Chapter

Recurrent Level Set Networks for Instance Segmentation

Thi Hoang Ngan Le, Khoa Luu, Marios Savvides, Kha Gia Quach and Chi Nhan Duong

Abstract

Level set (LS)-based segmentation has been widely used in medical imaging domain. It however has some difficulty when dealing with multi-instance objects in the real world. Furthermore, LS's performance is generally quite sensitive to some initial settings and parameters such as the number of iterations. To address these issues and promote the classic LS methods to a new degree of performance in a trainable deep learning framework, we are presenting a novel approach contextual recurrent level sets (CRLS) for object instance segmentation. In the proposed networks, the curve deformation process is formed as a hidden state evolution procedure in gated recurrent units (GRUs) and updated by minimizing an energy functional composed of fitting forces and contour length.

Keywords: level sets, convolutional neural networks (CNNs), recurrent neural networks (RNNs), image segmentation, semantic segmentation

1. Introduction

One of the central goals of computer vision is object recognition and scene understanding. Humans are able to (1) possess a remarkable ability to parse an image simply by looking at them. (2) analyze an image and separate all the components present in it (3) recognize a new object that has never seen before based on observing a set of objects. The Holy Grail of a computer vision system is to able to perform as same tasks as humans are with the ability of across huge variations in pose, appearance, viewpoint, illumination, occlusion, etc. Obviously, this is a very difficult computational problem and involves in eventually making intelligent machines. An image understanding system can be roughly defined in four different subproblems, i.e., image classification, object detection, semantic segmentation, semantic instance segmentation, according to its level of complexity.

1. **Image classification:** assign a label to an image. In this subproblem, all the objects presenting in a scene are given one label, independent of its location.

2. **Object detection:** predict the bounding box around an object as well as the class of each object. This subproblem is also known as object localization.

3. **Semantic segmentation:** predict the semantic class of the individual pixels in an image. However this subproblem unable to distinguish different instances of the same class.

4. **Instance segmentation:** label as well as provide pixel-level segmentation to each object instance in the image. This subproblem is the higher level of semantic segmentation. It can be considered as a combination of the second subproblem and third problem.

This chapter focus on the problem of semantic instance segmentation by reformulating the classic LS contour evolution in a new learnable deep framework *recurrent level set* (*RLS*) that employs gated recurrent unit under the energy minimization of a LS functional. In the proposed RLS framework, the curve deformation process in the classic LS acts as a hidden state evolution procedure and updated by minimizing an energy functional composed of fitting forces and contour length. By sharing the convolutional features and collaborating with region proposal detection in a *fully end-to-end trainable framework*, we extend RLS to *Contextual RLS* (*CRLS*) to address semantic segmentation in the wild.

This book chapter is organized as follows: Section 2 introduce basic concept of both LS and deep learning including convolutional neural networks (CNNs) and recurrent neural networks (RNNs). The next two sections focus on contextual recurrent level set for object instance segmentation. The last section is targeting at experimental results which have been conducted on PASCAL VOC and COCO datasets.

2. Background

This section starts with introducing the convolutional neural networks (CNNs) models, and recurrent neural networks (RNNs), specialized artificial neural networks (ANNs) architecture particularly useful for computer vision problems. Then, we introduce active contour (AC) and a classic variational level set (LS) mechanism which have been used in image segmentation.

2.1 Convolutional neural networks

Convolutional neural networks (CNNs) [1–4] are a special case of fully connected multi-layer perceptrons that are mainly focusing on weight sharing to process grid-like topology data, i.e., an image. In CNNs, the spatial correlation of the signal is used to constrain the architecture in a more sensible way. There are two keys properties in their architecture, i.e., spatially shared weights and spatial pooling. These properties are made based on biological visual system which help them extremely useful for image applications. In such kind of network, the pooling layers aim at producing feature that are shift-invariant whereas the convolution layers is to lean visual representation. Since 2012, one of the most notable results in deep learning is the use of convolutional neural networks to obtain a remarkable improvement in object recognition for ImageNet classification challenge.

A typical convolutional network has multiple stages. Each stage contains one or many convolution layer, activation layer, pooling layer. The output of each stage is feature maps which are a set of 2D arrays. In its more general form, a convolutional network can be written as:

$$\begin{aligned} \mathbf{h}^0 &= \mathbf{x} \\ \mathbf{h}^l &= pool^l\left(\sigma_l\left(\mathbf{w}^l\mathbf{h}^{l-1} + \mathbf{b}^l\right)\right), \forall l \in 1, 2, \ldots L \\ \mathbf{o} &= \mathbf{h}^L = f(\mathbf{x}, \theta) \end{aligned} \tag{1}$$

where $\mathbf{w}^l$, $\mathbf{b}^l$ are trainable parameters as in MLPs at layer l. $\mathbf{x} \in \mathrm{R}^{c \times h \times w}$ is vectorized from an input image with c is color channels, h is the image height and w is the image width. $\mathbf{o} \in \mathrm{R}^{n \times h' \times w'}$ is vectorized from an array of dimension $h' \times w'$ of output vector (of dimension n). $pool^l$ is a (optional) pooling function at layer l.

The main difference between multi-layer neural networks (MLPs) and CNNs lies in the parameter matrices $\mathbf{w}_l$. In MLPs, the matrices can take any general form, while in CNNs these matrices are constraints to be Toeplitz matrices. That is, the columns are circularly shifted versions of the signal of various shifts for each columns in order to implement spatial correlation operation using matrix algebra. Furthermore, these matrices are very sparse because the image size is much bigger than the kernel size. Thus, the hidden unit $\mathbf{h}_l$ can be expressed as a discrete-time convolution between the kernel $\mathbf{w}_l$ and the previous hidden unit $\mathbf{h}_{l-1}$. A point-wise non-linearity and pooling are then applied on the hidden unit. There are numerous variants of CNNs architectures in the literature. However, their basic components are very similar which contains of convolutional layer, pooling layer, activation layer and fully connected layer.

- **Convolutional layer:** the convolutional layer aims to learn feature representations of the inputs and it is composed of several convolution kernels which are used to compute different feature maps. In a convolutional layer, each neuron of a feature map is connected to a region of neighboring neurons in the previous layer. The new feature map can be obtained by first convolving the input with a kernel and then applying an element-wise nonlinear activation function on the convolved results. The feature maps are obtained by using a set of different kernels.

- **Activation layer:** the activation function introduces nonlinearities to CNNs. This layer controls how the signal flows from one layer to the next, emulating how neurons are fired in our brain.

- **Pooling layer:** pooling layer is also called sampling layer to smooth the input from the convolutional layer. The pooling layer helps to (1) reduce the sensitivity of the filters to noise and variations. Pooling operation is usually placed between two convolutional layers. Each feature map of a pooling layer is connected to its corresponding feature map of the preceding convolutional layer.

- **Loss layer:** it is important to choose an appropriate loss function for a specific task. *Softmax* loss is a commonly used loss function which is essentially a combination of multinomial logistic loss and softmax. Softmax turns the predictions into non-negative values and normalizes them to get a probability distribution over classes. Such probabilistic predictions are used to compute the multinomial logistic loss.

- **Regularization:** regularization aims to reduce the overfitting problem during training. l_p*-norm*, Dropout, and DropConnect are some common regularization techniques have been developed to solve this.

- **Optimization:** data augmentation, weight initialization, stochastic gradient descent, batch normalization, shortcut connections are some common optimization techniques that will be discussed. Popular *data augmentation* methods are sampling, various photometric transformations, shifting, and greedy strategy. *Stochastic gradient descent* [5] (SGD) is the most popular technique used to update the parameter during backpropagation.

2.2 Recurrent neural networks

The Recurrent Neural Networks (RNNs) is an extremely powerful sequence model and was introduced in the early 1990s [6]. A typical RNNs contains three parts, namely sequential input data ($\mathbf{x}_t$), hidden state ($\mathbf{h}_t$) and sequential output data ($\mathbf{o}_t$).

RNNs contains a sequence of elements and each element performs the similar task. The input of future element depends on the output of current element. The RNNs contains one or many input and one or many output depending on the applications. Some examples, of RNNs architecture are given in **Figure 1**.

In RNNs, the activation of the hidden states at timestep t is computed as a function $\mathbf{F}$ of the current input symbol $\mathbf{x}_t$ and the previous hidden states $\mathbf{h}_{t-1}$. The output at time t is calculated as a function $\mathbf{G}$ of the current hidden state $\mathbf{h}_t$ as follows:

$$\begin{aligned} \mathbf{h}_t &= \mathbf{F}(\mathbf{U}\mathbf{x}_t + \mathbf{W}\mathbf{h}_{t-1}) \\ \mathbf{o}_t &= \mathbf{F}(\mathbf{V}\mathbf{h}_t) \end{aligned} \tag{2}$$

where $\mathbf{W}$ is the state-to-state recurrent weight matrix, $\mathbf{U}$ is the input-to-hidden weight matrix, $\mathbf{V}$ is the hidden-to-output weight matrix. $\mathbf{F}$ is usually a logistic sigmoid function or a hyperbolic tangent function and $\mathbf{G}$ is defined as a softmax function.

