A Deep Learning Approach for the Computation of Curvature in the Level-Set Method

Luis Ángel Larios Cárdenas[a,*], Frederic Gibou[a,b]

[a]Department of Computer Science, University of California, Santa Barbara, CA 93106
[b]Department of Mechanical Engineering, University of California, Santa Barbara, CA 93106

Abstract

We propose a deep learning strategy to compute the mean curvature of an implicit level-set representation of an interface. Our approach is based on fitting neural networks to synthetic datasets of pairs of nodal ϕ values and curvatures obtained from circular interfaces immersed in different uniform resolutions. These neural networks are multilayer perceptrons that ingest sample level-set values of grid points along a free boundary and output the dimensionless curvature at the center vertices of each sampled neighborhood. Evaluations with irregular (smooth and sharp) interfaces, in both uniform and adaptive meshes, show that our deep learning approach is systematically superior to conventional numerical approximation in the L^2 and L^∞ norms. Our methodology is also less sensitive to steep curvatures and approximates them well with samples collected with fewer iterations of the reinitialization equation, often needed to regularize the underlying implicit function. Additionally, we show that an application-dependent map of local resolutions to neural networks can be constructed and employed to estimate interface curvatures more efficiently than using typically expensive numerical schemes while still attaining comparable or higher precision.

Keywords: Deep Learning, Interface Curvature, Level-Set Method

1. Introduction

Free boundary problems are a large class of models that arise from distant areas that share the same mathematical structure [1]. Among the most common applications one may find models for heat conduction, propagation of fire fronts, interface dynamics, image segmentation, and morphogenesis. There exist two general numerical approaches for evolving interfaces subject to velocity fields: a Lagrangian or explicit formulation, and an Eulerian or implicit formulation.

In the Lagrangian strategy, the interface is discretized into a finite number of pieces, and these are advected by solving an elementary ordinary differential equation. This method enjoys the advantages of simplicity for updating the elements position and thus lead to accurate volume preservation [2]. The Lagrangian approach's accuracy, however, deteriorate rather quickly when the velocity field causes pronounced topology deformations, and one does not resort to special procedures for maintaining boundary smoothness and regularity [3]. In addition, explicit methods can become quite demanding when considering merging and splitting of moving fronts. The combination of the latter numerical techniques together with the Lagrangian formulation to solve the corresponding evolution ODE are collectively referred to as front-tracking methods. Some representative developments on this branch include the simulation of the unsteady motion of bubbles using a method for incompressible multi-fluid flows in [4]; the numerical tool to study the stable and unstable solidification of pure substances by [5]; the numerical simulation of film boiling with a complex motion of a liquid-vapor interface in [6]; and the explicitly tracked unsteady motion of an infinitely thin premixed flame separating burned and un-burned gas in [7]. An excellent review is given in [2].

Implicit methods circumvent the problems with boundary element deformations, and complicated topology maintenance procedures by using an implicit function to evolve the interface [3]. Typical Eulerian formulations include the phase-field [8–18], the volume-of-fluid [19–31], and the level-set methods [3, 24, 29, 32–47]. The main advantage of these methods is their natural ability to handle complex changes in topology, as depicted in figure 1.

*Corresponding author: lal@cs.ucsb.edu

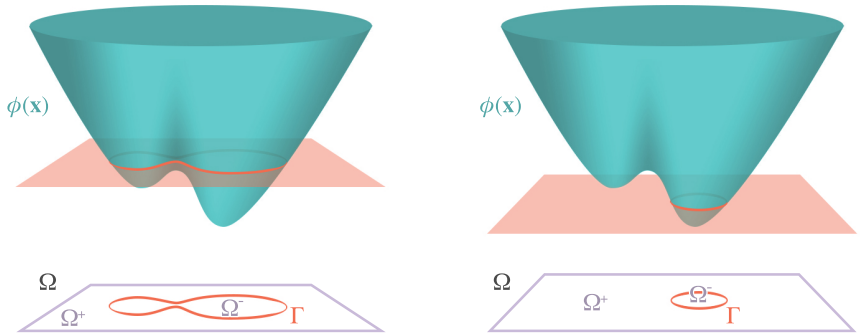


Figure 1: Time evolution of a level-set function $\phi(\mathbf{x})$, its zero isocontour Γ, and the associated computational domain Ω.

Phase field models are implicit methods that have been used extensively in the materials sciences to treat phase transitions or decompositions with complicated morphologies. The methodology is based on a numerical solution of the nonlinear Cahn-Hilliard or Cahn-Allen equation [9], and introduces a diffuse profile where an auxiliary continuous order parameter at a given point identifies the phase [8]. The phase field approach is a flexible model where the interfacial location needs not be explicitly tracked. For this reason, the framework has been considerably popular in the investigation of micro-structural pattern formations in alloys and in the simulation of solidification phenomena, such as dendritic crystallization under various conditions (see, for example, [14–16]). These methods, however, experience difficulties when handling large domains, and, notably, the ability to adjust the interface width to physically unrealistic values results in a loss of detail to represent sharp transitions [10]. In turn, these issues translate into severe time step restrictions and accuracy. It is also not clear how the parameters of the phase-field representation are related to physical quantities.

The volume of fluids approach (or VOF), pioneered by [19–21], is a popular and convenient method in computational fluid dynamics that uses volume fractions of fluid as index function values at each computational cell [31]. In two-phase flows, a unit function value corresponds to a cell full of fluid, while a zero value is associated with an empty cell. Consequently, any mixed element with an intermediate value contains part of a free surface or material interface [21] that is commonly assembled through a non-straightforward piecewise linear or parabolic reconstruction [25, 26]. By construction, VOF methods conserve volume and mass locally, treat intersecting free boundaries automatically, require a minimum of stored information, and can be easily carried on to three dimensions [21]. However, extending the algorithm to handling more materials is nontrivial [23], and employing only the volume fractions to accurately compute interface curvatures, normal vectors, and forces is challenging because of the discontinuous nature of the VOF representation. To address the difficulty and improve on the accuracy of interfacial curvature calculations in the VOF method, Qi *et al.* [31] proposed a machine learning strategy. The authors took the nine volume fractions of computational cells centered at a point of interest as inputs to a one-hidden-layer, feedforward neural network, and outputted the continuous dimensionless curvature for that cell. Their neural model was trained for a regular grid with a sufficiently large synthetic dataset, spanning a wide range of curvatures and orientations from circular interfaces. Qi *et al.* showed that their neural network produced satisfactory learning and evaluation results at distinct resolutions, both for a static sinusoidal wave and in a flow solver with an irregular, three-petaled free boundary.

The level-set method is a powerful framework for tracking arbitrary interfaces, where these are captured as the zero isocontour of an implicit function (see section 2). The interface is advected under a velocity field by solving a Hamilton-Jacobi equation using finite difference, finite volume, or finite element schemes. Consequently, the dynamics of the moving boundary imply no need for asymptotic analysis, and only standard time step restrictions for stability and consistency are required [39]. Because of its numerical nature, however, the most important level-set method's difficulties are concerning the smoothness of the underlying implicit surface and the conservation of mass when the interface undergoes severe stretching or tearing [24, 33]. Hybrid methods have been introduced to improve mass conservation. In the coupled level-set and VOF method (CLSVOF) by Sussman and Puckett [24], the VOF index function complements the level-set implicit function to transport the free boundary. There, a piecewise linear interface calculation (PLIC) is employed to correct for and preserve the enclosed fluid mass or volume, while the smooth level-set function is used to compute surface normal vectors and curvatures [44]. More recently, and also exploiting the advantages of both the VOF and level-set frameworks, Yang *et al.* [29] have presented an adaptive coupled level-set and volume-of-fluid (ACLSVOF) volume tracking method for unstructured triangular grids. Unlike previous works,

Yang and coauthors' adaptive-mesh algorithm is founded on an analytical PLIC for triangular grids that efficiently and accurately resolves complex topological changes, regions of steep curvature, and near contact regions of colliding fronts. Another hybrid approach to also address the mass-conserving shortcoming is given by Enright *et al.* [37], where a set of marker particles are randomly seeded near the interface and are passively advected with a flow field. These Lagrangian marker particles are subsequently used to rebuild the level-set function in regions that are under-resolved.

The level-set method's mass loss/gain problem is especially worse when coarse grids are used for discretization [29]. Thus, multiple studies have focused on applying mesh refinement techniques [48, 49] to efficiently handle large numbers of spatial elements near evolving fronts. On this line of research, Chen *et al.* [38, 42] and Min and Gibou [40, 41, 50, 51] have developed node-based, robust level-set tools and extrapolation and reinitialization schemes, that have been incorporated to adaptive solvers in a variety of experimental situations (see [45] and the references therein). The level-set methods on quadtrees (in 2D) and octrees (in 3D) proposed by Min and Gibou in [39] are representative of the application of local grid refinement to reduce computational costs associated with high resolutions. In addition, Mirzadeh *et al.* [43] have extended these level-set technologies on quad- and octrees to parallel algorithms for distributed memory machines using a domain decomposition technique. Mirzadeh and coauthors' algorithms were presented in modeling the solidification process by solving a Stefan problem and were implemented using a combination of the `MPI` standard [52] and the `p4est` [53] library. Local level-set methods have also been introduced by limiting the band of grid points processed around the interface [54] or by using efficient data structures that only records the set of grid points around the interface [55–57]. The authors in [55] noted that only recording the set of adjacent grids points can introduce numerical noise when reinitializing the level-set equation and that level-set methods based on octree have superior performance.

The interface mean curvature is one of the most important derived geometric properties that is used in free boundary problems for its relation to surface tension in physics and its regularization property in optimization problems. Thus, it is crucial to compute the mean curvature accurately from interface representations. In the case of the level-set method, the accuracy of the curvature approximation is dependent of the smoothness of the level-set function. The smoothness can be enforced by reinitializing the level-set function using iterative procedures. The work of [58] introduced a high-order accurate discretization of the reinitialization equation that produced second-order accurate curvature computation in the L^∞ norm; this work was based on the observation by Russo and Smereka [59] that standard level-set reinitialization schemes did not account for characteristic propagations. However, high-order reinitialization procedures are costly and cannot always be achieved on non-uniform grids. Furthermore, fast reinitialization schemes such as the Fast Marching Method [60–62] and the Fast Sweeping Method [63–66] do not produce smooth enough level-set functions to guarantee accurate curvatures.

Inspired by the achievements in [31], we propose a deep learning strategy to calculate the mean curvature in the level-set framework. Our approach is based on fitting one or more (deep) neural networks to synthetic datasets of pairs of nodal ϕ values and curvatures obtained from circular interfaces immersed in different uniform resolutions. The neural models we present are multilayer perceptrons that ingest sample level-set values of grid points along a free boundary and output the dimensionless curvature of the center nodes at each sampled neighborhood. Given an underlying implicit function, ϕ, our deep learning method further seeks to reduce the cost associated with level-set reinitialization into a signed distance function. Remarkably, our tests with irregular interfaces demonstrate that our deep learning approach (1) systematically outperforms the conventional numerical method and (2) makes it possible to estimate interface curvatures with higher precision at relatively cheaper level-set reinitialization operations.

We present the essentials of the level-set method in section 2 before describing the deep learning curvature approximation in section 3. Section 4 is devoted to numerical experiments, and section 5 draws some conclusions.

2. The Level-Set Method

The level-set method, introduced by Osher and Sethian in their seminal work [32], is an Eulerian formulation of the numerical evolution of an implicit interface, captured as the zero isocontour of the level-set function $\phi(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}$. Given a prescribed computational domain $\Omega \in \mathbb{R}^n$ and the interface Γ, ϕ is defined as the implicit, signed distance function to the interface:

$$\phi(\mathbf{x}) = \begin{cases} -d, & \mathbf{x} \in \Omega^-, \\ +d, & \mathbf{x} \in \Omega^+, \\ 0, & \mathbf{x} \in \Gamma, \end{cases} \tag{1}$$

where d is the Euclidean distance from $\mathbf{x}$ to Γ, and Ω^- and Ω^+ are the resulting inside and outside regions in the domain, which is correspondingly partitioned by the interface.

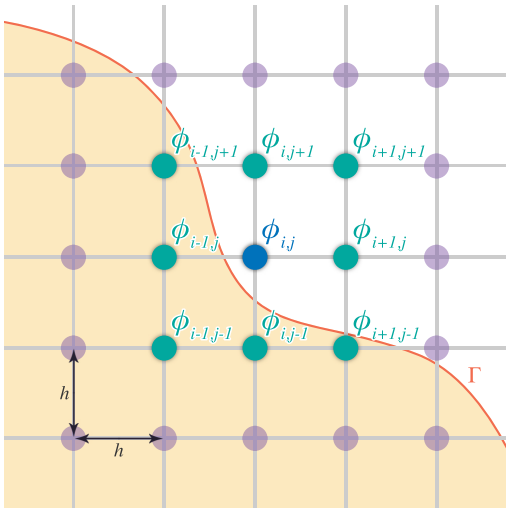


Figure 2: Level-set values used in the standard computation of the curvature at a grid point (i, j).

Assuming that $\mathbf{v}(\mathbf{x})$ is a velocity field defined for all $\mathbf{x} \in \Omega$, the evolution of the implicit function $\phi(\mathbf{x})$ satisfies:

$$\phi_t + \mathbf{v} \cdot \nabla\phi = 0, \tag{2}$$

which is known as the level-set equation.

Figure 2 depicts the zero level set, Γ, of some implicit function $\phi(x, y) : \mathbb{R}^2 \to \mathbb{R}$, traveling across a portion of a discretized, two-dimensional domain, Ω, and the 9 values of ϕ employed in the numerical estimation of the interface curvature, κ, at (i, j). The formula to approximate κ is:

$$\kappa = \nabla \cdot \frac{\nabla\phi}{|\nabla\phi|} = \frac{\phi_x^2\phi_{yy} - 2\phi_x\phi_y\phi_{xy} + \phi_y^2\phi_{xx}}{\left(\phi_x^2 + \phi_y^2\right)^{3/2}}, \tag{3}$$

where the partial derivatives of ϕ at the vertex (i, j) can be approximated using second-order-accurate central differences involving $\phi_{p,q}$, for $i - 1 \leq p \leq i + 1$ and $j - 1 \leq q \leq j + 1$ in the case of a uniform Cartesian grid.

