u^{j,l}_k, v^{j,l}_k$	j	l	k	-
Index meaning: index of	grid levels	sequential splittings	parallel splittings	-

In (4.14), θ_2 is a space decomposition representation for a discretization of θ_1. The operation procedure $\mathcal{N}_2$ is a numerical scheme solving (3.1). We can see that problem (4.14) is a discretization of the continuous Potts model (3.2) with some proper decomposition of the control variables.

Remark 4.1. We will show in section 4.3 that the structure of one iteration of Algorithm 4.1 is the same as a neural network. Take such a network as a block. The operator $\mathcal{N}_2$ applies Algorithm 4.1 N times, corresponding to chaining N such blocks sequentially. Thus $\mathcal{N}_2$ can be taken as a large network and θ_2 is the collection of all trainable parameters (control variables). Problem (4.14) can be solved by training this network to determine θ_2, for which PyTorch is used in our experiments.

4.3. Relationship to neural networks. For readers familiar with convolutional neural networks, it is immediately evident that the first two terms on the right-hand side of (4.15)

and (4.16) are computing the same thing as is done by the "conv2d" function in PyTorch. The architecture of Algorithm 4.1 is equivalent to a simple encoder-decoder-based neural network with $2J$ layers:

(i) The left and right branches of the V-cycle correspond to the encoder and decoder in neural networks, respectively. In the left branch, computations from fine grids to coarse grids are conducted, which is an encoding process. In the right branch, computations from coarse grids to fine grids are conducted, which is a decoding process. In general, Algorithm 4.1 is equivalent to a neural network with $2J$ layers: $2J-1$ layers for the encoder and decoder and one final layer.

(ii) Computations at grid level j in the left branch correspond to the jth layer of the corresponding network. At this grid level, the number of parallel splittings c_j corresponds to the width of the jth layer of the network. The index k corresponds to the kth channel of this layer.

(iii) The relaxation in the right branch corresponds to the skip pathways between encoders and decoders in the network.

4.4. A general decomposition for θ_1. In Algorithm 4.1, excluding the final substep, we only have two sequential steps at each grid level: one in the left branch and one in the right branch. As a consequence, the equivalent network has two layers for every grid level: one layer in the encoder and one layer in the decoder. In practice, many popular networks use several layers for each grid level, for example, UNet has four layers for each grid level. We then modify the splitting strategy of Algorithm 4.1 to generalize it so that it is equivalent to more general networks. We decompose the control variables θ_1 as follows:

(i)–(ii) The first two decompositions are the same as the first two decompositions in section 4.2, after which we get the decompositions of A, b, S and $\widetilde{A}, \widetilde{b}, \widetilde{S}$.

(iii) At grid level j, let L_j be a positive integer representing the number of substeps we want to perform at grid level j in the left (and right) branch. We decompose

$$A^j = \sum_{l=1}^{L_j} A^{j,l}, \quad b^j = \sum_{l=1}^{L_j} b^{j,l}, \quad S^j = \sum_{l=1}^{L_j} S^{j,l}. \tag{4.18}$$

We use a sequential splitting to divide the operators into L_j sets, where $A^{j,l}, b^{j,l}$, and $S^{j,l}$ are the sets of operators used at the lth sequential substep. We do the same decomposition for the right branch.

(iv) At grid level j and the lth sequential substep, we decompose

$$A^{j,l} = \sum_{k=1}^{c_j} A_k^{j,l}, \quad b^{j,l} = \sum_{k=1}^{c_j} b_k^{j,l}, \quad b^{j,l} = \sum_{k=1}^{c_j} S_k^{j,l}. \tag{4.19}$$

We use a hybrid splitting with c_j parallel splittings to treat all operators, where operators $A_k^{j,l}, b_k^{j,l}$, and $S_k^{j,l}$ are used in the kth parallel splitting. We do the same decomposition for the right branch.

(v) At grid level j, the lth sequential step, and the kth parallel splitting, we take the c_j outputs from the $(l-1)$-th sequential step as inputs and use kernels from $A_k^{j,l}$ to convolve with them. Therefore, we decompose $A_k^{j,l}$ into c_j kernels:

$$A_k^{j,l} = \sum_{s=1}^{c_{j,l}} A_{k,s}^{j,l} \quad \text{with} \quad c_{j,l} = \begin{cases} c_{j-1} & \text{if } l = 1, \\ c_j & \text{if } l > 1. \end{cases} \tag{4.20}$$

We do the same decomposition for the right branch:

$$\widetilde{A}_k^{j,l} = \sum_{s=1}^{\widetilde{c}_{j,l}} \widetilde{A}_{k,s}^{j,l} \quad \text{with} \quad \widetilde{c}_{j,l} = \begin{cases} c_{j+1} & \text{if } l = 1, \\ c_j & \text{if } l > 1. \end{cases} \tag{4.21}$$

(vi) We decompose A^* in the same way as the decomposition step (v) in section 4.2. After the decompositions, the control variables and operations are decomposed as

(4.22)

$$A(\mathbf{x},t) = \sum_{j=1}^{J}\sum_{l=1}^{L_j}\sum_{k=1}^{c_j}\sum_{s=1}^{c_{j,l}} A_{k,s}^{j,l}(\mathbf{x},t), \qquad \widetilde{A}(\mathbf{x},t) = \sum_{j=1}^{J}\sum_{l=1}^{L_j}\sum_{k=1}^{c_j}\sum_{s=1}^{\widetilde{c}_{j,l}} \widetilde{A}_{k,s}^{j,l}(\mathbf{x},t) + \sum_{s=1}^{c_1} A_s^*(\mathbf{x},t),$$

(4.23)

$$b(\mathbf{x},t) = \sum_{j=1}^{J}\sum_{l=1}^{L_j}\sum_{k=1}^{c_j} b_k^{j,l}(\mathbf{x},t), \qquad \widetilde{b}(\mathbf{x},t) = \sum_{j=1}^{J}\sum_{l=1}^{L_j}\sum_{k=1}^{c_j} \widetilde{b}_{k,s}^{j,l}(\mathbf{x},t) + b^*(\mathbf{x},t),$$

(4.24)

$$S(u) = \sum_{j=1}^{J}\sum_{l=1}^{L_j}\sum_{k=1}^{c_j} S_k^j(u), \qquad \widetilde{S}(u) = \sum_{j=1}^{J-1}\sum_{l=1}^{L_j}\sum_{k=1}^{c_j} \widetilde{S}_k^j(u) + S^*(u).$$

Algorithm 4.2. A general V-cycle multigrid control algorithm.

Data: The solution U^n at time t^n.
Result: The computed solution U^{n+1} at time step t^{n+1}.
Set $c_0 = 1, L_0 = 1, v^0 = U^n, v_1^{0,1} = U^n$.
for $j = 1, \ldots, J$ **do**
  Set $v^{j,0} = v^{j-1,L_{j-1}}$ and $v_k^{j,0} = v_k^{j-1,L_{j-1}}$, $k = 1, \ldots, c_{j-1}$,
  for $l = 1, \ldots, L_j$ **do**
    for $k = 1, \ldots, c_j$ **do**
      Compute $v_k^{j,l}$ by solving

$$\frac{v_k^{j,l} - v^{j,l-1}}{2^{j-1} c_j \Delta t} = \sum_{s=1}^{c_{j,l}} A_{k,s}^{j,l}(t^n) * v_s^{j,l-1} + b_k^{j,l}(t^n) + S_k^{j,l}(v_k^{j,l}), \tag{4.25}$$

      where $c_{j,l}$ is defined in (4.20) and (4.21).
    end for
    Compute $v^{j+1,l}$ as

$$v^{j+1,l} = \frac{1}{c_j}\sum_{k=1}^{c_j} v_k^{j,l}.$$

  end for
end for

Algorithm 4.2. *Continued.*

Set $\bar{u}^{J,L_J} = v^{J,L_J}$ and $u_k^{J,L_J} = v_k^{J,L_J}$ for $k = 1, 2, \dots c_J$.
for $j = J-1, \dots, 1$ **do**
  Set $u^{j,0} = \bar{u}^{j+1,L_{j+1}}$ and $u_k^{j,0} = u_k^{j+1,L_{j+1}}$, $k = 1, \dots, c_{j+1}$,
  for $l = 1, \dots, L_j$ **do**
    for $k = 1, 2, \cdots c_j$ **do**
      Compute $u_k^{j,l}$ by solving

$$\frac{u_k^{j,l} - u^{j,l-1}}{2^j c_j \Delta t} = \sum_{s=1}^{\widetilde{c}_{j,l}} \widetilde{A}_{k,s}^j(t^n) * u_s^{j,l-1} + \widetilde{b}_k^j(t^n) + \widetilde{S}_k^j(u_k^j), \tag{4.26}$$

      where $\widetilde{c}_{j,l}$ is defined in (4.20) and (4.21).
    end for
    Compute $u^{j,l}$ as

$$u^{j,l} = \frac{1}{c_j} \sum_{k=1}^{c_j} u_k^{j,l}.$$

  end for
  Compute $\bar{u}_k^{j,L_j}, \bar{u}^{j,L_j}$ as

$$\bar{u}_k^{j,L_j} = \frac{1}{2} u_k^{j,L_j} + \frac{1}{2} v_k^{j,L_j}, \quad \bar{u}^{j,L_j} = \frac{1}{c_j} \sum_{k=1}^{c_j} \bar{u}_k^{j,L_j} \tag{4.27}$$

end for
Compute U^{n+1} by solving

$$\frac{U^{n+1} - \bar{u}^{1,L_1}}{\Delta t} = \sum_{s=1}^{c_1} A_s^*(t^n) * u_s^{1,L_1} + b^*(t^n) + S^*(U^{n+1}). \tag{4.28}$$

The decomposed variables $A(\mathbf{x}, t), \widetilde{A}(\mathbf{x}, t), b(\mathbf{x}, t), \widetilde{b}(\mathbf{x}, t)$ and the operations S and $\widetilde{S}$ have forms similar to those variables and operations in Appendix D. Thus we can use the hybrid splitting method Algorithm D.2 to solve the control problem. The resulting scheme for updating U^n to U^{n+1} is summarized in Algorithm 4.2, where the dependency of the control variables on $\mathbf{x}$ is omitted. Note that Algorithm 4.2 is a special case of Algorithm D.2 by setting $M_j = M_{2J-j} = L_j$ for $1 \le j \le J$, $d_{j,m} = c_{j,m-1}, c_{j,m} = c_{2J-j,m} = c_j$ for $1 \le j \le J, 1 \le m \le L_j$. The architecture of Algorithm 4.2 is illustrated in Figure 3. The explanations of all indices for operators and variables of the left branch are summarized in Table 1.

Similar to (4.13), denote the collection of all parameters discussed in this subsection by θ_3 and denote

$$\mathcal{N}_3 : f \to H(f) \to U^1 \to \cdots \to U^N$$

as the mapping that maps f to U^N by applying Algorithm 4.2 for N times. Parameters θ_3 are learned by solving

$$\min_{\theta_3} \sum_{i=1}^{I} \mathcal{L}(\mathcal{N}_3(f_i, \theta_3), v_i). \tag{4.29}$$

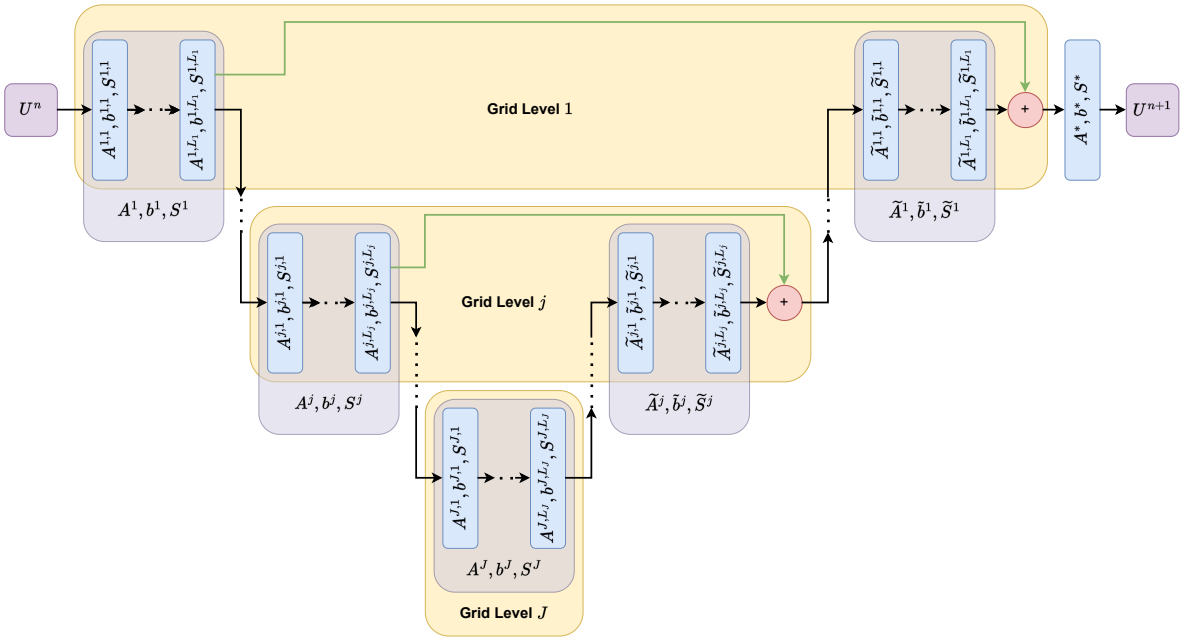

Figure 3. *An illustration of Algorithm 4.2.*

Remark 4.2. Similar to Remark 4.1, we can show that the structure of one iteration of Algorithm 4.2 is the same as a neural network. Take such a network as a block. The operator $\mathcal{N}_3$ applies Algorithm 4.2 N times, corresponding to chaining N such blocks sequentially. Thus $\mathcal{N}_3$ can be taken as a large network and θ_3 is the collection of all trainable parameters (control variables). Problem (4.29) can be solved by training this network to determine θ_3.

Connections to neural networks. Algorithm 4.2 gives a mathematical explanation of many popular encoder-decoder-based network architectures. In addition to the connections between Algorithm 4.1 with neural networks, Algorithm 4.2 has one more connection: the number of sequential splitting for each grid level in Algorithm 4.2 corresponds to the number of layers for each data resolution in an encoder-decoder-based network. Compared to Algorithm 4.1, this additional connection makes Algorithm 4.2 a mathematical explanation of a more general class of encoder-decoder-based networks.

Note that Algorithm 4.1 is a special case of Algorithm 4.2 by setting $L_j = 1$ for $j = 1, \ldots, J$. In the rest of this paper, we focus on Algorithm 4.2.

We summarize our notation in Table 2.

5. Numerical discretization.

Algorithm 4.2 is semiconstructive as we still need to discretize u, v in space and solve the semi-implicit problems (4.25), (4.26), and (4.28). We discuss spatial discretization in this section.

5.1. On the choices of $S, \widetilde{S}$.

In Algorithm 4.2, one needs to solve (4.25), (4.26), and (4.28), which includes components of $S, \widetilde{S}$. We discuss the choices of $S, \widetilde{S}$ and present how to solve (4.25), (4.26), and (4.28) in the next subsection.

Table 2
Summary of notation.

		Index	
j	grid level	l	sequential splitting
k	parallel splitting	s	output from the previous substep
n	index of time step		
		Variable and operator	
Δt	time step size	t^n	discrete time
U^n	numerical solution at t^n	ε	weight for the regularized softmax operator
J	total grid levels	L_j	total number of sequential splitting at grid level j
c_j	number of parallel splitting at each sequential splitting of grid level j in the left branch	$\widetilde{c}_j$	number of parallel splitting at each sequential splitting of grid level j in the left branch
$v_k^j, v_k^{j,l}$	intermediate variable in the left branch	$u_k^j, u_k^{j,l}$	intermediate variable in the right branch
$A_{k,s}^j, A_{k,s}^{j,l}$	convolution kernel in the left branch	$\widetilde{A}_{k,s}^j, \widetilde{A}_{k,s}^{j,l}$	convolution kernel in the right branch
$b_k^j, b_k^{j,l}$	bias in the left branch	$\widetilde{b}_k^j, \widetilde{b}_k^{j,l}$	bias in the right branch
$S_k^j, S_k^{j,l}$	nonlinear operator in the left branch	$\widetilde{S}_k, \widetilde{S}_k^j$	nonlinear operator in the right branch
A_s^*, b^*, S^*	control variables and operators at the final substep		
$H(f)$	initial condition	$\theta_1, \theta_2, \theta_3$	collection of learnable parameters
G_σ	Gaussian kernel with variance σ^2		

According to (4.2), $S+\widetilde{S}$ consists of two terms. (i) The first term is $-\varepsilon \ln \frac{u}{1-u}$, which results from the regularized softmax. This term enforces u to be between 0 and 1. (ii) The second term $-\eta G_\sigma * (1-2u)$ results from the length penalty, which promotes smooth boundaries in the segmentation u. As all substeps except the last substep in Algorithm 4.2 are used to extract and reconstruct important features of the input image, we set

$$S_k^{j,l} = -\frac{(2^{j-1}c_j)^{-1}}{\kappa}\varepsilon \ln \frac{u}{1-u}, \quad \widetilde{S}_k^{j,l} = -\frac{(2^j c_j)^{-1}}{\kappa}\varepsilon \ln \frac{u}{1-u} \tag{5.1}$$

with the normalization term

$$\kappa = \sum_{j=1}^{J}\sum_{l=1}^{L_j}\sum_{k=1}^{c_j}(2^{j-1}c_j)^{-1} + \sum_{j=1}^{J-1}\sum_{l=1}^{L_j}\sum_{k=1}^{c_j}(2^j c_j)^{-1} + 1. \tag{5.2}$$

At the last substep, the segmentation is reconstructed, for which we will use the length penalty term to regularize the segmentation. We set

$$S^* = -\frac{1}{\kappa}\varepsilon \ln \frac{u}{1-u} - \eta G_\sigma * (1-2u). \tag{5.3}$$

In (5.1), the factor on the right-hand side is chosen based on two considerations: (i) the functions S and $\widetilde{S}$ should satisfy (4.2), and (ii) the effect of $-\varepsilon \ln \frac{u}{1-u}$ is expected to be

the same for every intermediate variable $v_k^{j,l}$ and $u_k^{j,l}$, i.e., it is not affected by the factors on the left-hand side of (4.25) and (4.26), which result from parallel splitting and the relaxation step.

5.2. On the solution to (4.25), (4.26), and (4.28). In (4.25) and (4.26) when $l=1$, the problems involve computations between functions discretized on different grid levels. We will first convert all functions to $\mathcal{T}^j$ and then solve the problems.

For (4.25) when $l=1$, $v^{j,0}=v^{j-1,L_j}$ and $v_k^{j,0}=v_k^{j-1,L_j}$ are defined on grid level $j-1$. We use the average pooling (B.5) or the max pooling (B.6) to downsample these functions to grid level j and then assign them to $v^{j,0}$ and $v_k^{j,0}$. In our experiment, (B.6) is used.

For (4.26) when $l=1$, $u^{j,0}=\bar{u}^{j+1,L_j}$ and $u_k^{j,0}=\bar{u}_k^{j+1,L_j}$ are defined on grid level $j+1$. We use the piecewise constant upsampling (B.4) to upsample these functions to grid level j and then assign them to $u^{j,0}$ and $u_k^{j,0}$.

Observe that (4.25), (4.26), and (4.28) are in the form of

$$\frac{u-u^*}{\gamma\Delta t}=\sum_{s=1}^{c}\widehat{A}_s*u_s^*+\widehat{b}+\widehat{S}(u), \tag{5.4}$$

where γ is some constant, c is some integer, $u^*=\frac{1}{c}\sum_{s=1}^{c}u_s^*$ for some functions u_s^*'s, $\widehat{A}_s$'s are some convolution kernels, $\widehat{b}$ is some bias function, and $\widehat{S}$ is some nonlinear function. The solution to (5.4) can be computed using two substeps:

$$\begin{cases}\bar{u}=u^*+\gamma\Delta t\left(\sum_{s=1}^{c}\widehat{A}_s*u_s^*+\widehat{b}\right),\\ u=(I_{\mathrm{id}}-\gamma\Delta t\widehat{S})^{-1}(\bar{u}),\end{cases} \tag{5.5}$$

where I_{id} denotes the identity operator, and $(I_{\mathrm{id}}-\gamma\,\Delta t\widehat{S})^{-1}$ is the resolvent operator of $(I_{\mathrm{id}}-\gamma\Delta t\widehat{S})$. When $\widehat{S}$ is a nonlinear operator, (5.5) is the building block of neural networks: the first substep is a linear layer, and the second substep corresponds to some activation function. Therefore, Algorithm 4.2 is a convolutional network with $(I_{\mathrm{id}}-\Delta tS)^{-1}$ being the activation function.