Most work on RNNs has made use of the method of backpropagation through time (BPTT) [7] to train the parameter set $(\mathbf{U}, \mathbf{V}, \mathbf{W})$ and propagate error backward through time. In classic backpropagation, the error or loss function is defined as:

$$E(\mathbf{o}, \mathbf{y}) = \sum_t \left\| \mathbf{o}_t - \mathbf{y}_t \right\|^2 \tag{3}$$

where $\mathbf{o}_t$ is prediction and y_t is labeled groundtruth.

For a specific weight $\mathbf{W}$, the update rule for gradient descent is defined as $\mathbf{W}^{new} = \mathbf{W} - \gamma \frac{\partial E}{\partial \mathbf{W}}$, where γ is the learning rate. In RNNs model, the gradients of the error with respect to our parameters $\mathbf{U}$, $\mathbf{V}$ and $\mathbf{W}$ are learned using stochastic gradient descent (SGD) and chain rule of differentiation.

There are two popular improvements of RNNs in order to solve the problems of exploding and vanishing when dealing with long time dependency.

- **Long short time memory:** to address the issue of learning long-term dependencies, Hochreiter and Schmidhuber [8] proposed Long Short-Term Memory (LSTM), which is able to maintain a separate memory cell inside it that updates and exposes its content only when deemed necessary. Similar to RNNs, LSTMs are defined under a chain like structure, but the repeating module has a different structure. Instead of having a single neural network

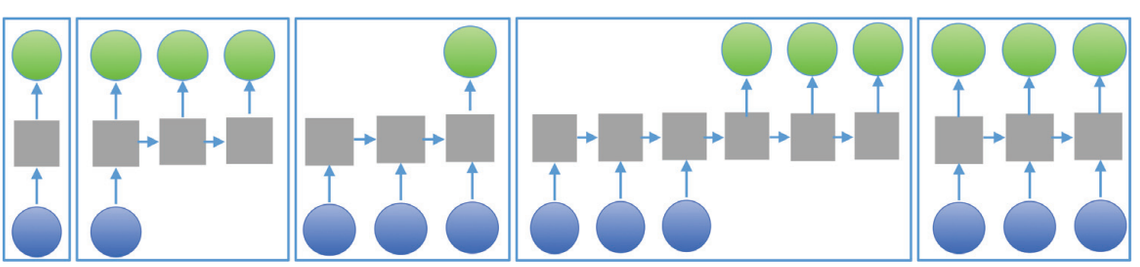

Figure 1.
Example of RNNs architecture.

layer (single tanh layer), LSTMS have four neural networks in each layer and they are interacting in a very special way. The key to LSTMs is the cell state (C_t) which has the ability to remove or add information by optionally letting information through. Gates are composed out of a sigmoid layer and a pointwise multiplication operation.

- **Gated recurrent units:** to make each recurrent unit adaptively capture dependencies of different time scales, [9] proposed gated recurrent unit (GRU) as an extension of RNN. Similar LSTM unit, GRU contains gating units that control the flow of information inside the unit. However, GRU does not have separate memory cells. By using leaky integration, GRU is able to fully expose its memory content each timestep and balance between the previous memory content and the new memory content strictly. Notably, its adaptive time constant is controlled by the update gate. In traditional RNNs, the content of activation unit is always replaces by a new value calculated from the current input and the previous hidden state. Instead of replacing, GRU jsut adds the new content on top and still keeps the existing content. This addition provides shortcut paths that bypass multiple temporal steps. In GRUs, thank to these shortcuts, the error is back-propagated easily without too quickly vanishing as a result of passing through multiple, bounded nonlinearities. Thus it help to reduce the difficulty due to vanishing gradients. Furthermore, using the addition, each unit is able to remember the existence of a specific feature in the input stream for a long series of steps.

2.3 Variational level set

Variational level set is an implicit implementation of active contour (AC). The main ideas of applying AC for image segmentation is to start with an initial random (guess) contour C which is represented in a form of closed curves. The curve is then iteratively adjusted by shrinking or expanding under an image-driven forces until it reach to the boundaries of the desired objects. The entire process is called contour evolution or curve evolution, denoted as $\frac{\partial C}{\partial t}$.

There are two main approaches in active contours: snakes and level sets. Snakes explicitly move predefined snake points based on an energy minimization scheme, while level set approaches move contours implicitly as a particular level of a function.

Level set (LS)-based or implicit active contour models have provided more flexibility and convenience for the implementation of active contours, thus, they have been used in a variety of image processing and computer vision tasks. The basic idea of the implicit active contour is to represent the initial curve C implicitly within a higher dimensional function, called the level set function $\phi(x,y) : \Omega \rightarrow R$, such as: $C = (x,y) : \phi(x,y) = 0, \forall (x,y) \in \Omega$, where Ω denotes the entire image plane.

A zero level set function is used to formulate the contour, i.e., the contour evolution is equivalent to the evolution of the level set function, i.e., $\frac{\partial C}{\partial t} = \frac{\partial \phi(x,y)}{\partial t}$. The reason of using the zero level set is that a contour can be defined as the border between a positive area and a negative area. Thus, everything on the zero area belongs to the contour and it is identified by signed distance function as follows:

$$\phi(x) = \begin{cases} d(x,C)) & \text{if } x \text{ is inside } C \\ 0 & \text{if } x \text{ is on } C \\ -d(x,C)) & \text{if } x \text{ is outside } C \end{cases} \tag{4}$$

where $d(x,C)$ denotes the distance from an arbitrary position to the curve.

The computation is performed on the same dimension as the image plane Ω, therefore, the computational cost of level set methods is high and the convergence speed is quite slow.

Under the scenarios of image segmentation, we consider an image in 2D space, Ω. The level set is to find the boundary C of an open set $\omega \in \Omega$, which is defined as: $C = \partial\omega$. In LS framework, the boundary C can be represented by the zero level set ϕ as follows:

$$\forall(x,y) \in \Omega \begin{cases} C = & \{(x,y) : \phi(x,y) = 0\} \\ inside(C)) & \{(x,y) : \phi(x,y) > 0\} \\ output(C)) & \{(x,y) : \phi(x,y) < 0\} \end{cases} \tag{5}$$

Under the image segmentation problem, entire domain of an image **I** is presented by Ω which contains three regions corresponding to: inside contour (¿0), on contour (=0) and outside the contour (¡0). Clearly, the zero LS function ϕ partitions the region Ω into two regions: region inside ω (foreground), denoted as inside (C) and region outside ω (background) denoted as outside (C). Using the zero level set function, the length of the contour C and the area inside the contour C are defined as follows:

$$\begin{aligned} Length(C) &= \int_{\Omega} |\nabla H(\phi(x,y))| dxdy = \int_{\Omega} \delta(\phi(x,y)) |\nabla \phi(x,y)| dxdy \\ Area(C) &= \int_{\Omega} H(\phi(x,y)) dxdy \end{aligned} \tag{6}$$

where $H_{\varepsilon}(\cdot)$ is a Heaviside function.

Typically, the LS-based segmentation methods start with an initial level set ϕ_0 and an given image **I**. The LS updating process is performed via gradient descent by minimizing an energy function which defined based on the difference of image features, such as color and texture, between foreground and background. The fitting term in LS model is defined by the inside contour energy (E_1) and outside contour energy (E_2).

$$E = E_1 + E_2 = \int_{insideC} \left(I_{(x,y)} - c_1\right)^2 dxdy + \int_{outsideC} \left(I_{(x,y)} - c_2\right)^2 dxdy \tag{7}$$

where c_1 and c_2 are the average intensity inside and outside the contour C, respectively.

One of the most popular region based active contour models is proposed by Chan-Vese (CV) [10]. In this model the boundaries are not defined by gradients and the curve evolution is based on the general Mumford-Shah (MS) [11] formulation of image segmentation as shown in Eq. (8).

$$E = \int_{\Omega} |\mathbf{I} - u|^2 dxdy + \int_{\Omega/C} |\nabla u|^2 dxdy + \nu Length(C) \tag{8}$$

CV's model is an alternative form of MS's model which restricts the solution to piecewise constant intensities and it has successfully segmented an image into two regions, each having a distinct mean of pixel intensity by minimizing the following energy functional.

$$E(c_1, c_2, \phi) = \mu \text{Area}(\omega_1) + \nu \text{Length}(C) + \lambda_1 \int_{\omega_1} |\mathbf{I}(x,y) - c_1|^2 dxdy + \lambda_2 \int_{\omega_2} |\mathbf{I}(x,y) - c_2|^2 dxdy \tag{9}$$

where c_1 and c_2 are two constants. The parameters μ, ν, λ_1, λ_2 are positive parameters and usually fixing $\lambda_1 = \lambda_2 = 1$ and $\mu = 0$. Thus, we can ignore the first term in Eq. (9). Thus the energy functional is rewritten as follows:

$$\begin{aligned} E(c_1, c_2, \phi) = \mu \int_{\Omega} H(\phi(x,y))dxdy + \nu \int_{\Omega} \delta(\phi(x,y))|\nabla\phi(x,y)|dxdy \\ + \lambda_1 \int_{\omega_1} |\mathbf{I}(x,y) - c_1|^2 dxdy + \lambda_2 \int_{\omega_2} |\mathbf{I}(x,y) - c_2|^2 dxdy \end{aligned} \tag{10}$$