In practice, one opts for a signed distance function, $\phi(x, y)$, because it produces robust numerical results [67]. However, as Γ evolves over time, ϕ diverges from its signed-distance property and develops noisy features that get amplified when used in the approximation of partial derivatives in equation (3). Consequently, it is standard to periodically reinitialize ϕ into a signed distance function. The equation for level-set reinitialization was first introduced by [67]:

$$\phi_t + S(\phi^0)(|\nabla\phi| - 1) = 0, \tag{4}$$

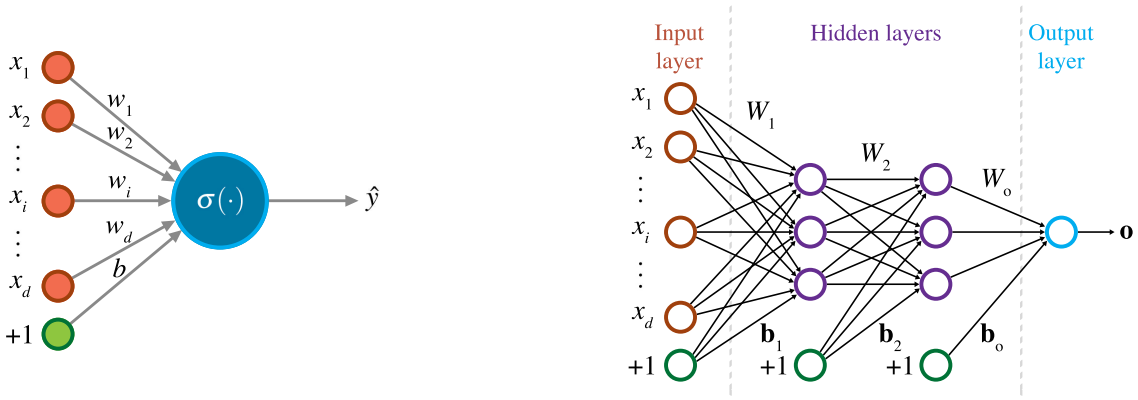
where $S(\phi^0)$ is a smoothed-out signed function defined using the initial values of the level-set function to be reinitialized, ϕ^0.

In order to evolve equation (4) to a steady state (i.e. $\phi_t = 0$ and $|\nabla\phi| = 1$), one uses a TVD Runge-Kutta scheme in time and a Godunov spatial discretization of the Hamiltonian $H(\nabla\phi) = S(\phi^0)(|\nabla\phi| - 1)$ in a combination of Euler steps. It is typical to use between 5 and 20 iterations to reinitialize a level-set function, with more iterations translating into smoother reinitialized ϕ values and thus a more accurate computation of κ. However, the computational cost associated with the reinitialization procedure is significant when using a large number of iterations. It is thus desirable to seek a method that accurately computes κ while being less dependent on the number of reinitialization steps.

3. Deep Learning and Neural Network Curvature Approximation

3.1. Fundamentals

An artificial neural network is a computational graph of elementary units that lie interconnected in some particular way to compute a function of its inputs by using connection weights as intermediate parameters [68]. Neural networks



(a) Perceptron with a single computational unit

(b) Multilayer perceptron with 2 hidden layers

Figure 3: The artificial neural network basic architectures.

are powerful supervised learning methods [69] that learn a target function by successively adapting their weights in order to minimize an error or loss function given their current parameter configuration and a set $\mathcal{D}$ of training data pairs.

The simplest neural network architecture is known as **perceptron** and consists of a single input layer and one non-linear output neuron (see figure 3a). Given an input layer with d linear units, $\mathbf{x} \in \mathbb{R}^d$, a vector of edge parameters, $\mathbf{w} \in \mathbb{R}^d$, and an offset bias, $b \in \mathbb{R}$, the neural prediction, $\hat{y}$, is computed as:

$$\hat{y} = \sigma(\mathbf{w} \cdot \mathbf{x} + b) = \sigma\left(\sum_{i=1}^{d} w_i x_i + b\right), \tag{5}$$

where $\sigma(\cdot)$ plays the role of an activation function. By varying the choice of σ, a perceptron can easily simulate machine learning models such as linear and logistic regression. Traditional $\sigma(\cdot)$ options include the step-function, sigmoids, and hyperbolic tangent, although more recent non-linearities have become commonplace, such as rectified linear units (ReLUs), leaky rectified linear units, and exponential linear units (ELUs) [69].

The training algorithm of a perceptron works by feeding each input sample $(\mathbf{x}_p, y_p) \in \mathcal{D}$ into the network and outputting a prediction $\hat{y}_p$. The weights are then updated based on the error $e(\mathbf{x}_p) = y_p - \hat{y}_p$ as follows:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha e(\mathbf{x}_p)\mathbf{x}_p, \qquad \text{or} \qquad \mathbf{w} \leftarrow \mathbf{w} + \alpha \sum_{(\mathbf{x}_p, y_p) \in \mathcal{B}} e(\mathbf{x}_p)\mathbf{x}_p, \tag{6}$$

where α is referred to as the neural network learning rate, and $\mathcal{B} \subset \mathcal{D}$ is a batch of randomly selected samples from the training dataset.

The perceptron learning process is iteratively executed until convergence. Every sample pair $(\mathbf{x}_p, y_p) \in \mathcal{D}$ may be cycled through many times, and each such iteration is known as an epoch [68]. In particular, the perceptron learning algorithm based on equation (6) is considered a (mini-batch) stochastic gradient-descent method; it minimizes the squared error of prediction by performing gradient-descent updates with respect to randomly selected (batches of) training samples [68].

Multilayer neural networks are powerful extension models of simple perceptrons. A multilayer, feedforward neural network consists of a number of computing (perceptron) units arranged in multiple layers, where the input layer transmits data into the network, the hidden layers perform some additional (non)linear calculations, and the output layer yields the final results or predictions. The output from each intermediate layer is carried over as the (weighted) input to the next level in the architecture, and this gives rise to a composition of functions that is evaluated at every individual node [68]. The output layer is usually a simple classifier (a logistic regression or a soft-max function for categorical data), or a linear regression function in the case of continuous outputs [69]. Our following discussion centers around neural networks built to predict a continuous output. Figure 3b depicts a multilayer, feedforward, and fully connected neural network with 2 hidden layers and a one-unit output layer.

The multilayer neural network parameters, $\boldsymbol{\Theta}$, are contained in bias vectors, $\mathbf{b}_m$, and weight matrices, W_m, for $1 \leq m \leq M+1$, where M is the number of hidden layers. The weight matrix for the connections between the input with d units and the first hidden layer with m_1 neurons is denoted as $W_1 \in \mathbb{R}^{d \times m_1}$; the weights between the r^{th} and the $(r+1)^{th}$ hidden layer are provided in $W_{r+1} \in \mathbb{R}^{m_r \times m_{r+1}}$; and the matrix that holds the weights between the last hidden layer and the output layer with o units is given by $W_o \in \mathbb{R}^{m_M \times o}$ [68]. The layer-wise, feed-forward network recurrence equations that transform an input, $\mathbf{x}$, into the predicted output, $\mathbf{o}$, are expressed as:

$$\begin{cases} \mathbf{h}_1 = \sigma(W_1^T \mathbf{x} + \mathbf{b}_1), & \text{from input to first hidden layer,} \\ \mathbf{h}_{r+1} = \sigma(W_{r+1}^T \mathbf{h}_r + \mathbf{b}_{r+1}), 1 \leq r \leq M-1, & \text{from } r^{th} \text{ to } (r+1)^{th} \text{ hidden layers,} \\ \mathbf{o} = \sigma(W_o^T \mathbf{h}_M + \mathbf{b}_o), & \text{from hidden to output layer,} \end{cases}$$

where $\sigma(\cdot)$ is, like in the perceptron case, a non-linear, element-wise activation function, and $\mathbf{b}_m$ is a vector of bias offsets.

Training feedforward networks consists of constructing a cost or loss function and using gradient descent to optimize it to find the best set of weights and biases, $\boldsymbol{\Theta}$ [69]. For continuous output data, the most common loss functions include the Mean Squared Error (or MSE)

$$e(\boldsymbol{\Theta}) = \frac{1}{n} \sum_{p=1}^{n} \left(y_p - \hat{y}_p(\boldsymbol{\Theta}) \right)^2, \tag{7}$$

and the Mean Absolute Error (or MAE)

$$e(\boldsymbol{\Theta}) = \frac{1}{n} \sum_{p=1}^{n} \left| y_p - \hat{y}_p(\boldsymbol{\Theta}) \right|, \tag{8}$$

where n is the cardinality of $\mathcal{D}$. It is also not uncommon to include regularization terms in equations (7) and (8) to reduce the impact of data intrinsic noise and aid in model generalization [69].

As in other supervised learning models, gradient descent is used to minimize equations (7) or (8). Basically, the neural network parameters, $\boldsymbol{\Theta}$, are iteratively updated by moving in the opposite direction of the gradient of the cost function, $\nabla_{\boldsymbol{\Theta}} e(\boldsymbol{\Theta})$. Some optimizers often employed to perform gradient descent include Stochastic Gradient Descent and, more recently, Nesterov, RMSProp, Adadelta, Adagrad, and Adam. We refer the interested reader to [68–70] for a more complete study of gradient descent and loss function optimizers.

Calculating the gradients for feedforward networks is done through the backpropagation algorithm, which exploits the layered architecture of the neural models for efficient computations [69]. Backpropagation computes the gradient of a composition of functions by a direct application of dynamic programming. The process contains two phases: a forward phase, in which the network outputs and the local derivatives are calculated, and a backward phase where the products of these local values are accumulated and the gradient of the loss function with respect to the different weights is learned and used to update the parameters, $\boldsymbol{\Theta}$ [68]. We refer the interested reader to [68, 69] for the details on the backpropagation derivation and its detailed application to neural networks training.

3.2. Training a Neural Network to Approximate Curvature

Recent advancements and experiments in deep learning have shown that multilayer perceptrons with several hidden layers possess the mathematical capacity and power to approximate complex functions [69]. Under this principle, and seeking to build a model that improves on curvature accuracy at a fraction of the number of iterations for level-set reinitialization (section 2), we follow the machine learning approach in [31] to propose a level-set curvature neural network. The goal of our neural model is to estimate $h\kappa$ from equation (3) through the function:

$$h\kappa_{i,j} = f \begin{pmatrix} \phi_{i-1,j+1}, & \phi_{i,j+1}, & \phi_{i+1,j+1}, \\ \phi_{i-1,j}, & \phi_{i,j}, & \phi_{i+1,j}, \\ \phi_{i-1,j-1}, & \phi_{i,j-1}, & \phi_{i+1,j-1} \end{pmatrix}, \tag{9}$$

where $h\kappa_{i,j}$ is the dimensionless curvature of the zero isocontour of the level-set function $\phi(\mathbf{x}) : \mathbb{R}^2 \to \mathbb{R}$, at the grid point (i, j) (see figure 2).

Our level-set curvature neural network is a supervised model that employs training samples $(\boldsymbol{\phi}_p, o_p) \in \mathcal{D}$, $\boldsymbol{\phi}_p \in \mathbb{R}^9$ and $o_p \in \mathbb{R}$, to learn the association in equation (9) between $o_p = h\kappa$ and the nodal ϕ values. As in [31], we train,

validate, and test our neural network with synthetic datasets extracted from circular interfaces, Γ, of varying radii. The learning interfaces correspond to the zero level sets of the three-dimensional implicit functions:

$$\phi_{cs}(x, y) = \sqrt{(x - x_0)^2 + (y - y_0)^2} - r, \tag{10}$$

and

$$\phi_{cn}(x, y) = (x - x_0)^2 + (y - y_0)^2 - r^2, \tag{11}$$

where (x_0, y_0) are the coordinates of the center of Γ, and r is the circle radius. We note that while equation (10) is a signed distance function, equation (11) is not and therefore needs to be reinitialized via the numerical techniques introduced in section 2. Following [31], we opt for circular interfaces because once their radii are known, the expected or target dimensionless curvatures can be readily calculated as h/r.

The neural network learning spatial domain is defined as the non-periodic unit square $\Omega \in [0, 1] \times [0, 1]$, which we discretize uniformly along the x and y Cartesian directions by a prescribed amount of grid points, ρ. Given a desired resolution and its interval $h = \Delta x = \Delta y = 1/(\rho - 1)$, the number of circular interfaces (i.e. signed and non-signed distance functions) can be obtained as:

$$\nu(\rho) = \left\lceil \frac{\rho - 8.2}{2} \right\rceil + 1, \tag{12}$$

which ensures that any resolution gets sufficient coverage with training interfaces.

To make sure that the learning circular interfaces are fully contained in Ω, we randomly set their center coordinates to $(x_0, y_0) \in [0.5 - h/2, 0.5 + h/2]^2$ and restrict their minimum and maximum radii to $1.6h$ and $1/2 - 2h$, respectively. Furthermore, we generate samples for each circle radius r_ι, $1 \leq \iota \leq \nu(\rho)$, up to 5 times, in order to introduce randomness (i.e. noise) during the learning stage. These policies, together with equation (12), guarantee that: (1) at least four grid points lie inside the smallest circle, (2) any node (i, j) adjacent to Γ has a well-defined stencil of 8 neighbors surrounding it (see figure 2), and (3) the center variation for the same circle radius helps with neural network generalization [68, 69].

The learning set, $\mathcal{D}$, is then a collection of N samples of the form:

$$\left(\boldsymbol{\phi}_p, o_p\right) = \left(\left[\phi_{i-1,j+1}, \phi_{i,j+1}, \phi_{i+1,j+1}, \phi_{i-1,j}, \phi_{i,j}, \phi_{i+1,j}, \phi_{i-1,j-1}, \phi_{i,j-1}, \phi_{i+1,j-1}\right], h\kappa_{i,j}\right), \tag{13}$$

where a sample data point is generated for any node (i, j) that either sits on Γ or has an outgoing vertical or horizontal edge that is crossed by the boundary. In particular, the input values, $\boldsymbol{\phi}_p$, in equation (13), are obtained from the corresponding circular interfaces in both equations (10) and (11) at some space resolution. In the case of $\phi_{cn}(x, y)$ above, we employ 5, 10, 15, and 20 iterations to reinitialize the level-set into a signed distance function. This allows us to inject a greater diversity of input level-set values, $\boldsymbol{\phi}_p$, for the same expected interface curvature. In addition, for all instances $(\boldsymbol{\phi}_p, o_p)$, we collect their negated version, $(-\boldsymbol{\phi}_p, -o_p)$, so that the neural network can also account for negative curvatures.

