Variation and Geodesic Methods for Image Analysis **2018/2019**

Reading Report — March 26, 2019

Lecturer: Laurent Cohen *Student: Changqing Fu*

Abstract

This paper[**?**] improves Neural Network models for delineation problem (curve/linear structure detection, or crack detection, or road segmentation) by introducing a new loss term called *topology loss*, rather than simply use traditional binary cross-entropy loss. The advantage of this method is that sub-pixel topological structure is considered, or more precisely, inner convolutional layers (in a pre-trained VGG19 net[**?**]) or "feature maps" are analyzed in order to eliminate small connected components. U-net model[**?**] is applied to show its efficiency. In order to achieve state-of-the-art performance, an *iterative refinement framework* technique [**?**] is also used.

The aim of this report is to summarize the mathematical formulation and partially implement the algorithm on the same datasets (**EM**: Electron Microscopy:[1]), and to give several attempts to improve this method.

1.1 Highlights of Ideas

1. Q — *What is the problem dealt in the article?*

   A — This article discusses curvilinear detection. In a broader sense, it's an image segmentation problem, which generally aims at separating the image to several parts, but here we only focus on curve structures. Mathematically it's a classification problem, where the input space $\mathcal{X} = ([0,1]^{M\times N})^3$ is an RGB color image, each color components being normalized to $[0,1]$. The target space $\mathcal{Y} = [0,1]^{M\times N}$ where (M,N) is the image size. Each value in the output 2D matrix represents the probability that each pixel belongs to the curve. We might also threshold the probability with 0.5 to ge a final result as logical (true/false) values (usually this is unecessary).

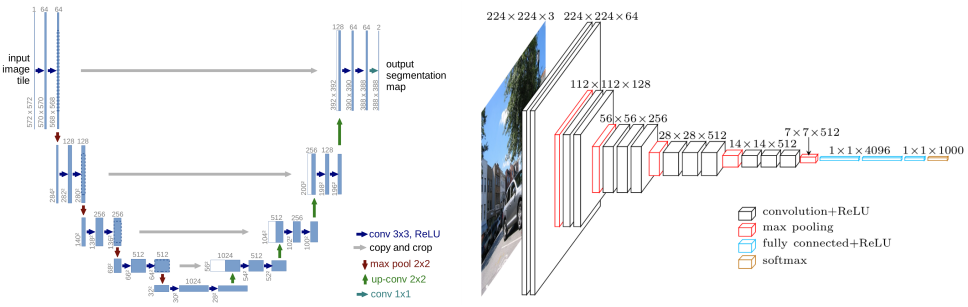

Figure 1.1. U-net and VGG Network Structures

2. Q — *What are the equations and numerical methods? Is it possible to give a better mathematical formulation to the problem?*

A — Two Deep Convolutional Neural Network models (Figure 1.1). The primary one is the U-net, which conducts the segmentation task. U-net is an hourglass network whose structure is similar to an auto-encoder. Standard Convolution Neural Network, pooling and upsampling routine will not be described in detail here, and we note that the "Concatenate" Operation in the U-net means copying-and-pasting the weights in some layers as additional input before applying next layer (avoiding "gradient vanishing/explosion", this skipping trick can be compared to ResNet: the network is now a directed graph instead of a well-ordered chain). The other is VGG19, which is used to help refine the model by extracting non-curvature structure as an extra punishing term. Only top layers are used in the refinement process. There is no PDE in the primary model, but we will discuss a link in the sequel (equation (1.1)).

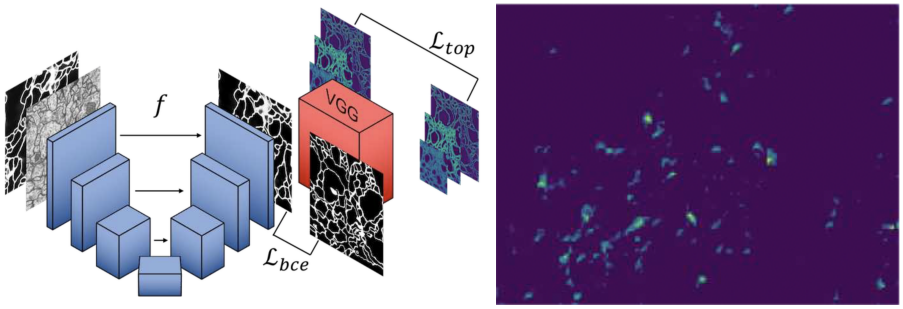

Figure 1.2. Left: Topology Loss. Right: Noise detected with Topology Loss.