For numerical approximations, the δ function needs a regularizing term for smoothing. In most cases, the Heaviside function H and Dirac delta function δ are defined as in Eq. (11) and Eq. (12), respectively.

$$H_\varepsilon(x) = \frac{1}{2}\left(1 + \frac{2}{\pi}\arctan\left(\frac{x}{\varepsilon}\right)\right) \text{and } \delta_\varepsilon(x) = H'(x) = \frac{1}{\pi}\frac{\varepsilon}{\varepsilon^2 + x^2} \tag{11}$$

$$\delta_\varepsilon(x) = H'(x) = \frac{1}{\pi}\frac{\varepsilon}{\varepsilon^2 + x^2} \tag{12}$$

As $\varepsilon \to 0$, $\delta_\varepsilon \to \delta$, and $H_\varepsilon \to H$. Using Heaviside function H, the Eq. 10 becomes Eq. 13.

$$\begin{aligned} E(c_1, c_2, \phi) = \mu \int_{\Omega} H(\phi(x,y))dxdy + \nu \int_{\Omega} \delta(\phi(x,y))|\nabla\phi(x,y)|dxdy \\ + \lambda_1 \int_{\Omega} |\mathbf{I}(x,y) - c_1|^2 H(\phi(x,y))dxdy + \lambda_2 \int_{\Omega} |\mathbf{I}(x,y) - c_2|^2 (1 - H(\phi(x,y)))dxdy \end{aligned} \tag{13}$$

In the implementation, they choose $\varepsilon = 1$. For fixed c_1 and c_2, gradient descent equation with respect to ϕ is:

$$\frac{\partial\phi(x,y)}{\partial t} = \delta_\varepsilon\, \phi(x,y)\left[\nu\kappa\, \phi(x,y) - \mu - \lambda_1(\mathbf{I}(x,y) - c_1)^2 + \lambda_2(\mathbf{I}(x,y) - c_2)^2\right] \tag{14}$$

where δ_ε is a regularized form of Dirac delta function and c_1, c_2 are the mean of inside the contour ω_{in} and the mean of the outside of the contour ω_{out}, respectively. The curvature κ is given by:

$$\kappa(\phi(x,y)) = -\text{div}\left(\frac{\Delta\phi}{|\Delta\phi|}\right) = -\frac{\phi_{xx}\phi_y^2 - 2\phi_x\phi_y\phi_{xy} + \phi_{yy}\phi_x^2}{\left(\phi_x^2 + \phi_y^2\right)^{1.5}} \tag{15}$$

where $\partial_x\varphi_t$, $\partial_y\varphi_t$ and $\partial_{xx}\varphi_t$, $\partial_{yy}\varphi_t$ are the first and second derivatives of φ_t with respect to x and y directions. For fixed ϕ, gradient descent equation with respect to c_1 and c_2 are:

$$c_1 = \frac{\sum_{x,y}\mathbf{I}(x,y)H(\phi(x,y))}{\sum_{x,y}H(\phi(x,y))} \text{and } c_2 = \frac{\sum_{x,y}\mathbf{I}(x,y)(1 - H(\phi(x,y)))}{\sum_{x,y}(1 - H(\phi(x,y)))} \tag{16}$$

herein, we use notation φ_t to indicate φ at the iteration t^{th} in order to distinguish the φ at different iterations. Under this formulation, the curve evolution is shown as a time series process which helps to give better visualization of reformulating LS. From this point, we redefine the curve updating in a time series form for the LS function φ_t as in Eq. (17).

$$\varphi_{t+1} = \varphi_t + \eta \frac{\partial \varphi_t}{\partial t} \tag{17}$$

The LS at time $t+1$ depends on the previous LS at time t and the curve evolution $\frac{\partial \varphi_t}{\partial t}$ with a learning rate η.

3. Recurrent level set for object instance segmentation

To reformulate level set under a RNNs/GRUs deep framework, we need to solve the following problems:

- **Input:** How to generate a sequential data x_t from a single image I if we treat ϕ_t in LS as hidden state h_t in RNNs with the same updating manner?

- **Output:** In the segmentation problem by LS, the binary mask corresponding to foreground and background is computed by one forward process with many iterations whereas the RNNs is computed by forward procedure and learnt by backward procedure. How to compute the error and perform backward procedure in LS framework?

- **Update rule:** Does the update rule in LS work in the same fashion as the forward procedure in RNNs/GRUs deep framework?

As shown in **Table** 1, one of the most challenging problems of reformulating LS as RNNs is data configuration. RNNs work on sequential data whereas LS uses single image as input and produce a mask as single image too. The first question here is *how to generate sequential data from a single image*. Moreover, there are two source inputs used in LS, i.e., an input image I and an initial contour which is treated as initial LS function ϕ_0 and updated by Eq. (17). That means we need to generate a sequential data x_t $(t = 1, \cdots, N)$ from single image $\mathbf{I}$ in our proposed RLS. In order to achieve this goal, we define a function $g(\mathrm{I}, \phi_{t-1})$ as in Eq. (18).

$$\mathbf{x}_t = g(\mathbf{I}, \phi_{t-1}) = \phi_{t-1} + \eta\left[\kappa(\phi_{t-1}) - \mathbf{U}_g(\mathbf{I} - c_1)^2 + \mathbf{W}_g(\mathbf{I} - c_2)^2\right] \tag{18}$$

In Eq. (18), c_1 and c_2 are average values of inside and outside of the contour presented by the LS function ϕ_{t-1} and defined in Eq. (16). κ denotes the curvature and defined in Eq. (15). $\mathbf{U}_g$ and $\mathbf{W}_g$ are two matrices that control the force inside and outside of the contour.

In such equation, the curve evolution in the proposed RLS functions as same as in the traditional LS defined by Eq. (14), i.e. the input at iteration t^{th}, $\mathbf{x}_t$, is updated based on the input image $\mathbf{I}$ and the previous LS function ϕ_{t-1}. In our proposed RLS, $\mathbf{x}_t$ is considered to be sequential input whereas LS updating ϕ_t is treated as hidden state updating. Notably, the initial contour in LS function plays the role as initial hidden state as randomly generated. The relationship among $\mathbf{I}$, $\mathbf{x}_t$, and ϕ_t are given in **Figure 2**.

So far, the question of generating input sequence from the single image $\mathbf{I}$ has been answered. In the proposed RLS, we use the same input as defined in LS problem, namely the input image $\mathbf{I}$ and the initial ϕ_0. We generate sequential data $\mathbf{x}_t$ from the input as single image. The next challenging problem is how to generate the hidden state ϕ_t from the input data x_t and the previous hidden state ϕ_{t-1}. Under the same intuition of proposing GRUs [9], the procedure of generating hidden state ϕ_t

	Input	Update	Output
CLS	Image $\mathbf{I}$ Initial LS function ϕ_0	$\phi_{t+1} = \phi_t + \eta \frac{\partial \phi_t}{\partial t}$ $\frac{\partial \phi_t}{\partial t} = \delta_\epsilon(\phi_t[\nu\kappa(\phi_t - \mu\ \lambda_1(\mathbf{I} - c_1)^2 + \lambda_2(\mathbf{I} - c_2)^2])$	ϕ_N
GRUs	Sequence $x_1, x_2, \ldots x_N$ Initial hidden state h_0	$\mathbf{z}_t = f(\mathbf{U}_z\mathbf{x}_t + \mathbf{W}_z\mathbf{h}_{t-1})$ $\mathbf{r}_t = f(\mathbf{U}_r\mathbf{x}_t + \mathbf{W}_r\mathbf{h}_{t-1})$ $\tilde{\mathbf{h}}_t = h(\mathbf{U}_h\mathbf{x}_t + \mathbf{W}_h(\mathbf{h}_{t-1} \circ \mathbf{r}_t))$ $\mathbf{h}_t = (1 - \mathbf{z}_t)\mathbf{h}_{t-1} + \mathbf{z}_t\tilde{\mathbf{h}}_t$ h: Tanh f: Sigmoid or Tanh	$\mathcal{G}(\mathbf{V}\mathbf{h}_N)$ $\mathcal{G}$: softmax
RLS	Image $\mathbf{I}$ Initial LS function ϕ_0	$\mathbf{x}_t = \kappa\left(\phi_{t-1} - \mathbf{U}_g(\mathbf{I} - c_1)^2 + \mathbf{W}_g(\mathbf{I} - c_2)^2\right)$ $\mathbf{z}_t = \sigma(\mathbf{U}_z\mathbf{x}_t + \mathbf{W}_z\phi_{t-1} + \mathbf{b}_z)$ $\mathbf{r}_t = \sigma(\mathbf{U}_r\mathbf{x}_t + \mathbf{W}_r\phi_{t-1} + \mathbf{b}_r)$ $\tilde{\mathbf{h}}_t = h(\mathbf{U}_y\mathbf{x}_t + \mathbf{W}_y(\phi_{t-1} \circ \mathbf{r}_t) + \mathbf{b}_y)$ $\phi_t = (1 - \mathbf{z}_t)\phi_{t-1} + \mathbf{z}_t\mathbf{h}t$ h: Tanh σ: Sigmoid	$\mathcal{G}(\mathbf{V}\phi_N + b_V)$ $\mathcal{G}$: softmax

Table 1.
Comparison of input, update rule and output used in traditional LS, GRUs, and our proposed RLS.