We have used **TensorFlow** [71] and **Keras** [70] to build a dictionary of three fully connected neural networks that address the level-set method curvature accuracy problem as outlined above. Each neural model is respectively designed for a space resolution, ρ, of 256, 266, and 276 grid points per unit length, and is trained to solve for $h\kappa$ given their individual sets of learning sample pairs, $\mathcal{D}$. In all of these instances, we followed the literature convention [68, 69] and split the (shuffled) datasets into three parts: training (70%), validation (15%), and testing (15%). Table 1 contains the number of samples we collected under this framework. We point out that the samples in the learning stage and in the upcoming analyses in section 4 were generated either in vanilla Python (equation (10)) or in C++ by implementing the parallel adaptive level-set method of [43] (equation (11) and other irregular interfaces described below).

In accordance to equation (13), our proposed level-set curvature neural networks have input layers with nine linear units and a single linear neuron in their output layers to estimate continuous $h\kappa$ values. Moreover, to minimize the

ρ	Training	Testing	Validation
256	3'145,410	674,017	674,017
266	3'399,948	728,560	728,560
276	3'664,188	785,184	785,184

Table 1: Number of samples collected for each neural network.

ρ	Training Epochs	Hidden Layers	Units per Hidden Layer	Testing MSE	Testing MAE	Max AE
256	103	4	[128, 128, 128, 128]	3.86×10^{-7}	2.91×10^{-4}	0.154
266	72	4	[140, 140, 140, 140]	3.64×10^{-7}	2.70×10^{-4}	0.122
276	33	4	[140, 140, 140, 140]	5.42×10^{-7}	3.01×10^{-4}	0.164

Table 2: Best neural network configurations, their testing set statistics, and their maximum absolute error over $\mathcal{D}$.

impact of possibly disparate feature ϕ values in $\mathcal{D}$, we normalized $\boldsymbol{\phi}_p$ in equation (13) and used z-scores during the networks' learning and evaluation stages. We also utilized only ReLU neurons in all of the intermediate layers because of their beneficial non-saturating property that ameliorates the problem of vanishing and exploding gradients [69]. For training, we considered batches of size 32, selected the mean square error loss function (see equation (7)), and employed the Adam optimizer [70] with a learning rate of 0.00015 to minimize the cost associated with the $h\kappa$ approximations. In addition, we exercised the early-stopping technique, which monitored the models' generalization performance through the mean absolute error (see equation (8)) of the validation sets. Furthermore, we empirically determined that a maximum patience of 30 epochs sufficed to halt long-lasting, overfitting-prone training processes. In the end, the learning stage for each of the three networks never surpassed 200 epochs, including the 30-epoch patience. The best configurations for our three-element neural dictionary are provided in table 2, together with some outcome statistics from the learning stage.

Figure 4 illustrates the quality of the fit achieved by our neural network adapted for a space resolution of $\rho = 266$ grid points per unit length. Similar plots for the other two neural models were also obtained. Those results demonstrate that our neural model is better at estimating the dimensionless curvature for circles of small radii (i.e. $|h\kappa| \to 1$) when compared against the results obtained with the finite-difference method. As we show in the next section with irregular interfaces, our neural networks are not just on par with the numerical technique for equation (3) but also superior in accuracy for sharp curvatures and cheaper with respect to the number of iterations for reinitializing the underlying level-set functions. We hypothesize that an application-dependent map of local resolutions to neural networks can be constructed and employed to estimate interface curvatures more efficiently and with higher precision than using typically expensive numerical methods.

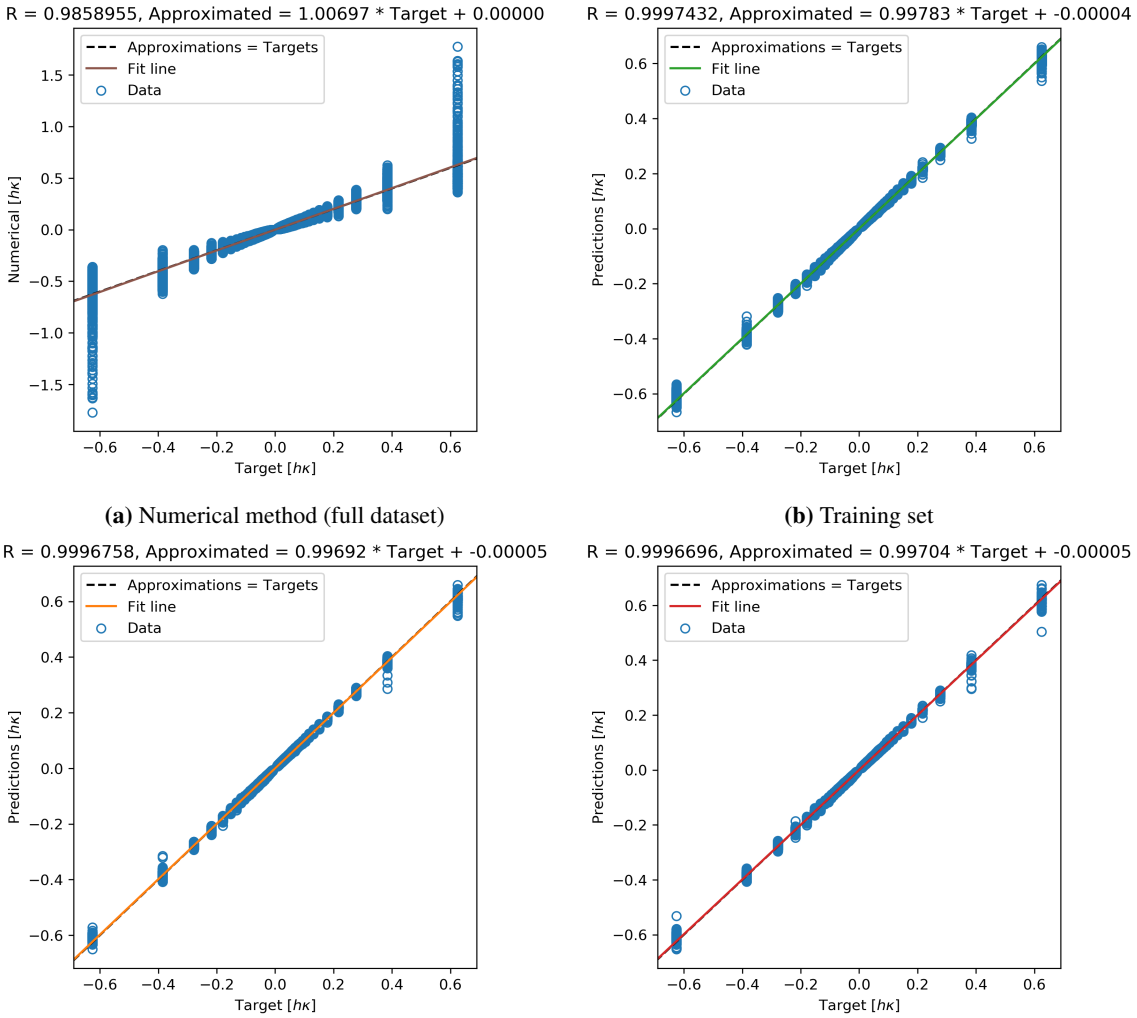
To conclude this preparatory section, we remark that our level-set curvature neural networks, their configurations, and the learning methodology described above have been the results of intense experimentation in the pursuit of high accuracy and efficiency. In regard to architectural design, for instance, we have assessed the incorporation of noise layers and layer-wise pretraining, (via stacked denoising autoencoders [68],) without noticing any major improvements in precision or generalization. Additionally, other forms of feature engineering have been attempted in order to build a "universal" neural model to approximate interfacial curvatures independently of the input's domain resolution. In this case, we scaled the learning sample pairs as $\frac{1}{h}(\boldsymbol{\phi}_p, o_p) \in \mathcal{D}$, with no further normalization. Nonetheless, the model we fitted under this scheme was also unable to generalize well when the local resolution of a testing free boundary modestly differed from the training resolution. In fact, this observation corroborates a well-known limitation of neural networks due to their interpolative nature: functions can only be approximated within a span of the sample data used for training [72, 73]. For this reason, we have opted for constructing a map of domain resolutions to neural networks. Our approach has proven to yield superior accuracy by breaking the curvature inference problem into more tractable components. Thus, we note that our framework remains valid (1) when the neural networks' training resolutions are chosen appropriately, and (2) when one extrapolates $h\kappa$ from an input local resolution that is within a narrow span of the learning resolutions. These constraints have been realized in the experiments we present in section 4.

4. Experiments and Results

We evaluate the accuracy of our level-set curvature neural networks on a two-dimensional irregular interface in both uniform and adaptive grids at different resolutions (see figures 5, 14, and 16). Our test interface corresponds to the zero isocontour of the implicit function given by:

$$\phi(x, y) = r(\theta) - a\cos(p\theta) - b, \tag{14}$$

with $r(\theta) = \sqrt{x^2 + y^2}$ where $\theta \in [0, 2\pi)$ is the angle of $(x, y) \in \Omega$ with respect to the horizontal, and $a, b, p \in \mathbb{R}$ are shape parameters that also determine the curvature sharpness of the irregular interface. For this interface, the mean curvature is analytically defined by:



(a) Numerical method (full dataset)

(b) Training set

(c) Testing set

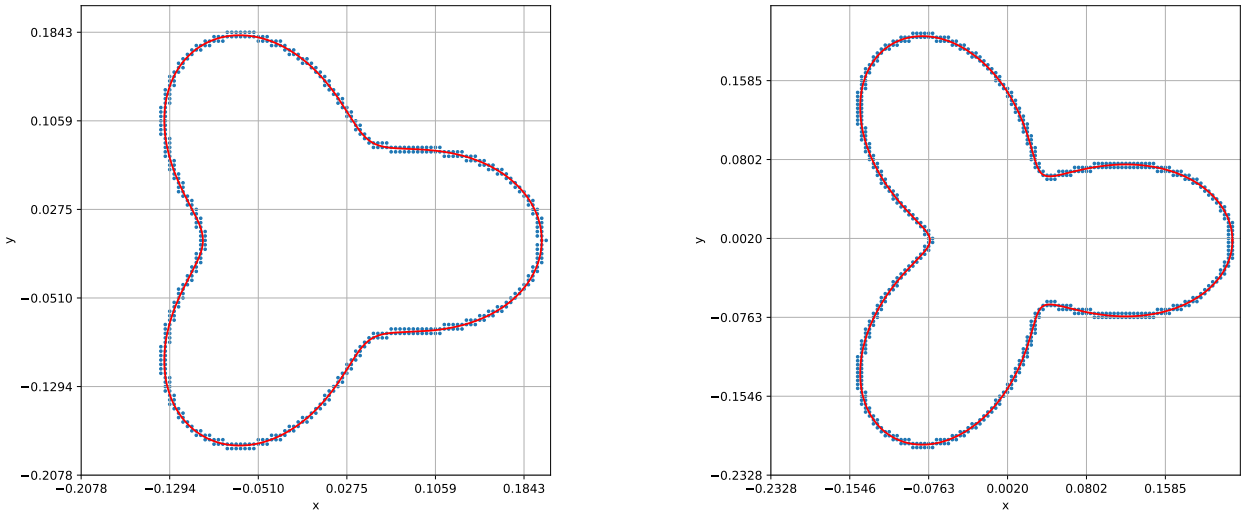
(d) Validation set

Figure 4: Correlation between expected and inferred or computed $h\kappa$ using the numerical method (a) and the neural network (b-d) after the learning stage for a unit grid resolution of 266×266 nodes.

$$\kappa(\theta) = \frac{r^2(\theta) + 2\left(r'(\theta)\right)^2 - r(\theta)r''(\theta)}{\left(r^2(\theta) + \left(r'(\theta)\right)^2\right)^{3/2}} \tag{15}$$

In the experiments that follow, we set $a = 0.05$ and $a = 0.075$ (while keeping $b = 0.15$ and $p = 3$ constant) to generate a three-petaled flower interface with two corresponding variations of curvature sharpness. We refer to the first configuration as the "smooth flower", Γ_s, and to the second one as the "acute flower", Γ_a. Our study compares the accuracy of our neural models' inferred dimensionless curvature, $h\kappa$, against the numerical approximation obtained with the finite-difference discretization of $h\nabla \cdot \nabla\phi(x_i, y_i)/|\nabla\phi(x_i, y_i)|$ (see also equation (3)), where x_i and y_i are the Cartesian coordinates of adjacent nodes to ϕ's zero isocontour.

The next series of analyses also motivate the thesis that a resolution-based dictionary of neural networks can be used to improve the accuracy of curvature calculations along interfaces. In our experiments we generate samples by using 5, 10, and 20 iterations to reinitialize the level-set function in equation (14), and these samples are extracted from grids whose resolutions are equivalent to regular unit square discretizations that yield spatial intervals of size $h \sim 1/255$, $h \sim 1/265$, and $h \sim 1/275$. We refer to these resolutions as "lower-end", "medium", and "higher-end", respectively, and they are intuitively matched to each of the trained neural networks described in section 3.2.



(a) Smooth flower

(b) Acute flower

Figure 5: Three-petaled flower interface embedded in lower-end resolution uniform grids.

4.1. Lower-End Resolution Regular Grid

We begin the accuracy analysis of our deep learning approach with the smooth interface, Γ_s, in a lower-end resolution regular grid. We define the domain $\Omega \in [-0.207843, 0.207843]^2$ and discretize it into 107 equally distributed nodes along the x and y directions (see figure 5a). This yields $h = 3.921569 \times 10^{-3}$, which is equivalent to a 256×256 grid-point unit square and allows us to employ the trained neural network for $\rho = 256$. Then, we collect 528 samples whose expected $h\kappa$ values are calculated with equation (15), having $\theta = \theta(x_i^{\perp}, y_i^{\perp})$, where $(x_i^{\perp}, y_i^{\perp})$ is the normal projection of the i^{th} sample node onto $r(\theta)$.

