Hybrid Graph U-Net for Image Segmentation

Cheng Qiu

September 18, 2024

Abstract

U-Net is designed to capture fine-grained details such as edges and patterns as well as macroscopic details. At the top level of the U-shaped architecture, the input are describing what seems to be closely related to pixels such as edges and pixel intensity. However, at lower levels of the network, the input is processed to resemble more of a feature map that describes the features of an object in the image. For most variants of the U Nets[4][5], the architecture propagates rectangular operation throughout the whole network. This may not be the most accurate approach to the segmentation of an image because, at the lower levels, the inputs are no longer pixel images but rather object descriptors. Therefore, this work propose a non-rectangular operations to better capture details.

1 Introduction

U-Nets are a very common architecture for convolutional neural networks and it's also a common image-to-image architecture, see e.g., the de-noising diffusion papers. Due to its popularity, there are many variants, but in general, there's only one blueprint for U-Net construction: convolutions followed by down-sampling/up-sampling, and between modules of corresponding scale, skip layers. The original paper oriented these networks toward medical image segmentation, attempting to capture the so-called "scale-space" intuition of vision. There is coarse-grained semantics in images (object locations, macro-relationships), medium-grained details (object shapes/boundaries, spatial orientations), and fine-grained textures (object surface details, image intensity variations).

U-Nets operates on regular rectangular images and images are usually represented in a rectangular grid of pixels. Alternatively, images can be represented as lattice graphs with sparsely connected nodes analogous to pixels. For U-Nets to segment objects in images, it requires both higher-level object information and lower-level local shading/texture/intensity information. At the top end of the U-Net, the network attempts to capture fine-grained details; local edges, textures, and patterns. Towards the bottleneck of the architecture, consensus intuition is that the network will learn macroscopic features. At these top levels, the representation of the input is closely related to rectangular pixel images, but toward the bottom, the representation is more closely related feature map that resembles an object. Despite the representational differences at different layers of the network, the U-Net employs convolution and pooling operations that process the representation in rectangular patches.

2 Related Works

Due to UNET's outstanding performance in image segmentation, researchers have explored new variations of the UNET to improve upon its architecture to attain higher accuracy. One of such is the NNU-NET which increased the overall accuracy of the UNET architecture by introducing minor modifications such as replacing the ReLU activation function with leaky ReLUs[5]. Other variations such as MultiResUNET included more drastic alternations to the architecture like the residual paths that apply convolution to the skips connection instead of the conventional skip connections to facilitate more effective learning[4].

Moreover,recent works have also adapted UNET to perform learning on unstructured data. However, one major setback is that UNET does not directly support data other than rectangular images. In graph representation learning, Gao et al. proposed the gPool and gUnpool for fully-connected graphs which computes the ranks of the nodes and selects a subset of the highest ranked nodes to create a new graph while

gUnpool reverses the pooling operation by keep track of the node used in pooling [2]. For 3D object representations like meshes, works such as [3] provided alternative methods applicable to mesh data to perform downsampling and upsampling by pooling non-uniformly edges according to the strength of the edge feature while unpooling reinstates the corresponding mesh connectivity before pooling.

3 Proposed Architecture

3.1 Representation of Image: Mesh

As mentioned in the introduction, the image can be represented in a graph. In this project, we are representing each image as a triangular mesh. Each pixel in the image corresponds to each vertex in the graph. Given a triangle mesh represented in a graph $G = (V, E)$, there exist edges between two vertices v_i and v_j if they are immediate neighbors. Since we are considering a simple triangle mesh, we are only considering one diagonal, therefore the other diagonal is ignored. See figure 1 for a graphical representation.

One of the issues that I've encountered using the triangular mesh is that operations on the triangular mesh, especially if the operation deals with boundary edges, tend to create non-manifold (non-triangular) faces. That is a problem because it creates instability within the mesh and makes everything more messy. One of the workarounds that I'm currently using is to pad the mesh with a layer of dummy boundary vertices so that operations can't be performed. I'm not exactly sure if there is a more elegant solution to the non-manifold issue.