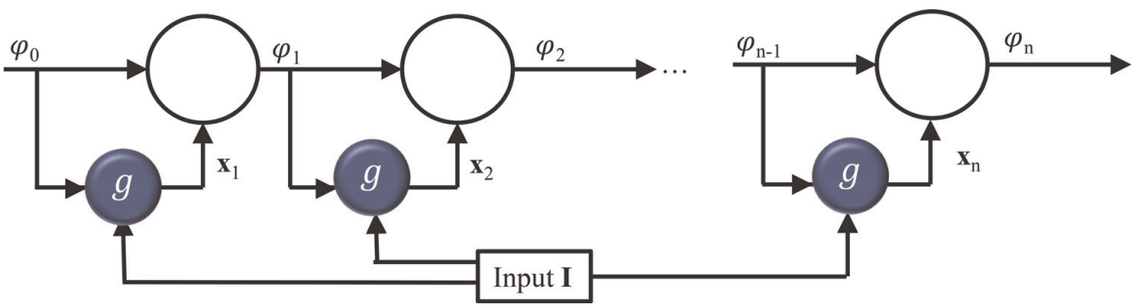

Figure 2.
The visualization of generating sequence input data $\mathbf{x}_t$ and hidden state ϕ_t from the input image $\mathbf{I}$ and the initial LS function ϕ_0.

is based on the updated gate $\mathbf{z}_t$, the candidate memory content $\tilde{\mathbf{h}}_t$ and the previous activation unit ϕ_{t-1} as the rule given in Eq. (19).

$$\phi_t = \mathbf{z}_t\tilde{\mathbf{h}}_t + (1 - \mathbf{z}_t)\phi_{t-1} \tag{19}$$

The update gate $\mathbf{z}_t$, which controls how much of the previous memory content is to be forgotten and how much of the new memory content is to be added is defined as in Eq. (20).

$$\mathbf{z}_t = \sigma(\mathbf{U}_z\mathbf{x}_t + \mathbf{W}_z\phi_{t-1} + \mathbf{b}_z) \tag{20}$$

where σ is a sigmoid function and $\mathbf{b}_z$ is the update bias. However, the propose RLS does not have any mechanism to control the degree to which its state is exposed, but exposes the whole state each time in such naive definition. To address that issue, the new candidate memory content y_t is computed. as in Eq. (21).

$$\tilde{\mathbf{h}}_t = tanh\left(\mathbf{U}_{\tilde{h}}\mathbf{x}_t + \mathbf{W}_{\tilde{h}}(\phi_{t-1} \odot \mathbf{r}_t) + \mathbf{b}_{\tilde{h}}\right) \tag{21}$$

where $\odot$ denotes an element-wise multiplication, $\mathbf{b}_{\tilde{h}}$ is the hidden bias. The reset gate $\mathbf{r}_t$ is computed similarly to the update gate as in Eq. (22).

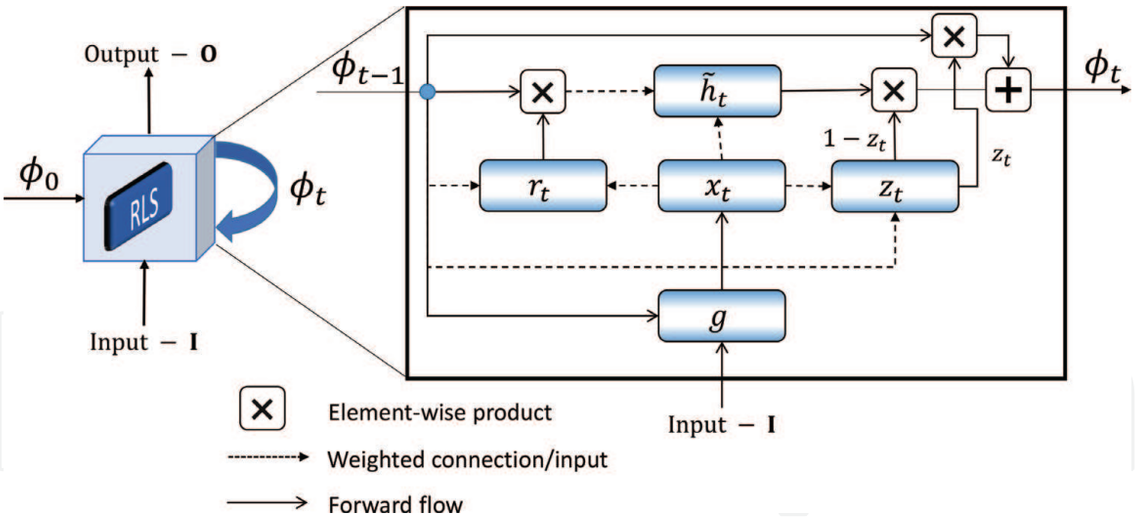

Figure 3.
The proposed RLS network for curve updating process under the sequential evolution and its forward computation of curve evolution from time $t-1$ to time t.

$$\mathbf{r}_t = \sigma(\mathbf{U}_r\mathbf{x}_t + \mathbf{W}_r\phi_{t-1} + \mathbf{b}_r) \tag{22}$$

where $\mathbf{b}_r$ is the reset bias. When $\mathbf{r}_t$ is close to 0 (off), the reset gate effectively makes the unit act as if it is reading the first symbol of an input sequence, allowing it to forget the previously computed state. The output $\mathbf{o}$ is computed from the current hidden states ϕ_t and then a softmax function is applied to obtain foreground/background segmentation $\hat{\mathbf{y}}$ given the input image as follows:

$$\begin{aligned} \hat{\mathbf{y}} &= \text{softmax}(\mathbf{o}) \\ \mathbf{o} &= \mathbf{V}\phi_t + \mathbf{b}_V \end{aligned} \tag{23}$$

where $\mathbf{V}$ is weighted matrix between hidden state and output. **Figure 4** (left) illustrate the proposed RLS in folded mode where the input of the network is defined as the same as the LS model, i.e., a given input image $\mathbf{I}$ and an initial LS function ϕ_0. In our proposed RLS, the curve evolution from ϕ_t at present time step t to the future ϕ_{t+1} at time $t+1$ is designed in the same fashion as the hidden state in GRUs and is illustrated in **Figure 4** (right) where ϕ_{t+1} depends on both previous LS function ϕ_t and the present input $\mathbf{x}_{t+1}$. We have summarized the comparison among LS, GRUs and our proposed RLS in **Table 1** and visulized the relationship between LS and GRUs as **Figure 3**. It is easy to see that RLS uses the same input as in traditional LS where the curve evolution (i.e., update procedure) and output in the proposed RLS follows similar fashion as in GRUs.

We summarize the proposed building block RLS in Algorithm 1.

Algorithm 1. Algorithm of the proposed building block RLS

Input: Given an image $\mathbf{I}$, an initial level set function ϕ_0, time step N, learning rate η, initial parameters $\theta = (\mathbf{U}_z, \mathbf{W}_z, \mathbf{b}_z, \mathbf{U}_r, \mathbf{W}_r, \mathbf{b}_r\ \mathbf{U}_{\tilde{h}}, \mathbf{W}_{\tilde{h}}, \mathbf{b}_{\tilde{h}}, \mathbf{V}, \mathbf{b}_V)$.

for each epoch **do**
  Set $\phi = \phi_0$
  for t = 1 : T **do**
    Generate RLS input $\mathbf{x}_t$: $\mathbf{x}_t \leftarrow g(\mathbf{I}, \phi_{t-1})$
    Compute update gate $\mathbf{z}_t$, reset gate $\mathbf{r}_t$ and intermediate hidden unit $\tilde{\mathbf{h}}_t$:
    $\mathbf{z}_t \leftarrow \sigma(\mathbf{U}_z\mathbf{x}_t + \mathbf{W}_z\phi_{t-1} + \mathbf{b}_z)$
    $\mathbf{r}_t \leftarrow \sigma(\mathbf{U}_r\mathbf{x}_t + \mathbf{W}_r\phi_{t-1} + \mathbf{b}_r)$

$$\tilde{\mathbf{h}}_t \leftarrow \tanh\left(\mathbf{U}_{\tilde{h}} x_t + \mathbf{W}_{\tilde{h}}\left(\phi_{t-1} \circ \mathbf{r}_t\right) + \mathbf{b}_{\tilde{h}}\right)$$

Update the zero LS ϕ_t: $\phi_t \leftarrow (1 - \mathbf{z}_t)\tilde{\mathbf{h}}_t + \mathbf{z}_t \phi_{t-1}$

end for

Compute the loss function L: $L \leftarrow -\sum_n \mathbf{y}_n \log \hat{\mathbf{y}}_n$

Compute the derivate w.r.t. θ: $\nabla\theta \leftarrow \frac{\partial L}{\partial \theta}$.