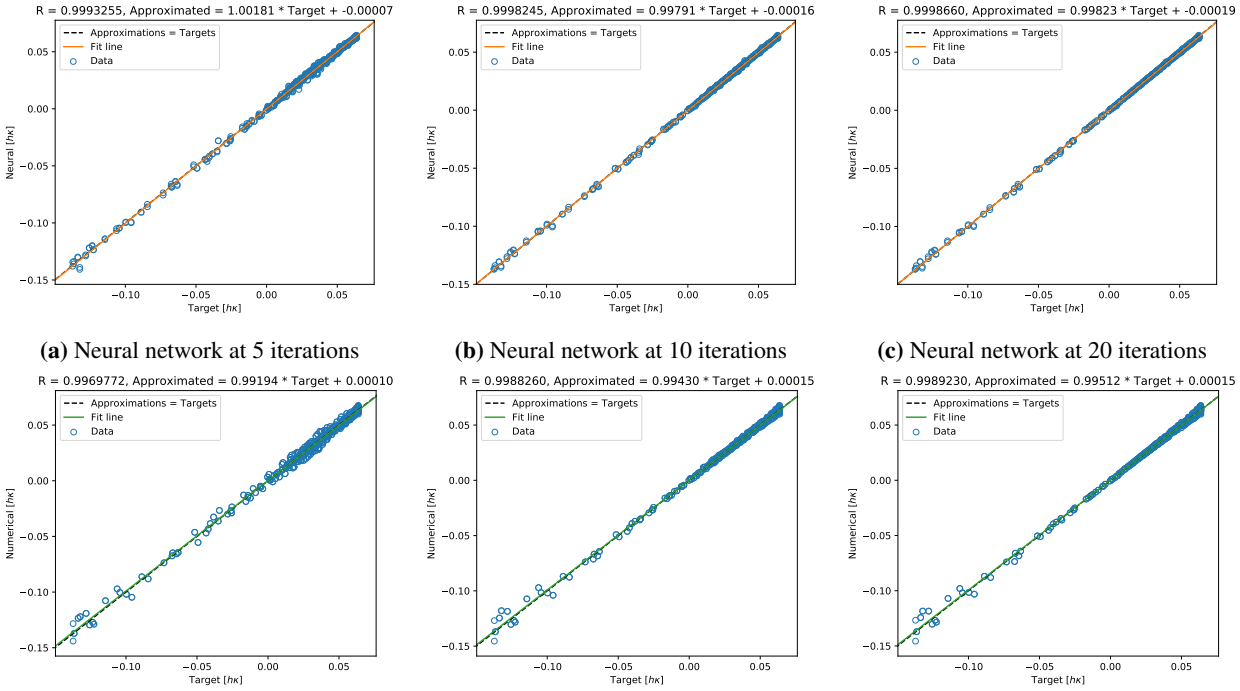
Figure 6 plots the quality of the neural and numerical $h\kappa$ approximations with respect to their target values, as one varies the iterations from 5 to 20 to reinitialize equation (14) into a signed distance function. The results visually confirm that our deep learning approach produces smaller error in the L^{∞}-norm as $|h\kappa| \to 0.15$. Another view of these findings is shown in figure 7, where the analytical $h\kappa(\theta)$ is contrasted with the neural and numerical approximations. It is evident that the finite-difference method's outputs are much noisier when compared to the inferred outputs from the neural network, especially when the number of iterations for level-set reinitialization is the smallest. In a typical free boundary value problem, this translates into a reduced need for a large number of iterations in the costly solution of the reinitialization equation.

We additionally provide an error statistical summary for Γ_s in a lower-end resolution regular grid in table 3. These results make evident our preceding observations by demonstrating that the deep learning approach improves on the mean curvature accuracy by at least 55% and on the error L^{∞}-norm by more than 17%, regardless of the number of iterations used for level-set reinitialization.

Next, we repeat our previous error analysis for the acute interface, Γ_a, in a lower-end resolution regular grid. In this case, we set the domain $\Omega \in [-0.232826, 0.232826]^2$ and discretize it into 120 equally spaced grid points along each Cartesian direction (see figure 5b). This discretization yields $h = 3.913043 \times 10^{-3}$, which corresponds to a unit-square

Iterations	Method	Mean Absolute Error	Max Absolute Error	Mean Square Error
5	Neural	1.138546×10^{-3}	8.424489×10^{-3}	2.568785×10^{-6}
	Numerical	2.599575×10^{-3}	1.020075×10^{-2}	1.142227×10^{-5}
10	Neural	5.828545×10^{-4}	4.830513×10^{-3}	7.112521×10^{-7}
	Numerical	1.290843×10^{-3}	1.427928×10^{-2}	4.454964×10^{-6}
20	Neural	4.759801×10^{-4}	4.287496×10^{-3}	5.615322×10^{-7}
	Numerical	1.089934×10^{-3}	1.381520×10^{-2}	4.083200×10^{-6}

Table 3: Error analysis for the smooth flower interface in a regular grid of 107×107 nodes.



(a) Neural network at 5 iterations **(b)** Neural network at 10 iterations **(c)** Neural network at 20 iterations

(d) Numerical method at 5 iterations **(e)** Numerical method at 10 iterations **(f)** Numerical method at 20 iterations

Figure 6: Correlation between expected and inferred or computed $h\kappa$ using the neural network and the numerical method for the smooth flower interface in a regular grid of 107×107 nodes.

resolution of roughly 256.56 nodes per unit length (i.e. within the resolution domain of our neural model for $\rho = 256$.) The amount of samples we collect is 624, and the quality of their $h\kappa$ neural and numerical approximations is illustrated in figure 8.

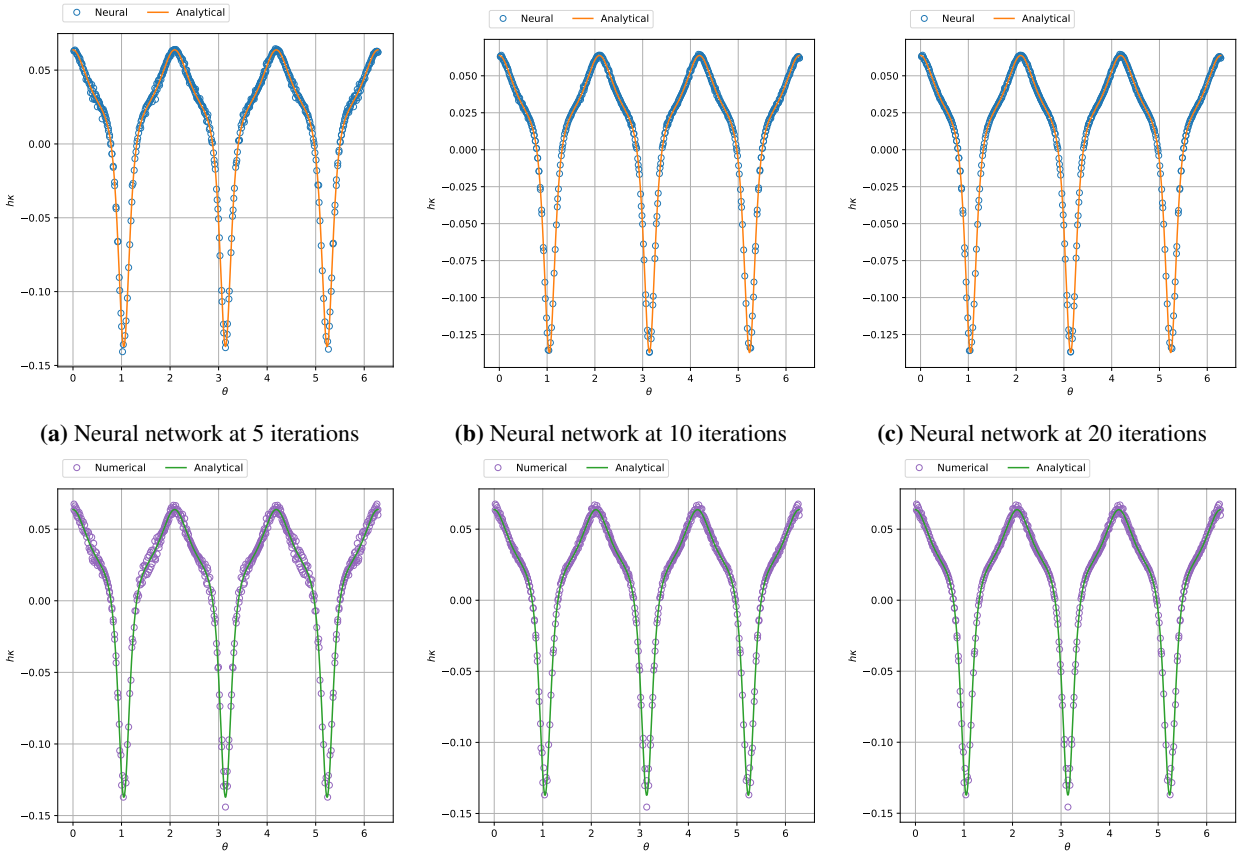
Once again, the higher correlation between the expected dimensionless curvature and its approximation occurs when one employs our neural network. Admittedly, in comparison to Γ_s, the steeper curvature at the steep petal junctions poses a harder test for both methods; however, the L^∞-norm errors in the deep learning approach are always better than the numerical technique. Indeed, we can raise the accuracy by increasing the number of iterations to reinitialize the underlying level-set; nonetheless, as we show in table 4, the maximum absolute error in the numerical method remains as large as 2 times the corresponding metric in the neural network. A similar analysis about the mean absolute error shows an accuracy improvement of at least 38% when one uses the neural model to solve for $h\kappa$ in equation (9).

4.2. Medium Resolution Regular Grid

We continue with our neural network accuracy analysis by assessing the $h\kappa$ approximations for the smooth interface, Γ_s, in a medium resolution regular grid. Following the experiment protocol outlined in the previous paragraphs, we start with a two-dimensional, computational domain given by $\Omega \in [-0.207547, 0.207547]^2$ and discretize it into 111

Iterations	Method	Mean Absolute Error	Max Absolute Error	Mean Square Error
5	Neural	2.733929×10^{-3}	4.417808×10^{-2}	3.051615×10^{-5}
	Numerical	4.552317×10^{-3}	8.752408×10^{-2}	6.947290×10^{-5}
10	Neural	1.338157×10^{-3}	4.306619×10^{-2}	1.921467×10^{-5}
	Numerical	2.277875×10^{-3}	8.782324×10^{-2}	5.406817×10^{-5}
20	Neural	1.184682×10^{-3}	4.412467×10^{-2}	1.860809×10^{-5}
	Numerical	1.914857×10^{-3}	8.794514×10^{-2}	5.509789×10^{-5}

Table 4: Error analysis for the acute flower interface in a regular grid of 120×120 nodes.



(a) Neural network at 5 iterations **(b)** Neural network at 10 iterations **(c)** Neural network at 20 iterations

(d) Numerical method at 5 iterations **(e)** Numerical method at 10 iterations **(f)** Numerical method at 20 iterations

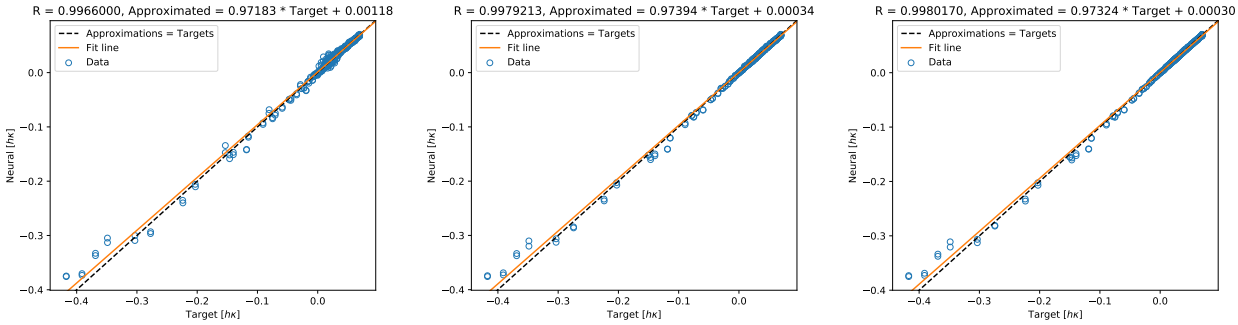
Figure 7: Dimensionless curvature as a function of θ when using the neural network and the numerical method for the smooth flower interface in a regular grid of 107×107 nodes.

uniform nodes on each Cartesian direction. This yields an effective cell's width of $h = 3.773585 \times 10^{-3}$ (i.e. it is equivalent to a 266×266-node unit square and compatible with our trained multilayer perceptron for $\rho = 266$), and the total number of samples we collect along the interface is 552. Since this interface exhibits practically the same shape as the smooth in a lower-end resolution uniform grid, for space-saving considerations, we refer the reader to figure 5a for a visual reference of the current experiment's Γ_s.

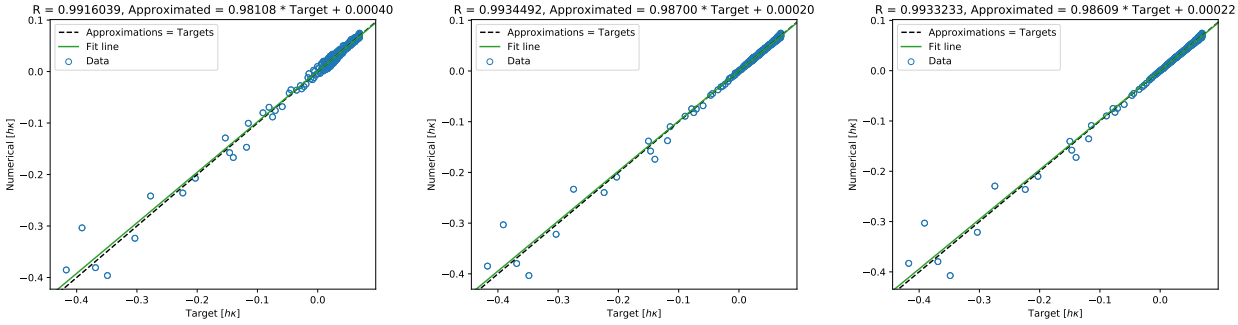
The quality of the neural and numerical $h\kappa$ approximations for Γ_s at a medium resolution uniform grid is evaluated in figure 9. Like in the previous case of Γ_s embedded in a lower-end resolution (see figure 8), the neural network's correlation factors are superior to those from the numerical method; our deep learning approach is more resilient at computing $h\kappa$ as the interface curvature tends away from 0, and it does it consistently across the number of iterations used for level-set reinitialization.

Statistically, our proposed method's L^∞-error shows an improvement of at least 42% with respect to the numerical method's maximum absolute error. Furthermore, as we show in table 5, the mean absolute error in the conventional numerical technique remains 2.1 times or more than the neural network's regardless of the number of iterations employed for level-set reinitialization.