In (5.5), there is no difficulty in solving for $\bar{u}$ as it is an explicit step. For u in (5.5), when $\widehat{S}=S_k^{j,l},\widetilde{S}_k^{j,l}$ or S^* as in (5.1) and (5.3), we need to solve a problem of the following form:

$$\frac{u-\bar{u}}{C_1\Delta t}+C_2G_\sigma*(1-2u)=-\varepsilon\ln\frac{u}{1-u}, \tag{5.6}$$

where

$$C_1=1/\kappa,\ C_2=0 \quad \text{for } \widehat{S}=S_k^{j,l},\widetilde{S}_k^{j,l},$$

$$C_1=1/\kappa,\ C_2=\eta \quad \text{for } \widehat{S}=S^*.$$

We use a fixed point method to solve (5.6).

First initialize $p^0=\bar{u}$. From p^k, we update p^{k+1} by solving

$$\frac{p^k-\bar{u}}{C_1\Delta t}+C_2\eta G_\sigma*(1-2p^k)=-\varepsilon\ln\frac{p^{k+1}}{1-p^{k+1}}, \tag{5.7}$$

for which we have the closed-form solution

$$p^{k+1} = \mathrm{Sig}\left(-\frac{1}{\varepsilon}\left(\frac{p^k - \bar{u}}{C_1 \Delta t} + C_2 \eta G * (1 - 2p^k)\right)\right), \tag{5.8}$$

where $\mathrm{Sig}(x) = \frac{1}{1+e^{-x}}$ is the sigmoid function. By repeating (5.8) so that p^{k+1} converges to some function p^*, we set $u = p^*$.

As U^n evolves during iterations, it is not necessary to repeat (5.8) until convergence for every intermediate variable. Instead, one may only use a few steps of (5.8). In particular, if only two steps of (5.8) are used, the resulting formula is a relaxed and regularized version of the sigmoid activation function.

Remark 5.1. In (5.6), if $C_2 = 0$, i.e., there is no length penalty term, at least two steps of (5.8) should be used. In fact, when $C_2 = 0$ and we initialize $p^0 = \bar{u}$, the first step is trivial as one always has $p^1 = 0.5$.

6. Relations of Algorithm 4.2 to existing networks.

Encoder-decoder-based neural networks have been widely used in image segmentation, such as UNet [58], UNet++ [75], and SegNet [3]. In sections 4.2 and 4.4, we showed that Algorithm 4.2 has an encoder-decoder architecture and its analogy to neural networks. In this section, we show that with minor modifications, Algorithm 4.2 can recover the architectures of most encoder-decoder-based neural networks, just with different activation functions.

We order the image resolution (corresponding to the grid levels in Algorithm 4.2) from the finest to coarsest by 1 to J, where J is the total levels of image resolution. With this ordering, level 1 corresponds to the finest resolution and level J corresponds to the coarsest resolution.

6.1. Relations to general convolutional neural networks.

For a convolutional neural network, the building block is

$$\begin{cases} \bar{v}_k = \sum_{s=1}^{c} W_{k,s} * v_s^* + b_k, \\ v_k = \chi(\bar{v}_k), \end{cases} \tag{6.1}$$

where v_s^*'s are the outputs from the previous layer, v_k is the output of the kth channels of the current layer, $W_{k,s}$'s are convolutional kernels, and χ is an activation function. For Algorithm 4.2, the building block is (5.4), which is solved by (5.5). In fact, (6.1) and (5.5) have the same form. With the proper choice of σ, they are equivalent to each other.

In the following, we show that if we choose $\chi = (I_{\mathrm{id}} - \gamma \Delta t \widehat{S})^{-1}$, $v_s^* = u_s^*$, and the proper convolution kernels $W_{k,s}$'s, we have $v_k = u$.

In the first equation of (5.5), substitute the expression of u^*, and we have

$$\bar{u} = \frac{1}{c}\sum_{s=1}^{c} u_s^* + \gamma \Delta t \left(\sum_{s=1}^{c} \widehat{A}_s * u_s^* + \widehat{b}\right) = \sum_{s=1}^{c}\left(\frac{1}{c}\mathbb{1} + \gamma \Delta t \widehat{A}_s\right) * u_s^* + \gamma \Delta t \widehat{b}, \tag{6.2}$$

where $\mathbb{1}$ denotes the identity kernel satisfying $\mathbb{1} * g = g$ for any function g. Set

$$W_{k,s} = \frac{1}{c}\mathbb{1} + \gamma \Delta t \widehat{A}_s, \quad b_k = \widehat{b}. \tag{6.3}$$

We have $\bar{v}_k = \bar{u}$. By choosing $\chi = (I_{\mathrm{id}} - \gamma \Delta t \widehat{S})^{-1}$, we have $v_k = u$. Essentially, Algorithm 4.2 and convolutional neural networks have the same building block. Most encoder-decoder-based neural networks are instances of Algorithm 4.2, i.e., an operator-splitting scheme for some control problems.

6.2. Relations to networks with skip pathways. Some popular encoder-decoder networks have skip pathways to improve performance, such as UNet [58] and UNet++ [75]. In Algorithm 4.2, skip pathways are realized using the relaxation step (4.27). The implementation of skip pathways in Algorithm 4.2 may be slightly different from that in UNet and other networks. While one can make minor modifications to Algorithm 4.2 to exactly recover the architecture of these networks, here for simplicity, we only address the modification needed to recover UNet's architecture. For other architectures, one can revise Algorithm 4.2 in a similar manner.

In UNet at resolution j, the skip pathway copies features from the encoder part to the upsampled features from resolution $j+1$ in the decoder part, which is just before computations at resolution j in the decoder part are conducted. In our Algorithm 4.2, the intermediate variables v_k^{j,L_j}'s at grid level j in the encoder part are added to the output of grid level j in the decoder part, which is after all computations at grid level j in the decoder part are finished.

To recover the architecture of UNet, we replace the initialization of $u^{j,0}$ and $u_k^{j,0}$ by

$$u_k^{j,0} = \begin{cases} \frac{1}{2}u_k^{j+1,L_{j+1}} + \frac{1}{2}v_k^{j,L_j} & \text{for } 1 \le k \le \min\{c_j, c_{j+1}\}, \\ \frac{1}{2}u_k^{j+1,L_{j+1}} + \frac{1}{2}v^{j,L_j} & \text{for } c_j < k \le c_{j+1} \text{ if } c_{j+1} > c_j, , \\ \frac{1}{2}u^{j+1,L_{j+1}} + \frac{1}{2}v_k^{j,L_j} & \text{for } c_{j+1} < k \le c_j \text{ if } c_j > c_{j+1}, \end{cases} \quad u^{j,0} = \frac{1}{c_j}\sum_{j=1}^{c_j} u_k^{j,0}, \tag{6.4}$$

replace (4.26) by

$$\frac{u_k^{j,l} - u^{j,l-1}}{2^{j-1}c_j\tau} = \sum_{s=1}^{\widetilde{c}_{j,l}} \widetilde{A}_{k,s}^j(t^n) * u_k^{j,l-1} + \widetilde{b}_k^j(t^n) + \widetilde{S}_k^j(u_k^j), \tag{6.5}$$

and remove (4.27).

One can show that the modified algorithm is an operator-splitting scheme for the control problem (4.12) with proper choices of convolution kernels, biases, and nonlinear operators, and it has the same architecture as UNet. Putting this property with the one discussed in section 6.1, we see that the modified algorithm is nothing else but UNet with different activation functions.

Compared to UNet, UNet++ has additional nested, dense skip pathways. For UNet, the skip pathways are realized by using relaxation steps. To realize the additional skip pathways in UNet++, one needs to add more relaxation steps correspondingly. With the proper

decomposition of $A, \widetilde{A}, b, \widetilde{b}, S, \widetilde{S}$ and the introduction of new relaxations, one can easily modify Algorithm 4.2 so that it has the same architecture with UNet++.

6.3. Relations to networks without skip pathways. SegNet [3] is another popular encoder-decoder network for image segmentation. Compared to UNet and UNet++, SegNet does not have any skip pathways and only contains an encoder and a decoder. To recover the architecture of SegNet, we only need to make two changes to Algorithm 4.2: (i) in the relaxation step (4.27), directly set $\bar{u}_k^{j,L_j} = u_k^{j,L_j}$; (ii) remove the factors 2^{j-1} in (4.25) and 2^j in (4.26). The revised algorithm has the same architecture as SegNet and one can show that it is an operator-splitting scheme for a control problem.

6.4. Differences of Algorithm 4.2 from other networks. Even though Algorithm 4.2 has an architecture that is similar to some existing encoder-decoder-based network, there are still some differences. Specifically, it is different from other networks in three aspects:

1. Algorithm 4.2 is derived from the classical Potts model, which has a strong mathematical background. In PottsMGNet, control variables are introduced to approximate the function g in (2.4), which is assumed to be dependent on the input image f. Thus we intentionally design some control variables that conduct operations on f, as shown in (7.1). This is different from existing models, which only directly operate on f at the input.
2. Algorithm 4.2 is an approximate gradient flow of the original Potts model. By controlling the time step in the discretized scheme, the evolution of the segmentation mask will not change too much from layer to layer, which in some sense ensures the stability of the network against noise. Even though the time step factor can be absorbed by convolution kernels, we cannot control what will be learned and the learned kernels may not have this property, i.e., small scales. We believe explicitly using this time step factor helps improve the performance.
3. Compared to SOTA models, our model explicitly uses the perimeter penalty at several layers, which helps improve the robustness against noise. Even though such a regularizer can be realized by a convolution layer, we believe explicitly using it helps improve the performance.

As will be demonstrated in section 7, these differences make PottsMGNet outperform other models in dealing with images with various levels of noise.

We would like to mention that the STD model introduced in [47] is also derived from the Potts model and uses a perimeter penalty as a regularizer. The STD model appends some basic networks, such as DeepLabV3+ [14], by STD layers, which are fixed-point iterations solving an optimization problem with the perimeter penalty. However, the basic network is still a black box whose architecture is not explainable. PottsMGNet designs each step (layer) using the multigrid method and operator-splitting and provides mathematical explanations for each layer of the resulting network. The first two differences mentioned above also apply to the STD model.

7. Experiments. We compare Algorithm 4.2 with popular networks. We show that when using a single network to segment images with various noise levels, Algorithm 4.2 has better performance. Our PyTorch code is available at https://github.com/liuhaozm/PottsMGNet.

7.1. Implementation details. The building block of Algorithm 4.2 is (5.5). We call the first and second steps the convolution step and activation step, respectively. Our implementation details are as follows:

- *Initial condition* $H(f)$: We set $H(f)$ as a convolution layer with f:
$$H(f) = \mathrm{Sig}\left(\sum_{k=1}^{3} B_{0,k} * f^k\right)$$
for $f = \{f^1, f^2, f^3\}$.
- *Choices of* b: In the control problem for Potts model (3.1), there are two control variables: W and d. Variable W is a kernel function used to compute the convolution with u. Variable d is independent of u and not directly applied to u. While in the Euler–Lagrange equation of Potts model (2.3), there is a term g which is independent of u and only depends on the input image $f = \{f^1, f^2, f^3\}$, it is natural to set d, and thus b and $\widetilde{b}$ (see (4.4)), as a function of f. In our implementation, we set
$$b_k^{j,l} = \begin{cases} \sum_{s=1}^{3} B_{k,s}^{j,l} * f^s & \text{if } l = 1, \\ \beta_k^{j,l} & \text{if } l > 1, \end{cases} \quad \widetilde{b}_k^{j,l} = \begin{cases} \sum_{s=1}^{3} \widetilde{B}_{k,s}^{j,l} * f^s & \text{if } l = 1, \\ \widetilde{\beta}_k^{j,l} & \text{if } l > 1 \end{cases} \tag{7.1}$$
for some learnable kernels $B_{k,s}^{j,l}, \widetilde{B}_k^{j,l}$ defined on grid level j, and $\beta_k^{j,l}, \widetilde{\beta}_k^{j,l}$ being some bias constants to be learned.
- *Convolution step in* (5.5): The convolution step is an explicit step that can be easily implemented using discretized convolution.
- *Activation step in* (5.5): The activation step is implemented as (5.8) with two iterations. For simplicity, we set $C_1 = 1, C_2 = 0$ when computing $u_s^{k,l}$ (or $u_s^{k,l}$). At the last substep (of Algorithm 4.2), we set $C_1 = 1, C_2 = \lambda$.
- *Batch normalization*: To improve the stability of the training process, a batch normalization step is added before each activation step except for the last substep.
- *Downsampling and upsampling*: For downsampling and upsampling, maxpooling (B.6) and piecewise constant upsampling (B.4) are used, respectively.
- *Number of time steps*: One iteration of Algorithm 4.2 is one time step. In our implementation, four steps are used.
- *Skip pathways*: We replace the weight $1/2$ in (4.27) by parameters that will be learned in training.
- *Initial condition*: We set $U^0 = \sum_{s=1}^{3} W_s^0 * f_s$ for some learnable kernel W_s^0's.
- *Network hyper parameters*: Without specification, we test Algorithm 4.2 with five grid levels ($J = 5$) and the following hyper parameters:
$$\{L_1, L_2, L_3, L_3, L_5\} = \{3, 3, 3, 5, 5\}, \quad \{c_1, c_2, c_3, c_4, c_5\} = \{32, 32, 64, 128, 256\}.$$
In the activation function step (5.8), except for the parameters C_1, C_2 arising from the skip pathways, this step only depends on $\varepsilon\Delta t$ and η/ε. In our experiments, we set $\Delta t = 0.5, \varepsilon\Delta t = 1, \eta/\varepsilon = 40$, and $N = 4$ i.e., four time steps. We set $\sigma = 0.5$ in G_σ. For the kernel size in the convolution layers, we use 3×3 kernels in the first layer (initialization) and the coarsest grid level. In other layers, we use 5×5 kernels.

7.2. Models and datasets. Algorithm 4.2 is a mathematical explanation of encoder-decoder-based neural networks. We compare the performance of Algorithm 4.2 with popular encoder-decoder-based networks, such as Unet [58], UNet++ [75], DeepLabV3+ [14], and SegNet [3]. The implementation of UNet++ and DeepLabV3+ used the segmentation models PyTorch package [38].

We test all networks on two data sets: the cell segmentation dataset (CSD) [40] and the MSRA10K dataset [16]. CSD is used in the 2018 data science bowl competition. The stage 1 dataset contains 536 training images and 134 test images. We resize all images to a size of 96×96. MSRA10K contains 10,000 salient object images with manually annotated masks. We choose 800 images for training and 200 images for testing. We resize all images to a size of 128×192.

7.3. Training strategy. We aim to train our networks to be robust to several noise levels. If a network is trained on data with a certain noise level and tested on a data set with higher noise, the performance can be poor; but if the network is tested on a data set with lower noise, it will still provide good results. So we will train our network with a high noise level. To make the training more stable, we use a progressive training strategy: we first train the network on a clean data set and then gradually increase the noise level. We add Gaussian noise with standard deviation (SD) $\sigma \in \{0, 0.3, 0.5, 0.8, 1\}$. For each noise level, we use 500 epochs to train our network and use the trained parameters as initial values to train the network for the next noise level. In our experiments, we normalize all images so that all pixel values are between 0 and 1.

In our training, we tried two settings. In the first setting, for each noise level with SD=a, each pixel has a noise SD that is randomly generated from $[0, a]$. In the second setting, for each noise level with SD=a, all pixels have noise with SD=a. The comparisons of both settings are visualized in Figure 4, in which various models are tested on images with Gaussian

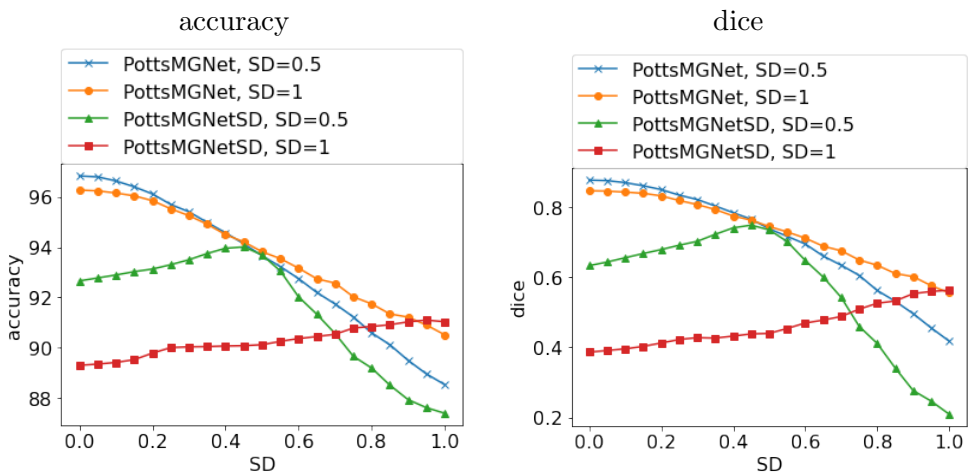

Figure 4. *Results on CSD. Comparisons of different training strategies. PottsMGNet is trained under the first setting, PottsMGNetSD is trained under the second setting, and SD=a denotes the model that are progressively trained with largest SD=a.*

noise varying from 0 to 1. In the figure, PottsMGNet is trained under the first setting, PottsMGNetSD is trained under the second setting, and SD=a denotes the model that is progressively trained with the largest SD=a, i.e., the training process stops at SD=a. For the second setting, if the model stops training at SD = 0.5, then the model has the highest test accuracy on noise images with SD around 0.5. Further training the model on higher noise only makes the test accuracy better when tested on higher noise but makes the test accuracy worse on low noise levels. While the first setting always gives robust results, similar phenomena are observed for other models.

In the following comparisons, the first setting is always used with the highest SD being 1.

7.4. Robustness to hyper parameters. In this section, we show that PottsMGNet is not sensitive to hyper parameters. We compare the performances of PottsMGNet with hyper parameters specified in Table 3.

In Table 3, Model 1 is the default model specified in section 7.1. With the training strategy described in section 7.3 and for CSD, the testing accuracy and dice score of all models are presented in Figure 5. All models have a similar performance, implying that PottsMGNet is not sensitive to hyper parameters.

Table 3
PottsMGNet with different hyper parameters considered in section 7.4.

	N	$\{L_1, L_2, L_3, L_3, L_5\}$	$\{c_1, c_2, c_3, c_4, c_5\}$
Model 1	4	$\{3,3,3,5,5\}$	$\{32,32,64,128,256\}$
Model 2	3	$\{3,3,3,5,5\}$	$\{32,32,64,128,256\}$
Model 3	4	$\{3,3,3,3,3\}$	$\{32,32,64,128,256\}$
Model 4	4	$\{5,5,5,5,5\}$	$\{32,32,64,128,256\}$
Model 5	4	$\{3,3,3,5,5\}$	$\{32,64,128,256,256\}$

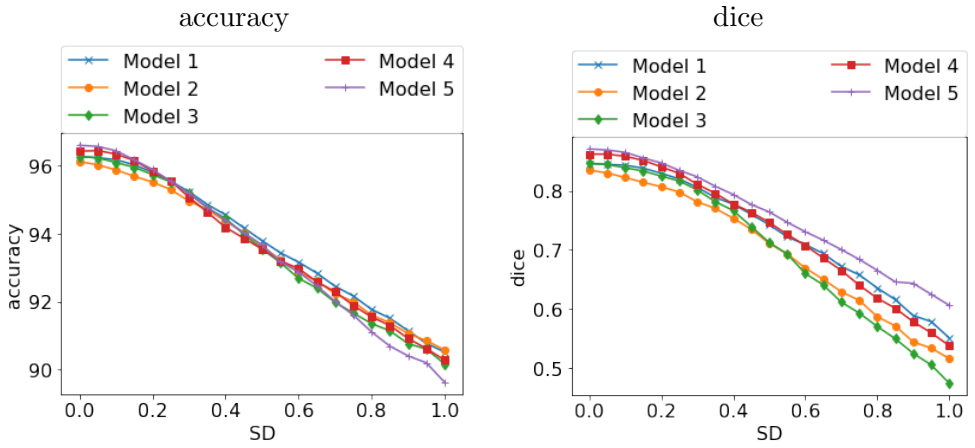

Figure 5. *Results on CSD. Comparisons of different PottsMGNet with hyper parameters specified in Table 3.*

Table 4

Results on CSD. Comparisons of accuracy and dice score of different models for noise SD=0, 0.5, 0.8, 1.