Previously and commonly used **Binary Cross Entropy Loss (BCE)**:

$$\mathcal{L}_{\text{bce}} = \sum_{i=1}^{N} y_i \log f_i(x, \mathbf{w}) + (1 - y_i) \log(1 - f_i(x, \mathbf{w}))$$

The new **Topology Loss (TOP)**: M_n being the number of channels each feature map. W_n, H_n being height and width of the feature maps respectively

$$\mathcal{L}_{\text{top}} = -\sum_{n=1}^{N} \frac{1}{M_n W_n H_n} \sum_{m=1}^{M_n} \|\ell_n^m(y) - \ell_n^m(f(x, \mathbf{w}))\|_2^2$$

With these definitions, the final **Total Loss** is a combination of the two with scale parameter μ:

$$\mathcal{L}(x, y, \mathbf{w}) = \mathcal{L}_{\text{bce}}(x, y, \mathbf{w}) + \mu \mathcal{L}_{\text{top}}(x, y, \mathbf{w})$$

[1] https://imagej.net/Segmentation_of_neuronal_structures_in_EM_stacks_challenge_-_ISBI_2012

3. Q — *Can you locate this article over studied methods in the course, and compare to close subjects presented in the course.*

A — Traditional linear structure algorithms have been proven to be successful; such as Active Contour via Level Sets, or Fast Marching, can be taken as analogies. One of the advantages of these methods is that they are unsupervised. Here is a table for an incomplete comparison in brief:

Method	**Explanation**	**Comments**
Edge Detectors	1. Thresholding Gradient; 2. Zero crossing of the Laplacian; 3. Hessian× Gradient; 4. Convolution (e.g. Sobel, Prewitt)	1 to 3 requires pre-blurring.
Active Contour and Level Set	Minimizing Energy Functionals on curves, or image force plus regularization (can be solved with PDE)	Initialization position is required. Only for simple curve.
Fast Marching	Dynamic Programming, evolution from minimum energy along the image structure.	Can be adapted to line structure detection.

Apart from these comparisons, there is another interesting connection to the course in "discussion" section.

4. Q — *What is the originality of the work (according to the authors)?*

A — The topology loss.

5. Q — *What are the important new results arising therefrom.*

A — This new loss can be heuristic to other pixel-wise image tasks in supervised setting when small sub-pixel structures are crucial. For instance, as a generalization, other segmentation problems include foreground/background detection (also binary for each pixel), object localization (binary or multi-class), etc. In multi-class setting, we use a target space $\mathcal{Y} = ([0,1]^{M\times N})^K$ instead, where at each pixel, $[0,1]^K$ represents the *one-hot encoding*, *i.e.* probability that the pixel belongs to each class. The rich structure of U-net makes it possible to succeed not only in our problem, but also in other sub-cases in a supervised setting according to different training data.

6. Q — *Do you see weaknesses in the approach presented and do you have ideas to cope with them?*

A — Several criticisms are formulated, with attempts to improve them.

- **Data inconsistency**. Detection is trained on a fixed dataset, ImageNet, which is totally irrelevant to the original dataset. Thus generality became questionable. Once taken from Image Net, wights for the Topology Loss is unchanged, and information from the dataset is not exploited.

  Solution: We propose to use an *adaptive topology loss (AdaTOP)*, in which weights for topology loss in VGG could be updated while training. Back-propagation enables to implement this easily.

- **Well-definedness of Topology Loss**. There is a hidden selection phase of the algorithm that is not mentioned in the article: NOT all the inner layers of the VGG net work for small component detection! We say it's hidden because the best feature maps, or certain channels of convolution layers are selected manually by looking into inner layers (64, 128, 256 layers respectively), another hyperparameter μ is tuned to 0.1 manually as well. However, due to the black-box nature of deep neural nets, we need further selection criteria to find the right channel(s). Worse still, random initialization and symmetry of the layers make the weights and orders of VGG layers different each time we re-train the model. Without downloading the pre-trained weights from the official VGG implementation, the result of the article is, strictly speaking, irreproducible, due to these random and human-oriented characteristics.