We now proceed to examine the accuracy of our deep learning approach at approximating the acute interface $h\kappa$ values in a medium resolution regular grid. Following our previous set up process, we begin by defining a two-dimensional computational domain $\Omega \in [-0.232563, 0.232563]^2$ that is subsequently discretized into 124 uniform nodes along each Cartesian direction. Thus, we obtain a spatial step of size $h = 3.781513 \times 10^{-3}$, which makes our regular domain equivalent to a unit square with 265.44 nodes per side. These settings allow us to collect 648 samples on or sufficiently close to the irregular interface (see figure 5b) and ensure that the resolution scope for the $\rho = 266$ neural network domain is satisfied.



(a) Neural network at 5 iterations **(b)** Neural network at 10 iterations **(c)** Neural network at 20 iterations



(d) Numerical method at 5 iterations **(e)** Numerical method at 10 iterations **(f)** Numerical method at 20 iterations

Figure 8: Correlation between expected and inferred or computed $h\kappa$ using the neural network and the numerical method for the acute flower interface in a regular grid of 120×120 nodes.

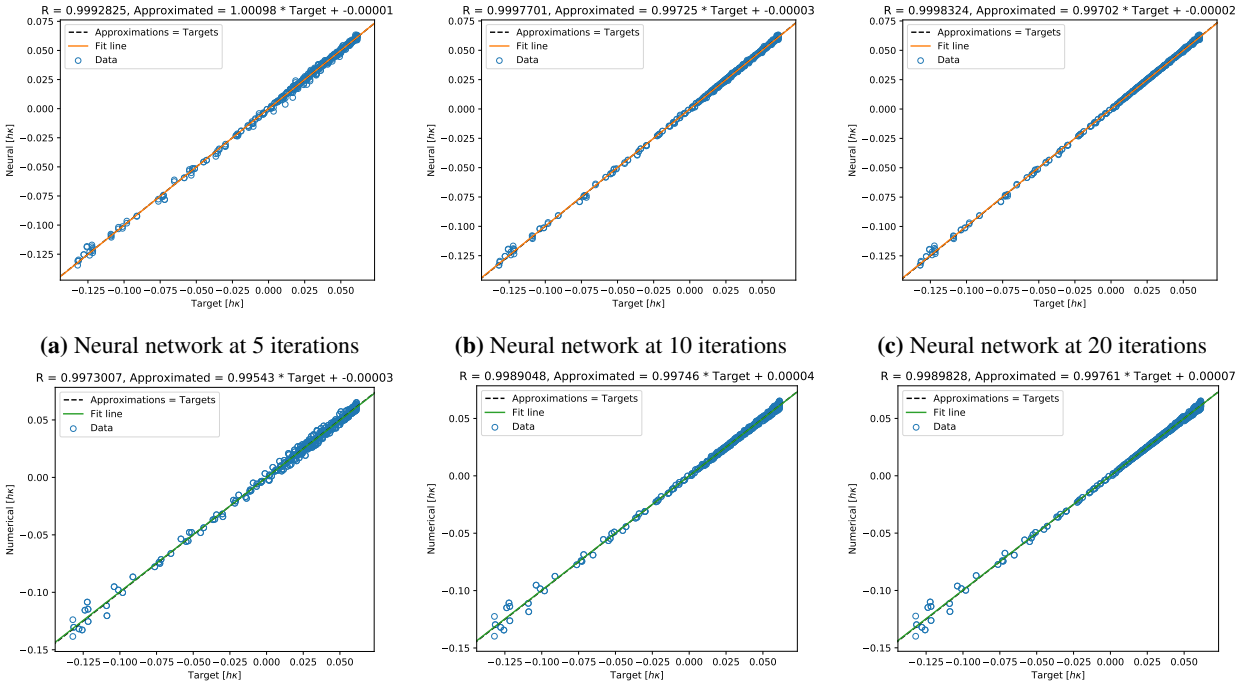
The quality of the $h\kappa$ approximations when using the neural and numerical methods for Γ_a in a medium resolution regular grid is contrasted in figure 10. It is evident that our deep learning approach is consistently better than the traditional numerical method in spite of the number of iterations used for level-set reinitialization. This is true both in terms of the correlation factor and in the calculation of steep curvatures. A more detailed error report is shown in table 6.

The statistical summary for the acute interface in a medium resolution regular grid demonstrates that our deep learning approach yields L^∞-errors that are as small as 37% the maximum absolute error incurred by the numerical method. Moreover, our neural network achieves a mean accuracy improvement of at least 43% with respect to the $h\kappa$ values obtained with the numerical scheme. Although the errors for this experiment are relatively higher than in the lower-end resolution regular grid, one can still confirm that our neural model is more precise at solving for the dimensionless curvature in equation (9) than the finite-difference method regardless of the number of costly iterations utilized for level-set reinitialization. More importantly, the reader can verify that the numerical error L^∞-norm, even with 20 iterations, never gets as small as the cheapest-reinitialization neural network's maximum absolute error.

4.3. Higher-End Resolution Regular Grid

Iterations	Method	Mean Absolute Error	Max Absolute Error	Mean Square Error
5	Neural	1.105992×10^{-3}	7.895379×10^{-3}	2.557006×10^{-6}
	Numerical	2.359816×10^{-3}	1.374033×10^{-2}	9.587425×10^{-6}
10	Neural	5.832588×10^{-4}	6.367638×10^{-3}	8.334263×10^{-7}
	Numerical	1.243609×10^{-3}	1.146487×10^{-2}	3.884756×10^{-6}
20	Neural	4.649859×10^{-4}	6.305406×10^{-3}	6.142720×10^{-7}
	Numerical	1.035165×10^{-3}	1.220033×10^{-2}	3.608487×10^{-6}

Table 5: Error analysis for the smooth flower interface in a regular grid of 111×111 nodes.



(a) Neural network at 5 iterations **(b)** Neural network at 10 iterations **(c)** Neural network at 20 iterations

(d) Numerical method at 5 iterations **(e)** Numerical method at 10 iterations **(f)** Numerical method at 20 iterations

Figure 9: Correlation between expected and inferred or computed $h\kappa$ using the neural network and the numerical method for the smooth flower interface in a regular grid of 111×111 nodes.

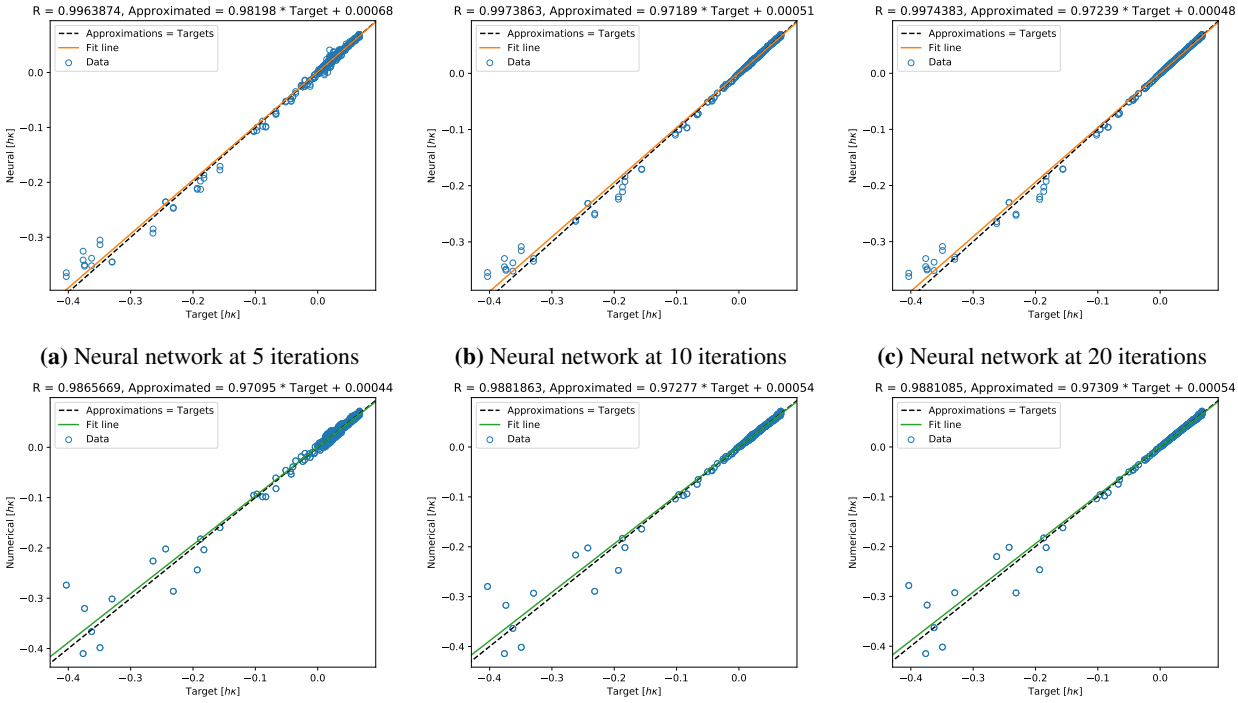
Next, we assess our level-set curvature neural network accuracy for the smooth interface in a higher-end resolution regular grid. We begin by setting the two-dimensional computational domain to $\Omega \in [-0.207339, 0.207339]^2$ and discretizing it into 114 equally-spaced points on each Cartesian direction. As a result, we collect 564 samples along Γ_s in an embedding uniform grid characterized by a spatial node interval of $h = 3.669724 \times 10^{-3}$. This cell width is equivalent to a unit-square resolution of 273.5 nodes per side length and is compatible with the resolution domain of our neural network adapted to a 276×276 local resolution. We invite the reader to revisit figure 5a to get a sense of the smooth flower interface shape that we are currently studying.

Figure 11 contrasts the quality of the smooth interface $h\kappa$ estimations obtained with our neural network and the numerical method in a higher-end resolution uniform grid. As in previous experiments at lower resolutions, our deep learning approach is proven again superior to the traditional finite-difference technique, especially at interface sections where curvature grows steeper, and as one varies the number of iterations from 5 to 20 to reinitialize the level-set function. We back the correlation results from figure 11 with the error summary provided in table 7.

The error statistics for the smooth interface in a higher-end regular grid show that the numerical method's error L^∞-norm is never smaller than the neural network's in spite of the number of iterations used for level-set reinitialization. In fact, one can observe that (1) the neural model's maximum absolute error always remains at 46% or less than the corresponding metric in the numerical method; and (2) the mean absolute error drops by 37% or more when one

Iterations	Method	Mean Absolute Error	Max Absolute Error	Mean Square Error
5	Neural	2.823062×10^{-3}	5.093963×10^{-2}	3.424025×10^{-5}
	Numerical	5.066497×10^{-3}	1.293456×10^{-1}	1.241324×10^{-4}
10	Neural	1.556722×10^{-3}	4.877330×10^{-2}	2.667084×10^{-5}
	Numerical	2.873739×10^{-3}	1.236361×10^{-1}	1.090646×10^{-4}
20	Neural	1.365779×10^{-3}	4.704585×10^{-2}	2.610181×10^{-5}
	Numerical	2.427686×10^{-3}	1.251736×10^{-1}	1.097577×10^{-4}

Table 6: Error analysis for the acute flower interface in a regular grid of 124×124 nodes.



(a) Neural network at 5 iterations **(b)** Neural network at 10 iterations **(c)** Neural network at 20 iterations

(d) Numerical method at 5 iterations **(e)** Numerical method at 10 iterations **(f)** Numerical method at 20 iterations

Figure 10: Correlation between expected and inferred or computed $h\kappa$ using the neural network and the numerical method for the acute flower interface in a regular grid of 124×124 nodes.

compares the precision attained with our deep learning approach versus the traditional finite-difference discretization.

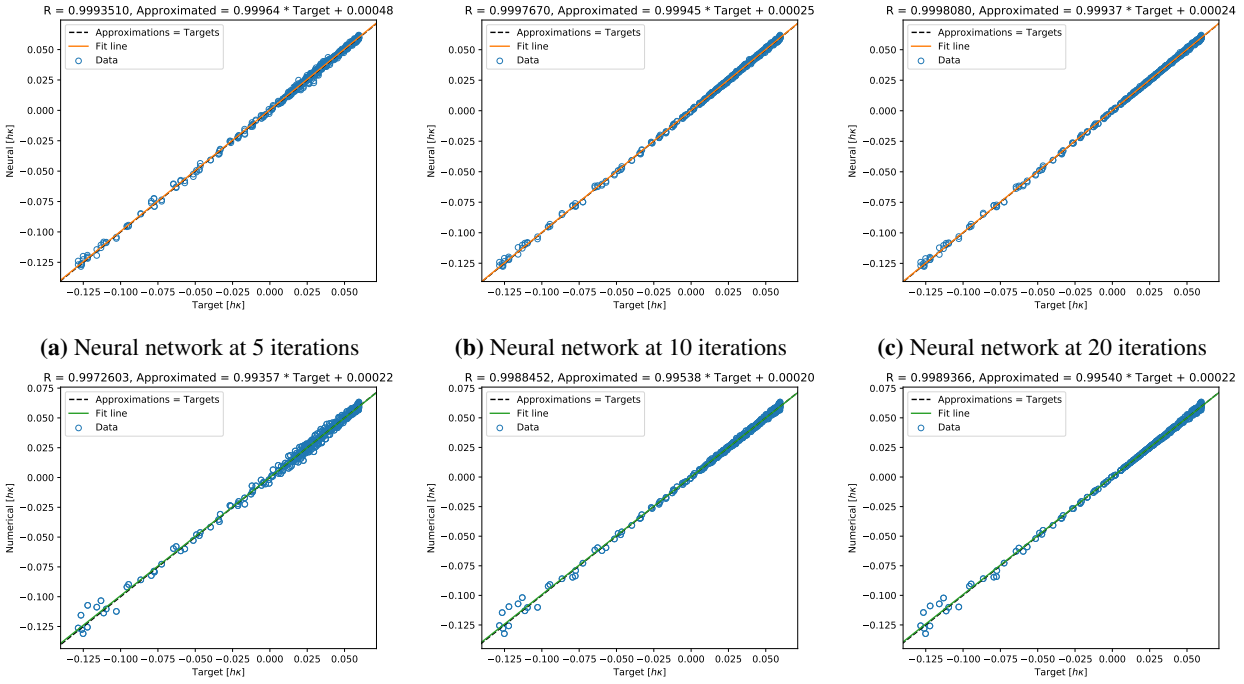
The last curvature accuracy assessment for the irregular flower interface in a uniform grid corresponds to Γ_a (see figure 5b) embedded in a higher-end resolution. Adhering to the preceding experiments methodology, we first set the computational domain to $\Omega \in [-0.232258, 0.232258]^2$ and then discretize it into a 129×129 regular mesh of nodes. This yields a constant cell width of $h = 3.629032 \times 10^{-3}$, which is equivalent to a local unit square with 276.56 grid points per side length. The configuration just described enables us to apply our trained neural network for $\rho = 276$ and to gather 672 samples along the acute interface. The correlation plots between the target $h\kappa$ values and the neural and numerical approximations are shown in figure 12.

