1. Details regarding Point cloud Transformer layer proposed in [2]

Here, we provide additional low-level details relevant to our point cloud learning experiments.

Let Q, K, V denote the query, key and value matrices respectively, generated by linear transformations of the input features $F_{in} \in R^{N \times d_e}$ as follows:

$$(Q, K, V) = F_{in} \cdot (W_q, W_k, W_v) \tag{1}$$

$$Q, K \in R^{N \times d_a}, \quad V \in R^{N \times d_e} \tag{2}$$

$$W_q, W_k \in R^{d_e \times d_a}, \quad W_v \in R^{d_e \times d_e} \tag{3}$$

where W_q, W_k and W_v are the shared learnable linear transformations, and d_a is the dimension of the query and key vectors. Note that d_a may not be equal to d_e.

Next, we calculate the attention weights:

$$\tilde{A} = (\tilde{\alpha})_{i,j} = Q \times K^T \tag{4}$$

The weights are then normalized to give $A = (\alpha)_{i,j}$:

$$\bar{\alpha}_{i,j} = \frac{\tilde{\alpha}_{i,j}}{\sqrt(d_a)} \tag{5}$$

$$\alpha_{i,j} = \text{softmax}(\bar{\alpha}_{i,j}) = \frac{\exp(\bar{\alpha}_{i,j})}{\sum_k \exp(\bar{\alpha}_{i,k})} \tag{6}$$

The self-attention output $F_s a$ is computed as:

$$F_{sa} = A \cdot V \tag{7}$$

In [2], they design an offset-attention which is used on top of F_{sa}. After computing F_{sa}, the final output of the transformer layer is specified as

$$F_{out} = \text{FF}(F_{in} - F_{sa}) + F_{in} \tag{8}$$

where FF is the feed-forward layer comprised of the linear layer, normalization layer and nonlinear layer.

2. Details regarding GNNs for few-shot learning proposed in [3]

Here, we provide additional low-level details relevant to our few-shot learning experiments.

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E}; \mathcal{T})$ be the graph constructed with samples from the task $\mathcal{T}$, where $\mathcal{V} = \{V_i\}_{i=1,\ldots,|\mathcal{T}|}$ and $\mathcal{E} := \{E_{ij}\}_{i,j=1,\ldots,|\mathcal{T}|}$ denote the set of nodes and edges of the graph, respectively. Let v_i and e_{ij} be the node feature of V_i and the edge feature of E_{ij}, respectively. We take $|\mathcal{T}| = N \times K + T$ to be the total number of samples in the task $\mathcal{T}$. Each ground-truth edge-label y_{ij} is defined by the ground-truth node labels as $y_{ij} = 1$ if $y_i = y_j$; 0 otherwise.

In the proposed GNN in [3], in the experiments, node feature update and edge feature update are done iteratively for L layers. The node features are initialized by the output of the convolutional embedding network and the edge features represent the strengths of the intra- and inter-class relations between two connected nodes. We treat the node feature update function as f and use our TMDlayer on top of it. In [3], the proposed node feature update function is:

$$v_i^l = g_v^l([\sum_j \tilde{e}_{ij1}^{l-1} v_j^{l-1} || \sum_j \tilde{e}_{ij2}^{l-1} v^{l-1} v_j]; \theta_v^j) \tag{9}$$

where each edge feature $e_{ij} = \{e_{ijd}\}_{d=1}^2$ is a 2-dimensional vector $\tilde{e}_{ijd} = \frac{e_{ijd}}{\sum_k e_{ikd}}$, and g_d^l is the feature transformation network with the parameter set θ_v^l.

3. Details regarding deep active contour model

Here, we provide additional details relevant to our segmentation experiments with our proposed deep active contour model. In our segmentation experiments, we treat the entire update module introduced here as f and use our TMDlayer on top of it.

Overview. Given an input image I, our model learns to perform weighted length minimization and learns feature-value sets in the image region, which are then used to perform T steps of a level set update from the initial configuration ϕ_0. Our framework also enables a semi-supervised segmentation once we add a modified variational energy term which our level set updates will explicitly optimize as a loss function (for the unlabeled images). The level set update takes place within the network, which seamlessly enables end-to-end training of our whole network.