	PottsMGNet	UNet	UNet++	DeepLab V3+	SegNet
			SD = 0		
Accuracy	96.27%	94.24%	94.10%	93.88%	94.48
Dice	0.8460	0.7401	0.7332	0.7231	0.7474
			SD = 0.5		
Accuracy	93.85%	91.11%	91.12%	91.87%	91.66
Dice	0.7445	0.5659	0.5571	0.5996	0.5591
			SD = 0.8		
Accuracy	91.78%	89.73%	89.08%	90.50%	90.31
Dice	0.6353	0.4938	0.4652	0.4988	0.4622
			SD = 1		
Accuracy	90.50%	89.10%	87.48%	89.79%	89.57
Dice	0.5554	0.4625	0.4224	0.4582	0.4087

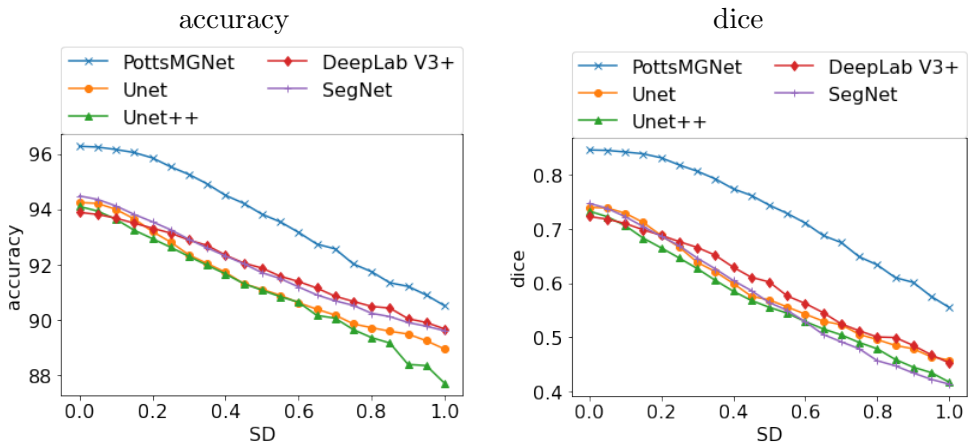

Figure 6. *Results on CSD. Comparisons of accuracy and dice score of different models for noise SD varying from 0 to 1.*

7.5. CSD results. On CSD, we test the trained models on several noise levels. The comparisons with SD = 0, 0.5, 0.8, and 1 are shown in Table 4. Comparisons on more noise levels are visualized in Figure 6. We observe that for all noise levels, PottsMGNet always provides the highest accuracy and dice score. Some sample comparisons are presented in Figure 7. The columns from left to right correspond to noise SD = 0, 0.3, 0.5, and 0.7. Even with very high noise (SD = 0.7), PottsMGNet still identifies a large portion of cells.

7.6. MSRA10K. For MSRA10K, we train all models with noise SD $\{0, 0.3, 0.5, 0.8\}$. For each noise level, 500 epochs are used. We test the trained models on several noise levels. The comparisons with SD = 0, 0.5, and 0.8 are shown in Table 5. Comparisons on more noise levels are visualized in Figure 8. Similar to our observation for CSD, for most noise levels,

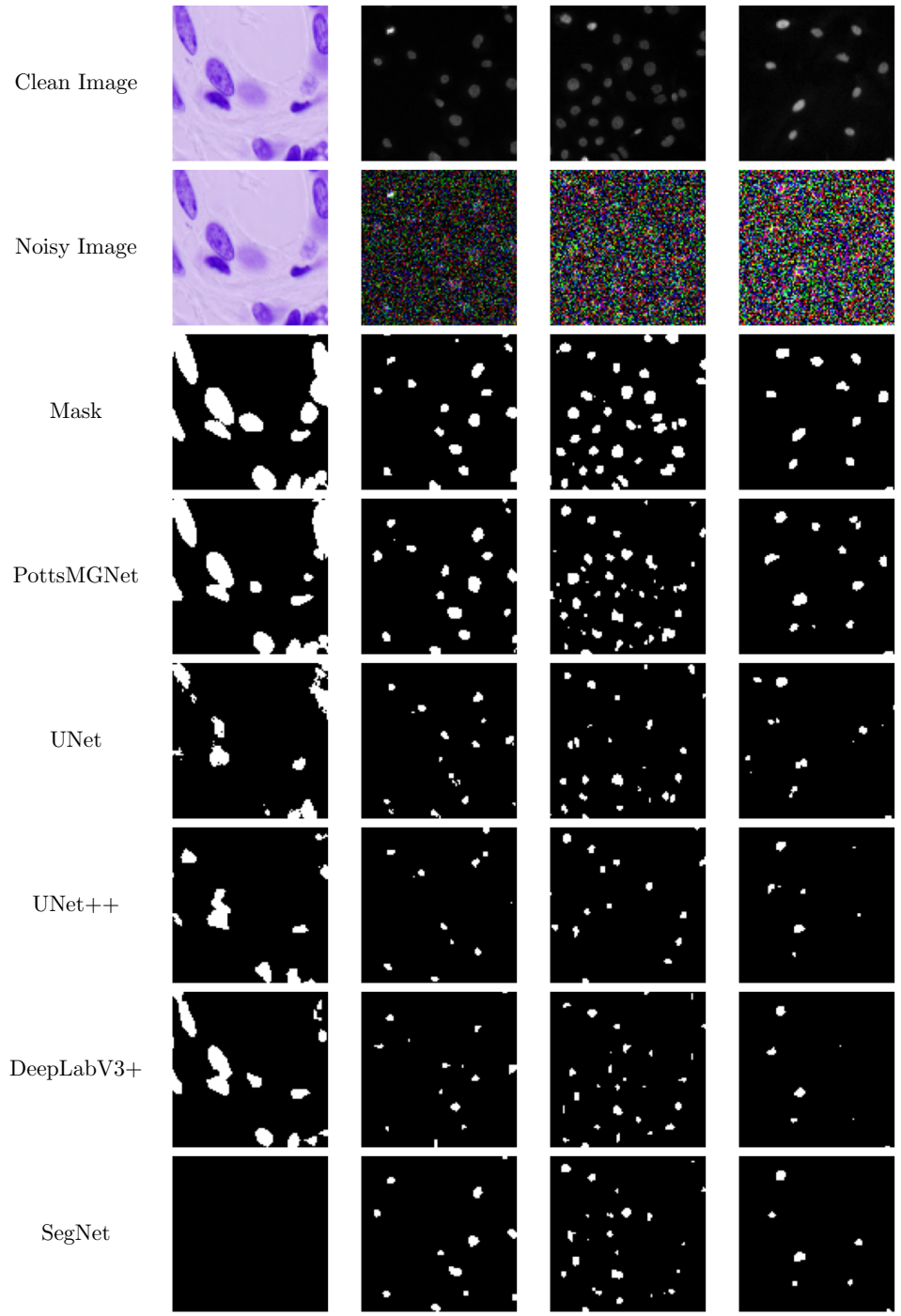

Figure 7. *Results on CSD. Examples of results by different models. Columns* 1–4 *correspond to noise SD*=0, 0.3, 0.5, 0.7.

Table 5

Results on MSRA10K. Comparisons of accuracy and dice score of different models for noise SD=0, 0.5, 0.8.

	PottsMGNet	UNet	UNet++	DeepLab V3+	SegNet
			SD = 0		
Accuracy	93.28%	92.14%	92.49%	92.47%	92.20
Dice	0.8417	0.8012	0.8190	0.8170	0.8108
			SD = 0.5		
Accuracy	92.69%	91.83%	91.77%	91.96%	91.99
Dice	0.8278	0.7976	0.8016	0.8074	0.8071
			SD = 0.8		
Accuracy	91.46%	90.94%	90.28%	90.89%	91.23
Dice	0.7969	0.7726	0.7728	0.7802	0.7907

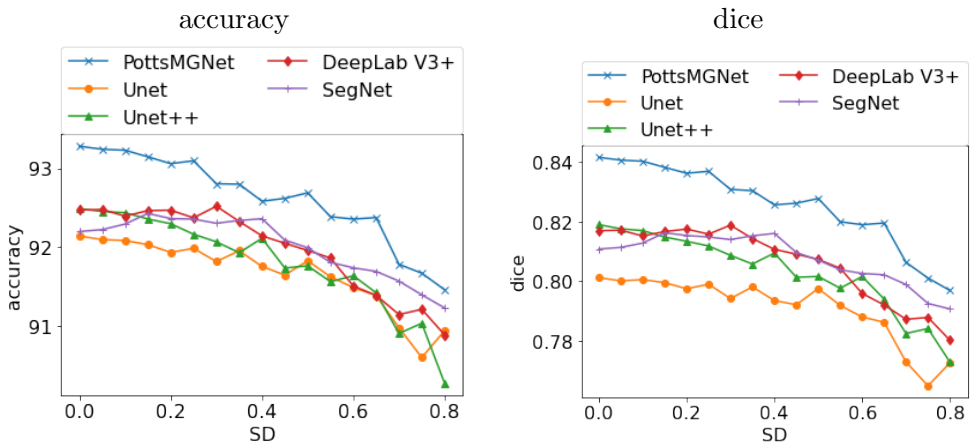

Figure 8. *Results on MSRA10K. Comparisons of accuracy and dice score of different models for noise SD varying from 0 to 1.*

PottsMGNet always provides the highest accuracy and dice score. Some sample comparisons are presented in Figure 9. The columns from left to right correspond to noise SD = 0, 0.3, 0.5, and 0.8. In all examples, PottsMGNet can better segment the target object. The segmented images are presented in Figure 10.

8. Conclusion. In this paper we propose the PottsMGNet algorithm for image segmentation. Starting from the two-phase Potts model, a control problem is considered. The proposed Potts net is a first-order operator-splitting algorithm that solves the control problem. The Potts net has two ingredients: (i) it is based on a novel hybrid splitting strategy for initial value problems, and (ii) it uses a multigrid idea to split all computations into several substeps according to the grid level. The PottsMGNet essentially is a neural network. From the algorithmic perspective, it provides mathematical explanations for most encoder-decoder type neural networks for image segmentation: these networks are operator-splitting schemes for some control problems. Our numerical experiments show that PottsMGNet is more robust to different noise levels compared to existing networks.

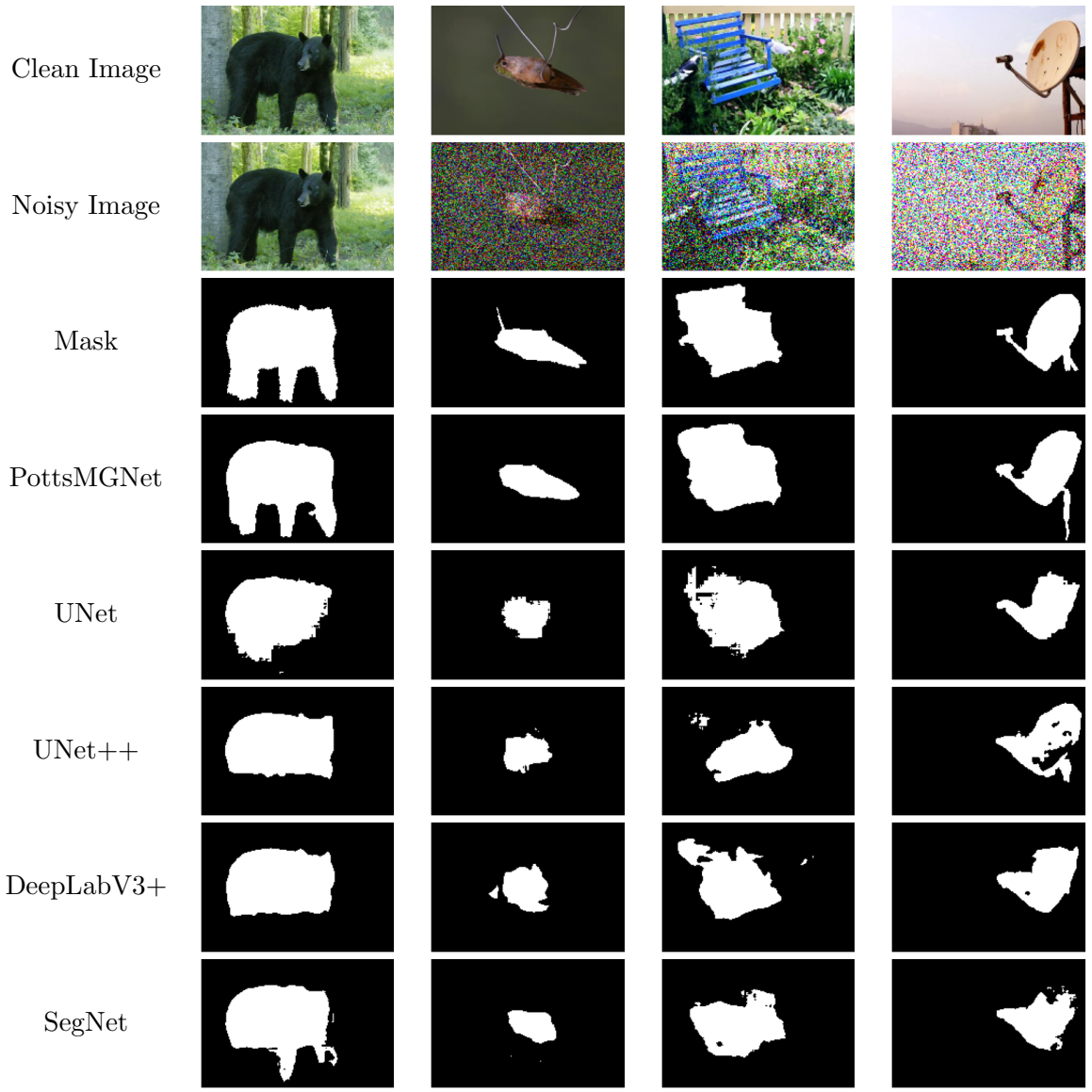

Figure 9. *Results on MSRA10K. Examples of results by different models. Columns* 1–4 *correspond to noise SD*=0, 0.3, 0.5, 0.8.

Appendix A. Gradient flow for the Potts model. The Potts model is a popular approach used in image segmentation and classification tasks with ideas originating from [55]. It extends the Ising model to multiple states, allowing it to be used for classification tasks [25, 9]. We discuss the derivation of the gradient flow of the Potts model in a multiphase setting. The flow for the two-phase model used in (2) is a special case.

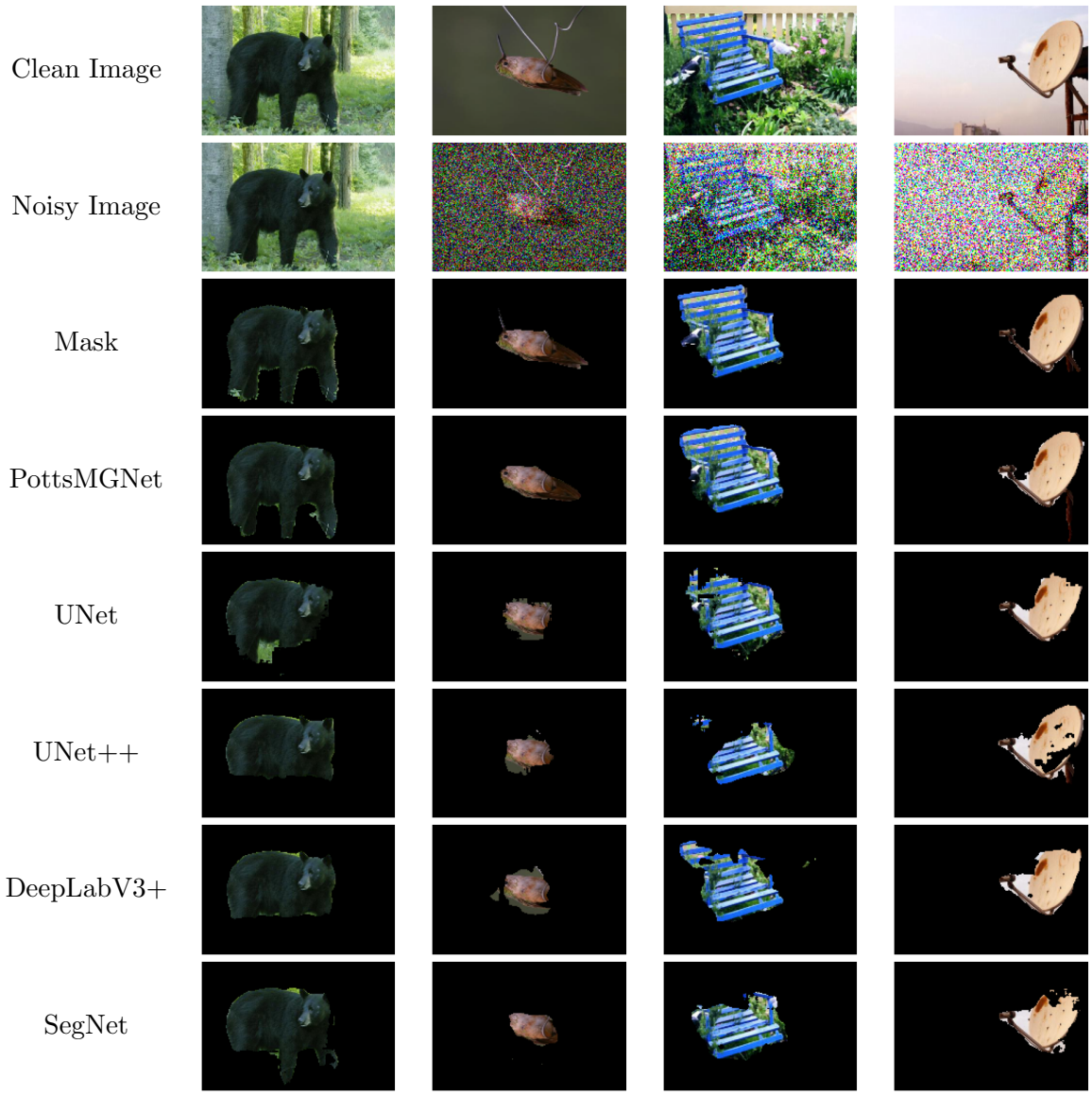

Figure 10. *Results on MSRA10K. Segmented images in Figure 9.*

Let Ω be the image domain. The continuous K–phase Potts model is in the form of [71, 63]

$$\text{(A.1)} \qquad \begin{cases} \min\limits_{\Omega_k,\ k=1,2\cdots K} \sum\limits_{k=1}^{K} \left(\int_{\Omega_k} f_k(\mathbf{x}) d\mathbf{x} + \lambda |\partial \Omega_k| \right), \\ \cup_{k=1}^{K} \Omega_k = \Omega,\ \ \Omega_{k_1} \cap \Omega_{k_2} = \emptyset \quad \forall k_1 \neq k_2, \end{cases}$$

where $|\partial\Omega_k|$ is the perimeter of $\partial\Omega_k$, $\lambda \geq 0$ is a given constant, and f_k's are some nonnegative given functions. A popular choice of f_k is $f_k(\mathbf{x}) = (f(\mathbf{x}) - c_k)^2/\alpha$ in which α is a scaling parameter and c_k is the estimated mean density of $f(\mathbf{x})$ on Ω_k. With such a choice, the Potts

model (A.1) reduces to the piecewise Mumford–Shah functional, which is also known as the Chan–Vese model [12].