Update θ: $\theta \leftarrow \theta + \eta\nabla\theta$

end for

The proposed RLS described in the previous section performs the image segmentation task, i.e., dividing an image into two parts corresponding foreground and background segments. Given a real-world image, RLS however performs neither the instance segmentation nor image understanding which are significant tasks in many computer vision application. In order solve this requirement, we introduce contextual recurrent level sets (CRLS) for semantic object segmentation which is an extension of our proposed RLS model to address the multi-instance object segmentation in the wild. The proposed CRLS is able to (1) localize objects existing in the given image; (2) segment the objects out of the background; (3) classify the objects in the image. The output of our CRLS is multiple values (each value is corresponding to one object class) instead of two values (foreground and background) as in RLS.

Our proposed CRLS inherits the merits of RLS and faster-RCNN [12] for semantic segmentation which simultaneously performs three tasks which are addressed by three stages, i.e., detection, segmentation and classification in a fully end-to-end trainable framework as shown in **Figure 5**. In our CRLS, the network

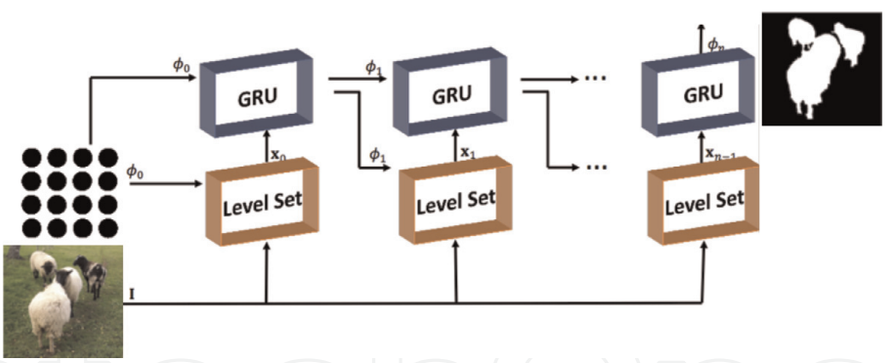

Figure 4.
Visualization of relationship between LS and GRUs in a RLS unit.

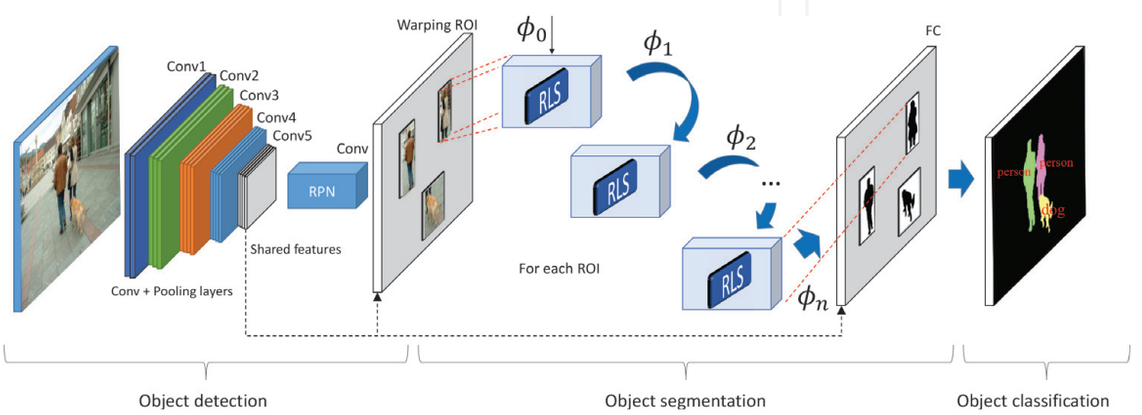

Figure 5.
The flowchart of our proposed CRLS for semantic instance segmentation with three tasks which are addressed by three stages, i.e. object detection by Faster R-CNN [12], object segmentation by RLS and classiffication.

takes an image of arbitrary size as the input, and outputs instance-aware semantic segmentation results. The network contains three components corresponding three stages of a semantic instance segmentation: object detection for proposing box-level, object segmentation is for mask-level and object classification is for categorizing each instance. These three stages are designed to share convolutional features. Each stage involves a loss term and the loss of later stage relies on the output of an predecessor stage, so the three loss terms are not independent. We train the entire network end-to-end with a unified loss function.

3.1 Stage 1: object detection

One of the most important approaches to the object detection and classification problems is the generations of region-based convolutional neural networks (R-CNN) family [12–15].

In order to propose a fully end-to-end trainable network, we adapt the region proposal network (RPN) introduced in Faster R-CNN [12] to predict the object bounding boxes and the objectness scores. Aiming at reducing time consuming, both the detection and segmentation procedure share the convolutional features of a deep VGG-16 network [16].

As for the share convolutional features, we utilize a VGG-16 networks with 13 convolution layers where each convolution layer is followed by a ReLU layer but only four pooling layers are placed right after the convolution layer to reduce the spatial dimension.

In this stage, object detection, the network proposes object instances in the form of bounding boxes which are predicted with an objectness score. The network structure and loss function of this stage follow the work of Region Proposal Networks (RPNs) [12]. RPNs take an image of any size as the input and predicts bounding box locations and objectness scores in a fully-convolutional form. A 3×3 convolutional layer for reducing the dimension is applied on top of share convolutional features. The lower dimension features are fed into two sibling 1×1 convolutional layers: one is for a box-regression layer and the other is for box-classification layer.

From the share feature maps and at each sliding-window location, multiple region proposals are generated. Denote k is the maximum possible proposals for each location. Each box is presented by four values corresponding to locations (x, y w, h) and two scores corresponding the probability of object or not object. Each anchor, which is centered at the sliding window, is associated with an aspect ratio and scale. In the experiments, each sliding position has 9 ($k = 9$) anchors corresponding to three scales and three aspect ratios. If the sharing feature maps of size $W \times H$, there are $k \times W \times H$ anchors. For the purpose of object detection, there are only two anchors considered:

- *Positive anchor*: there are two cases are taken into account, i.e., (i) an anchor with the highest Intersection-over-Union (IoU) overlap with a ground-truth box, (ii) an anchor that has an IoU overlap higher than 0.7 with any ground-truth box.

- *Negative anchor*: an anchor with IoU score is lower than 0.3 with all ground-truth boxes.

- *Ignore anchor*: an anchor that is neither positive nor negative do not contribute to the training objective.

There are two sibling output layers: probability of classification and bounding box regression in the end of RPNs, therefore, the RPNs loss is based on two losses: boxes regression loss and object classification loss.

$$L_{RPN}\left(p^i, b^i\right) = \frac{1}{N_{cls}} \sum_i L_{cls}\left(p^i, p^{i*}\right) + \lambda \frac{1}{N_{reg}} \sum_i p^{i*} L_{reg}\left(b^i, b^{i*}\right) \tag{24}$$

In this equation:

- i is the index of an anchor in a mini-batch.
- p^i is the predicted probability of anchor i being an object and p^i is computed by a softmax over the K outputs of a fully connected layer (K categories).
- N_{cls} and N_{reg} are the two normalization terms.
- λ is a balancing parameter.
- p^{i*} is groundtruth and it sets as 1 if the anchor is positive, and 0 if the anchor is negative.
- $b^i = \left(b_x^i, b_y^i, b_w^i, b_h^i\right)$ is a vector representing the four parameterized coordinates of the predicted bounding box.
- $b^{i*} = \left(b_x^{i*}, b_y^{i*}, b_w^{i*}, b_h^{i*}\right)$ is groundtruth bounding of positive box and is a vector representing the four parameterized coordinates. Let denote $(x^{i*}, y^{i*}, w^{i*}, h^{i*})$, (x^i, y^i, w^i, h^i), (x^a, y^a, w^a, h^a) are four coordinations (box center, width, and height) of groundtruth box, predicted box and anchor box, the parameterization of the four coordinates are as follows:

$$\begin{aligned} b_x^{i*} &= \left(x^{i*} - x^a\right)/w^a \quad b_x^i = \left(x^i - x^a\right)/w^a \\ b_y^{i*} &= \left(y^{i*} - y^a\right)/h^a \quad b_y^i = \left(y^i - y^a\right)/h^a \\ b_w^{i*} &= \log\left(w^{i*}/w^a\right) \quad b_w^i = \log\left(w^i/w^a\right) \\ b_h^{i*} &= \log\left(h^{i*}/h^a\right) \quad b_h^i = \log\left(h^i/h^a\right) \end{aligned} \tag{25}$$

- The bounding boxes regression is defined by $smooth_{l_1}$ as follows:

$$L_{reg} = \sum_{u \in x, y, w, h} smooth_{l_1}\left(b_u^i - b_u^{i*}\right) \tag{26}$$

$$smooth_{l_1}(x) = \begin{cases} \frac{1}{2}x^2 & if \quad |x| < 1 \\ |x| - 0.5 & otherwise \end{cases} \tag{27}$$

The classification loss (L_{cls}) is log loss over two classes (object vs. not object). The RPNs loss is defined as in Eq. (28) where i is the ground truth class.

$$L_{cls}(p^i, p^{i*}) = -\log(p^i). \tag{28}$$

To easy to follow, we denote the loss of this stage as: $L_{RPN} = L_{RPN}(B(\Theta))$

where Θ represents all network parameters to be optimized. B is the network output of this stage, representing a list of bounding boxes $B = \{b\}_i$. Each bounding box is presented by four coordinations (box center, width, and height) and predicted objectness probability $b_i = (b_x, b_y, b_w, b_h, p_i)_i$.