3.1. Weighted Length Minimization

In active contour models, a length minimization term is used to yield curvature-based evolution that is responsible for a smooth curve representation of the desired object boundary. More precisely, one uses the arc length minimization [1]:

$$\inf_C \int_0^1 |C'(s)| ds \tag{10}$$

However, minimizing the curve length alone will not suffice as a means to segment object boundaries. A weighted curve length is more meaningful in this context, especially if the weighting function is derived from image data and sensitive to edges in the image. Thus, Caselles et al. proposed geodesic active contour (GAC), which can be viewed as a weighted length minimization written as,

$$\inf_C \int_0^1 g(C(s)) ds \tag{11}$$

where $g(C(s)) := \frac{1}{1+|\nabla G_\sigma * I|^2}$, G_σ is a Gaussian with a variance parameter σ. Note that this choice of weighting function is hand-crafted like so many others that have

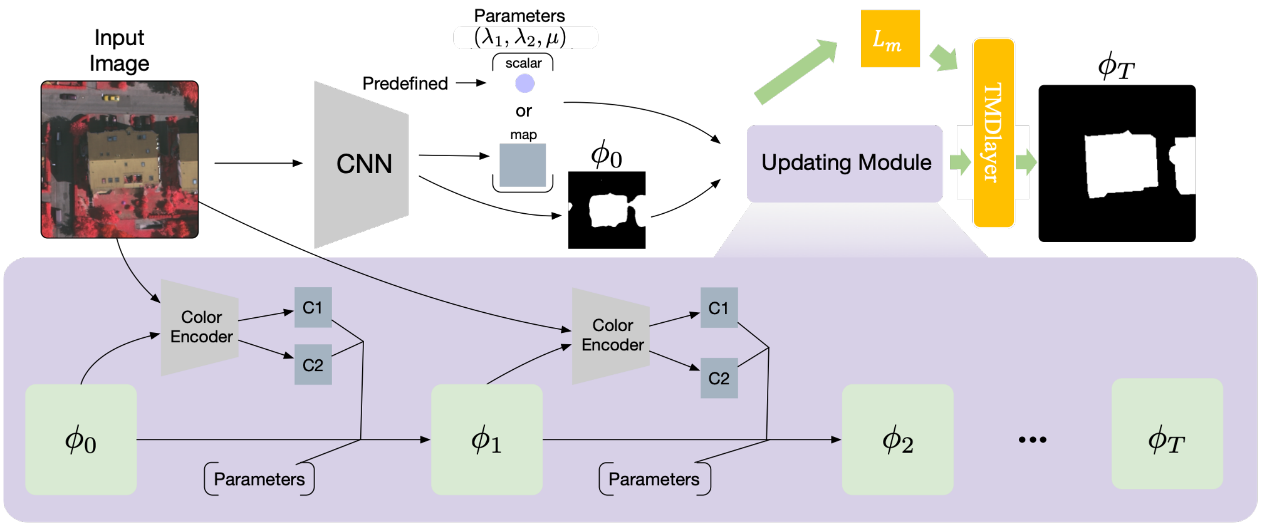

Figure 1: **Architecture of our model:** The image is fed into the CNN encoder to produce initial level set function and needed parameters. The parameters could be chosen to be a learned map or simply a constant. The updating module will evolve the level set function using initial ϕ_0 and the parameters, producing the final ϕ_T. Within the updating module, in every step the color encoder learns $C1, C2$ from the image and the current ϕ_t. The images of ϕ_0 and ϕ_T are binarized for a clear view. Our TMDlayer is added to every layer in the updating module. Here we only show once for clear view.

been used in literature. Indeed, such hand-crafted weighting functions in general can be the right choice for a small class of images and tasks but is often suboptimal in general. To overcome this limitation, we replace the image gradient based weighting function $g(C(s))$ with a term parameterized by deep neural networks thereby allowing the weighting function to be data driven as follows:

$$\inf_C \int_0^1 \mu(C(s))ds \quad \text{where} \quad \mu(\cdot) = CNN(I) \tag{12}$$

where $\mu(\cdot)$, which can be viewed as a 2-D function in the discrete case, is parameterized using a convolutional neural network. As we see in our experiments, this leads to better segmentation results.

In a level set framework, this weighted length minimization is usually performed by solving the strong form given by the following PDE:

$$\left[\frac{\partial \phi}{\partial t}\right]_{\text{length}} = \text{div}\left(\mu \frac{\nabla \phi}{|\nabla \phi|}\right) |\nabla \phi| \tag{13}$$

3.2. Chan-Vese in Learned Feature (Latent) Space

We now develop a Chan-Vese based active contour energy term that is set in a latent space i.e., a feature space where the features are learned by a deep neural network (DNN). To make the description simple, we will derive the loss function for a single image I (or X in the main paper), and so in this section (x, y) represents 2D pixel coordinates in X. As usual, the total loss function is given by summing over the examples in the training set. Let us first recall the classical Chan-Vese active contour energy functional [1] given by,

$$F(c_1, c_2, C) = \lambda_1 \int_{in(C)} |I(x,y) - c_1|^2 \, dxdy + \tag{14}$$

$$\lambda_2 \int_{out(C)} |I(x,y) - c_2|^2 \, dxdy \tag{15}$$

In (14), c_1 is the color averaged over pixels inside the contour C and c_2 is the color averaged over the pixels outside.