For any set $\Omega_k \subset \Omega$, define the indicator function as

$$\mathbf{1}_{\Omega_k}(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{x} \in \Omega_k, \\ 0 & \text{if } \mathbf{x} \notin \Omega_k. \end{cases} \tag{A.2}$$

If we set $\mathbf{v} = (v_1, v_2, \ldots, v_K)$, $v_k(\mathbf{x}) = \mathbf{1}_{\Omega_k}(\mathbf{x})$, $\mathbf{f} = (f_1, f_2, \ldots, f_K)$, then solving the Potts model (A.1) is equivalent to solving

$$\min_{\mathbf{v}(\mathbf{x}) \in \widehat{\mathcal{S}}} \left[\int_\Omega \mathbf{v}(\mathbf{x}) \cdot \mathbf{f}(\mathbf{x}) d\mathbf{x} + \sum_{k=1}^{K} |\partial \Omega_k| \right], \tag{A.3}$$

where

$$\widehat{\mathcal{S}} = \left\{ \mathbf{v} : \mathbf{v} = (v_1, v_2, \ldots, v_K), \sum_{k=1}^{K} v_k(\mathbf{x}) = 1, v_k(\mathbf{x}) \in \{0, 1\} \right\}. \tag{A.4}$$

The minimizer of (A.3), denoted by $\mathbf{u}$, segments the image into K regions: region Ω_k is represented by $\{\mathbf{x} : u_k(\mathbf{x}) = 1\}$. The term for the perimeter regularization can be approximated by a smoothed version using threshold dynamics [1, 51, 22, 47]:

$$\sum_{k=1}^{K} |\partial \Omega_k| \approx \sum_{k=1}^{K} \sqrt{\frac{\pi}{\sigma}} \int_\Omega u_k(\mathbf{x}) (G_\sigma * (1 - u_k))(\mathbf{x}) d\mathbf{x}, \tag{A.5}$$

where G_σ is the Gaussian kernel $G_\sigma(\mathbf{x}) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{\|\mathbf{x}\|^2}{2\sigma^2}\right)$. One can show that the right-hand term in (A.5) converges to $\sum_{k=1}^{K} |\partial \Omega_k|$ as $\sigma \to 0$ [1, 51, 52]. Then the functional we are minimizing is

$$\min_{\mathbf{v} \in \widehat{\mathcal{S}}} \left[\int_\Omega \mathbf{v}(\mathbf{x}) \cdot f(\mathbf{x}) d\mathbf{x} + \frac{\lambda}{2} \int_\Omega \mathbf{v}(\mathbf{x}) \cdot (G_\sigma * (1 - \mathbf{v}))(\mathbf{x}) d\mathbf{x} \right], \tag{A.6}$$

where

$$G_\sigma * (1 - \mathbf{v}) = \begin{bmatrix} G_\sigma * (1 - v_1) & \cdots & G_\sigma * (1 - v_K) \end{bmatrix}^\top. \tag{A.7}$$

We then relax the constraint $\mathbf{v}(\mathbf{x}) \in \{0, 1\}$ to $\mathbf{v}(\mathbf{x}) \in [0, 1]$ and consider the following minimization problem with a smoothing parameter ε:

$$\min_{\mathbf{v} \in \mathcal{S}} \left[\int_\Omega \mathbf{v} \cdot \mathbf{f} d\mathbf{x} + \varepsilon \int_\Omega \mathbf{v} \cdot \ln \mathbf{v} d\mathbf{x} + \frac{\lambda}{2} \int_\Omega \mathbf{v}(\mathbf{x}) \cdot (G_\sigma * (1 - \mathbf{v}))(\mathbf{x}) d\mathbf{x} \right] \tag{A.8}$$

with

$$\mathcal{S} = \left\{ \mathbf{v} : \mathbf{v} = (v_1, v_2, \ldots, v_K), \sum_{k=1}^{K} v_k(\mathbf{x}) = 1, v_k(\mathbf{x}) \geq 0 \right\}. \tag{A.9}$$

If $\mathbf{u} \in \mathcal{S}$ is a minimizer of the above energy functional, we have the following lemma.

Lemma A.1. *When $\varepsilon \mapsto 0$, we have $\mathbf{u} \to \{0,1\}^K$ for any $\mathbf{x} \in \Omega$.*

Proof of Lemma A.1. Denote

$$F_\varepsilon(\mathbf{v}) = \int_\Omega \mathbf{v}\cdot\mathbf{f}d\mathbf{x} + \varepsilon\int_\Omega \mathbf{v}\cdot\ln\mathbf{v}d\mathbf{x} + \frac{\lambda}{2}\int_\Omega \mathbf{v}(\mathbf{x})\cdot(G_\sigma * (1-\mathbf{v}))(\mathbf{x})d\mathbf{x}. \tag{A.10}$$

Let $\{F_{\varepsilon(n)}\}_{n=1}^\infty$ be any sequence so that $\lim_{n\to\infty}\varepsilon(n) = 0$. Let $\mathbf{v}^*$ be any elements in $\mathcal{S}$ and $\{\mathbf{v}_n\}_{n=1}^\infty \subset \mathcal{S}$ be a sequence satisfying $\lim_{n\to\infty}\mathbf{v}_n = \mathbf{v}^*$. Since F_ε is continuous in ε and $\mathbf{v}$, we have

$$\lim_{n\to\infty} F_{\varepsilon(n)}(\mathbf{v}_n) = F(\mathbf{v}). \tag{A.11}$$

Therefore, $\{F_{\varepsilon(n)}\}_{n=1}^\infty$ Γ-converges to $\mathcal{F}$. Proving Lemma A.1 reduces to proving $\mathbf{u} \in \{0,1\}^K$ for the limiting case, i.e., when $\varepsilon = 0$.

Our proof borrows techniques from [67]. When $\varepsilon = 0$, we have

$$\begin{aligned}\mathbf{u} &= \min_{\mathbf{v}\in\mathcal{S}} \int_\Omega \mathbf{v}\cdot\mathbf{f}d\mathbf{x} + \frac{\lambda}{2}\int_\Omega \mathbf{v}\cdot G_\sigma * (1-\mathbf{v})d\mathbf{x} \\ &= \min_{\mathbf{v}\in\mathcal{S}} \sum_{k=1}^K \left[\int_\Omega v_k f_k d\mathbf{x} + \frac{\lambda}{2}\int_\Omega v_k G_\sigma * (1-v_k)d\mathbf{x}\right], \end{aligned} \tag{A.12}$$

which is concave in each v_k. In (A.8), we can represent $v_K = 1 - \sum_{k=1}^{K-1} v_k$. Denote $\widetilde{\mathbf{v}} = [v_1,\ldots,v_{K-1}]^\top$ and $\widetilde{\mathbf{f}} = [f_1 - f_K,\ldots,f_{K-1} - f_K]^\top$. Denote the set $\widetilde{\mathcal{S}} = \{\widetilde{\mathbf{v}} : 0 \le \widetilde{v}_k \le 1$ for $k = 1,\ldots,K-1$, and $0 \le \sum_{k=1}^{K-1}\widetilde{v}_k \le 1\}$. If $\mathbf{u}$ is a minimizer of (A.12), then $\widetilde{\mathbf{u}}$ is a minimizer of

$$\min_{\widetilde{\mathbf{v}}\in\widetilde{S}} \sum_{k=1}^{K-1}\left[\int_\Omega \widetilde{v}_k(f_k - f_K)d\mathbf{x} + \frac{\lambda}{2}\int_\Omega \widetilde{v}_k G_\sigma * (1-\widetilde{v}_k) - \widetilde{v}_k G_\sigma * \left(\sum_{k=1}^{K-1}\widetilde{v}_k\right)d\mathbf{x}\right]. \tag{A.13}$$

We prove the lemma by contradiction. Assume $\widetilde{\mathbf{u}} \in \widetilde{\mathcal{S}}$ is not binary. There are two cases. *Case* 1: There exist a constant $c > 0$ and a set $\mathcal{K}$ on which

$$c < \widetilde{u}_{k^*} < 1 - c. \tag{A.14}$$

for some $1 \le k^* \le K-1$ and $\widetilde{u}_k = 0$ for $k \ne k^*$. Let δ be the indicator function of $\mathcal{K}$ and $\boldsymbol{\chi} = [\chi_1,\ldots,\chi_{K-1}]^\top$ such that $\chi_{k^*} = \delta$ and $\chi_k = 0$ for $k \ne k^*$. Then $\widetilde{\mathbf{u}} + t\boldsymbol{\chi} \in \widetilde{\mathcal{S}}$ for $|t| \le c$. We deduce that

$$\begin{aligned}\frac{d(\widetilde{\mathbf{u}} + t\boldsymbol{\chi})}{dt} &= \int_\Omega \delta(f_{k^*} - f_{K-1})d\mathbf{x} + \frac{\lambda}{2}\int_\Omega \delta G_\sigma * (1 - \widetilde{u}_{k^*} - t\delta) - \delta G_\sigma * (\widetilde{u}_{k^*} + t\delta)d\mathbf{x} \\ &\quad - \frac{\lambda}{2}\int_\Omega \delta G_\sigma * \left(t\delta + \sum_{k=1}^{K-1}\widetilde{u}_k\right) + \delta G_\sigma * (\widetilde{u}_{k^*} + t\delta)d\mathbf{x},\end{aligned}$$

and

$$\frac{d^2(\widetilde{\mathbf{u}}+t\boldsymbol{\chi})}{dt^2}=-2\lambda\int_\Omega \delta G_\sigma * \delta d\mathbf{x}<0,$$

which contradicts that $\widetilde{\mathbf{u}}$ is a minimizer.

Case 2. There exist a constant $c>0$ and a set $\mathcal{K}$ on which

$$c<\widetilde{u}_{k_1^*}<1-c, \quad c<\widetilde{u}_{k_2^*}<1-c \tag{A.15}$$

for some $1\le k_1^*,k_2^*\le K-1$ and $\widetilde{u}_k=0$ for $k\neq k_1^*,k_2^*$. Let δ be the indicator function of $\mathcal{K}$ and $\boldsymbol{\chi}=[\chi_1,\dots,\chi_{K-1}]^\top$ such that $\chi_{k_1^*}=\delta$, $\chi_{k_2^*}=-\delta$, and $\chi_k=0$ for $k\neq k_1^*,k_2^*$. Then $\widetilde{\mathbf{u}}+t\boldsymbol{\chi}\in\widetilde{\mathcal{S}}$ for $|t|\le c$.

We deduce that

$$\begin{aligned}\frac{d(\widetilde{\mathbf{u}}+t\boldsymbol{\chi})}{dt}=&\int_\Omega \delta(f_{k_1^*}-f_{K-1})-\delta(f_{k_2^*}-f_{K-1})d\mathbf{x}\\&+\frac{\lambda}{2}\int_\Omega \delta G_\sigma*(1-\widetilde{u}_{k_1^*}-t\delta)-\delta G_\sigma*(\widetilde{u}_{k_1^*}+t\delta)d\mathbf{x}\\&+\frac{\lambda}{2}\int_\Omega -\delta G_\sigma*(1-\widetilde{u}_{k_2^*}+t\delta)+\delta G_\sigma*(\widetilde{u}_{k_2^*}-t\delta)d\mathbf{x}\end{aligned}$$

and

$$\frac{d^2(\widetilde{\mathbf{u}}+t\boldsymbol{\chi})}{dt^2}=-2\lambda\int_\Omega \delta G_\sigma * \delta d\mathbf{x}<0,$$

which contradicts that $\widetilde{\mathbf{u}}$ is a minimizer.

In conclusion, $\widetilde{\mathbf{u}}$ must be binary and thus $\mathbf{u}$ must be binary. ∎

The Euler–Lagrangian equation of (A.8) reads as

$$\varepsilon(1+\ln\mathbf{u})+\lambda G_\sigma*(1-2\mathbf{u})+\mathbf{f}+\partial I_{\mathcal{S}}(\mathbf{u})\ni\mathbf{0}\quad\forall\mathbf{x}\in\Omega,$$

where $I_{\mathcal{S}}(\cdot)$ is the indicator function for set $\mathcal{S}$ defined as

$$I_{\mathcal{S}}(\mathbf{u})=\begin{cases}0 & \text{if } \mathbf{u}\in\mathcal{S},\\ \infty & \text{otherwise,}\end{cases}$$

and $\partial I_{\mathcal{S}}(\mathbf{u})$ is its subdifferential at $\mathbf{u}$. The gradient flow for this problem is

$$\begin{cases}\frac{\partial\mathbf{u}}{\partial t}+\varepsilon(1+\ln\mathbf{u})+\lambda G_\sigma*(1-2\mathbf{u})+\mathbf{f}+\partial I_{\mathcal{S}}(\mathbf{u})\ni\mathbf{0},\ (\mathbf{x},t)\in\Omega\times(0,T],\\ \mathbf{u}(\mathbf{x},0)=\mathbf{u}_0(\mathbf{x}),\mathbf{x}\in\Omega.\end{cases}$$

For the two-phase Potts model, i.e., $K=2$, we use $v=v_1,v_2=1-v$, and then the minimization problem (A.8) becomes (2.2).

We want to mention that (A.8) is a regularized softmax in the sense that it reduces to the softmax function when $\lambda=0$ and $\varepsilon=1$. It has been used in [47] to embed variational models into traditional neural networks. If we replace the Gaussian kernel $G_\sigma(\mathbf{x})$ by $B(\mathbf{x}/\sigma)$,

where $B(x)$ is the indicator function for the unit ball, then the length approximation is still correct (see [68] for a proof). It has been used in [46] in the same way as threshold dynamics for image segmentation.

Appendix B. Multigrids and image functions over the grids. In the discrete setting, an image can be viewed as a piecewise constant function on a grid. Images with different resolutions can then be viewed as functions on grids of different sizes. Without loss of generality, we assume the original discrete image has a grid (resolution) $\mathcal{T}$ of size $m \times n$ and grid step size h with

$$m = 2^{s_1}, \quad n = 2^{s_2}$$

for some $h > 0$ and integers $s_1, s_2 > 0$. The image f has a constant value on each small patch $[\alpha_1 h, (\alpha_1+1)h) \times [\alpha_2 h, (\alpha_2+1)h)$ for $\alpha_1 = 1, \ldots, m$ and $\alpha_2 = 1, \ldots, n$.

Starting with $\mathcal{T}^1 = \mathcal{T}$, we consider a sequence of coarse grids $\{\mathcal{T}^j\}_{j=1}^J$ so that $\mathcal{T}^j$ has grid size $m_j \times n_j$ and grid step size h_j with

$$m_j = 2^{s_1 - j + 1}, \quad n_j = 2^{s_2 - j + 1}, \quad h_k = 2^{j-1} h.$$

A sequence of grids with $m = n = 16, h = 1$ are illustrated in the first row of Figure 11.

Denote $\mathcal{I}^j = \{\boldsymbol{\alpha} : \boldsymbol{\alpha} = (\alpha_1, \alpha_2), \alpha_1 = 1, \ldots, m_j,\ \alpha_2 = 1, \ldots, n_j\}$. For a given grid $\mathcal{T}^j$, we can define a set of piecewise-constant basis functions $\{\phi_\alpha^j\}_{\alpha \in I^j}$ so that

$$\phi_{\boldsymbol{\alpha}}^j(x,y) = \begin{cases} 1 & \text{if } (x,y) \in [\alpha_1 h_j, (\alpha_1+1)h_j) \times [\alpha_2 h_j, (\alpha_2+1)h_j), \\ 0 & \text{otherwise.} \end{cases} \tag{B.1}$$

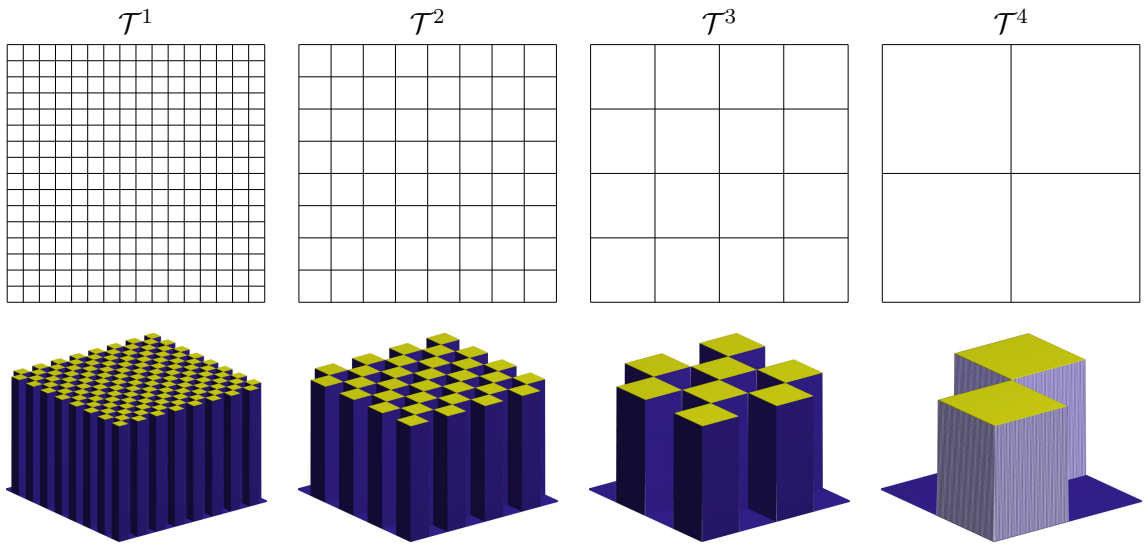

Figure 11. *An illustration of a sequence of grids with $J = 4$. First row: visualization of grids. Second row: visualization of half of the basis functions for each grid.*

In the second row of Figure 11, we illustrate the grids with $m=n=16, J=4$, and half of the basis functions for each grid. Let $\mathcal{V}^j=\text{span}(\{\phi_\alpha^j\}_{\alpha\in I^j})$ be the linear space containing all the piecewise constant functions over grid $\mathcal{T}^j$, and then we have

$$\mathcal{V}^1\supset\mathcal{V}^2\supset\cdots\supset\mathcal{V}^J. \tag{B.2}$$

For each $f\in\mathcal{V}^j$, it can be expressed as $f(x,y)=\sum_{\boldsymbol{\alpha}\in\mathcal{I}^j} f_{\boldsymbol{\alpha}}^j\phi_{\boldsymbol{\alpha}}^j(x,y)$ with $f_{\boldsymbol{\alpha}}^j=f(\alpha_1h_j,\alpha_2h_j)$.

Let $\mathcal{T}^j$ and $\mathcal{T}^{j+1}$ be two grids. We next discuss the downsampling and upsampling operations. Consider $f^{j+1}\in\mathcal{V}^{j+1}$. According to (B.2), there exists a function $f^j\in\mathcal{V}^j$ satisfying $f^j=f^{j+1}$. We denote the upsampling operator $\mathcal{U}^{j+1}:\mathcal{V}^{j+1}\to\mathcal{V}^j$ so that

$$f^j=\mathcal{U}^{j+1}(f^{j+1}). \tag{B.3}$$

It is easy to see that for $\boldsymbol{\alpha}\in\mathcal{I}^j$, we have

$$(\mathcal{U}^j(f^j))_{\boldsymbol{\alpha}}=f_{\boldsymbol{\alpha}'}^{j+1}\text{ with }\alpha_1',\alpha_2'\text{ satisfying }2\alpha_1'-1\le\alpha_1\le2\alpha_1',\ 2\alpha_2'-1\le\alpha_2\le2\alpha_2'. \tag{B.4}$$

Given a function $f^j\in\mathcal{V}^j$, there are many ways to define a downsampling operator $\mathcal{D}^j:\mathcal{V}^j\to\mathcal{V}^{j+1}$. For example, we can define $\mathcal{D}^j$ as an averaging downsampling operator:

$$f^{j+1}=(\mathcal{U}^j(f^j))_{\boldsymbol{\alpha}}=\frac{1}{4}\sum_{\alpha_1'=2\alpha_1-1}^{2\alpha_1}\sum_{\alpha_2'=2\alpha_2-1}^{2\alpha_2}f_{\alpha_1',\alpha_2'}^j. \tag{B.5}$$

Another choice is the max pooling operator, which is widely used in deep learning:

$$f^{j+1}=(\mathcal{U}^k(f^j))_{\boldsymbol{\alpha}}=\max_{\substack{\alpha_1'=2\alpha_1-1,2\alpha_1\\ \alpha_2'=2\alpha_2-1,2\alpha_2}}f_{\alpha_1',\alpha_2'}^j. \tag{B.6}$$

Appendix C. Parallel and sequential splitting scheme.

Operator splitting methods have been widely used for scientific computing and many other applications. Among the rich literature on it, we could point to the following recent publications and surveys [27, 31, 32]. Consider an initial value problem

$$\begin{cases}u_t+\sum_{m=1}^M(A_m(\mathbf{x},t;u)+S_m(\mathbf{x},t;u)+f_m(\mathbf{x},t))=0\text{ on }\Omega\times[0,T],\\ u(0)=u_0,\end{cases} \tag{C.1}$$

where $A_m(\mathbf{x},t;u)$ and $S_m(u,\mathbf{x},t)$ are operators (linear or nonlinear), and $f_m(\mathbf{x},t)$ are functions independent of u. Ω is a given domain and T is a given time. In the following, we omit $\mathbf{x}$ for the simplicity of notation. We discuss in this section parallel and sequential splitting schemes to solve (C.1). As usual, we denote $\Delta t=T/N, t^n=n\Delta t$ for $n=0,1,\dots N$ with a given step number N.