A visual study of the plots in figure 12 reveals that our deep learning approach is invariably better at estimating curvature than the numerical method. Indeed, the correlation factor is not only higher in the case of the neural network across the number of iterations for level-set reinitialization, but also the error L^∞-norm remains lower when compared to the finite-difference discretization as $|h\kappa| \to 0.4$. We complement these findings with the short statistical analysis contained in table 8.

The error summary for the current acute interface's study indicates that our deep learning approach $h\kappa$ accuracy is always superior to the traditional finite-difference method at estimating equation (3). While the mean absolute error

Iterations	Method	Mean Absolute Error	Max Absolute Error	Mean Square Error
5	Neural	1.164241×10^{-3}	6.791887×10^{-3}	2.423722×10^{-6}
	Numerical	2.301648×10^{-3}	1.481315×10^{-2}	9.299462×10^{-6}
10	Neural	6.946607×10^{-4}	4.271347×10^{-3}	8.500361×10^{-7}
	Numerical	1.200980×10^{-3}	1.252291×10^{-2}	3.936283×10^{-6}
20	Neural	6.165151×10^{-4}	4.365713×10^{-3}	7.043411×10^{-7}
	Numerical	9.802761×10^{-4}	1.311203×10^{-2}	3.633351×10^{-6}

Table 7: Error analysis for the smooth flower interface in a regular grid of 114×114 nodes.



(a) Neural network at 5 iterations **(b)** Neural network at 10 iterations **(c)** Neural network at 20 iterations

(d) Numerical method at 5 iterations **(e)** Numerical method at 10 iterations **(f)** Numerical method at 20 iterations

Figure 11: Correlation between expected and inferred or computed $h\kappa$ using the neural network and the numerical method for the smooth flower interface in a regular grid of 114×114 nodes.

decreases by at least 39%, the error L^∞-norm improves by more than 57% when one makes use of the trained neural network to solve equation (9). We note that these results are again consistent with our findings above, at different resolutions, regardless of the cost we might pay for reinitializing the underlying level-set function.

4.4. Embedding the Irregular Interface in an Adaptive Grid

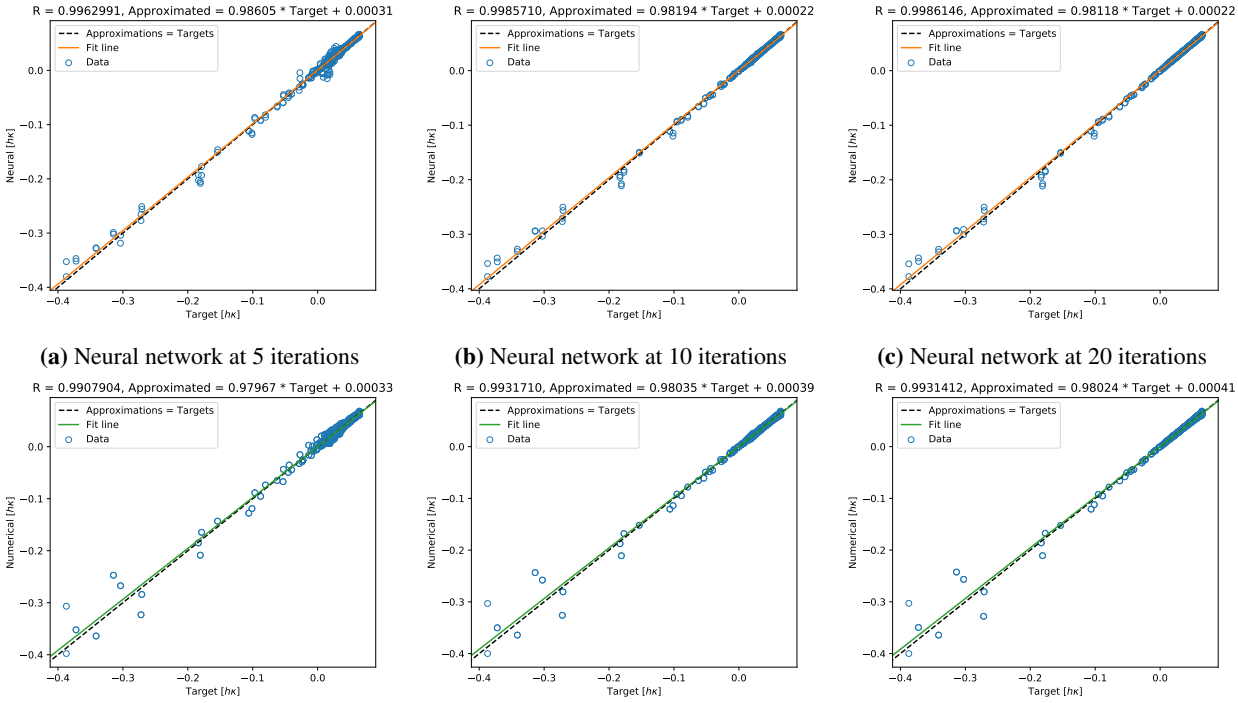
We close the results section with the analysis of neural network $h\kappa$ approximations for both Γ_s and Γ_a using an adaptive grid discretization. The adaptive grid is a non-uniform, quadtree mesh of vertices that covers the entire computational domain, Ω. A quadtree is a rooted data structure where each tree cell, C either has 4 children, i.e. quadrants, or is a leaf [74]. The tree cells are organized into $L \geq 0$ levels, where the cell vertices (i.e. Cartesian nodes) store positional information as well as the value of $\phi(x, y)$ [49]. An example quadtree is shown in figure 13.

The process for building a quadtree consists of a recursive subdivision of cells that initiates at the root (i.e. $l_0 \equiv \Omega$) and stops when the tree reaches an application-dependent maximum level of refinement. At every subdivision step, we determine whether a cell C needs to be split based on its distance to Γ [49], as it is succinctly expressed through the following criterion [39]:

$$\min_{v \in \mathcal{V}(C)} |\phi(v)| \leq \mathrm{Lip}(\phi(v)) \times \mathrm{diag}_\ell(C), \tag{16}$$

Iterations	Method	Mean Absolute Error	Max Absolute Error	Mean Square Error
5	Neural	2.815830×10^{-3}	3.444712×10^{-2}	2.717994×10^{-5}
	Numerical	4.598858×10^{-3}	8.030253×10^{-2}	6.706189×10^{-5}
10	Neural	1.196355×10^{-3}	3.333340×10^{-2}	1.127559×10^{-5}
	Numerical	2.178264×10^{-3}	8.390227×10^{-2}	4.979660×10^{-5}
20	Neural	1.032981×10^{-3}	3.302984×10^{-2}	1.105503×10^{-5}
	Numerical	1.716480×10^{-3}	8.419102×10^{-2}	5.001022×10^{-5}

Table 8: Error analysis for the acute flower interface in a regular grid of 129×129 nodes.



(a) Neural network at 5 iterations **(b)** Neural network at 10 iterations **(c)** Neural network at 20 iterations

(d) Numerical method at 5 iterations **(e)** Numerical method at 10 iterations **(f)** Numerical method at 20 iterations

Figure 12: Correlation between expected and inferred or computed $h\kappa$ using the neural network and the numerical method for the acute flower interface in a regular grid of 129×129 nodes.

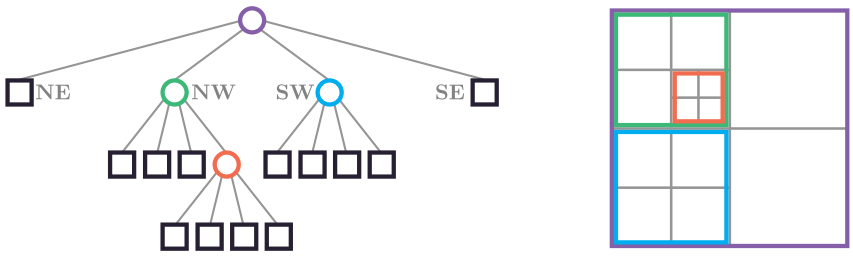
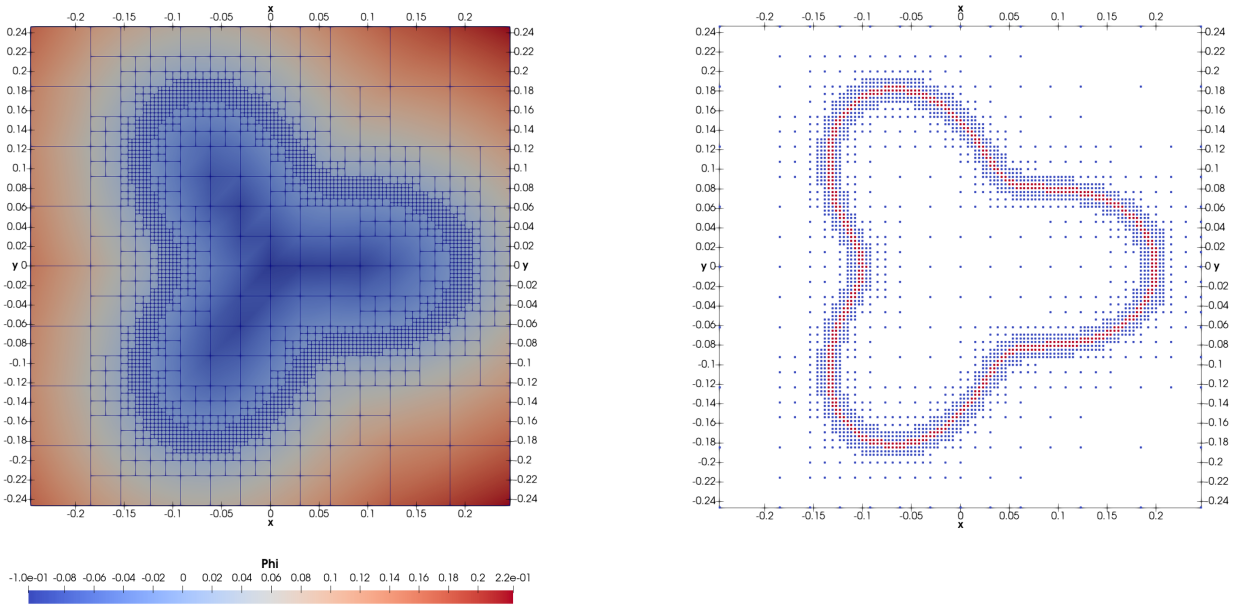


Figure 13: A quadtree and its cell subdivisions (adapted from [74]).

where $\mathcal{V}$ is the set of C's vertices, $\text{diag}_\ell(C)$ defines the diagonal length of C, and $\text{Lip}(\phi(v))$ is the Lipschitz constant of ϕ (here set to 1.2 for all cells).

First, we analyze the quality of the neural $h\kappa$ approximation for the smooth interface, Γ_s, in a non-uniform grid. To this end, we define the two-dimensional computational domain $\Omega \in [-0.246154, 0.246154]^2$, discretize it using one quadtree with $l_{max} = 7$ (see figure 14a), and obtain a minimum $h = 3.846154 \times 10^{-3}$. This is equivalent to a local resolution of 261 uniform grid points per unit length and is compatible with the neural model we trained for a uniform mesh of 266×266 nodes. The former settings allow us to collect 536 samples along Γ_s, which we show in red in figure 14b.

Figure 15 plots the correlation between the expected $h\kappa$ values and their approximations obtained with our deep learning approach and the numerical method when Γ_s is embedded in a non-uniform grid. In analogy to the former regular grid test cases, we observe that the neural network accuracy is superior to the numerical method's regardless of the number of iterations used for level-set reinitialization. One can cross-check the visual results in figure 15 with the error statistical summary given in table 9. Once more, the numerical method's mean absolute error is at least 1.4 times the neural network's, while the L^∞-error ratio from the numerical to the neural method remains at 2.1 times or more as the number of iterations increases. In particular, the neural network maximum absolute error reduces up to 71% with



(a) Quadtree discretization

(b) Sampled nodes along interface

Figure 14: Smooth flower interface embedded in a non-uniform grid.

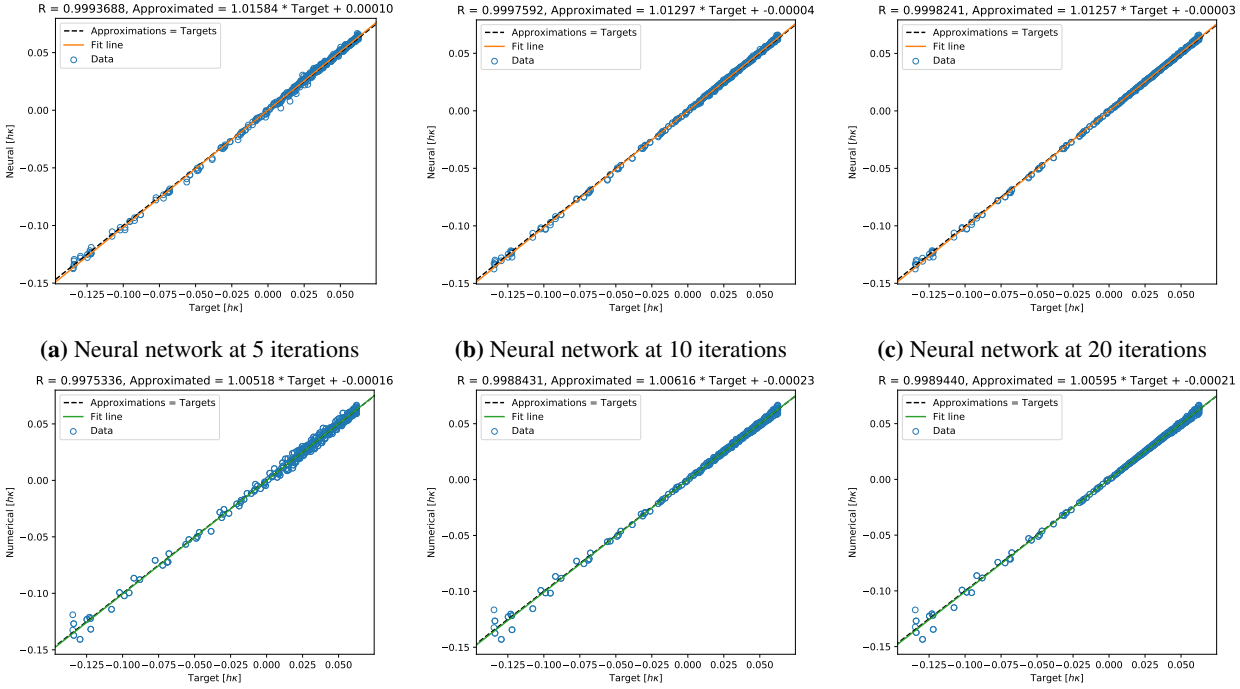
respect to the numerical method's when the number of iterations is the largest. In fact, the numerical method cannot attain the neural network's 5-iteration accuracy even if one quadruplicates the number of iterations to 20.