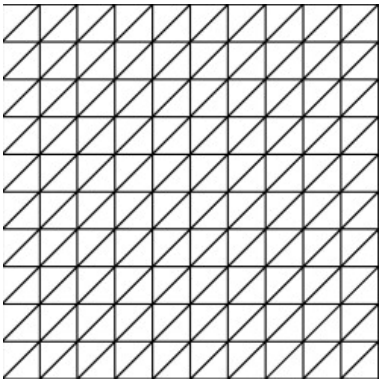


Figure 1: The graphical mesh representation of a rectangular image

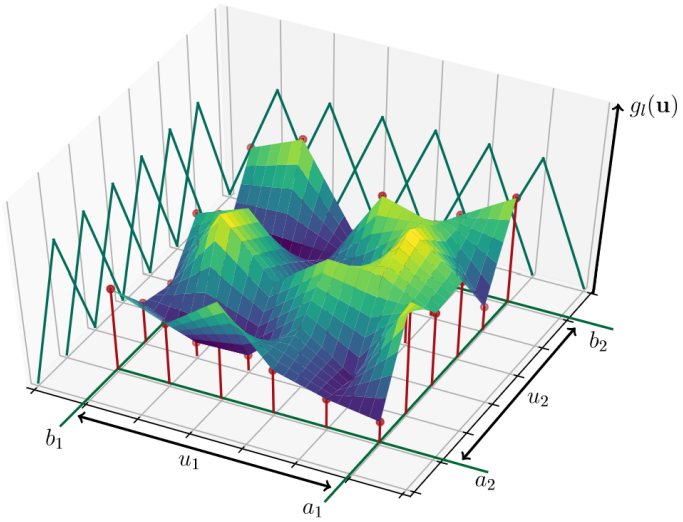
3.2 Convolution

As for convolutions, there are multiple options for convolution that I would like to experiment with just to see the results. There are the Graph Convolution for Node classification task and Spline Convolution (splineconv) [1]. On a high level, the splineconv aggregates feature in a local neighborhood based on some pseudo-coordinates U using a trainable b-spline kernel.

Given an input graph, $\mathcal{G} = (\mathcal{V}, \mathcal{E}, U)$ with V being the vertices, E being the set of directed edges and U being $U \in [0, 1]^d$ where d is the dimension of the coordinate (in our case is going to be 2D). Let $f(i)$ denote a vector of M_{in} input features for each node $i \in V$ and $N(i)$ to be the neighborhood of vertex i. Since the objective of this function is to process a feature input f of M_{in} to M_{out} output features, we can define the spline convolution as the following where f is the feature and g is the kernel function:

$$(f * g)(i) = \sum_{l=1}^{M_{out}} \sum_{j \in N(i)} f_l(j) \cdot g_l(u(i, j)) \tag{1}$$

To put it into my own words (checking my own understanding), equation1 and figure 2, what spline convolution is actually doing is that it is aggregating the product of all the neighbors of vertex i with the



(a) Linear B-spline basis functions

Figure 2: A linear spline with basis of degree of 1 of dimensionality of 2. The height of the red dots is the trainable parameter.

kernel value associated with the vertex i over all of the kernel filters. The kernel value is then influenced by the pseudo-coordinate of $u(i, j)$ as well as the trainable parameters in the basis function.

3.3 Pooling/Unpooling

So far, I've explored a few pooling/unpooling schemes, specifically gPool/gUnpool proposed in [2] as well as mesh pool and mesh unpool proposed in [3]. The one issue that we had with gPool/gUnpool is that this pooling scheme works for node classification tasks since the graph is fully connected. However, for sparsely connected graphs like an image, the pooling operation will likely result in undesirable alternations to the relationship between vertices (e.g. pooling vertices that may result in disconnecting the sparse graph). Similarly, for gUnpool, we do not have information on how to restore the original connectivity. Therefore, this approach seems to be invalid for our specific task.

Another approach that we've looked at and currently are trying to implement is the mesh pool and mesh unpool. As discussed in Section 2, pooling in mesh might result in non-manifolds, and dealing with non-manifolds can be complicated. Therefore, there are a few constraints that are put into place to prevent such a scenario from happening. The following conditions have to be met for two vertices and the edge that connects to be merged together:

1. The edge does not contain any dummy vertices (meaning the padding)/boundary vertices
2. There must exist exactly two overlapping neighbors between the two vertices(that are not the two vertices) (I'm not sure if this condition is needed since after adding the padding, the graph will maintain its regular triangle shape no matter how it's pooled)

During the merging of vertices, the neighbors of the neighbors of the merged vertex will be updated to include all the neighbors of the two vertices. In addition, mesh pooling also maintains a record of the order of merge and the connectivity of the graph prior to pooling. With this information, unpooling simply becomes reinstating the connectivity before the pooling and copying the values over to the pooled vertices. This operation is shown in Figure 3