C.1. Parallel splitting schemes.

The authors in [49] proposed a parallel splitting scheme to solve (C.1):

$$\begin{cases}u^0=u_0,\\ \text{for } n\ge0,\text{we compute } u^{n+1}\text{ from }u^n\text{ by solving first}\\ \dfrac{u_m^n-u^n}{M\Delta t}+A_m(t^{n+1};u^n)+S_m(t^{n+1};u_m^n)+f_m(t^{n+1})=0,\ \text{for } m=1,\dots,M,\\ u^{n+1}\text{ is computed by averaging } u^{n+1}=\dfrac{1}{M}\sum_{m=1}^M u_m^n.\end{cases} \tag{C.2}$$

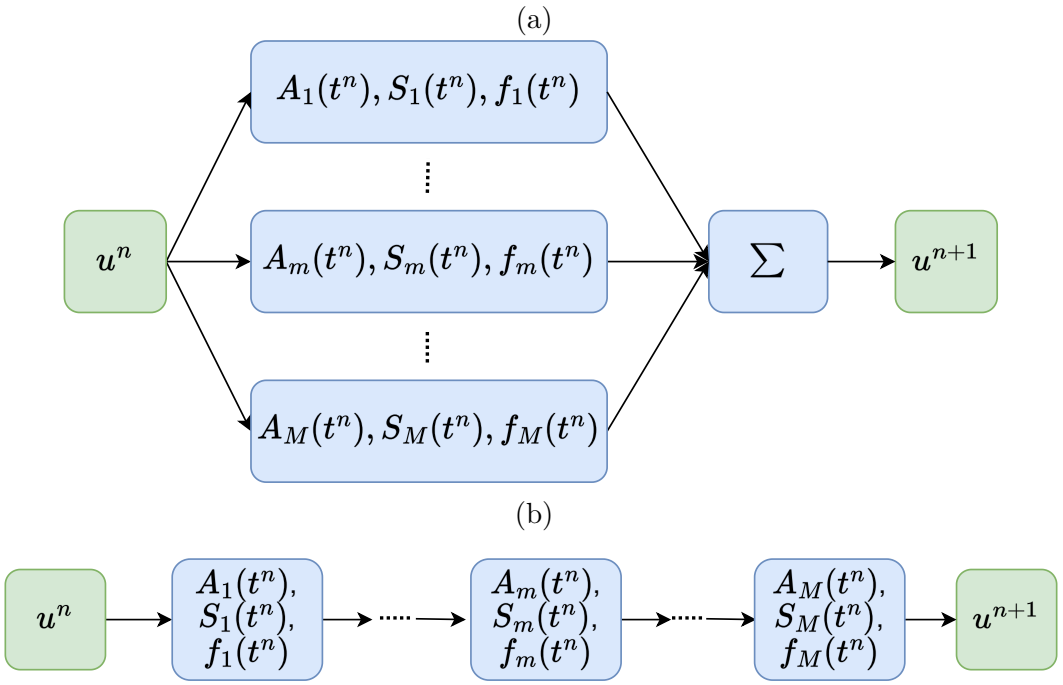

Figure 12. *Illustrations of the corresponding network architecture for* (a) *parallel splitting and* (b) *sequential splitting.*

See also [32, sect. 2.8] for some more explanations about this scheme. It has been proved that (C.2) is $O(\Delta t)$ accurate when A and B are linear operators. In (C.2), all operators are treated implicitly. If the operators A_m are treated explicitly, the parallel splitting scheme becomes

$$\text{(C.3)}\qquad \begin{cases} u^0 = u_0, \\ \text{for } n \geq 0, \text{we compute } u^{n+1} \text{ from } u^n \text{ by solving first} \\ \dfrac{u_m^n - u^n}{M\Delta t} + A_m(t^n; u^n) + S_m(t^{n+1}; u_m^n) + f_m(t^{n+1}) = 0 \text{ for } m = 1, \ldots, M, \\ u^{n+1} \text{ is computed by averaging } u^{n+1} = \dfrac{1}{M}\displaystyle\sum_{m=1}^{M} u_m^n. \end{cases}$$

The structure of (C.3) is illustrated in Figure 12(a).

C.2. Sequential splitting schemes. As a sequential splitting scheme, the Marchuk–Yanenko scheme [50, 26] solving (C.1) reads as

$$\text{(C.4)}\qquad \begin{cases} u^0 = u_0, \\ \text{for } n \geq 0, \text{we compute } u^{n+1} \text{ from } u^n \text{ by solving} \\ \dfrac{u^{n+m/M} - u^{n+(m-1)/M}}{\Delta t} + A_m(t^{n+1}; u^{n+m/M}) + S_m(t^{n+1}; u^{n+m/M}) + f_m(t^{n+1}) = 0, \\ \text{for } m = 1, \ldots, M. \end{cases}$$

See also [32, sect. 2.2] for some more explanations about this scheme. One can show that this scheme is also $O(\Delta t)$ accurate. In (C.4), all operators are treated implicitly. If the operators A_m are treated explicitly, the sequential splitting scheme becomes

(C.5)
$$\begin{cases} u^0 = u_0, \\ \text{for } n \geq 0, \text{we compute } u^{n+1} \text{ from } u^n \text{ by solving} \\ \dfrac{u^{n+m/M} - u^{n+(m-1)/M}}{\Delta t} + A_m(t^n; u^{n+(m-1)/M}) + S_m(t^{n+1}; u^{n+m/M}) + f_m(t^{n+1}) = 0 \\ \text{for } m = 1, \ldots, M. \end{cases}$$

The structure of (C.5) is illustrated in Figure 12(b).

Appendix D. A hybrid splitting scheme to solve initial value problems. Algorithm 4.2 uses a hybrid splitting scheme to split the control problem (4.12) into several subproblems. In this section, we introduce the hybrid splitting scheme and give an error analysis.

D.1. A hybrid splitting scheme. Consider the following initial value problem:

$$\text{(D.1)} \quad \begin{cases} u_t + \displaystyle\sum_{m=1}^{M} \left(\sum_{k=1}^{c_m} \sum_{s=1}^{d_m} A_{k,s}^m(\mathbf{x}, t; u) + \sum_{k=1}^{c_m} S_k^m(\mathbf{x}, t; u) + \sum_{k=1}^{c_m} f_k^m(\mathbf{x}, t) \right) = 0 \text{ on } \Omega \times [0, T], \\ u(0) = u_0. \end{cases}$$

Algorithm D.1. A hybrid splitting scheme.

Data: The solution u^n at time step t^n.
Result: The computed solution u^{n+1} at time step t^{n+1}.
Set $d_1 = 1, u_1^n = u^n$.
for $m = 1, \ldots, M$ **do**
  for $k = 1, \ldots, c_j$ **do**
    Compute $u_k^{n+m/M}$ by solving

$$\text{(D.2)} \quad \frac{u_k^{n+m/M} - u^{n+(m-1)/M}}{c_m \Delta t} = -\sum_{s=1}^{d_m} A_{k,s}^m(t^n; u_s^{n+(m-1)/M}) - S_k^m(t^{n+1}; u_k^{n+m/M}) - f_k^m(t^n).$$

  end for
  Compute $u^{n+m/M}$ as

$$\text{(D.3)} \quad u^{n+m/M} = \frac{1}{c_m} \sum_{k=1}^{c_m} u_k^{n+m/M}.$$

end for

Above $\{c_m\}_{m=1}^M, \{d_m\}_{m=1}^M$ are some known positive integers and $A_{k,s}^m, S_k^m$ are operators. As usual, f_k^m is used to denote functions independent of u. We propose a hybrid splitting scheme, which is a mixture of the parallel and the sequential splitting schemes, to solve (D.1). In our splitting, we split all operators into M sequential substeps, each of which consists of c_m parallel splittings. The computation of each parallel splitting uses d_m intermediate results from the previous substep, which requires the condition $d_m \le c_{m-1}$. The algorithm is summarized in Algorithm D.1. The operators can be treated implicitly or explicitly. In (D.2), we purposely treat $A_{k,s}^m$ explicitly and S_k^m implicitly.

The following theorem shows that Algorithm D.1 converges with first-order at least when the split operators are all linear.

Theorem D.1. *For a fixed $T>0$ and a positive integer N, set $\Delta t = T/N$. Let u^{n+1} be the numerical solution by Algorithm D.1. Assume $A_{k,s}^m$'s and S_k^m's are Lipschitz with respect to $t, \mathbf{x}$ and are linear symmetric positive definite operators with respect to u. Assume Δt is small enough (i.e., N is large enough). We have*

$$\|u^{n+1} - u(t^{n+1})\|_\infty = O(\Delta t) \tag{D.4}$$

for any $0 \le n \le N$.

Theorem D.1 is proved in Appendix E.1.

D.2. A more general hybrid splitting scheme with relaxation. When applying operator splitting schemes to neural network constructions, we need to handle a more general dynamical system in the following form:

$$\begin{cases} u_t + \displaystyle\sum_{j=1}^{2J-1} \sum_{m=1}^{M_j} \left(\sum_{k=1}^{c_{j,m}} \sum_{s=1}^{d_{j,m}} A_{k,s}^{j,m}(\mathbf{x},t;u) + \sum_{k=1}^{c_{j,m}} S_k^{j,m}(\mathbf{x},t;u) + \sum_{k=1}^{c_{j,m}} f_k^{j,m}(\mathbf{x},t) \right) \\ \quad + \displaystyle\sum_{s=1}^{d_{2J}} A_s^*(\mathbf{x},t;u) + S^*(\mathbf{x},t;u) + f^*(\mathbf{x},t) = 0 \text{ on } \Omega \times [0,T], \\ u(0) = u_0, \end{cases} \tag{D.5}$$

where the positive integers $J, M_j, c_{j,m}, d_{j,m}$, the operators $A_{k,s}^{j,m}(t), S_k^{j,m}$, and functions $f_k^{j,m}(t)$ are supposed to be given. Suppose $A_{k,s}^{j,m}(t), S_k^{j,m}(t)$, and $f_k^{j,m}(t)$ are time dependent. We assume $d_{j,m} \le c_{j,m-1}$ for $1 \le m \le M_j$, and $d_{2J} \le c_{2J-1,M_{2J-1}}$, where $c_{j,0} = c_{j-1,M_{j-1}}$ is used. We also assume $c_{j,M_j} = c_{2J-j,M_{2J-j}}$. These conditions are needed to make the algorithm meaningful. In (D.5), the operators A_s^*, S^*, and f^* can be absorbed by the second term. Here we write them down explicitly so that the operators in (D.5) have the same form as the decomposed control variables in section 4.4. We will show that Algorithm 4.2 is a special case of the general hybrid splitting scheme discussed in this section.

The number of operations in (D.5) is about $2J-1$ times of that in (D.1). A simple way to solve (D.5) is to apply Algorithm D.1 for $(2J-1)$ times. In many numerical methods for PDEs, relaxation steps are used to improve the accuracy and stability of the numerical method, such as the Runge–Kutta method [56, 57]. We will propose a new hybrid splitting

Algorithm D.2. A general hybrid splitting algorithm.

Data: The solution u^n at time t^n.
Result: The computed solution u^{n+1} at time step t^{n+1}.
Set $d_{1,1}=1, u_1^n=u^n$.
for $j=1,\dots,J$ **do**
  Set $u^{n,j,0}=u^{n,j-1,M_{j-1}}$, $u_k^{n,j,0}=u_k^{n,j-1,M_{j-1}}$ for $k=1,\dots,c_{j-1,M_{j-1}}$.
  for $m=1,\dots,M_j$ **do**
    for $k=1,\dots,c_{j,m}$ **do**
      Compute $u_k^{n,j,m}$ by solving

$$\frac{u_k^{n,j,m}-u^{n,j,m-1}}{2^{j-1}c_{j,m}\Delta t}=-\sum_{s=1}^{d_{j,m}}A_{k,s}^{j,m}(t^n;u_s^{n,j,m-1})-S_k^m(t^{n+1};u_k^{n,j,m})-f_k^{j,m}(t^n). \tag{D.6}$$

    end for
    Compute $u^{n,j,m}$ as

$$u^{n,j,m}=\frac{1}{c_{j,m}}\sum_{k=1}^{c_{j,m}}u_k^{n,j,m}. \tag{D.7}$$

  end for
end for
Set $\bar{u}^{n,J,M_J}=u^{n,J,M_J}$ and $\bar{u}_k^{n,J,M_J}=u_k^{n,J,M_J}$ for $k=1,2,\dots,c_{J,M_J}$.
for $j=J+1,\dots,2J-1$ **do**
  Set $u^{n,j,0}=\bar{u}^{n,j-1,M_{j-1}}$ and $u_k^{n,j,0}=\bar{u}_k^{n,j-1,M_{j-1}}$, $k=1,\dots,c_{j-1,M_{j-1}}$.
  for $m=1,\dots,M_j$ **do**
    for $k=1,2,\cdots c_{j,m}$ **do**
      Compute $u_k^{n,j,m}$ by solving

$$\frac{u_k^{n,j,m}-u^{n,j,m-1}}{2^{j}c_{j,m}\Delta t}=-\sum_{s=1}^{d_{j,m}}A_{k,s}^{j,m}(t^n;u_s^{n,j,m-1})-S_k^m(t^{n+1};u^{n,j,m})-f_k^{j,m}(t^n). \tag{D.8}$$

    end for
    Compute $u^{n,j,m}$ as

$$u^{n,j,m}=\frac{1}{c_{j,m}}\sum_{k=1}^{c_{j,m}}u_k^{n,j,m}. \tag{D.9}$$

  end for
  Compute $\bar{u}_k^{n,j,M_j},\bar{u}^{n,j,M_j}$ as

$$\bar{u}_k^{n,j,M_j}=\frac{1}{2}u_k^{n,j,M_j}+\frac{1}{2}u_k^{n,2J-j,M_{2J-j}},\quad \bar{u}^{n,j,M_j}=\frac{1}{c_{j,M_j}}\sum_{k=1}^{c_{j,M_j}}\bar{u}_k^{n,j,L_j} \tag{D.10}$$

end for
Compute u^{n+1} by solving

$$\frac{u^{n+1}-\bar{u}^{n,2J-1,M_{2J-1}}}{\Delta t}=-\sum_{s=1}^{d_{2J}}A_s^*(t^n;u_s^{n,2J-1,M_{2J-1}})-S^*(t^{n+1};u^{n+1})-f^*(t^n). \tag{D.11}$$

scheme that incorporates relaxation steps. Our general idea is to split all operators into $2J$ sequential parts according to the index j. For the first $2J-1$ part, the jth part consists of M_j sequential substeps. Algorithm D.1 will be used $2J-1$ times for these parts. The $2J$th part only has one substep. There are infinitely many ways to conduct the relaxations. In this section, we discuss one way as an example. Other ways can be implemented similarly. In our algorithm, for $j = J+1, \ldots, 2J-1$, we will pass the intermediate variable from part $2J-j$ to part j. We simply use averaging for these relaxations, which uses the condition $c_{j,M_j} = c_{2J-j,m_{2J-j}}$. The new scheme is summarized in Algorithm D.2.

The following theorem gives an error estimation of Algorithm D.2 when $S_k^{j,m}$'s and S^* are linear operators.

Theorem D.2. *Assume $S_k^{j,m}$'s and S^* are linear operators and $A_{k,s}^{j,m}, A_s^*, S_k^{j,m}, S^*$ are Lipschitz in $\mathbf{x}, t$ and are symmetric positive definite with respect to u. From an initial condition $u(0) = u_0$, we use Algorithm D.2 to solve (D.5) until time $t = T$ with time step Δt so that $T = N\Delta t$ for some integer $N > 0$. Denote the numerical solution at t^n by u^n. We have*

$$\|u^{n+1} - u(t^{n+1})\|_\infty = O(\Delta t) \tag{D.12}$$

for any $0 \le n \le N$.

Theorem D.2 is proved in Appendix E.2, which shows that under certain conditions, Algorithm D.2 is a first-order numerical scheme in solving the control problem (D.5).

Appendix E. Proof of theorems.

E.1. Proof of Theorem D.1.

Proof of Theorem D.1. In this proof, we focus on the case $d_m = c_{m-1}$. Other cases can be proved similarly. Due to linearity, we denote $A_{k,s}^m(t^n; u), S_k^m(t^n; u)$ by $A_{k,s}^m(t^n)u, S_k^m(t^n)u$. Denote $\widetilde{u}^{n+1}$ the solution of (D.1) at $t = t^{n+1}$ using u^n as the initial condition at $t = t^n$. Denote

$$W(t)u = \sum_{m=1}^{M}\sum_{k=1}^{c_m}\left(\sum_{s=1}^{c_{m-1}} A_{k,s}^m(t)u + S_k^m(t)u\right) \tag{E.1}$$

and $\mathcal{I}$ as the identity operator (matrix). We further denote

$$f(t) = \sum_{m=1}^{M}\sum_{k=1}^{c_m} f_k^m(\mathbf{x}, t). \tag{E.2}$$

We have

$$\begin{aligned}\widetilde{u}^{n+1} = &\exp\left(\int_{t^n}^{t^{n+1}} -W(\tau)d\tau\right)u^n \\ &- \exp\left(\int_{t^n}^{t^{n+1}} -W(\tau)d\tau\right)\int_{t^n}^{t^{n+1}} \exp\left(\int_{t^n}^{\tau} W(\theta)d\theta\right) f(\tau)d\tau.\end{aligned} \tag{E.3}$$

Using Taylor expansion, we have

$$\begin{aligned}\int_{t^n}^{t^{n+1}} \exp\left(\int_{t^n}^{\tau} W(\theta)d\theta\right) f(\tau)d\tau &= \Delta t \exp\left(\int_{t^n}^{t^n} W(\theta)d\theta\right) f(t^n) + O(\Delta t^2) \\ &= \Delta t f(t^n) + O(\Delta t^2)\end{aligned} \tag{E.4}$$

and

$$\exp\left(\int_{t^n}^{t^{n+1}} -W(t)d\tau\right) = 1 - \int_{t^n}^{t^{n+1}} W(t)d\tau + O(\Delta t^2)$$

$$= 1 - \Delta t\widetilde{W}(t^n) + O(\Delta t^2) \tag{E.5}$$

with

$$\widetilde{W}(t^n) = \sum_{m=1}^{M}\sum_{k=1}^{c_m}\left(\sum_{s=1}^{c_{m-1}} A_{k,s}^m(t^n) + S_k^m(t^{n+1})\right). \tag{E.6}$$

Substituting (E.4) and (E.5) into (E.3) gives rise to

$$\widetilde{u}^{n+1} = u^n - \Delta t\widetilde{W}(t^n)u^n - \Delta t f(t^n) + O(\Delta t^2). \tag{E.7}$$

We next focus on u^{n+1}. We show that for any $1 \le m^* \le M$, we have

$$u_{k^*}^{n+m^*/M} = u^n - \Delta t\left(\sum_{m=1}^{m^*-1}\sum_{k=1}^{c_m}\left(\left(\sum_{s=1}^{c_{m-1}} A_{k,s}^m(t^n) + S_k^m(t^{n+1})\right)u^n + f_k^m(t^n)\right)\right.$$

$$\left. + c_{m^*}\left(\sum_{s=1}^{c_{m^*-1}}\left(A_{k^*,s}^m(t^n) + S_{k^*}^m(t^{n+1})\right)u^n + f_{k^*}^m(t^n)\right)\right) + O(\Delta t^2), \tag{E.8}$$

$$u^{n+m^*/M} = u^n - \Delta t\sum_{m=1}^{m^*}\sum_{k=1}^{c_m}\left(\left(\sum_{s=1}^{c_{m-1}} A_{k,s}^m(t^n) + S_k^m(t^{n+1})\right)u^n + f_k^m(t^n)\right) + O(\Delta t^2) \tag{E.9}$$

for any $1 \le k^* \le c_{m^*}$. We will prove (E.8)–(E.9) by mathematical induction. For $m^* = 1$, we have $c_0 = 1$. According to (D.2) and (D.3), we have

$$\begin{aligned} u_k^{n+1/M} &= (\mathcal{I} + \Delta t c_1 S_k^m(t^{n+1}))^{-1}\left(u^n - \Delta t c_1 A_{k,1}^1(t^n)u^n - \Delta t c_1 f_k^1(t^n)\right) \\ &= (\mathcal{I} - \Delta t c_1 S_k^m(t^{n+1}))\left(u^n - \Delta t c_1 A_{k,1}^1(t^n)u^n - \Delta t c_1 f_k^1(t^n)\right) + O(\Delta t^2) \\ &= u^n - \Delta t c_1\left(A_{k,1}^1(t^n) + S_k^m(t^{n+1})\right)u^n - \Delta t c_1 f_k^1(t^n) + O(\Delta t^2) \end{aligned} \tag{E.10}$$

and

$$u^{n+1/M} = \frac{1}{c_1}\sum_{k=1}^{c_1}\left(u^n - \Delta t c_1\left(A_{k,1}^1(t^n) + S_k^m(t^{n+1})\right)u^n - \Delta t c_1 f_k^1(t^n)\right) + O(\Delta t^2)$$

$$= u^n - \Delta t\sum_{k=1}^{c_1}\left(\left(A_{k,1}^1(t^n) + S_k^1(t^{n+1})\right)u^n + f_k^1(t^n)\right) + O(\Delta t^2). \tag{E.11}$$

Thus (E.8)–(E.9) hold.