Lastly, we prove that the neural network $h\kappa$ approximation for the acute interface is also superior to the numerical method's when we embed Γ_a in a non-uniform grid. We now set the computational domain to $\Omega \in [-0.244068, 0.244068]^2$ and discretize it using one quadtree with $l_{max} = 7$ (see figure 16a). This yields a minimum cell's side of length $h = 3.813559 \times 10^{-3}$, which is equivalent to a unit square with 263.22 regularly distributed nodes on each Cartesian direction. These settings enable us to both collect 644 samples along the Γ_a interface and use the neural network trained for $\rho = 266$ to carry out the accuracy assessment. We show the irregular interface samples in figure 16b together with the rest of the adaptive grid vertices.

Finally, figure 17 compares the quality of the approximated $h\kappa$ in the neural and the numerical methods when one uses an adaptive grid to capture Γ_a. In spite of the challenging, steep curvatures at the kink-like petal junctions, our deep learning approach consistently outperforms the well-established numerical technique across the evaluated iterations for level-set reinitialization (especially as $|h\kappa| \to 0.4$.) These conclusions are complemented with the error statistics reported in table 10. The statistical analysis demonstrates that the neural network's error L^∞-norm never grows larger than 61% of the numerical method's maximum absolute error, no matter how many iterations are used. And, more importantly, one can realize that the maximum absolute error in the traditional finite-difference discretization is never as small as the cheapest-iteration neural network's corresponding metric, even if we raise the number of iterations from 5 to 20.

Iterations	Method	Mean Absolute Error	Max Absolute Error	Mean Square Error
5	Neural	1.328745×10^{-3}	7.396850×10^{-3}	2.975383×10^{-6}
	Numerical	2.314025×10^{-3}	1.560937×10^{-2}	9.011782×10^{-6}
10	Neural	8.280843×10^{-4}	5.092966×10^{-3}	1.243497×10^{-6}
	Numerical	1.249718×10^{-3}	1.806956×10^{-2}	4.279880×10^{-6}
20	Neural	7.128068×10^{-4}	5.050182×10^{-3}	9.832609×10^{-7}
	Numerical	1.033626×10^{-3}	1.753976×10^{-2}	3.903604×10^{-6}

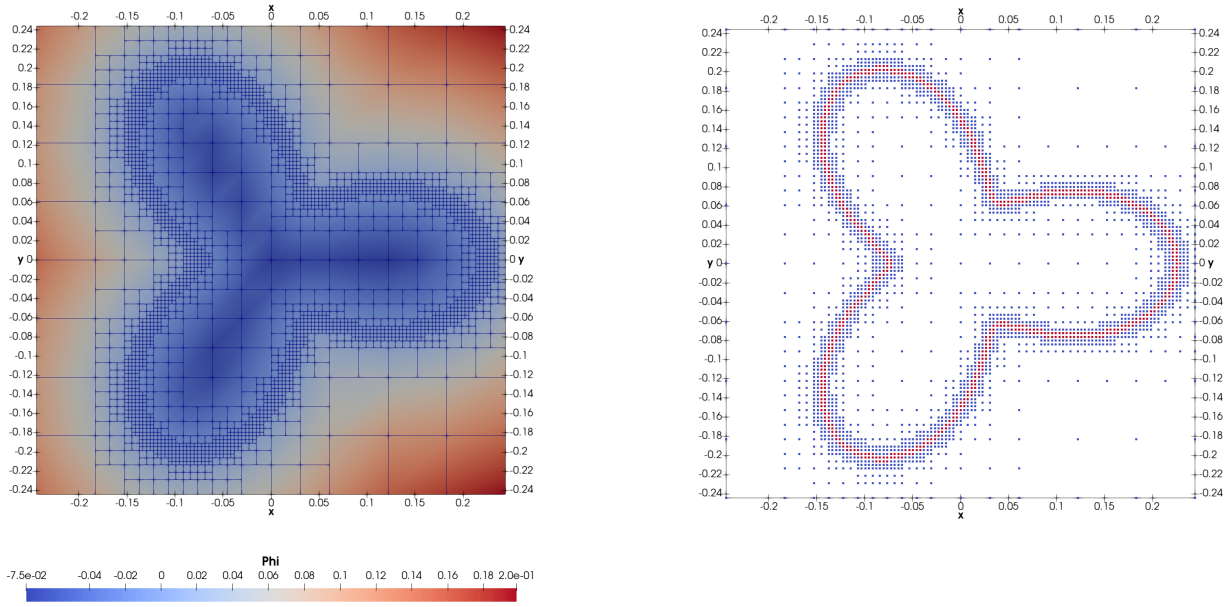
Table 9: Error analysis for the smooth flower interface in an adaptive grid.



(a) Neural network at 5 iterations **(b)** Neural network at 10 iterations **(c)** Neural network at 20 iterations

(d) Numerical method at 5 iterations **(e)** Numerical method at 10 iterations **(f)** Numerical method at 20 iterations

Figure 15: Correlation between expected and inferred or computed $h\kappa$ using the neural network and the numerical method for the smooth flower interface in an non-uniform grid.

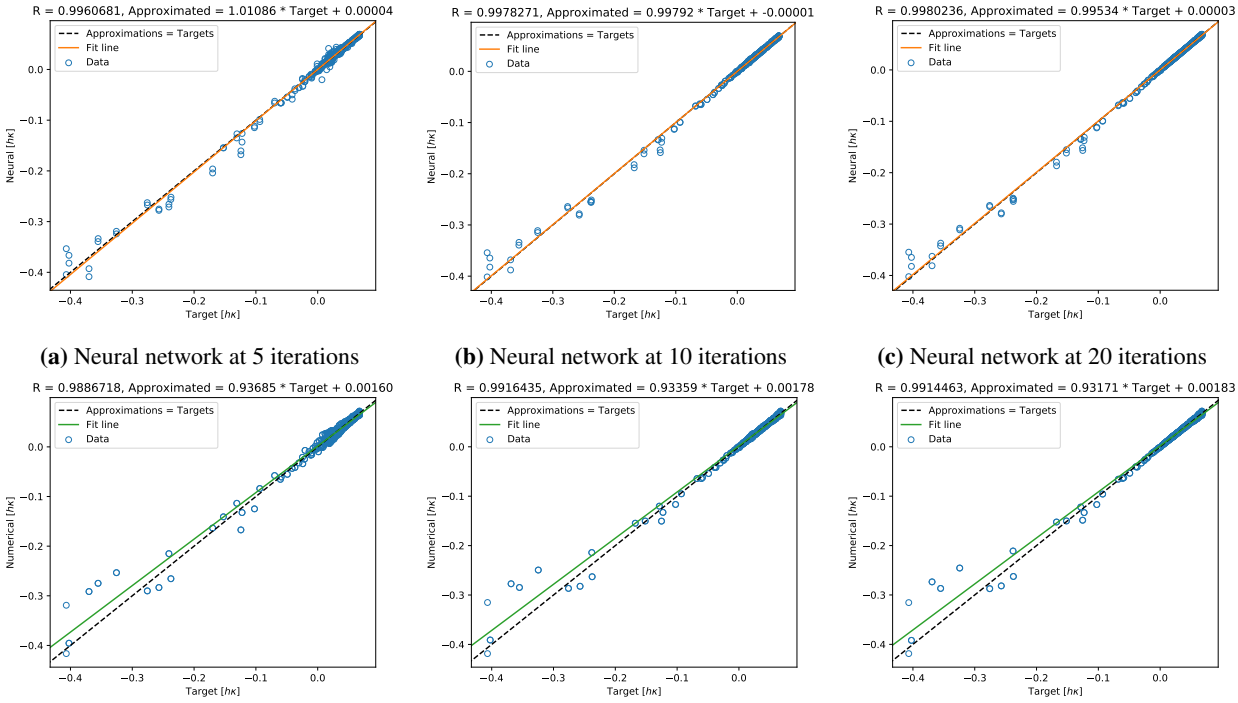


(a) Quadtree discretization **(b)** Sampled nodes along interface

Figure 16: Acute flower interface embedded in a non-uniform grid.

5. Conclusions

We have presented a novel deep learning approach for approximating and enhancing the mean curvature computation along two-dimensional, zero-isocontours of discretized implicit surfaces in the level-set method. Our experiments



(a) Neural network at 5 iterations **(b)** Neural network at 10 iterations **(c)** Neural network at 20 iterations

(d) Numerical method at 5 iterations **(e)** Numerical method at 10 iterations **(f)** Numerical method at 20 iterations

Figure 17: Correlation between expected and inferred or computed $h\kappa$ using the neural network and the numerical method for the acute flower interface in an non-uniform grid.

on sharp and smooth irregular interfaces, embedded in both uniform and adaptive grids, indicate that our feedforward neural networks consistently attain higher $h\kappa$ accuracy than well-established numerical methods.

We recognize that the neural networks we built are limited in scope. Indeed, a deployed, pre-configured model cannot transfer to a domain where the needed resolution extends too far from the spatial discretizations considered during the learning stage [72]. However, given some a priori knowledge about the domain of a level-set equation, we claim that a dictionary of multilayer perceptrons can be constructed for more than one resolution. Then, this map can be used to solve for the dimensionless curvature in equation (9) for virtually any interface. In effect, a dictionary of neural networks exhibits two advantages over a universal neural model: (1) it is modular and faster to train, and (2) it is more efficient to evaluate in both time and space. Nevertheless, as part of our future work, we plan to carry out a rigorous study to establish the resolution span over which a dictionary's neural network can be applied with a high degree of confidence. For instance, even though our $\rho = 266$ neural model (see section 3.2) accurately infers $h\kappa$ for a local resolution equivalent to $\rho = 261$ (see section 4.4), it still remains to conduct an exhaustive resolution-span analysis. This study would evaluate a broader variety of interfaces and set some general resolution bounds for any level-set curvature neural network.

The curvature deep learning framework we outlined in the preceding sections has a couple of important advantages

Iterations	Method	Mean Absolute Error	Max Absolute Error	Mean Square Error
5	Neural	2.938585×10^{-3}	5.339007×10^{-2}	3.624541×10^{-5}
	Numerical	5.098362×10^{-3}	8.773449×10^{-2}	1.071264×10^{-4}
10	Neural	1.473968×10^{-3}	5.257256×10^{-2}	1.911673×10^{-5}
	Numerical	2.572642×10^{-3}	9.163997×10^{-2}	8.466861×10^{-5}
20	Neural	1.276491×10^{-3}	5.206994×10^{-2}	1.737252×10^{-5}
	Numerical	2.185198×10^{-3}	9.558052×10^{-2}	8.709143×10^{-5}

Table 10: Error analysis for the acute flower interface in an adaptive grid.

over its numerical counterpart. First, it is less sensitive to sharp curvatures. Also, it requires fewer iterations for level-set reinitialization to achieve higher accuracy than the conventional numerical methods. While applications typically demand twenty iterations at every advection step, we have provided evidence that a neural model is able to estimate interface curvatures with the same or better precision by inputting samples generated with just five iterations. On this regard, faster, yet first-order accurate, reinitialization schemes, such as the Fast Marching Method [60–62] and the Fast Sweeping Method [63–66], open up the possibility for future endeavors aiming at further efficiency improvements. Additionally, the techniques we use to generate the synthetic learning pairs suggest that more diverse curvature data can be extracted from other shapes, like squares, triangles, and merging circles. In turn, these samples may be employed for training neural networks to solve for interfacial curvatures at kinks and under-resolved regions, where traditional numerical methods are well-known to fail.

Finally, although we have not carried out an extensive execution-time assessment for our deep learning approach, we report our findings for the current implementation of the level-set curvature neural networks presented in section 3.2. Given a batch of 14,400 samples and a one-processor system, the numerical method requires 83ms, in C++, to approximate $h\kappa$ when using 20 iterations to reinitialize the level-set function. On the contrary, our neural networks take an average execution time of 351ms to produce the corresponding outputs with TensorFlow in Python. At first sight, the traditional finite-difference procedure might seem more efficient by a factor of 4, but this factor is dwarfed by the gain in the reduction of reinitialization procedure. In addition, there are two promising venues that we are yet to explore in order to boost the neural networks' efficiency: (1) neural parameters migration and full inference evaluation in C++ and PETSc [75], and (2) model pruning [76] to favor sparsity and reduce the number of operations. We foresee that once these tasks are complete, not only will our neural networks outweigh the numerical method's correctness but also will a dictionary of multilayer perceptrons be integrated as a solver for $h\kappa$ in a complex and highly accurate numerical framework. Our trained level-set curvature neural networks have been made publicly available at `https://github.com/UCSB-CASL/LSCurvatureDL`, where we have also included the training statistics for the correct normalization of the networks' inputs.

Acknowledgments

This research acknowledges support from ONR N00014-17-1-2676. Use was made of computational facilities purchased with funds from the National Science Foundation (CNS-1725797) and administered by the Center for Scientific Computing (CSC). The CSC is supported by the California NanoSystems Institute and the Materials Research Science and Engineering Center (MRSEC; NSF DMR 1720256) at UC Santa Barbara.

References

References

[1] G. Carinci, A. de Masi, C. Giardinà, and E. Presutti. Free Boundary Problems in PDEs and Particle Systems. In *SpringerBriefs in Mathematical Physics*. Springer International Publishing, 2016.

[2] G. Tryggvason, B. Bunner, A. Esmaeeli, D. Juric, N. Al-Rawahi, W. Tauber, J. Han, S. Nas, and Y.-J. Jan. A Front-Tracking Method for the Computations of Multiphase Flow. *Journal of Computational Physics*, 169:708–759, 2001.