3.2 Stage 2: object segmentation

The second stage takes the share features and the results from the first stage (predicted box) as inputs. The output of this stage is binary segmentation which contains foreground (object) and background.

For each predicted box, we make use of RoI warping layer to crop and warp a region on the feature map into the target fixed size by interpolation. Each predicted box from the deep feature map (conv5) is now resized to $m \times m$ where $m = 21$ in our experiment. To reduce the size of a region, RoI warping layer is performed by using a pooling payer to convert features inside any valid region of interest into a small feature map. Each RoI is defined by top-left corner (r, c) and its height and width (h, w). That means, it is defined by a four-tuple (r, c, h, w). The feature map of size $h \times w$ is partitioned into $m \times m$ grid of sub-windows of approximate size $\frac{h}{m} \times \frac{w}{m}$. The max pooling is then applied into each sub-window. As a result, RoI warping layer outputs a grid cell.

The fixed size extracted features are passed through the proposed RLS together with a randomly initial ϕ_0 to generate a sequence input data $\mathbf{x}_t$ based on Eq. (18). The curve evolution procedure is performed via LS updating process given in Eqs. (14) and (17). This task outputs a binary mask as given in Eq. (23) sized $m \times m$ and parameterized by an m^2 dimensional vector.

Given a set of predicted bounding box from the first stage, the loss term L_{SEG} of the second stage for foreground segmentation is given by:

$$L_{SEG} = L_{SEG}(S(\Theta)|B(\Theta)) \tag{29}$$

Here, S is the network designed by RLS and is presenting a list of segmentations $S = \{S\}_i$. Each segmentation S_i is an $m \times m$ mask.

3.3 Stage 3: object classification via fully-connected network

The third stage takes the share feature, bounding box from the first stage, object segmentation from the second stage as inputs. The output of this stage is class score for each object instance.

For each predicted bounding box from the first stage, we first extract the feature by ROI pooling (f_{ROI}). The feature is then go through the proposed RLS and presented by a binary mask ($f_{SEG}(\Theta)$) from the second stage. The input of this stage (f_{MSK}) is the masked feature which depends on the segmentation results ($f_{SEG}(\Theta)$) and ROI feature (f_{ROI}) and computed by element-wise product of those, $f_{MSK}(\Theta) = f_{ROI}(\Theta) * f_{SEG}(\Theta$. To predict the category of a region, we first apply two fully-connected layers to masked feature f_{MSK}. As a result, we receive mask-based feature vector which is then concatenated with another box-based feature vector to build a joined feature vector. Notably, the box-based feature vector is computed by applying two fully-connected layers to ROI feature f_{ROI}. Finally, two fully-connected layers are attached to the joined feature and each gives class scores and

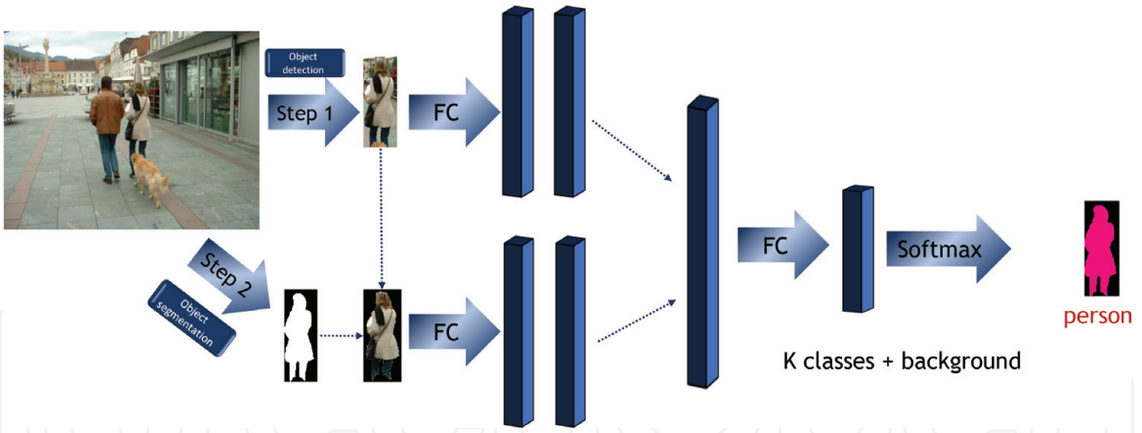

Figure 6.
An illustration of Stage 3 for object classification using $f_{ROI}(\Theta) * f_{SEG}(\Theta)$ as inputs.

refined bounding boxes using softmax classification of $(K+1)$ classes (including background). The entire procedure of Stage 3 is illustrated in **Figure 6**.

Let C is the network output of this stage and representing a list of category predictions for all instances: $C = \{C\}_i$. The loss term L_{CLS} of the third stage is expressed in Eq. (30).

$$L_{CLS} = L_{CLS}(C(\Theta)|B(\Theta), S(\Theta)) \tag{30}$$

The loss of entire proposed CRLS network is defined as in Eq. (32).

$$\begin{aligned} L(\Theta) &= L_{RPN} + L_{SEG} + L_{CLS} \\ &= L_{RPN}(B(\Theta)) + L_{SEG}(S(\Theta)|B(\Theta)) + L_{CLS}(C(\Theta)|B(\Theta), S(\Theta)) \end{aligned} \tag{31}$$

4. Inference

In our experiment, the top-scored 300 ROIs are first chosen by RPNs proposed boxes giving an image. Using Non-maximum suppression (NMS) with IoU and threshold is chosen as 0.7 to filter out highly overlapping and redundant candidates. The proposed RLS is then applied on to each ROI to partition it into object (foreground) and non-object (background). From each ROI, we obtain a segmenting mask and a category score prediction via two fully-connected layers followed by a soft-max layer.

Besides softmax loss, which is to optimize when classifying a region is object or non-object in the first stage, there are three "dependent" losses in which the later loss depends on the predecessor loss. Three losses are assigned for each ROI, i.e., (1) smooth l_1 loss—bounding box regression loss in the first state, (2) cross entropy loss —foreground segmentation loss in the second stage (3) softmax loss—instance classification loss in the third stage. Among all the losses, computing the gradient w. r.t. predicted box positions and address the dependency on the bounding box $B(\Theta)$ is the most difficulty. Given a full size feature map $\mathcal{H}(\Theta)$, we crop a bounding box region (predicted box) $b_i(\Theta) = (b_x, b_y, b_w, b_h)$ and warp it to a fixed size by interpolation. The wrapped region ROI is presented as:

$$\mathcal{F}_{ROI}(\Theta) = \mathcal{I}(b_i(\Theta))\mathcal{F}(\Theta) \tag{32}$$

where $\mathcal{I}$ is cropping and warping operations. As for dimension, $\mathcal{F}(\Theta) \in \mathrm{R}^{\mathrm{N}}$ is a vector reshaped from the image, $N = W \times H$. The cropping and warping matrix

$\mathcal{I} \in \mathrm{R}^M \times \mathrm{R}^N$. $\mathcal{F}_{ROI}(\Theta)$ is a target region sized $w \times h$ and $M = w \times h$. $\mathcal{I}(b_i(\Theta))$ presents transforming a that box $b_i(\Theta)$ from size of $b_w \times b_h$ into the size of $w \times h$. Let (x', y') and (x, y) are two points on target feature map $\mathcal{F}_{ROI}(\Theta)$ size of $w \times h$ and original map $\mathcal{F}(\Theta)$ size of $(W \times H)$, respectively. Using interpolation with (G) is the bilinear interpolation function, $\mathcal{I}(b_i(\Theta))$ is computed as follows:

$$\begin{aligned} \mathcal{I}(b_i(\Theta)) &= (G)\left(b_x + \frac{x'}{w} - x\right)(g)\left(b_y + \frac{y'}{h} - y\right) \\ g(z) &= \max(0, 1 - |z|) \end{aligned} \tag{33}$$

5. Implementation details

The proposed RLS approach is implemented using the TensorFlow system [17] whereas the extended CRLS semantic segmentation is implemented using Caffe environment [18]. Three functional sub-networks corresponding to three tasks, i.e., detection, segmentation and classification (shown in **Figure 5**), are connected together to construct an end-to-end network. The first and the third sub-networks, i.e., object detection and object classification, are adopted from Faster R-CNN framework [12]. In the second sub-network, we re-implement the defined layers from RLS-Tensorflow to RLS-Caffe for both forward and backward operations in Python. Since TensorFlow supports automatic differentiation capabilities, RLS-TensorFlow is therefore easier to implement than the new layers in RLS-Caffe.

In the experiment, the pre-trained VGG-16 model with 13 convolution layers is utilized to obtain the share features. Each convolution layer is always followed by a ReLU layer. There are 4 max-pooling layers are placed right after the convolution layer.