Assume (E.8)–(E.9) hold for $m^* = m_1 < M$, i.e.,

$$u_{k^*}^{n+m_1/M} = u^n - \Delta t \Bigg(\sum_{m=1}^{m_1-1} \sum_{k=1}^{c_m} \Bigg(\Bigg(\sum_{s=1}^{c_{m-1}} A_{k,s}^m(t^n) + S_k^m(t^{n+1}) \Bigg) u^n + f_k^m(t^n) \Bigg)$$

$$+ c_{m_1} \Bigg(\sum_{s=1}^{c_{m_1-1}} \left(A_{k^*,s}^m(t^n) + S_{k^*}^m(t^{n+1}) \right) u^n + f_{k^*}^m(t^n) \Bigg) \Bigg) + O(\Delta t^2), \tag{E.12}$$

$$u^{n+m_1/M} = u^n - \Delta t \sum_{m=1}^{m_1} \sum_{k=1}^{c_m} \Bigg(\Bigg(\sum_{s=1}^{c_{m-1}} A_{k,s}^m(t^n) + S_k^m(t^{n+1}) \Bigg) u^n + f_k^m(t^n) \Bigg) + O(\Delta t^2) \tag{E.13}$$

for any $1 \le k^* \le c_{m_1}$. When $m^* = m_1 + 1$, we have

$$\begin{aligned} u_{k^*}^{n+(m_1+1)/M} &= \left(\mathcal{I} + \Delta t c_{m_1+1} S_{k^*}^{m_1+1}(t^{n+1}) \right)^{-1} \Bigg(u^{n+m_1/M} - \Delta t c_{m_1+1} \sum_{s=1}^{m_1} A_{k^*,s}^{m_1+1}(t^n) u_s^{n+m_1/M} \\ &\quad - \Delta t c_{m_1+1} f_{k^*}^{m_1+1}(t^n) \Bigg) \\ &= \left(\mathcal{I} - \Delta t c_{m_1+1} S_{k^*}^{m_1+1}(t^{n+1}) \right) \Bigg(u^{n+m_1/M} - \Delta t c_{m_1+1} \sum_{s=1}^{m_1} A_{k^*,s}^{m_1+1}(t^n) u_s^{n+m_1/M} \\ &\quad - \Delta t c_{m_1+1} f_{k^*}^{m_1+1}(t^n) \Bigg) + O(\Delta t^2) \\ &= u^{n+m_1/M} - \Delta t c_{m_1+1} \sum_{s=1}^{c_{m_1}} A_{k^*,s}^{m_1+1}(t^n) u_s^{n+m_1/M} - \Delta t c_{m_1+1} S_{k^*}^{m_1+1}(t^{n+1}) u^{n+m_1/M} \\ &\quad - \Delta t c_{m_1+1} f_{k^*}^{m_1+1}(t^n) + O(\Delta t^2). \end{aligned} \tag{E.14}$$

Substituting (E.12)–(E.13) into (E.14) gives rise to

$$\begin{aligned} u_{k^*}^{n+(m_1+1)/M} &= u^n - \Delta t \sum_{m=1}^{m_1} \sum_{k=1}^{c_m} \Bigg(\Bigg(\sum_{s=1}^{c_{m-1}} A_{k,s}^m(t^n) + S_k^m(t^{n+1}) \Bigg) (t^{n+1}) u^n + f_k^m(t^n) \Bigg) \\ &\quad - \Delta t c_{m_1+1} \sum_{s=1}^{c_{m_1}} A_{k^*,s}^{m_1+1}(t^n) u_s^{n+m_1/M} - \Delta t c_{m_1+1} S_{k^*}^{m_1+1}(t^{n+1}) u^n \\ &\quad - \Delta t c_{m_1+1} f_{k^*}^{m_1+1}(t^n). \end{aligned} \tag{E.15}$$

According to (D.3), we have

$$\begin{aligned} u^{n+(m_1+1)/M} &= u^n - \Delta t \sum_{m=1}^{m_1}\sum_{k=1}^{c_m}\left(\left(\sum_{s=1}^{c_{m-1}} A_{k,s}^m(t^n) + S_k^m\right)(t^{n+1})u^n + f_k^m(t^n)\right) \\ &\quad - \Delta t \sum_{k=1}^{c_{m_1+1}}\left(\sum_{s=1}^{c_{m_1}} A_{k,s}^{m_1+1}(t^n)u_s^{n+m_1/M} - S_k^{m_1+1}(t^{n+1})u^n - f_k^{m_1+1}(t^{n+1})\right) \\ &\quad + O(\Delta t^2) \\ &= u^n - \Delta t \sum_{m=1}^{m_1+1}\sum_{k=1}^{c_{m+1}}\left(\left(\sum_{s=1}^{c_m} A_{k,s}^m(t^n) + S_k^m(t^{n+1})\right)u^n + f_k^m(t^n)\right) + O(\Delta t^2). \end{aligned} \tag{E.16}$$

Therefore, (E.8)–(E.9) hold for $m = m_1 + 1$. Combining (E.10)–(E.11) and (E.15)–(E.16), we have that (E.8)–(E.9) hold for any $1 \le m \le M$.

Setting $m = M$ gives rise to

$$\begin{aligned} u^{n+1} &= u^n - \Delta t \sum_{m=1}^{M}\sum_{k=1}^{c_m}\left(\left(\sum_{s=1}^{c_{m-1}} A_{k,s}^m(t^n) + S_k^m(t^{n+1})\right)u^n + f_k^m(t^n)\right) + O(\Delta t^2) \\ &= u^n - \Delta t\widetilde{W}(t^n)u^n - \Delta t \sum_{m=1}^{M}\sum_{k=1}^{c_m} f_k^m(t^n) + O(\Delta t^2), \end{aligned} \tag{E.17}$$

where $\widetilde{W}(t^n)$ is defined in (E.6).

Comparing (E.17) with (E.7), we have the local error

$$u^{n+1} - \widetilde{u}^{n+1} = O(\Delta t^2). \tag{E.18}$$

Therefore, the global error is $O(\Delta t)$. ∎

E.2. Proof of Theorem D.2.

Proof of Theorem D.2. We consider the case $d_{j,m} = c_{j,m-1}$ and $d_{2J} = c_{2J-1,M_{2j-1}}$. Other cases can be proved similarly. We denote $A_{k,s}^{j,m}(t;u), S_k^{j,m}(u)$ by $A_{k,s}^{j,m}(t)u, S_k^{j,m}u$.

To simplify the notation, we denote

$$\begin{aligned} Z_j(t) &= \sum_{m=1}^{M_j}\sum_{k=1}^{c_{j,m}}\left(\sum_{s=1}^{c_{j,m-1}} A_{k,s}^{j,m}(t) + S_k^{j,m}(t)\right), \quad Z_*(t) = \sum_{s=1}^{c_{2J-1,M_{2J-1}}} A_s^*(t) + S^*(t), \\ f_j(t) &= \sum_{m=1}^{M_j}\sum_{k=1}^{c_{j,m}} f_k^{j,m}(t), \\ Z(t) &= \sum_{j=1}^{2J-1} Z_j(t) + Z^*(t), \quad f(t) = \sum_{j=1}^{2J-1} f_j(t) + f^*(t) \end{aligned}$$

and

$$\widetilde{Z}_j(t^n)=\sum_{m=1}^{M_j}\sum_{k=1}^{c_{j,m}}\left(\sum_{s=1}^{c_{j,m-1}}A_{k,s}^{j,m}(t^n)+S_k^{j,m}(t^{n+1})\right),\ \widetilde{Z}^*(t^n)=\sum_{s=1}^{c_{2J-1,M_{2J-1}}}A_s^*(t^n)+S^*(t^{n+1}),$$

$$\widetilde{Z}(t^n)=\sum_{j=1}^{2J-1}\widetilde{Z}_j(t^n)+\widetilde{Z}^*(t^n),\ f(t^n)=\sum_{j=1}^{2J-1}f_j(t^n)+f^*(t^n). \tag{E.19}$$

Denote $\widetilde{u}^{n+1}$ the solution of (D.5) at $t=t^{n+1}$ using u^n as an initial condition at $t=t^n$. We have

$$\begin{aligned}\widetilde{u}^{n+1}&=\exp\left(\int_{t^n}^{t^{n+1}}-Z(\tau)d\tau\right)u^n\\ &\quad-\exp\left(\int_{t^n}^{t^{n+1}}-Z(\tau)d\tau\right)\int_{t^n}^{t^{n+1}}\exp\left(\int_{t^n}^{\tau}Z(\theta)d\theta\right)f(\tau)d\tau.\end{aligned} \tag{E.20}$$

Using Taylor expansion, we have

$$\begin{aligned}\int_{t^n}^{t^{n+1}}\exp\left(\int_{t^n}^{\tau}Z(\theta)d\theta\right)f(\tau)d\tau&=\Delta t\exp\left(\int_{t^n}^{t^n}Z(\theta)d\theta\right)f(t^n)+O(\Delta t^2)\\ &=\Delta t f(t^n)+O(\Delta t^2)\end{aligned} \tag{E.21}$$

and

$$\begin{aligned}\exp\left(\int_{t^n}^{t^{n+1}}-Z(t)d\tau\right)&=1-\int_{t^n}^{t^{n+1}}Z(t)d\tau+O(\Delta t^2)\\ &=1-\Delta t\widetilde{Z}(t^n)+O(\Delta t^2).\end{aligned} \tag{E.22}$$

Substituting (E.21) and (E.22) into (E.20) gives rise to

$$\widetilde{u}^{n+1}=u^n-\Delta t\widetilde{Z}(t^n)u^n-\Delta t f(t^n)+O(\Delta t^2). \tag{E.23}$$

We then focus on u^{n+1}. For any $j\le J$ in Algorithm D.2, it is a standard hybrid splitting. From the proof of Theorem D.1, in particular (E.8)–(E.9), for any $1\le j^*\le J, 1\le k^*\le c_{j^*,M_{j^*}}$, we have

$$
\begin{aligned}
u_{k^*}^{n,j^*,M_{j^*}} &= u^n - \Delta t\Bigg(\sum_{j=1}^{j^*-1} 2^{j-1}\left(\sum_{m=1}^{M_j}\sum_{k=1}^{c_{j,m}}\left(\sum_{s=1}^{c_{j,m-1}} A_{k,s}^{j,m}(t^n) + S_k^{j,m}(t^{n+1})\right)u^n + f_k^{j,m}(t^n)\right)\\
&\quad + 2^{j^*-1}\left(\sum_{m=1}^{M_{j^*}-1}\sum_{k=1}^{c_{j^*,m}}\left(\sum_{s=1}^{c_{j^*,m-1}} A_{k,s}^{j^*,m}(t^n) + S_k^{j^*,m}(t^{n+1})\right)u^n + f_k^{j^*,m}(t^n)\right)\\
&\quad + 2^{j^*-1}c_{j^*,M_{j^*}}\left(\sum_{s=1}^{c_{j^*,M_{j^*}-1}} A_{k^*,s}^{j^*,M_{j^*}}(t^n) + S_{k^*}^{j^*,M_{j^*}}(t^{n+1})\right)u^n + f_{k^*}^{j^*,M_{j^*}}(t^n)\Bigg) + O(\Delta t^2)\\
&= u^n - \Delta t\Bigg(\sum_{j=1}^{j^*-1} 2^{j-1}\left(\widetilde{Z}_{1,j}u^n + f_j(t^n)\right)\\
&\quad - 2^{j^*-1}\left(\sum_{m=1}^{M_{j^*}-1}\sum_{k=1}^{c_{j^*,m}}\left(\sum_{s=1}^{c_{j^*,m}} A_{k,s}^{j^*,m}(t^n) + S_k^{j^*,m}(t^{n+1})\right)u^n + f_k^{j^*,m}(t^n)\right)\\
&\quad - 2^{j^*-1}c_{j^*,M_{j^*}}\left(\sum_{s=1}^{c_{j^*,M_{j^*}-1}} A_{k^*,s}^{j^*,M_{j^*}}(t^n) + S_{k^*}^{j^*,M_{j^*}}(t^{n+1})\right)u^n + f_{k^*}^{j^*,M_{j^*}}(t^n)\Bigg) + O(\Delta t^2)
\end{aligned}
\tag{E.24}
$$

$$
\begin{aligned}
u^{n,j^*,M_{j^*}} &= u^n - \Delta t\left(\sum_{j=1}^{j^*} 2^{j-1}\left(\sum_{l=1}^{M_j}\sum_{k=1}^{c_{j,m}}\left(\sum_{s=1}^{c_{j,m-1}} A_{k,s}^{j,l}(t^n) + S_k^{j,l}(t^{n+1})\right)u^n\right)\right)\\
&\quad - \Delta t\left(\sum_{j=1}^{j^*} 2^{j-1}\left(\sum_{m=1}^{M_j}\sum_{k=1}^{c_{j,m}} f_k^{j,m}(t^n)\right)\right) + O(\Delta t^2)\\
&= u^n - \Delta t\left(\sum_{j=1}^{j^*} 2^{j-1}\left(\widetilde{Z}_j(t^n)u^n + f_j(t^n)\right)\right) + O(\Delta t^2).
\end{aligned}
\tag{E.25}
$$

Setting $j^* = J$ and from Algorithm D.2, we have

$$
\begin{aligned}
\bar{u}_{k^*}^{n,J,M_J} = u_{k^*}^{n,J,M_J} &= u^n - \Delta t\Bigg(\sum_{j=1}^{J-1} 2^{j-1}\left(\widetilde{Z}_j(t^n)u^n + f_j(t^n)\right)\\
&\quad - 2^{J-1}\left(\sum_{m=1}^{M_J-1}\sum_{k=1}^{c_{J,m}}\left(\sum_{s=1}^{c_{J,m-1}} A_{k,s}^{J,m}(t^n) + S_k^{J,m}(t^{n+1})\right)u^n + f_k^{J,m}(t^n)\right)\\
&\quad - 2^{J-1}c_{J,M_J}\left(\sum_{s=1}^{c_{J,M_j-1}} A_{k^*,s}^{J,M_J}(t^n) + S_{k^*}^{J,M_J}(t^{n+1})\right)u^n + f_k^{J,M_J}(t^n)\Bigg) + O(\Delta t^2)
\end{aligned}
\tag{E.26}
$$

$$\bar{u}^{n,J,M_J} = u^{n,J,M_J} = u^n - \Delta t \left(\sum_{j=1}^{J} 2^{j-1} \left(\widetilde{Z}_j(t^n) u^n + f_j(t^n) \right) \right) + O(\Delta t^2). \tag{E.27}$$

For the simplicity of notation, we denote $\widetilde{j}^* = 2J - j^*$. We next use mathematical induction to show that for any $J+1 \le j^* \le 2J-1$ and $1 \le k^* \le c_{j^*,M_{j^*}}$, we have

$$\begin{aligned}
\bar{u}_{k^*}^{n,j^*,M_{j^*}} = u^n &- \Delta t \left(\sum_{j=1}^{\widetilde{j}^*-1} 2^{j-1} \left(\widetilde{Z}_j(t^n) u^n + f_j(t^n) \right) + \sum_{j=\widetilde{j}^*+1}^{J} 2^{\widetilde{j}^*-1} \left(\widetilde{Z}_j(t^n) u^n + f_j(t^n) \right) \right) \\
&- \Delta t 2^{\widetilde{j}^*-2} \left(\widetilde{Z}_{\widetilde{j}^*}(t^n) u^n + f_{\widetilde{j}^*}(t^n) \right) \\
&- \Delta t 2^{\widetilde{j}^*-2} \sum_{m=1}^{M_{\widetilde{j}^*}-1} \sum_{k=1}^{c_{\widetilde{j}^*,m}} \left(\left(\sum_{s=1}^{c_{\widetilde{j}^*,m-1}} A_{k,s}^{\widetilde{j}^*,m}(t^n) + S_k^{\widetilde{j}^*,m}(t^{n+1}) \right) u^n + f_k^{\widetilde{j}^*,m}(t^n) \right) \\
&- \Delta t 2^{\widetilde{j}^*-2} c_{\widetilde{j}^*,M_{\widetilde{j}^*}} \left(\left(\sum_{s=1}^{c_{\widetilde{j}^*,M_{\widetilde{j}^*}-1}} A_{k^*,s}^{\widetilde{j}^*,M_{\widetilde{j}^*}}(t^n) + S_{k^*}^{\widetilde{j}^*,M_J}(t^{n+1}) \right) u^n + f_{k^*}^{\widetilde{j}^*,M_{\widetilde{j}^*}}(t^n) \right) \\
&- \Delta t 2^{\widetilde{j}^*-1} \left(\left(\sum_{j=J+1}^{j^*-1} \left(\widetilde{Z}_j(t^n) u^n + f_j(t^n) \right) \right) \right. \\
&\left. + \left(\sum_{m=1}^{M_{j^*}-1} \sum_{k=1}^{c_{j^*,m}} \left(\sum_{s=1}^{c_{j^*,m-1}} A_{k,s}^{j^*,m}(t^n) + S_k^{j^*,m}(t^{n+1}) \right) u^n + f_k^{j^*,m}(t^n) \right) \right. \\
&\left. + c_{j^*,M_{j^*}} \left(\left(\sum_{s=1}^{c_{j^*,M_{j^*}-1}} A_{k^*,s}^{J,M_{j^*}}(t^n) + S_{k^*}^{j^*,M_{j^*}}(t^{n+1}) \right) u^n + f_{k^*}^{j^*,M_{j^*}}(t^n) \right) \right) + O(\Delta t^2),
\end{aligned} \tag{E.28}$$

$$\begin{aligned}
\bar{u}^{n,j^*,M_{j^*}} = u^n &- \Delta t \left(\sum_{j=1}^{\widetilde{j}^*-1} 2^{j-1} \left(\widetilde{Z}_j(t^n) u^n + f_j(t^n) \right) + \sum_{j=\widetilde{j}^*}^{J} 2^{\widetilde{j}^*-1} \left(\widetilde{Z}_j(t^n) u^n + f_j(t^n) \right) \right) \\
&- \Delta t 2^{\widetilde{j}^*-1} \left(\sum_{j=J+1}^{j^*} \left(\widetilde{Z}_j(t^n) u^n + f_j(t^n) \right) \right) + O(\Delta t^2).
\end{aligned} \tag{E.29}$$