[3] S. Osher and R. Fedkiw. *Level Set Methods and Dynamic Implicit Surfaces*. Springer-Verlag, 2002. New York, NY.

[4] S. O. Unverdi and G. Tryggvason. A Front-Tracking Method for Viscous, Incompressible, Multi-fluid Flows. *Journal of Computational Physics*, 100(1):25–37, 1992.

[5] D. Juric and G. Tryggvason. A Front Tracking Method for Dendritic Solidification. *Journal of Computational Physics*, 123:127–148, 1996.

[6] D. Juric and G. Tryggvason. Computations of Boiling Flows. *International Journal of Multiphase Flow*, 24(3):387–410, 1998.

[7] J. Qian, G. Tryggvason, and C.K. Law. A Front Tracking Method for the Motion of Premixed Flames. *Journal of Computational Physics*, 144(1):52–69, July 1998.

[8] R. Brawun and M. Murray. Adaptive Phase-Field Computations of Dendritic Crystal Growth. *Journal Crystal Growth*, 174:41, 1997.

[9] E. O. Ávila-Dávila, V. M. López-Hirata, and M. L. Saucedo-Muñoz. Application of Phase-Field Method to the Analysis of Phase Decomposition of Alloys. In *Modeling and Simulation in Engineering Sciences*. IntechOpen, 2016.

[10] R. S. Qin and H. K. Bhadeshia. Phase Field Method. *Materials Science and Technology*, 26(7):803–811, 2010.

[11] K. R. Elder, M. Grant, N. Provatas, and J. M. Kosterlitz. Sharp Interface Limits of Phase-Field Models. *Physical Review E*, 64(2):21604, 2001.

[12] J.-H. Jeong, N. Goldenfeld, and J. A. Dantzig. Phase Field Model for Three-Dimensional Dendritic Growth with Fluid Flow. *Physical Review E*, 64:41602, 2001.

[13] A. Karma and W.-J. Rappel. Phase-Field Modeling Method for Computationally Efficient Modeling of Solidification with Arbitrary Interface Kinetics. *Physical Review E*, 53(4):R3017–R3020, 1996.

[14] A. Karma and W.-J. Rappel. Quantitative Phase-Field Modeling of Dendritic Growth in Two and Three Dimensions. *Physical Review E*, 57(4):4323–4349, 1998.

[15] A. Karma. Phase-Field Formulation for Quantitative Modeling of Alloy Solidification. *Physical Review Letters*, 87(11):115701, 2001.

[16] B. Nestler, D. Danilov, and P. Galenko. Crystal Growth of Pure Substances: Phase-Field Simulations in Comparison with Analytical and Experimental Results. *Journal of Computational Physics*, 207:221–239, 2005.

[17] N. Provatas, N. Goldenfeld, and J. Dantzig. Efficient Computation of Dendritic Microstructures Using Adaptive Mesh Refinement. *Physical Review Letters*, 80(15):3308, 1998.

[18] N. Provatas, N. Goldenfeld, and J. Dantzig. Adaptive Mesh Refinement Computation of Solidification Microstructure using Dynamic Data Structures. *Journal of Computational Physics*, 148:265–290, 1999.

[19] R. B. DeBar. Fundamentals of the KRAKEN Code. Technical Report UCID- 17366, Lawrence Livermore National Laboratory, 1974.

[20] W. F. Noh and P. Woodward. SLIC (Simple Line Interface Calculation). In *5th International Conference on Numerical Methods in Fluid Dynamics*, pages 330–340, 1976.

[21] C. W. Hirt and B. D. Nichols. Volume of Fluid (VOF) Method for the Dynamics of Free Boundaries. *Journal of Computational Physics*, 39:201–225, 1981.

[22] D. L. Youngs. An Interface Tracking Method for a 3D Eulerian Hydrodynamics Code. Technical Report AWRE/44/92/35, Atomic Weapons Research Establishment, 1984.

[23] D. J. Benson. Computational Methods in Lagrangian and Eulerian Hydrocodes. *Computer Methods in Applied Mechanics and Engineering*, 99:235–394, 1992.

[24] M. Sussman and E. G. Puckett. A Coupled Level Set and Volume-of-Fluid Method for Computing 3D and Axisymmetric Incompressible Two-Phase Flows. *Journal of Computational Physics*, 162(2):301–337, 2000.

[25] Y. Renardy and M. Renardy. PROST: A Parabolic Reconstruction of Surface Tension for the Volume-of-Fluid Method. *Journal of Computational Physics*, 183(2):400–421, 2002.

[26] D. J. Benson. Volume of Fluid Interface Reconstruction Methods for Multi-Material Problems. *Applied Mechanics Reviews*, 55(2):151–165, 2002.

[27] E. Aulisa, S. Manservisi, R. Scardovelli, and S. Zaleski. A Geometrical Area-Preserving Volume-of-Fluid Advection Method. *Journal of Computational Physics*, 192(1):355–364, 2003.

[28] V. Dyadechko and M. Shashkov. Moment-of-Fluid Interface Reconstruction. Technical Report LA-UR-05-7571, Los Alamos National Laboratory, 2006.

[29] X. Yang, A. J. James, J. Lowengrub, X. Zheng, and V. Cristini. An Adaptive Coupled Level-Set/Volume-of-Fluid Interface Capturing Method for Unstructured Triangular Grids. *Journal of Computational Physics*, 217(2):364–394, 2006.

[30] Z. Wang, J. Yang, and F. Stern. A New Volume-of-Fluid Method with a Constructed Distance Function on General Structured Grids. *Journal of Computational Physics*, 231(9):3703–3722, 2012.

[31] Y. Qi, J. Lu, R. Scardovelli, S. Zaleski, and G. Tryggvason. Computing Curvature for Volume of Fluid Methods Using Machine Learning. *Journal of Computational Physics*, 377:155–161, 2019.

[32] S. Osher and J. A. Sethian. Fronts Propagating with Curvature Dependent Speed: Algorithms Based on Hamilton-Jacobi Formulations. *Journal of Computational Physics*, 79(1):12–49, 1988.

[33] M. Moog, R. Keck, and A. Zemitis. Some Numerical Aspects of the Level Set Method. *Mathematical Modeling and Analysis*, 3(1):140–151, 1998.

[34] M. Sussman, E. Fatemi, P. Smereka, and S. Osher. An Improved Level Set Method for Incompressible Two-Phase Flows. *Computers and Fluids*, 27(5-6):663–680, 1998.

[35] J. A. Sethian. *Level Set Methods and Fast Marching Methods*, volume 3 of *Cambridge Monographs on Applied and Computational Mathematics*. Cambridge University Press, Cambridge, second edition, 1999. Evolving interfaces in computational geometry, fluid mechanics, computer vision, and materials science.

[36] S. Osher and R. P. Fedkiw. Level Set Methods: An Overview and Some Recent Results. *Journal of Computational Physics*, 169(2):463–502, 2001.

[37] D. Enright, R. Fedkiw, J. Ferziger, and I. Mitchell. A Hybrid Particle Level Set Method for Improved Interface Capturing. *Journal of Computational Physics*, 183(1):83–116, 2002.

[38] H. Chen, C. Min, and F. Gibou. A Supra-Convergent Finite Difference Scheme for the Poisson and Heat Equations on Irregular Domains and Non-Graded Adaptive Cartesian Grids. *Journal of Scientific Computing*, 31(1-2):19–60, March 2007.

[39] C. Min and F. Gibou. A Second Order Accurate Level Set Method on Non-Graded Adaptive Cartesian Grids. *Journal of Computational Physics*, 225(1):300–321, July 2007.

[40] C. Min and F. Gibou. Geometric Integration over Irregular Domains with Application to Level-Set Methods. *Journal of Computational Physics*, 226(2):1432–1443, October 2007.

[41] C. Min and F. Gibou. Robust Second-Order Accurate Discretizations of the Multi-Dimensional Heaviside and Dirac Delta Functions. *Journal of Computational Physics*, 227(22):9686–9695, November 2008.

[42] H. Chen, C. Min, and F. Gibou. A Numerical Scheme for the Stefan Problem on Adaptive Cartesian Grids with Supralinear Convergence Rate. *Journal of Computational Physics*, 228(16):5803–5818, 2009.

[43] M. Mirzadeh, A. Guittet, C. Burstedde, and F. Gibou. Parallel Level-Set Methods on Adaptive Tree-Based Grids. *Journal of Computational Physics*, 322:345–364, 2016.

[44] M. Griebel and M. Klitz. CLSVOF as a Fast and Mass-Conserving Extension of the Level-Set Method for the Simulation of Two-Phase Flow Problems. *Numerical Heat Transfer, Part B: Fundamentals*, 71(1):1–36, 2017.

[45] F. Gibou, R. Fedkiw, and S. Osher. A Review of Level-Set Methods and Some Recent Applications. *Journal of Computational Physics*, 353:82–109, 2018.

[46] M. Theillard, F. Gibou, and T. Pollock. A Sharp Computational Method for the Simulation of the Solidification of Binary Alloys. *Journal of Scientific Computing*, 63:330–354, 2014.

[47] C. H. Rycroft and F. Gibou. Simulations of a Stretching Bar Using a Plasticity Model from the Shear Transformation Zone Theory. *Journal of Computational Physics*, 231(5):2155–2179, 2012.

[48] M. Berger and J. Oliger. Adaptive Mesh Refinement for Hyperbolic Partial Differential Equations. *Journal of Computational Physics*, 53(3):484–512, March 1984.

[49] J. Strain. Tree Methods for Moving Interfaces. *Journal of Computational Physics*, 151:616–648, 1999.

[50] C. Min, F. Gibou, and H. Ceniceros. A Supra-Convergent Finite Difference Scheme for the Variable Coefficient Poisson Equation on Non-Graded Grids. *Journal of Computational Physics*, 218(1):123–140, October 2006.

[51] C. Min and F. Gibou. A Second Order Accurate Projection Method for the Incompressible Navier-Stokes Equations on Non-Graded Adaptive Grids. *Journal of Computational Physics*, 219(2):912–929, December 2006.

[52] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI: Portable Parallel Programming with the Message-Passing Interface*. Scientific and Engineering Computation. The MIT Press, third edition, 2014.

[53] C. Burstedde, L. C. Wilcox, and O. Ghattas. `p4est`: Scalable Algorithms for Parallel Adaptive Mesh Refinement on Forests of Octrees. *SIAM Journal on Scientific Computing*, 33(3):1103–1133, 2011.

[54] D. Peng, B. Merriman, S. Osher, H. Zhao, and M. Kang. A PDE-Based Fast Local Level Set Method. *Journal of Computational Physics*, 155:410–438, 1999.

[55] E. Brun, A. Guittet, and F. Gibou. A Local Level-Set Method Using a Hash Table Data Structure. *Journal of Computational Physics*, 231:2528–2536, 2012.

[56] D. Adalsteinsson and J. Sethian. A Fast Level Set Method for Propagating Interfaces. *Journal of Computational Physics*, 118:269–277, 1995.

[57] M. B. Nielsen and K. Museth. Dynamic Tubular Grid: An Efficient Data Structure and Algorithms for High Resolution Level Sets. *Journal of Scientific Computing*, 26(3):261–299, January 2006.

[58] A. du Chene, C. Min, and F. Gibou. Second-Order Accurate Computation of Interface Curvature in a Level Set Framework. *Journal of Scientific Computing*, 35:114–131, 2008.

[59] G. Russo and P. Smereka. A Level-Set Method for the Evolution of Faceted Crystals. *SIAM Journal on Scientific Computing*, 21:2073, 2000.

[60] J. Sethian. A Fast Marching Level Set Method for Monotonically Advancing Fronts. *Proc. Natl. Acad. Sci.*, 93:1591–1595, 1996.

[61] A. Chacon and A. Vladimirsky. A Parallel Two-Scale Method for Eikonal Equations. *SIAM Journal on Scientific Computing*, 37(A156-A180), 2015.

[62] A. Chacon and A. Vladimirsky. Fast Two-Scale Methods for Eikonal Equations. *SIAM Journal on Scientific Computing*, pages 1–26, 2012.

[63] H. Zhao. A Fast Sweeping Method for Eikonal Equations. *Mathematics of Computation*, 74:603–627, 2004.

[64] H. Zhao. Parallel Implementations of the Fast Sweeping Method. *Journal of Computational Mathematics*, 25:421–429, 2007.

[65] M. Detrixhe, F. Gibou, and C. Min. A Parallel Fast Sweeping Method for the Eikonal Equation. *Journal of Computational Physics*, 237:46–55, 2013.

[66] M. Detrixhe and F. Gibou. Hybrid Massively Parallel Fast Sweeping Method for static Hamilton-Jacobi Equations. *In preparation*, 2014.

[67] M. Sussman, P. Smereka, and S. Osher. A Level Set Approach for Computing Solutions to Incompressible Two-Phase Flow. *Journal of Computational Physics*, 114:146–159, 1994.

[68] C. C. Aggarwal. *Neural Networks and Deep Learning – A Textbook*. Springer, 2018.

[69] P. Mehta, M. Bukov, C. Wang, A. G. R. Day, C. Richardson, C. K. Fisher, and D. J. Schwabd. A High-Bias, Low-Variance Introduction to Machine Learning for Physicists. *Physics Reports*, 810:1–124, 2019.

[70] F. Chollet et al. Keras. https://keras.io, 2015.

[71] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, R. Jozefowicz, Y. Jia, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, M. Schuster, R. Monga, S. Moore, D. Murray, C. Olah, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, 2015. Software available from tensorflow.org.

[72] S. L. Brunton, B. R. Noack, and P. Koumoutsakos. Machine Learning for Fluid Mechanics. *Annual Review of Fluid Mechanics*, 52:477–508, 2020.

[73] F. Gibou, D. Hyde, and R. Fedkiw. Sharp Interface Approaches and Deep Learning Techniques for Multiphase Flows. *Journal of Computational Physics*, 380:442–463, 2019.

[74] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry - Algorithms and Applications*. Springer, second edition, 2000.

[75] S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, A. Dener, V. Eijkhout, W. D. Gropp, D. Karpeyev, D. Kaushik, M. G. Knepley, D. A. May, L. Curfman McInnes, R. Tran Mills, T. Munson, K. Rupp, P. Sanan, B. F. Smith, S. Zampini, H. Zhang, and H. Zhang. PETSc Web page. https://www.mcs.anl.gov/petsc, 2019.

[76] M. Zhu and S. Gupta. To Prune, or Not to Prune: Exploring the Efficacy of Pruning for Model Compression. In *ICLR Workshop*, 2018.