  Solution 1: We can use other small component detection methods as an alternative to get rid of this routine. A simple example could be applying *level set methods* on the prediction result. Since this changes the topology of the image, it automatically eliminates small components. This option completely keeps us away from the VGG net, and build connections with the topology loss for neural network to methods in the course, but hyper-parameter tuning is still required to perform geodesic marching.

  Solution 2: Brute force may be used alternatively. Since hyper-parameter tuning is costly and is non-convex, we can only tune μ by trying different values/with cross-validation, and try out all the channels of each layer to look for the correct one that is sensitive to small components. This could be done either by observing each channel or by minimizing the loss. With abundant computing power, this is still feasible (see implementation section for a rough time benchmarking).

- **Complexity/Scalability**. Unlike Dynamic Programming algorithms like *fast marching*, Neural Networks demand huge computing power. However, since this is only for offline training phase, we don't regard this as an obstacle. Please refer to implementation section to get a quantitative sense.

- **Algorithmic Imperfection**. Overcommitment: Excessive elimination of small cracks. For instance, in the figure below (taken from [**?**]), noise in the green circle is correctly detected but not those in the red circle. This is a systematic error that cannot be avoided, due to the fact that human-labeled data cannot be precise and that quality of image has limitation.

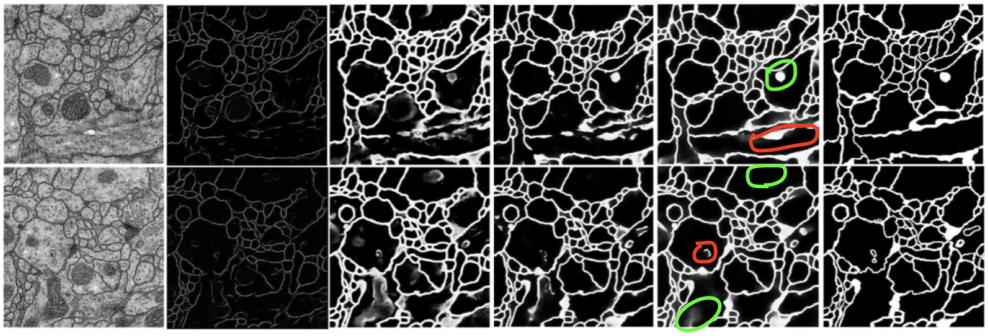

Figure 7: **EM**. From left to right: image, **Reg-AC**, **U-Net**, **OURS-NoRef** and **OURS** prediction, ground-truth.

1.2 Discussion

Here is a complementary motivation of the article from my understanding. The proposed method can be seen as a "data augmentation" trick. On one hand, in order to extract excessive small structures in the output, the VGG19 network is pre-trained on ImageNet dataset to get the desired weights to compute topology loss. That is to say, from an information theory perspective, we have conplementary data to improve the model. On the other hand, if we consider a "pure" topology loss model (in the case where μ is very large), with only one feature map, the model reduces to a singe "hourglass + downsample a bit" model but trained separately – the second part is trained with another dataset and then fixed as constant (see above for an idea to improve). Training data of this model is no longer the same. In short, we have additional transformed data.

Another point of view could be *Transfer Learning*. This algorithm use common knowledge (ImageNet classification knowledge) to aid the prediction job on curve detection datasets. Inspired by bionics, this method is being actively researched in NLP area, etc.

There is a link to our course on Geodesic Methods: During the (nonconvex) optimization of the deep network, we can adapt to the methods in the course to smooth the energy using a Gaussian Kernel G_γ (Baldassi et al., '15, '16 [**?**]):

$$w^* = \text{argmin}_w f(w) = \text{argmin}_w - \log(e^{-f(w)}) \simeq \text{argmin}_w - \log(G_\gamma * e^{-f(w)})$$

With this surrogate, our new optimization problem is written explicitly as:

$$w^* = \text{argmin}_w f_\gamma(w) = \text{argmin}_w - \log \int_{w'} \exp\Big(- f(w') - \frac{1}{2\gamma}\|w - w'\|^2 \mathrm{d}w'\Big))$$