We first show that (E.28)–(E.29) hold when $j^* = J+1$. Set $u^{n,J+1,0} = \bar{u}^{n,J,M_J}$ and $u_k^{n,J+1,0} = \bar{u}_k^{n,J,M_J}$ for $k = 1, 2, \ldots, c_{J,M_J}$. For any $1 \le k^* \le c_{J+1,1}$, we have

$$
\begin{aligned}
u_{k^*}^{n,J+1,1} &= \left(\mathcal{I} + \Delta t 2^{J-1} c_{J+1,1} S_{k^*}^{J+1,1}(t^{n+1})\right)^{-1} \left(u^{n,J+1,0} \right. \\
&\quad \left. - \Delta t 2^{J-1} c_{J+1,1} \left(\sum_{s=1}^{c_{J+1,0}} A_{k^*,s}^{J+1,1}(t^n) u_s^{n,J+1,0} + f_{k^*}^{J+1,1}(t^n) \right) \right) \\
&= \left(\mathcal{I} - \Delta t 2^{J-1} c_{J+1,1} S_{k^*}^{J+1,1}(t^{n+1})\right) \left(u^{n,J+1,0} \right. \\
&\quad \left. - \Delta t 2^{J-1} c_{J+1,1} \left(\sum_{s=1}^{c_{J+1,0}} A_{k^*,s}^{J+1,1}(t^n) u_s^{n,J+1,0} + f_{k^*}^{J+1,1}(t^n) \right) \right) + (\Delta t^2) \\
&= u^{n,J+1,0} - \Delta t 2^{J-1} c_{J+1,1} \left(\sum_{s=1}^{c_{J+1,0}} A_{k^*,s}^{J+1,1}(t^n) u_s^{n,J+1,0} + S_{k^*}^{J+1,1}(t^{n+1}) u^{n,J+1,0} \right. \\
&\quad \left. + f_{k^*}^{J+1,1}(t^n) \right) + O(\Delta t^2) \\
&= u^n - \Delta t \left(\sum_{j=1}^{J} 2^{j-1} \left(\widetilde{Z}_j(t^n) u^n + f_j(t^n) \right) \right) \\
&\quad - \Delta t 2^{J-1} c_{J+1,1} \left(\left(\sum_{s=1}^{c_{J+1,0}} A_{k^*,s}^{J+1,1}(t^n) + S_{k^*}^{J+1,1}(t^{n+1}) \right) u^n + f_{k^*}^{J+1,1}(t^n) \right) + O(\Delta t^2)
\end{aligned} \tag{E.30}
$$

and

$$
\begin{aligned}
u^{n,J+1,1} &= \frac{1}{c_{J+1,1}} \sum_{k=1}^{c_{J+1,1}} u_k^{n,J+1,1} \\
&= u^n - \Delta t \left(\sum_{j=1}^{J} 2^{j-1} \left(\widetilde{Z}_j(t^n) u^n + f_j(t^n) \right) \right) \\
&\quad - \Delta t 2^{J-1} \sum_{k=1}^{c_{J+1,1}} \left(\left(\sum_{s=1}^{c_{J+1,0}} A_{k,s}^{J+1,1}(t^n) + S_k^{J+1,1}(t^{n+1}) \right) u^n + f_k^{J+1,1}(t^n) \right) + O(\Delta t^2).
\end{aligned} \tag{E.31}
$$

Repeating the process, we can show that for any $1 \le m^* \le M_{J+1}$, we have

$$
\begin{aligned}
u_{k^*}^{n,J+1,m^*} &= u^n - \Delta t \left(\sum_{j=1}^{J} 2^{j-1} \left(\widetilde{Z}_j(t^n) u^n + f_j(t^n) \right) \right) \\
&\quad - \Delta t 2^{J-1} \sum_{m=1}^{m^*-1} \sum_{k=1}^{c_{J+1,m}} \left(\left(\sum_{s=1}^{c_{J+1,m-1}} A_{k,s}^{J+1,m}(t^n) + S_k^{J+1,m}(t^{n+1}) \right) u^n + f_k^{J+1,m}(t^n) \right) \\
&\quad - \Delta t 2^{J-1} c_{J+1,m^*} \left(\left(\sum_{s=1}^{c_{J+1,m^*}} A_{k^*,s}^{J+1,m^*}(t^n) + S_{k^*}^{J+1,m^*}(t^{n+1}) \right) u^n + f_{k^*}^{J+1,m^*}(t^n) \right) \\
&\quad + O(\Delta t^2)
\end{aligned} \tag{E.32}
$$

and

$$
\begin{aligned}
u^{n,J+1,m^*} &= \frac{1}{c_{J+1,m^*}} \sum_{k=1}^{c_{J+1,m^*}} u^{n,J+1,m^*} \\
&= u^n - \Delta t \left(\sum_{j=1}^{J} 2^{j-1} \left(\widetilde{Z}_j(t^n) u^n + f_j(t^n) \right) \right) \\
&\quad - \Delta t 2^{J-1} \sum_{m=1}^{m^*} \sum_{k=1}^{c_{J+1,m}} \left(\left(\sum_{s=1}^{c_{J+1,m-1}} A_{k,s}^{J+1,m}(t^n) + S_k^{J+1,m}(t^{n+1}) \right) u^n \right. \\
&\quad \left. + f_k^{J+1,l}(t^n) \right) + O(\Delta t^2).
\end{aligned} \tag{E.33}
$$

Set $m^* = M_{J+1}$ in (E.32) and (E.33). We get

$$
\begin{aligned}
u_{k^*}^{n,J+1,M_{J+1}} &= u^n - \Delta t \left(\sum_{j=1}^{J} 2^{j-1} \left(\widetilde{Z}_j(t^n) u^n + f_j(t^n) \right) \right) \\
&\quad - \Delta t 2^{J-1} \sum_{m=1}^{M_{J+1}-1} \sum_{k=1}^{c_{J+1,m}} \left(\left(\sum_{s=1}^{c_{J+1,m-1}} A_{k,s}^{J+1,m}(t^n) + S_k^{J+1,m}(t^{n+1}) \right) u^n \right. \\
&\quad \left. + f_k^{J+1,m}(t^n) \right) \\
&\quad - \Delta t 2^{J-1} c_{J+1,M_{J+1}} \left(\left(\sum_{s=1}^{c_{J+1,M_{J+1}-1}} A_{k^*,s}^{J+1,M_{J+1}}(t^n) + S_{k^*}^{J+1,M_{J+1}}(t^{n+1}) \right) u^n \right. \\
&\quad \left. + f_{k^*}^{J+1,M_{J+1}}(t^n) \right) + O(\Delta t^2)
\end{aligned} \tag{E.34}
$$

and

$$
\begin{aligned}
u^{n,J+1,M_{J+1}} &= u^n - \Delta t \left(\sum_{j=1}^{J} 2^{j-1} \left(\widetilde{Z}_j(t^n) u^n + f_j(t^n) \right) \right) \\
&\quad - \Delta t 2^{J-1} \left(\widetilde{Z}_{J+1}(t^n) + f_{J+1}(t^n) \right) + O(\Delta t^2).
\end{aligned} \tag{E.35}
$$

For any $1 \le k^* \le M_{J+1}$, we compute

$$
\begin{aligned}
&\bar{u}_{k^*}^{n,J+1,M_{J+1}} \\
&\quad = \frac{1}{2} u_{k^*}^{n,J+1,M_{J+1}} + \frac{1}{2} u_{k^*}^{n,J-1,M_{J-1}} \\
&\quad = u^n - \Delta t \left(\sum_{j=1}^{J-2} 2^{j-1} \left(\widetilde{Z}_j(t^n) u^n + f_j(t^n) \right) + 2^{J-2} \left(\widetilde{Z}_J(t^n) u^n + f_J(t^n) \right) \right) \\
&\qquad - \Delta t 2^{J-3} \left(\widetilde{Z}_{J-1}(t^n) u^n + f_{J-1}(t^n) \right) \\
&\qquad - \Delta t 2^{J-3} \sum_{m=1}^{M_{J-1}-1} \sum_{k=1}^{c_{J-1,m}} \left(\left(\sum_{s=1}^{c_{J-1,m-1}} A_{k,s}^{J-1,m}(t^n) + S_k^{J-1,m}(t^{n+1}) \right) u^n + f_k^{J-1,m}(t^n) \right) \\
&\qquad - \Delta t 2^{J-3} c_{J-1,M_{J-1}} \left(\left(\sum_{s=1}^{c_{J-1,M_{J-1}}-1} A_{k^*,s}^{J-1,M_{J-1}}(t^n) + S_{k^*}^{J-1,M_{J-1}}(t^{n+1}) \right) u^n + f_{k^*}^{J-1,M_{J-1}}(t^n) \right) \\
&\qquad - \Delta t 2^{J-2} \sum_{m=1}^{M_{J+1}-1} \sum_{k=1}^{c_{J+1,m}} \left(\left(\sum_{s=1}^{c_{J+1,m-1}} A_{k,s}^{J+1,m}(t^n) + S_k^{J+1,m}(t^{n+1}) \right) u^n + f_k^{J+1,m}(t^n) \right) \\
&\qquad - \Delta t 2^{J-2} c_{J+1,M_{J+1}} \left(\left(\sum_{s=1}^{c_{J+1,M_{J+1}}-1} A_{k^*,s}^{J+1,M_{J+1}}(t^n) + S_{k^*}^{J+1,M_{J+1}}(t^{n+1}) \right) u^n \right. \\
&\qquad \left. + f_{k^*}^{J+1,M_{J+1}}(t^n) \right) + O(\Delta t^2)
\end{aligned} \tag{E.36}
$$

and

$$
\begin{aligned}
\bar{u}^{n,J+1,M_{J+1}} &= \frac{1}{c_{J+1,M_{J+1}}} \sum_{k=1}^{c_{J+1,M_{J+1}}} \bar{u}_k^{n,J+1,M_{J+1}} \\
&= u^n - \Delta t \left(\sum_{j=1}^{J-2} 2^{j-1} \left(\widetilde{Z}_j(t^n) u^n + f_j(t^n) \right) + \sum_{j=J-1}^{J} 2^{J-2} \left(\widetilde{Z}_j(t^n) u^n + f_j(t^n) \right) \right) \\
&\quad - \Delta t 2^{J-2} \left(\widetilde{Z}_{J+1}(t^n) u^n + f_{J+1}(t^n) \right) + O(\Delta t^2).
\end{aligned} \tag{E.37}
$$

Therefore, (E.28)–(E.29) hold for $j^* = J - 1$.

Assume (E.28)–(E.29) hold for $j^* = j_1 - 1 \geq J+1$. Set $u^{n,j_1,0} = \bar{u}^{n,j_1-1,M_{j_1-1}}$ and $u_k^{n,j_1,0} = \bar{u}_k^{n,j_1-1,M_{j_1-1}}$ for $k = 1, 2, \ldots, c_{j_1-1,M_{j_1-1}}$. Denote $\widetilde{j}_1 = 2J - j_1$. For any $1 \leq k^* \leq M_{j_1}$, we compute

$$
\begin{aligned}
u_{k^*}^{n,j_1,1} &= \left(\mathcal{I} + \Delta t 2^{\widetilde{j}_1} c_{j_1,1} S_{k^*}^{j_1,1}(t^{n+1})\right)^{-1} \left(u^{n,j_1,0} + \Delta t 2^{\widetilde{j}_1} c_{j_1,1} \left(\sum_{s=1}^{c_{j_1,0}} A_{k^*,s}^{j_1,1}(t^n) u_s^{n,j_1,0} + f_{k^*}^{j_1,1}(t^n)\right)\right) \\
&= \left(\mathcal{I} - \Delta t 2^{\widetilde{j}_1} c_{j_1,1} S_k^{j_1,1}(t^{n+1})\right) \left(u^{n,j_1,0} + \Delta t 2^{\widetilde{j}_1} c_{j_1,1} \left(\sum_{s=1}^{c_{j_1,0}} A_{k^*,s}^{j_1,1}(t^n) u_s^{n,j_1,0} + f_{k^*}^{j_1,1}(t^n)\right)\right) \\
&\quad + (\Delta t^2) \\
&= u^{n,j_1,0} - \Delta t 2^{\widetilde{j}_1} c_{j_1,1} \left(\left(\sum_{s=1}^{c_{j_1,0}} A_{k^*,s}^{j_1,1}(t^n) + S_{k^*}^{j_1,1}(t^{n+1})\right) u_s^{n,j_1,0} + f_{k^*}^{j_1,1}(t^n)\right) + O(\Delta t^2) \\
&= u^n - \Delta t \left(\sum_{j=1}^{\widetilde{j}_1} 2^{j-1} \left(\widetilde{Z}_j(t^n) u^n + f_j(t^n)\right) + \sum_{j=\widetilde{j}_1+1}^{J} 2^{\widetilde{j}_1} \left(\widetilde{Z}_j(t^n) u^n + f_j(t^n)\right)\right) \\
&\quad - \Delta t 2^{\widetilde{j}_1} \left(\sum_{j=J+1}^{j_1-1} \left(\widetilde{Z}_j(t^n) u^n + f_j(t^n)\right)\right) \\
&\quad - \Delta t 2^{\widetilde{j}_1} c_{j_1,1} \left(\left(\sum_{s=1}^{c_{j_1,0}} A_{k^*,s}^{j_1,1}(t^n) + S_{k^*}^{j_1,1}(t^{n+1})\right) u^n + f_{k^*}^{j_1,1}(t^n)\right) + O(\Delta t^2)
\end{aligned}
\tag{E.38}
$$

and

$$
\begin{aligned}
u^{n,j_1,1} &= \frac{1}{c_{j_1,1}} \sum_{k=1}^{c_{j_1,1}} u_k^{n,j_1,1} \\
&= u^n - \Delta t \left(\sum_{j=1}^{\widetilde{j}_1} 2^{j-1} \left(\widetilde{Z}_j(t^n) u^n + f_j(t^n)\right) + \sum_{j=\widetilde{j}_1+1}^{J} 2^{\widetilde{j}_1} \left(\widetilde{Z}_j(t^n) u^n + f_j(t^n)\right)\right) \\
&\quad - \Delta t 2^{\widetilde{j}_1} \left(\sum_{j=J+1}^{j_1-1} \left(\widetilde{Z}_j(t^n) u^n + f_j(t^n)\right)\right) \\
&\quad - \Delta t 2^{\widetilde{j}_1} \sum_{k=1}^{c_{j_1,1}} \left(\left(\sum_{s=1}^{c_{j_1,1}} A_{k,s}^{j_1,1}(t^n) + S_k^{j_1,1}(t^{n+1})\right) u^n + f_k^{j_1,1}(t^n)\right) + O(\Delta t^2).
\end{aligned}
\tag{E.39}
$$

Repeating the process, we can show that for any $1 \le m^* \le L_{J-1}$, we have

$$
\begin{aligned}
u_{k^*}^{n,j_1,m^*} &= u^n - \Delta t \left(\sum_{j=1}^{\widetilde{j}_1} 2^{j-1} \left(\widetilde{Z}_j(t^n) u^n + f_j(t^n) \right) + \sum_{j=\widetilde{j}_1+1}^{J} 2^{\widetilde{j}_1} \left(\widetilde{Z}_j(t^n) u^n + f_j(t^n) \right) \right) \\
&\quad - \Delta t 2^{\widetilde{j}_1} \left(\sum_{j=J+1}^{j_1-1} \left(\widetilde{Z}_j(t^n) u^n + f_j(t^n) \right) \right) \\
&\quad - \Delta t 2^{\widetilde{j}_1} \sum_{m=1}^{m^*-1} \sum_{k=1}^{c_{j_1,m}} \left(\left(\sum_{s=1}^{c_{j_1,m-1}} A_{k,s}^{j_1,m-1}(t^n) + S_k^{j_1,m-1}(t^{n+1}) \right) u^n + f_k^{j_1,m-1}(t^n) \right) \\
&\quad - \Delta t 2^{\widetilde{j}_1} c_{j_1,m^*} \left(\left(\sum_{s=1}^{c_{j_1,m^*-1}} A_{k^*,s}^{j_1,m^*}(t^n) + S_{k^*}^{j_1,m^*}(t^{n+1}) \right) u^n + f_{k^*}^{j_1,m^*}(t^n) \right) + O(\Delta t^2)
\end{aligned} \tag{E.40}
$$

and

$$
\begin{aligned}
u^{n,j_1,m^*} &= \frac{1}{c_{j_1,m^*}} \sum_{k=1}^{c_{j_1,m^*}} u^{n,j_1,m^*} \\
&= u^n - \Delta t \left(\sum_{j=1}^{\widetilde{j}_1} 2^{j-1} \left(\widetilde{Z}_j(t^n) u^n + f_j(t^n) \right) + \sum_{j=\widetilde{j}_1+1}^{J} 2^{\widetilde{j}_1} \left(\widetilde{Z}_j(t^n) u^n + f_j(t^n) \right) \right) \\
&\quad - \Delta t 2^{\widetilde{j}_1} \left(\sum_{j=J-1}^{j_1-1} \left(\widetilde{Z}_j(t^n) u^n + f_j(t^n) \right) \right) \\
&\quad - \Delta t 2^{\widetilde{j}_1} \sum_{m=1}^{m^*} \sum_{k=1}^{c_{j_1,m}} \left(\left(\sum_{s=1}^{c_{j_1,m-1}} A_{k,s}^{j_1,m}(t^n) + S_k^{j_1,m}(t^{n+1}) \right) u^n + f_k^{j_1,m}(t^n) \right) + O(\Delta t^2).
\end{aligned} \tag{E.41}
$$

Setting $m^* = M_{j_1}$ in (E.40)–(E.41), we compute

$$
\begin{aligned}
\bar{u}_{k^*}^{n,j_1,M_{j_1}} &= \frac{1}{2} u_{k^*}^{n,j_1,M_{j_1}} + \frac{1}{2} u_{k^*}^{n,\widetilde{j}_1,M_{\widetilde{j}_1}} \\
&= u^n - \Delta t \left(\sum_{j=1}^{\widetilde{j}_1-1} 2^{j-1} \left(\widetilde{Z}_j(t^n) u^n + f_j(t^n) \right) + \sum_{j=\widetilde{j}_1+1}^{J} 2^{\widetilde{j}_1-1} \left(\widetilde{Z}_j(t^n) u^n + f_j(t^n) \right) \right) \\
&\quad - \Delta t 2^{\widetilde{j}_1-2} \left(\widetilde{Z}_{\widetilde{j}_1}(t^n) u^n + f_{\widetilde{j}_1}(t^n) \right) \\
&\quad - \Delta t 2^{\widetilde{j}_1-2} \sum_{m=1}^{M_{\widetilde{j}_1}-1} \sum_{k=1}^{c_{\widetilde{j}_1,m}} \left(\left(\sum_{s=1}^{c_{\widetilde{j}_1,m-1}} A_{k,s}^{\widetilde{j}_1,m}(t^n) + S_k^{\widetilde{j}_1,m}(t^{n+1}) \right) u^n + f_k^{\widetilde{j}_1,m}(t^n) \right) \\
&\quad - \Delta t 2^{\widetilde{j}_1-2} c_{\widetilde{j}_1,M_{\widetilde{j}_1}} \left(\left(\sum_{s=1}^{c_{\widetilde{j}_1,M_{\widetilde{j}_1}}} A_{k^*,s}^{\widetilde{j}_1,M_{\widetilde{j}_1}}(t^n) + S_{k^*}^{\widetilde{j}_1,M_{\widetilde{j}_1}}(t^{n+1}) \right) u^n + f_{k^*}^{\widetilde{j}_1,M_{\widetilde{j}_1}}(t^n) \right)
\end{aligned}
$$

$$
\begin{aligned}
&- \Delta t 2^{\widetilde{j}_1-1}\left(\sum_{j=J+1}^{j_1-1}\left(\widetilde{Z}_j(t^n)u^n + f_j(t^n)\right)\right)\\
&- \Delta t 2^{\widetilde{j}_1-1}\sum_{m=1}^{M_{j_1}-1}\sum_{k=1}^{c_{j_1,m}}\left(\left(\sum_{s=1}^{c_{j_1,m-1}} A_{k,s}^{j_1,m}(t^n) + S_k^{j_1,m}(t^{n+1})\right)u^n + f_k^{j_1,m}(t^n)\right)\\
&- \Delta t 2^{\widetilde{j}_1-1} c_{j_1,M_{j_1}}\left(\left(\sum_{s=1}^{c_{j_1,M_{j_1}-1}} A_{k^*,s}^{j_1,M_{j_1}}(t^n) + S_{k^*}^{j_1,M_{j_1}}(t^{n+1})\right)u^n + f_{k^*}^{j_1,M_{j_1}}(t^n)\right) + O(\Delta t^2)
\end{aligned}
\tag{E.42}
$$

and

$$
\begin{aligned}
\bar{u}^{n,j_1,M_{j_1}} &= \frac{1}{c_{j_1,M_{j_1}}}\sum_{k=1}^{c_{j_1,M_{j_1}}}\bar{u}_k^{n,j_1,M_{j_1}}\\
&= u^n - \Delta t\left(\sum_{j=1}^{\widetilde{j}_1-1} 2^{j-1}\left(\widetilde{Z}_j(t^n)u^n + f_j(t^n)\right) + \sum_{j=\widetilde{j}_1}^{J} 2^{\widetilde{j}_1-1}\left(\widetilde{Z}_j(t^n)u^n + f_j(t^n)\right)\right)\\
&\quad - \Delta t 2^{\widetilde{j}_1-1}\sum_{j=J+1}^{j_1}\left(\widetilde{Z}_j(t^n)u^n + f_j(t^n)\right) + O(\Delta t^2).
\end{aligned}
\tag{E.43}
$$

Therefore (E.28)–(E.29) hold for any $J+1 \le j^* \le 2J-1$.