In the first task of CRLS, i.e., object detection, we use a 3×3 convolutional layer to reduce feature dimensions and learn the feature representation and then two consecutive 1×1 convolutional layers predicts object's locations and object's presenting scores. Furthermore, we choose the two normalization terms (N_{cls}, N_{reg}) are chosen as $N_{cls} = 256$ and $N_{reg} = 2400$. The balancing parameter λ is set as $\lambda = 10$. As for non-maximum suppression (NMS), which is used to reduce the number of boxes generated from the first stage ($\sim 10^4$ regressed boxes are produced from the first stage), the threshold of the Intersection-over-Union (IoU) ratio is chosen as 0.7. As a result, the top-ranked 300 boxes are kept for the second stage.

In the second task of CRLS, i.e., object segmentation by the proposed RLS, we first extract a fixed-size (21×21) deep feature from an arbitrary box predicted using the object detection task. The proposed RLS takes the extracted feature as the input together with the randomly initial ϕ_0 to generate a sequence input data x_t based on Eq. (18). The curve evolution procedure is performed via LS updating process given in Eq. (14). This task outputs a binary mask as given in Eq. (23) sized $m \times m$ and parameterized by an m^2 dimensional vector.

In the final task of CRLS, i.e., object classification, using the shared convolutional features inside the bounding box region, we extract a feature representation for each ROI. Through the second task of CRLS (object segmentation), we obtain the segmenting mask prediction for that ROI. The masked feature goes through two fully-connected layers to produce the classification score for that ROI.

The proposed CRLS framework was implemented on Caffe environment [18] and under SGD optimization. For each training image, if its shorter side is larger than 600, the image is down-scaled to 600 on the shorter side. To perform the

experiments on PASCAL VOC [19], we train the network with 32 and 8 k iterations at learning rates of 0.001–0.0001, respectively. On the larger dataset like MSCOCO [20], we train the network with 180 and 20 k iterations at learning rates of 0.001–0.0001, respectively.

6. Experimental results of CRLS

In this section, we conduct two experiments corresponding to the object segmentation by the proposed RLS method and the semantic instance segmentation by the proposed CRLS system on PASCAL VOC [19] and COCO [20] datasets.

6.1 Datasets

6.1.1 Synthetic dataset

The synthetic and medical dataset containing 720 images are artificially created from images reported in [10, 21]. There are actually about 20 images reported in [10, 21]. To avoid over-fitting problem as well as make sure the proposed RLS is able to deal with both intensity homogeneity and inhomogeneity, different kinds of degradation and affined have been used. As for degradation, we use many kinds of noise and blurring at different ratio. As for affine transformations, we use rotation, translation, scale, and flip at different degrees. As a result, we obtain 720 images on this dataset. Half of this data set is used for training, i.e., 360 images which were artificially generated from first 10 images. The rest of 360 images is used for testing. Furthermore, we use the one object dataset from Weizmann [22] for the conducting the experiment on natural images. Using the same strategy of augmentation as in medical images, we obtain 4700 images and we used 2350 images for training and 2350 images for testing.

6.1.2 PASCAL VOC

The PASCAL VOC 2012 [19] are a commonly used database for evaluating semantic segmentation. The PASCAL VOC 2012 training and validation set has 11,540 images containing 27,450 bounding box annotated objects and 6929 segmentations in 20 categories and one background category. The data is divided into 5717 images for training and 5823 images for testing (val). The 20 object classes that have been selected are: person, bird, cat, cow, dog, horse, sheep, aeroplane, bicycle, boat, bus, car, motorbike, train, bottle, chair, dining table, potted plant, sofa, and tv/monitor.

6.1.3 MS COCO

The MS COCO training set contains about 80,000 images for training and 40,000 images for validation [20]. This dataset consists of ~500,000 annotated objects of 80 classes and one background class. COCO is a more challenging dataset as it contains objects in a wide range of scales from small ($<32^2$) to large ($>96^2$) objects.

6.2 Experiment 1: object segmentation by RLS

This section compares our object segmentation RLS against Chan-Vese's LS [10], DRLSE [23], Li's method. [24], L2S [25] and a simple Neural Network with two fully-connected layers followed by a ReLU layer in between and we name it

F-ConNet. The experiments are validated on both synthetic images, medical images and natural images collected in the wild. The input images are resized to 64 × 64, thus, the number of units in fully-connected layers and hidden cell of GRU is 4096.

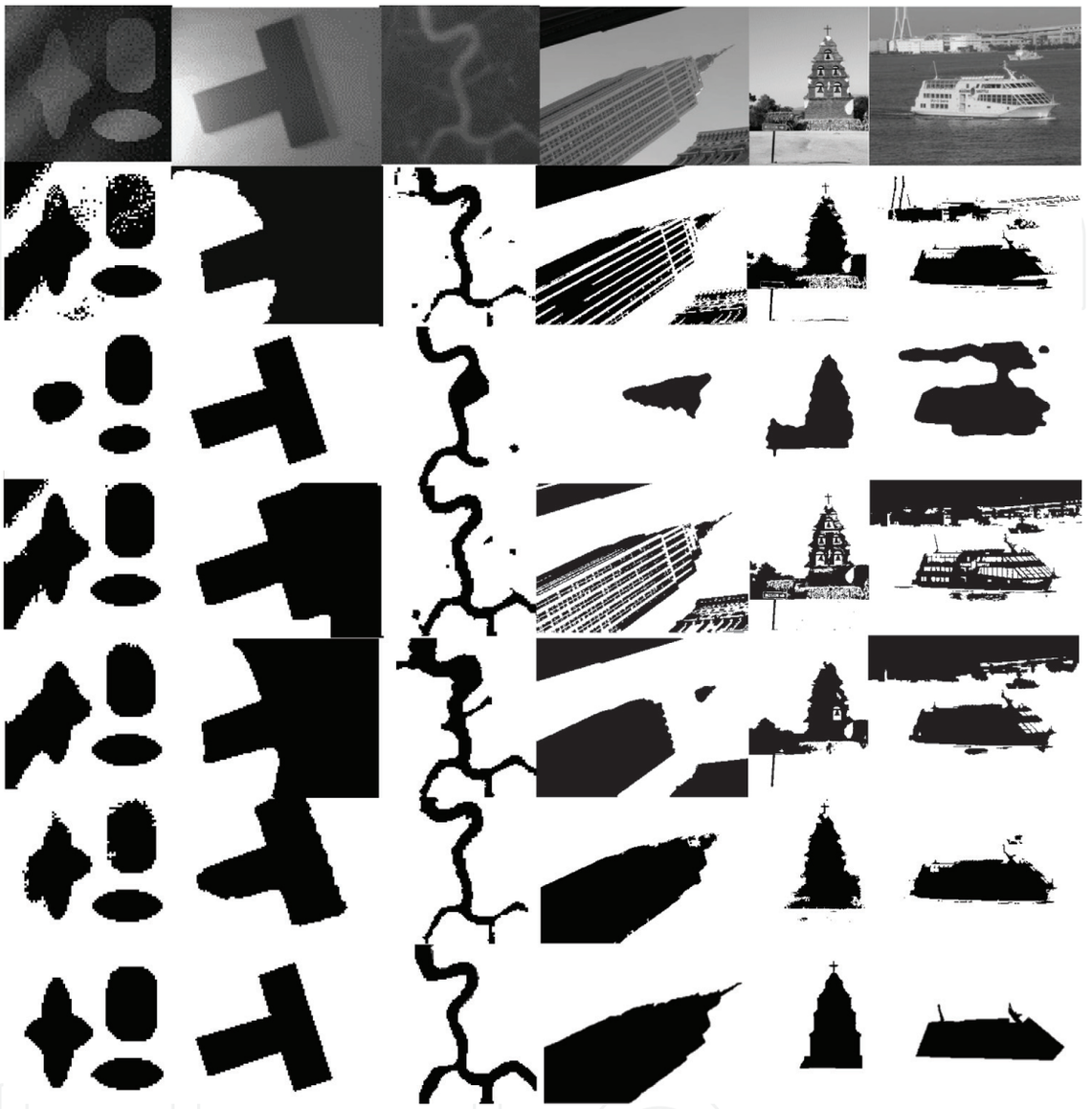

Figure 7.
Comparison between the proposed RLS against other LS-based methods: 1st row—input images, 2nd row—CLS [10], 3rd row—DRLSE [23], 4th row—Li et al. [24], 5th row—L2S [25], 6th row—our RLS, and 7th row—groundtruth. The best results for [10, 23–25].

Methods	FM (GT1)	FM (GT2)	Testing time
CV [10]	88.51	87.51	13.5(s)
DRLSE [23]	80.93	71.76	23.5(s)
Li et al. [24]	79.65	63.87	20.4(s)
L2S [25]	81.36	71.03	10.2(s)
F-ConNet	93.30	93.26	0.001(s)
RLS	99.16	99.17	0.008(s)

Table 2.
Average F-measure (FM) and testing time obtained by CV's model, DRLSE [23], Li et al. [24], L2S [25], F-ConNet and our proposed RLS with two different ground truth (GT1 and GT2) across Weizmann database.