$$c_1 = \frac{\int \mathbf{1}_{\phi>0} \cdot \phi dxdy}{\int \mathbf{1}_{\phi>0} dxdy}, \; c_2 = \frac{\int \mathbf{1}_{\phi<0} \cdot \phi dxdy}{\int \mathbf{1}_{\phi<0} dxdy} \tag{16}$$

This assumption has been shown to work well for simple images, but for most natural images that we may want to segment, this assumption may be too restrictive. In this paper, we replace the weighting factors $(I(x, y) - c_i)^2$ in the Chan-Vese energy with an energy in the learned feature space obtained using a deep network. The intuition here is that rather than perform variance minimization in the native space, we can perform this operation in a learned feature space which better captures the unknown homogeneity property of each region within the image. Note that the transformation from image space to the feature space can be achieved by simply modifying original scalar λ into a spatially variant map. Thus, the final energy functional of our active contour is given by,

$$F(c_1, c_2, C) = \int_{in(C)} \lambda_1(x,y) \cdot |I(x,y) - c_1|^2 \, dxdy + \int_{out(C)} \lambda_2(x,y) \cdot |I(x,y) - c_2|^2 \, dxdy \tag{17}$$

Another observation one can make is that if we restrict $\lambda(\cdot)$ to learn a nonnegative function, it becomes similar to the localized kernel weighting function. Thus, the functional based on the image data denoted by $[F(\phi)]_{C_1}$ becomes:

$$\underbrace{\sum_{i=1,\ldots,n} \int_{\Omega} \lambda_{1i}(x,y) \cdot (|I(x,y) - C_{1i}|^2) H_\epsilon(\phi) dxdy}_{[F(\phi)]_{C_1}} \tag{18}$$

where C_{1i} refers to the ith channel of C_1 (e.g., there are 3 channels in a color image) and we follow [1] to define a Heaviside function H_ϵ which acts as an differentiable approximation to the indicator function $\mathbf{1}$ defined as,

$$H_\epsilon(\cdot) := \frac{1}{2}\left(1 + \frac{2}{\pi} \arctan\left(\frac{\cdot}{\epsilon}\right)\right), \tag{19}$$

where ϵ is a specified constant (hyperparameter).

After deriving the corresponding Euler-Lagrange equation of (18) and parameterizing the descent direction with an artificial time parameter $t \geq 0$, we obtain,

$$\left[\frac{\partial \phi}{\partial t}\right]_{C1} = -\delta_\epsilon(\phi) \left[\sum_{i=1,\ldots,n} \lambda_{1i}(x,y) \cdot (|I(x,y) - C_{1i}|^2)\right] \tag{20}$$

Similarly, we can have an energy and corresponding evolution equation for the region outside outside the active contour,

$$\left[\frac{\partial \phi}{\partial t}\right]_{C2} = \delta_\epsilon(\phi) \left[\sum_{i=1,\ldots,n} \lambda_{2i}(x,y) \cdot (|I(x,y) - C_{2i}|^2)\right] \tag{21}$$

where we follow [1] to define $\delta_\epsilon = H'_\epsilon$.

3.3. Initial Level Set Function

Active contour models usually need manual initialization, which often influences the quality of the final result. This makes it impractical in large-scale applications. We propose to learn the initial level set function by using a segmentation neural network. That is,

$$\phi_0 = f(I) \tag{22}$$

where I is the given image and f denotes an encoder CNN. We see in our experiments that despite the "multi-stage" nature of this setup (CNN followed by level set updates), it can indeed be trained end-to-end.

3.4. The Full Evolution Model

The full ('total') model evolution is simply the sum of all the terms above:

$$\left[\frac{\partial \phi}{\partial t}\right]_{\text{total}} = \left[\frac{\partial \phi}{\partial t}\right]_{C1} + \left[\frac{\partial \phi}{\partial t}\right]_{C2} + \left[\frac{\partial \phi}{\partial t}\right]_{\text{length}} \tag{23}$$

3.5. Training Procedure

For training, we first learn an initial level set function ϕ_0 from an encoder CNN shown in (22). Then, we perform T update steps according to (23). The final update after ϕ_T steps will be used to calculate the loss based on the ground truth segmentation mask label M_{GT}

$$Loss = CE(H_\epsilon(\phi_T), M_{\text{GT}}) \tag{24}$$

where CE refers to the standard cross-entropy loss.

References

[1] Tony F Chan and Luminita A Vese. Active contours without edges. *IEEE Transactions on image processing*, 10(2):266–277, 2001.

[2] Meng-Hao Guo, Jun-Xiong Cai, Zheng-Ning Liu, Tai-Jiang Mu, Ralph R Martin, and Shi-Min Hu. Pct: Point cloud transformer. *arXiv preprint arXiv:2012.09688*, 2020.

[3] Jongmin Kim, Taesup Kim, Sungwoong Kim, and Chang D Yoo. Edge-labeling graph neural network for few-shot learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11–20, 2019.