This is also referred to as "local entropy". An PDE understanding could be given by a Hamilton-Jacobi equation, or with SDE (Stochastic Control):

$$(PDE) \begin{cases} u_t = -\frac{1}{2}|\nabla u|^2 + \frac{1}{2}\Delta u \\ u(w,0) = f(w) \end{cases} \quad (SDE) \begin{cases} \mathrm{d}w = -\alpha(w,s)\mathrm{d}s + \mathrm{d}B(s) \quad , t \le s \le T \\ w(t) = w \end{cases} \tag{1.1}$$

where $\alpha(w,t) = \nabla u(w,t)$. The link between u and w is that $u(w,t) = \min_\alpha \mathbb{E}[f(w(T)) + \frac{1}{2}\int_t^T \|\alpha(s)\|^2 \mathrm{d}s]$ (quadratic loss for Gradient Descent). Furthermore, if we use the vanishing viscosity method (let the diffusion part in the PDE go to zero), we obtain instead a solution of the form (known as Hopf-Lax formula):

$$u(w,t) = \inf_{w'} f(w') + \frac{1}{2w}\|w - w'\|^2$$

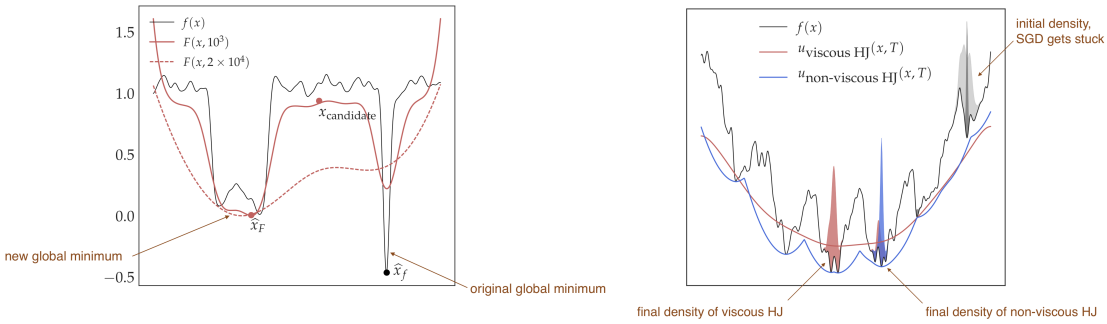

Figure 1.3. Left: smoothing with local entropy. Right: viscosity solution

1.3 Implementation

In our numerical test for U-net[2], we used an image size of 224×224 (for the purpose of reusability in the VGG model), which is slightly smaller of the proposed network at each layer, but this does not change the network size, since the number of parameters in convolution layers does not depend on image size. We used Adam as optimizer with learning rate 10^{-4} following the parameters in the paper. We also extracted the layers specified in the article *i.e.* **relu(conv1_2), relu(conv2_2) and relu(conv3_4)** so that we don't need to select them via searching. Iterative refinement framework[**?**] is not implemented since it's not the main idea of this article.

For our implementation, which is a medium-size problem (3M parameters), a 4-core Intel i5 CPU + 16G memory + 16 G Tesla P100 GPU machine is used (rented from Google Cloud Platform). We spent 20 minutes to optimize 10000 iterations. On my personal laptop (2.3 GHz Intel Core i5 + 8G memory without NVIDIA GPU support), the same process takes a couple of days to complete (approximately 30 minutes per 200 iterations).

[2]code can be found here: `https://github.com/EvergreenTree/mosinska2018beyond`

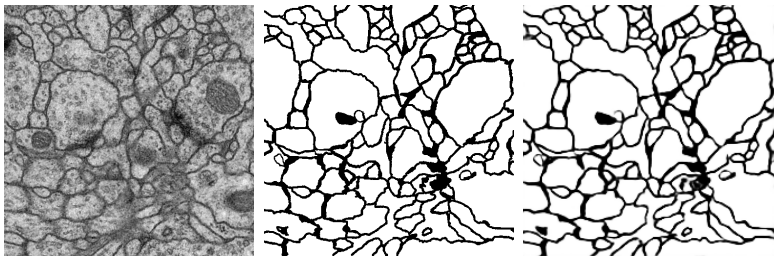

Figure 1.4. Left: Input. Middle: Ground truth. Right: Output