Figure 7 shows comparison between our proposed RLS against other methods on image segmentation where CLS model [10] is used as baseline, DRLSE [23] as for global approach, Li et al. [24, 25], L2S [25] as for the inhomogeneity approach, F-ConNet as a baseline deep learning approach on the synthetic medical dataset and natural dataset. The best results from CLS's method, DRLSE, Li's method are L2S are given in the second, third, fourth, fifth rows respectively. The performance of our proposed RLS is given in the sixth row whereas the last row shows the ground truth. Quantity assessment on F-measure is given in **Table 2** with two separated

Methods	mAP^r @.5	mAP^r @.7	Time (s)
SDS (AlexNet)	49.7%	25.3%	48
Hypercolumn	60.0%	40.4%	>80
CFM	60.7%	39.6%	32
MNC	63.5%	41.5%	0.36
CRLS (Ours)	66.7%	44.6%	0.54

Table 3.
Quantitative results and comparisons against existing CNN-based semantic segmentation methods (SDS (AlexNet) [27], Hypercolumn [28], CFM [29], MNC [26]) on the PASCAL VOC 2012 validation set.

Figure 8.
Some examples of semantic segmentation on PASCAL VOC 2012 database. The input image (1st column), MNC [25] (2nd column), our semantic segmentation CRLS (3rd column), and the ground truth (4th column) (best viewed in color).

groundtruth versions (GT1 and GT2) provided by two different annotator on Weizmann database [22]. Compare to other methods, RLS achieves the best segmentation performance in this experiment on both groundtruth annotated by two different people.

In terms of time consuming, the baseline method CLS takes 13.5 s while Li et al.'s and DRLSE approaches have similar time consuming of 20.4 and 23.5 s on average to process one image with original size. L2S consumes less time (10.2 s) than the others CLS, Li's and DRLSE whereas the proposed RLS takes 0.008 s and F-ConNet takes least time consuming with 0.001 s.

Figure 9.
Some examples of semantic segmentation on MS COCO database on validation set. The input image (1st column), MNC [26] our semantic segmentation CRLS (2nd column) and the ground truth (3rd column) (best viewed in color).

Methods	mAP^r @[.5:.95]	mAP^r @.5
MNC	19.5%	39.7%
CRLS (Ours)	20.5%	40.1%

Table 4.
Quantitative results and comparisons against existing CNN-based semantic segmentation method MNC [26] on the MS COCO 2014 database.

6.3 Experiment 2: semantic instance segmentation by CRLS

We demonstrate our proposed CRLS approach on two common semantic segmentation dataset, i.e., PASCAL VOC 2012 and MS COCO. The end-to-end CRLS network is trained using the ImageNet pre-trained VGG-16 model. We follow the same protocols used in recent papers [26–29] to evaluate the semantic segmentation task. We also use the same metrics reported in recent semantic object segmentation papers [26–29].

On PASCAL VOC dataset, we evaluate mAP^r with IoU at 0.5 and 0.7. As shown in **Table 3**, we compare our proposed CRLS against state-of-the-art CNN-based semantic segmentation methods including SDS [27], Hypercolumn [28], CFM [29] and MNC [26]. As can be seen from the table, our CRLS achieves higher mAP^r (about 3%) at both 0.5 and 0.7 than previous methods using the same testing protocol. Not only higher segmentation accuracy, the experimental results also show that our proposed CRLS gives very better testing time (0.54 second per image). Some illustrations of multi-instance object segmentation by our proposed CRLS on PASCAL VOC 2012 dataset are shown in **Figures 8** and **9**.

FOn MS COCO 2014, we use 80 k images for training and 20 k images in the test set (*test-dev*) for evaluating. The performance is measured on (mAP^r) using IoU between 0.5 and 0.95 and mAP^r using IoU at 0.5 (as PASCAL VOC metrics). As shown in **Table 4**, our CRLS achieves better results than the previous method (MNC) on the COCO dataset.

Author details

Thi Hoang Ngan Le[1*], Khoa Luu[1], Marios Savvides[2], Kha Gia Quach[3] and Chi Nhan Duong[3]

1 Department of Computer Science and Computer Engineering, University of Arkansas, Fayetteville AR, USA

2 Department of Electrical Computer Engineering, Carnegie Mellon University, Pittsburgh, PA, USA

3 PdActive Inc, Denver CO, USA

*Address all correspondence to: thihoanl@andrew.cmu.edu

References

[1] LeCun Y, Touresky D, Hinton G, Sejnowski T. A theoretical framework for back-propagation. In: Proceedings of the 1988 Connectionist Models Summer School. CMU, Pittsburgh, PA: Morgan Kaufmann; 1988. pp. 21-28

[2] LeCun Y, Bottou L, Orr GB, Müller K-R. Efficient backprop. In: Neural Networks: Tricks of the Trade. Springer; 1998. pp. 9-50

[3] LeCun Y, Bottou L, Bengio Y, Haffner P. Gradient-based learning applied to document recognition. In: CVPR. 1998. pp. 2278-2324

[4] LeCun YL, Boser B, Denker JS, Howard RE, Habbard W, Jackel LD, Henderson D. Handwritten digit recognition with a back-propagation network. In: Nips. 1990. pp. 396-404

[5] Qian N. On the momentum term in gradient descent learning algorithms. Neural Networks. 1999;**12**(1):145-151

[6] Jordan MI. Attractor dynamics and parallelism in a connectionist sequential machine. In: Diederich J, editor. Artificial Neural Networks. 1990. pp. 112-127

[7] Rumelhart DE, Hinton GE, Williams RJ. Learning representations by back-propagating errors. In: Anderson JA, Rosenfeld E, editors. Neurocomputing: Foundations of Research. Cambridge, MA, USA: MIT Press; 1988. pp. 696-699

[8] Hochreiter S, Schmidhuber J. Long short-term memory. Neural Computation. 1997;**9**(8):1735-1780

[9] Cho K, van Merrienboer B, Gülçehre Ç, Bougares F, Schwenk H, Bengio Y. Learning phrase representations using RNN encoder-decoder for statistical machine translation. Computing Research Repository. 2014;**abs/1406**: 1078

[10] Chan TF, Vese LA. Active contours without edges. IEEE Transactions on Image Processing. 2001;**10**(2):266-277

[11] Mumford D, Shah J. Optimal approximation by piecewise smooth functions and associated Variational problems. Communications on Pure and Applied Mathematics. 1989;**42**(5): 577-685

[12] Ren S, He K, Girshick RB, Sun J. Faster R-CNN: Towards real-time object detection with region proposal networks. Computing Research Repository. 2015;**abs/1506**:01497

[13] Girshick R, Donahue J, Darrell JMT. Region-based convolutional networks for accurate object detection and semantic segmentation. Transactions on Pattern Analysis and Machine Intelligence. 2015;**38**:142-158

[14] Girshick R. Fast r-cnn. In: ICCV; 2015. pp. 1440-1448

[15] He K, Zhang X, Ren S, Sun J. Spatial pyramid pooling in deep convolutional networks for visual recognition. TPAMI. 2015;**37**(9):1904-1916

[16] Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv: 1409.1556. 2014

[17] Abadi M, Agarwal A, Barham P, Brevdo E, et al. TensorFlow: Large-scale machine learning on heterogeneous systems, software available from te nsorow.org. [Online]. 2015. Available from: http://tensorow.org/

[18] Jia Y, Shelhamer E, Donahue J, Karayev S, Long J, Girshick R, et al. Caffe: Convolutional architecture for fast feature embedding. In: ACM Intl. Conf. On Multimedia; 2014. pp. 675-678

[19] Everingham M, Van Gool L, Williams CKI, Winn J, Zisserman A. The pascal visual object classes (voc) challenge. International Journal of Computer Vision. 2010;**88**(2):303-338

[20] Lin T, Maire M, Belongie SJ, Bourdev LD, Girshick RB, Hays J, et al. Microsoft COCO: Common objects in context. Computing Research Repository. 2014;**abs/1405**:0312

[21] Li C, Kao CY, Gore JC, Ding Z. Minimization of region-scalable fitting energy for image segmentation. IEEE Transactions on Image Processing. 2008

[22] Alpert S, Galun M, Basri R, Brandt A. Image segmentation by probabilistic bottom-up aggregation and cue integration. In: CVPR; 2007

[23] Li C, Xu C, Gui C, Fox MD. Distance regularized level set evolution and its application to image segmentation. IEEE Transactions on Image Processing. 2010;**19**(12):3243-3254

[24] Li C, Huang R, Ding Z, Gatenby C, Metaxas DN, Gore JC. A level set method for image segmentation in the presence of intensity inhomogene ities with application to mri. IEEE Transactions on Image Processing. 2011; **20**(7):2007-2016

[25] Mukherjee S, Acton S. Region based segmentation in presence of intensity inhomogeneity using legendre polynomials. IEEE Signal Processing Letters. 2015;**22**(3):298-302

[26] Dai J, He K, Sun J. Instance-fully convolutional instance-aware semantic segmentation via multi-task network cascades

[27] Hariharan B, Arbeláez P, Girshick R, Malik J. Simultaneous detection and segmentation. In: European Conference on Computer Vision. Springer; 2014. pp. 297-312

[28] Hariharan B, Arbelaez P, Girshick R, Malik J. Hyper-columns for object segmentation and fine-grained localization. In: CVPR. 2015

[29] Dai J, He K, Sun J. Convolutional feature masking for joint object and stuff segmentation. In: CVPR; 2015. pp. 21