Setting $j^* = 2J-1$ in (E.28)–(E.29), we have

$$
\begin{aligned}
\bar{u}_{k^*}^{n,2J-1,M_{2J-1}} &= u^n - \Delta t\left(\sum_{j=2}^{J}\left(\widetilde{Z}_j(t^n)u^n + f_j(t^n)\right)\right)\\
&\quad - \Delta t 2^{-1}\left(\widetilde{Z}_1(t^n)u^n + f_1(t^n)\right)\\
&\quad - \Delta t 2^{-1}\sum_{m=1}^{M_1-1}\sum_{k=1}^{c_{1,m}}\left(\left(\sum_{s=1}^{c_{1,m-1}} A_{k,s}^{1,m}(t^n) + S_k^{1,m}(t^{n+1})\right)u^n + f_k^{1,m}(t^n)\right)\\
&\quad - \Delta t 2^{-1} c_{1,M_1}\left(\left(\sum_{s=1}^{c_{1,M_1-1}} A_{k^*,s}^{1,M_1}(t^n) + S_{k^*}^{1,M_1}(t^{n+1})\right)u^n + f_{k^*}^{1,M_1}(t^n)\right)\\
&\quad - \Delta t\left(\left(\sum_{j=J+1}^{2J-2}\left(\widetilde{Z}_j(t^n)u^n + f_j(t^n)\right)\right)\right.
\end{aligned}
$$

$$
\begin{aligned}
&- \left(\sum_{m=1}^{M_{2J-1}-1} \sum_{k=1}^{c_{2J-1,m}} \left(\sum_{s=1}^{c_{2J-1,m-1}} A_{k,s}^{2J-1,m}(t^n) + S_k^{2J-1,m}(t^{n+1}) \right) u^n + f_k^{2J-1,m}(t^n) \right) \\
&- c_{2J-1,M_{2J-1}} \left(\sum_{s=1}^{c_{2J-1,M_{2J-1}-1}} \left(A_{k^*,s}^{2J-1,M_{2J-1}}(t^n) + S_{k^*}^{2J-1,M_{2J-1}}(t^{n+1}) \right) u^n + f_k^{2J-1,M_{2J-1}}(t^n) \right) \right)
\end{aligned}
\tag{E.44}
$$

$$
+ O(\Delta t^2),
$$

$$
\bar{u}^{n,2J-1,M_{2J-1}} = u^n - \Delta t \sum_{j=1}^{J} \left(\widetilde{Z}_j(t^n) u^n + f_j(t^n) \right) - \Delta t \sum_{j=J+1}^{2J-1} \left(\widetilde{Z}_j(t^n) u^n + f_j(t^n) \right) + O(\Delta t^2). \tag{E.45}
$$

According to (D.11), we have

$$
\begin{aligned}
u^{n+1} &= \left(\mathcal{I} + \Delta t S^*(t^{n+1}) \right)^{-1} \left(\bar{u}^{n,2J-1,M_{2J-1}} - \Delta t \left(\sum_{s=1}^{c_{2J-1,M_{2J-1}}} A_s^*(t^n) \bar{u}_k^{n,2J-1,M_{2J-1}} + f^*(t^n) \right) \right) \\
&= \left(\mathcal{I} - \Delta t S^*(t^{n+1}) \right) \left(\bar{u}^{n,2J-1,M_{2J-1}} - \Delta t \left(\sum_{s=1}^{c_{2J-1,M_{2J-1}}} A_s^*(t^n) \bar{u}_k^{n,2J-1,M_{2J-1}} + f^*(t^n) \right) \right) \\
&\quad + O(\Delta t^2) \\
&= \bar{u}^{n,2J-1,M_{2J-1}} - \Delta t S^*(t^{n+1}) \bar{u}^{n,2J-1,M_{2J-1}} - \Delta t \left(\sum_{s=1}^{c_{2J-1,M_{2J-1}}} A_s^*(t^n) \bar{u}_k^{n,2J-1,M_{2J-1}} \right. \\
&\quad \left. + f^*(t^n) \right) + O(\Delta t^2) \\
&= u^n - \Delta t \sum_{j=1}^{J} \left(\widetilde{Z}_j(t^n) u^n + f_j(t^n) \right) - \Delta t \sum_{j=J+1}^{2J-1} \left(\widetilde{Z}_j(t^n) u^n + f_j(t^n) \right) \\
&\quad - \Delta t \left(\widetilde{Z}^*(t^n) u^n + f^*(t^n) \right) + O(\Delta t^2)
\end{aligned}
$$

$$
= u^n - \Delta t \widetilde{Z}(t^n) u^n - \Delta t f(t^n) + O(\Delta t^2), \tag{E.46}
$$

where $f(t^n)$ is defined in (E.19).

Putting (E.46) and (E.23) together, we have

$$\|\widetilde{u}^n - u^n\|_\infty = O(\Delta t^2). \tag{E.47}$$

The local error is of $O(\Delta t^2)$. Thus the global error is of $O(\Delta t)$. ∎

REFERENCES

[1] G. Alberti and G. Bellettini, *A non-local anisotropic model for phase transitions: Asymptotic behaviour of rescaled energies*, European J. Appl. Math., 9 (1998), pp. 261–284.

[2] S. Anwar and N. Barnes, *Real image denoising with feature attention*, in Proceedings of the IEEE/CVF International Conference on Computer Vision, 2019, pp. 3155–3164.

[3] V. Badrinarayanan, A. Kendall, and R. Cipolla, *Segnet: A deep convolutional encoder-decoder architecture for image segmentation*, IEEE Trans. Pattern Anal. Mach. Intell., 39 (2017), pp. 2481–2495.

[4] E. Bae, J. Yuan, and X.-C. Tai, *Global minimization for continuous multiphase partitioning problems using a dual approach*, Int. J. Comput. Vision, 92 (2011), pp. 112–129.

[5] M. Benning, E. Celledoni, M. J. Ehrhardt, B. Owren, and C.-B. Schhönlieb, *Deep learning as optimal control problems: Models and numerical methods*, J. Comput. Dyn., 6 (2019), pp. 171–198.

[6] A. Bonito, A. Caboussat, and M. Picasso, *Operator splitting algorithms for free surface flows: Application to extrusion processes*, in Splitting Methods in Communication, Imaging, Science, and Engineering, Springer, New York, 2016, pp. 677–729.

[7] Y. Boykov and G. Funka-Lea, *Graph cuts and efficient nd image segmentation*, Int. J. Comput. Vision, 70 (2006), pp. 109–131.

[8] Y. Boykov and V. Kolmogorov, *An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision*, IEEE Trans. Pattern Anal. Mach. Intell., 26 (2004), pp. 1124–1137.

[9] Y. Boykov, O. Veksler, and R. Zabih, *Fast approximate energy minimization via graph cuts*, IEEE Trans. Pattern Anal. Mach. Intell., 23 (2001), pp. 1222–1239.

[10] E. Celledoni, M. J. Ehrhardt, C. Etmann, R. I. McLachlan, B. Owren, C.-B. Schonlieb, and F. Sherry, *Structure-preserving deep learning*, European J. Appl. Math., 32 (2021), pp. 888–936.

[11] T. F. Chan and T. P. Mathew, *Domain decomposition algorithms*, Acta Numer., 3 (1994), pp. 61–143.

[12] T. F. Chan and L. A. Vese, *Active contours without edges*, IEEE Trans. Image Process., 10 (2001), pp. 266–277.

[13] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, *DeepLab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected CRFs*, IEEE Trans. Pattern Anal. Mach. Intell., 40 (2017), pp. 834–848.

[14] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, *Encoder-decoder with atrous separable convolution for semantic image segmentation*, in Proceedings of the European Conference on Computer Vision (ECCV), 2018, pp. 801–818.

[15] C.-W. Cheng, C. Runkel, L. Liu, R. H. Chan, C.-B. Schönlieb, and A. I. Aviles-Rivero, *Continuous U-Net: Faster, Greater and Noiseless*, http://arxiv.org/abs/2302.00626, 2023.

[16] M.-M. Cheng, N. J. Mitra, X. Huang, P. H. Torr, and S.-M. Hu, *Global contrast based salient region detection*, IEEE Trans. Pattern Anal. Mach. Intell., 37 (2014), pp. 569–582.

[17] L.-J. Deng, R. Glowinski, and X.-C. Tai, *A new operator splitting method for the Euler elastica model for image smoothing*, SIAM J. Imaging Sci., 12 (2019), pp. 1190–1230.

[18] Y. Duan, Q. Zhong, X.-C. Tai, and R. Glowinski, *A fast operator-splitting method for Beltrami color image denoising*, J. Sci. Comput., 92 (2022), pp. 1–28.

[19] W. E, *A proposal on machine learning via dynamical systems*, Commun. Math. Stat., 5 (2017), pp. 1–11, https://doi.org/10.1007/s40304-017-0103-z.

[20] W. E, C. Ma, and L. Wu, *Machine learning from a continuous viewpoint,* I, Sci. China Math., 63 (2020), pp. 2233–2266, https://doi.org/10.1007/s11425-020-1773-8.

[21] J. Eckstein and D. P. Bertsekas, *On the Douglas-Rachford splitting method and the proximal point algorithm for maximal monotone operators*, Math. Program., 55 (1992), pp. 293–318.

[22] S. Esedoḡlu and F. Otto, *Threshold dynamics for networks with arbitrary surface tensions*, Comm. Pure Appl. Math., 68 (2015), pp. 808–864.
[23] M. Fortin and R. Glowinski, *On decomposition-coordination methods using an augmented Lagrangian*, in Augmented Lagrangian Methods: Applications to the Numerical Solution of Boundary-Value Problems, Stud. Math. Appl. 15, Elsevier, New York, 1983, pp. 97–146.
[24] K. Funahashi, *On the approximate realization of continuous mappings by neural networks*, Neural Networks, 2 (1989), pp. 183–192.
[25] S. Geman and D. Geman, *Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images*, IEEE Trans. Pattern Anal. Mach. Intell., 6 (1984), pp. 721–741.
[26] R. Glowinski, *Finite element methods for incompressible viscous flow*, Handb. Numer. Anal., 9 (2003), pp. 3–1176.
[27] R. Glowinski and P. Le Tallec, *Augmented Lagrangian and Operator-Splitting Methods in Nonlinear Mechanics*, Stud. Appl. Numer. Math. 9, SIAM, Philadelphia, 1989.
[28] R. Glowinski, S. Leung, and J. Qian, *A penalization-regularization-operator splitting method for Eikonal based traveltime tomography*, SIAM J. Imaging Sci., 8 (2015), pp. 1263–1292.
[29] R. Glowinski, H. Liu, S. Leung, and J. Qian, *A finite element/operator-splitting method for the numerical solution of the two dimensional elliptic Monge–Ampère equation*, J. Sci. Comput., 79 (2019), pp. 1–47.
[30] R. Glowinski, S. Luo, and X.-C. Tai, *Fast operator-splitting algorithms for variational imaging models: Some recent developments*, in Processing, Analyzing and Learning of Images, Shapes, and Forms: Part 2, Handb. Numer. Anal. 20, Elsevier, New York, 2019, pp. 191–232.
[31] R. Glowinski, S. J. Osher, and W. Yin, *Splitting Methods in Communication, Imaging, Science, and Engineering*, Springer, New York, 2017.
[32] R. Glowinski, T.-W. Pan, and X.-C. Tai, *Some facts about operator-splitting and alternating direction methods*, in Splitting Methods in Communication, Imaging, Science, and Engineering, Springer, New York, 2016, pp. 19–94.
[33] E. Haber and L. Ruthotto, *Stable architectures for deep neural networks*, Inverse Problems, 34 (2018), pp. 1–23, https://doi.org/10.1088/1361-6420/aa9a90.
[34] E. Haber, L. Ruthotto, E. Holtham, and S. H. Jun, *Learning across scales—Multiscale methods for convolution neural networks*, in Proceedings of the 32nd AAAI Conference on Artificial Intelligence, 2018, pp. 3142–3148, https://doi.org/10.1609/aaai.v32i1.11680.
[35] J. He and J. Xu, *MgNet: A unified framework of multigrid and convolutional neural network*, Sci. China Math., 62 (2019), pp. 1331–1354, https://doi.org/10.1007/s11425-019-9547-2.
[36] Y. He, S. H. Kang, and H. Liu, *Curvature regularized surface reconstruction from point clouds*, SIAM J. Imaging Sci., 13 (2020), pp. 1834–1859.
[37] K. Hornik, M. Stinchcombe, and H. White, *Multilayer feedforward networks are universal approximations*, Neural Networks, 2 (1989), pp. 359–366, https://doi.org/10.1016/0893-6080(89)90020-8.
[38] P. Iakubovskii, *Segmentation Models PyTorch*, https://github.com/qubvel/segmentation_models.pytorch, 2019.
[39] A. Khapalov, *Bilinear control for global controllability of the semilinear parabolic equations with superlinear terms*, Control Of Nonlinear Distributed Parameter Systems, 2 (2001), https://doi.org/10.1201/9780203904190.ch7.
[40] Kaggle, 2018 *Data Science Bowl Dataset*, https://www.kaggle.com/competitions/data-science-bowl-2018/overview, 2018.
[41] Y. Lan, Z. Li, J. Sun, and Y. Xiang, *DOSnet as a Non-Black-Box PDE Solver: When Deep Learning Meets Operator Splitting*, preprint, arXiv:2212.05571, 2022.
[42] H. Liu, R. Glowinski, S. Leung, and J. Qian, *A finite element/operator-splitting method for the numerical solution of the three dimensional Monge–Ampere equation*, J. Sci. Comput., 81 (2019), pp. 2271–2302.
[43] H. Liu, X.-C. Tai, and R. Glowinski, *An operator-splitting method for the Gaussian curvature regularization model with applications to surface smoothing and imaging*, SIAM J. Sci. Comput., 44 (2022), pp. A935–A963.
[44] H. Liu, X.-C. Tai, R. Kimmel, and R. Glowinski, *A color elastica model for vector-valued image regularization*, SIAM J. Imaging Sci., 14 (2021), pp. 717–748.

[45] H. Liu and D. Wang, *Fast operator splitting methods for obstacle problems*, J. Comput. Phys., 477 (2023), 11941, https://doi.org/10.1016/j.jcp.2023.111941.
[46] J. Liu, X.-c. Tai, H. Huang, and Z. Huan, *A fast segmentation method based on constraint optimization and its applications: Intensity inhomogeneity and texture segmentation*, Pattern Recognit., 44 (2011), pp. 2093–2108, https://doi.org/10.1016/j.patcog.2011.02.022.
[47] J. Liu, X. Wang, and X.-C. Tai, *Deep convolutional neural networks with spatial regularization, volume and star-shape priors for image segmentation*, J. Math. Imaging Vision, 64 (2022), pp. 625–645.
[48] J. Long, E. Shelhamer, and T. Darrell, *Fully convolutional networks for semantic segmentation*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 3431–3440.
[49] T. Lu, P. Neittaanmaki, and X.-C. Tai, *A parallel splitting-up method for partial differential equations and its applications to Navier-Stokes equations*, ESAIM Math. Model. Numer. Anal., 26 (1992), pp. 673–708.
[50] G. I. Marchuk, *Splitting and alternating direction methods*, Handb. Numer. Anal., 1 (1990), pp. 197–462.
[51] B. Merriman, J. K. Bence, and S. Osher, *Diffusion Generated Motion by Mean Curvature*, Department of Mathematics, University of California, Los Angeles, 1992.
[52] M. Miranda, Jr., D. Pallara, F. Paronetto, and M. Preunkert, *Short-time heat flow and functions of bounded variation in RN*, Ann. Fac. Sci. Toulouse Math., 16 (2007), pp. 125–145.
[53] A. Mrad, A. Caboussat, and M. Picasso, *A splitting method for the numerical simulation of free surface flows with sediment deposition and resuspension*, Internat. J. Numer. Methods Engrg., 94 (2022), pp. 1724–1743.
[54] T. Pock, A. Chambolle, D. Cremers, and H. Bischof, *A convex relaxation approach for computing minimal partitions*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, IEEE, 2009, pp. 810–817.
[55] R. B. Potts, *Some Generalized Order-Disorder Transformations*, Math. Proc. Cambridge Philos. Soc. 48, Cambridge University Press, Cambridge, 1952, pp. 106–109.
[56] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes 3rd Edition: The Art of Scientific Computing*, Cambridge University Press, Cambridge, 2007.
[57] H. Ranocha, L. Lóczi, and D. I. Ketcheson, *General relaxation methods for initial-value problems with application to multistep schemes*, Numer. Math., 146 (2020), pp. 875–906.
[58] O. Ronneberger, P. Fischer, and T. Brox, *U-Net: Convolutional networks for biomedical image segmentation*, in Proceedings of the International Conference on Medical Image Computing and Computer-Assisted Intervention, Springer, New York, 2015, pp. 234–241.
[59] D. Ruiz-Balet and E. Zuazua, *Neural ODE control for classification, approximation, and transport*, SIAM Rev., 65 (2023), pp. 735–773.
[60] L. Ruthotto and E. Haber, *Deep neural networks motivated by partial differential equations*, J. Math. Imaging Vision, 62 (2020), pp. 352–364.
[61] H. Sun, X.-C. Tai, and J. Yuan, *Efficient and convergent preconditioned ADMM for the Potts models*, SIAM J. Sci. Comput., 43 (2021), pp. B455–B478.
[62] D. Sussillo and O. Barak, *Opening the black box: Low-dimensional dynamics in high-dimensional recurrent neural networks*, Neural Comput., 25 (2013), pp. 626–649.
[63] X. Tai, L. Li, and E. Bae, *The Potts model with different piecewise constant representations and fast algorithms: A survey*, in Handbook of Mathematical Models and Algorithms in Computer Vision and Imaging, Springer, New York, 2021, pp. 1–41.
[64] X.-C. Tai, *Rate of convergence for some constraint decomposition methods for nonlinear variational inequalities*, Numer. Math., 93 (2003), pp. 755–786.
[65] X.-C. Tai and J. Xu, *Subspace correction methods for convex optimization problems*, in Proceedings of the 11th International Conference on Domain Decomposition Methods, London, 1998, pp. 130–139.
[66] X.-C. Tai and J. Xu, *Global and uniform convergence of subspace correction methods for some convex optimization problems*, Math. Comp., 71 (2002), pp. 105–124.

[67] D. Wang, *An efficient iterative method for reconstructing surface from point clouds*, J. Sci. Comput., 87 (2021), 38.
[68] J. Wang, L. Ju, and X. Wang, *An edge-weighted centroidal Voronoi tessellation model for image segmentation*, IEEE Trans. Image Process., 18 (2009), pp. 1844–1858.
[69] K. Wei, K. Yin, X.-C. Tai, and T. F. Chan, *New region force for variational models in image segmentation and high dimensional data clustering*, Ann. Math. Sci. Appl., 3 (2018), pp. 255–286.
[70] J. Xu, *Iterative methods by space decomposition and subspace correction*, SIAM Rev., 34 (1992), pp. 581–613.
[71] J. Yuan, E. Bae, and X.-C. Tai, *A study on continuous max-flow and min-cut approaches*, in Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, IEEE, 2010, pp. 2217–2224.
[72] J. Yuan, E. Bae, X.-C. Tai, and Y. Boykov, *A continuous max-flow approach to Potts model*, in Proceedings of the European Conference on Computer Vision, Springer, New York, 2010, pp. 379–392.
[73] K. Zhang, W. Zuo, Y. Chen, D. Meng, and L. Zhang, *Beyond a Gaussian denoiser: Residual learning of deep CNN for image denoising*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, IEEE, 2017, pp. 5743–5752.
[74] D. X. Zhou, *Universality of deep convolutional neural networks*, Appl. Comput. Harmon. Anal., 48 (2020), pp. 787–794, https://doi.org/10.1016/j.acha.2019.06.004.
[75] Z. Zhou, M. M. R. Siddiquee, N. Tajbakhsh, and J. Liang, *UNet++: Redesigning skip connections to exploit multiscale features in image segmentation*, IEEE Trans. Med. Imaging, 39 (2019), pp. 1856–1867.