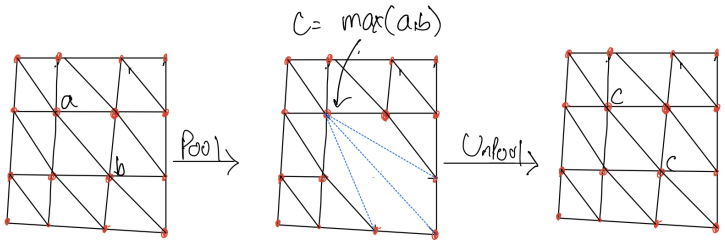


Figure 3: Very rough drawing/outline of the pooling and unpooling operation. The blue dotted line is the neighbors that have been reconnected after the merge operation. However, as you may notice, the spatial position of the vertices after pooling remains unchanged which may be an issue

	Test Accuracy (Dice Score)	**Converge Epoch**
U-Net	≈ 0.90	≈ 75
Hybrid-UNet (3 layer)	≈ 0.80	N/A

Table 1: Information on the performance of the models on CVC-ClinicDB Dataset

4 Experiment And Discussion

The experimentation was conducted using Nvidia A100s using the CVC-ClinicDB for binary image segmentation. Both the baseline U-Net and Hybrid UNet were configured to run for 200 epochs. Table 1 shows the result of the segmentation models. Based on the dice score, the proposed Hybrid-UNet did not outperform the baseline model with an accuracy of 80%. One area of potential issue that we are trying to explore is the receptive of the spline convolution. Since we are dealing with pseudo-coordinate system, if the distance between each vertex is too large and the receptive field of spline convolution is too small, the convolution might not be taking into account all the neighbors of the vertex. In addition, with mesh pooling, the spaces between each vertex is changing and it may result in spline convolution not capturing any useful neighbors in its neighborhood. To understand what the convolution is doing, consider graphing out the vertices in its 2d spatial position to see the distance between each vertices. An in-depth analysis of the the kernel function and its radius when performing convolution on pooled graphs will provide more insights into how the model is learning.

5 Future Directions

As I've mentioned in Section 4, there are quite a bit of flaws with my current implementations. The most serious of which is the runtime of the model. Currently, the model takes around 10 days to train around 200 epochs with 600 images in each epoch and this time simplifies to around 4 seconds per image which is really slow. There are also other theoretical flaws/questions in the implementation such as the receptive field of the spline convolution as well as the changes that come with the pooling step. Here are some of the things that I'll try to tackle as I dig deeper into this project.

6 Conclusion

We explored a novel approach to image representation and processing by utilizing triangular meshes for convolution and pooling operations. Despite the theoretical advantages of mesh-based methods, our experiments revealed that the proposed Hybrid-UNet architecture under-performed compared to the baseline U-Net in terms of segmentation accuracy on the CVC-ClinicDB dataset. Several factors likely contributed to this outcome, including the challenges associated with the receptive field of the spline convolution and the impact of pooling on the graph's structure.

One of the key issues identified was the sensitivity of spline convolutions to vertex spacing in the mesh, particularly after pooling operations. This raises questions about the effectiveness of the current pooling strategy, especially in sparse regions of the graph where useful neighborhood information may be lost. Further investigation into the spatial distribution of vertices and how it interacts with the convolution kernel is required to optimize the model's performance.

Moreover, the current runtime of the model is a significant bottleneck. Reducing the computational cost, particularly during the mesh generation and pooling steps, will be critical for future work. Addressing these limitations through adjustments in pooling mechanisms, as well as optimizing the kernel radius for convolution operations, will provide valuable insights into enhancing the efficiency and accuracy of mesh-based image processing models.

References

[1] M. Fey, J. E. Lenssen, F. Weichert, and H. Müller. SplineCNN: Fast Geometric Deep Learning with Continuous B-Spline Kernels, May 2018. arXiv:1711.08920 [cs].

[2] H. Gao and S. Ji. Graph U-Nets. In *Proceedings of the 36th International Conference on Machine Learning*, pages 2083–2092. PMLR, May 2019. ISSN: 2640-3498.

[3] R. Hanocka, A. Hertz, N. Fish, R. Giryes, S. Fleishman, and D. Cohen-Or. MeshCNN: A Network with an Edge. *ACM Transactions on Graphics*, 38(4):1–12, Aug. 2019. arXiv:1809.05910 [cs, stat].

[4] N. Ibtehaz and M. S. Rahman. MultiResUNet : Rethinking the U-Net Architecture for Multimodal Biomedical Image Segmentation. *Neural Networks*, 121:74–87, Jan. 2020. arXiv:1902.04049 [cs].

[5] F. Isensee, J. Petersen, A. Klein, D. Zimmerer, P. F. Jaeger, S. Kohl, J. Wasserthal, G. Koehler, T. Norajitra, S. Wirkert, and K. H. Maier-Hein. nnU-Net: Self-adapting Framework for U-Net-Based Medical Image Segmentation, Sept. 2018. arXiv:1809.10486 [cs].